SEKE2020

Proceedings of the 32nd International Conference on Software Engineering and Knowledge Engineering

PROCEEDINGS

SEKE 2020

The 32nd International Conference on
Software Engineering &
Knowledge Engineering

Sponsored by

KSI Research Inc. and Knowledge Systems Institute Graduate School, USA

Technical Program

July 9 – 19, 2020

KSIR Virtual Conference Center, Pittsburgh, USA

Organized by

KSI Research Inc. and Knowledge Systems Institute Graduate School, USA

FOREWORD

Welcome to the 32nd International Conference on Software Engineering and Knowledge Engineering (SEKE), in KSIR Virtual Conference Center, Pittsburgh, PA, USA. In the last 30 years, SEKE has established itself as a major international forum to foster, among academia, industry, and government agencies, discussion and exchange of ideas, research results and experience in software engineering and knowledge engineering. The SEKE community has grown to become a very important and influential source of ideas and innovations on the interplays between software engineering and knowledge engineering, and its impact on the knowledge economy has been felt worldwide. On behalf of the Program Committee, it is my great pleasure to invite you to participate in the technical program of SEKE.

This year, we received 160 submissions from 39 countries. Through a rigorous review process where a majority of the submitted papers received three reviews, and the rest with two reviews, we were able to select 66 full papers for the general conference (41 percent), 44 short papers (27.5 percent) and 44 rejects (27.5 percent). There are also 6 withdrawals. Out of that 100 papers are scheduled for presentation in eleven sessions during the conference.

The high quality of the SEKE 2020 technical program would not have been possible without the tireless effort and hard work of many individuals. First of all, we would like to express our sincere appreciation to all the authors whose technical contributions have made the final technical program possible. We are very grateful to all the Program Committee members whose expertise and dedication made our responsibility that much easier. Our gratitude also goes to the keynote speakers who graciously agreed to share their insight on important research issues, to the conference organizing committee members for their superb work, and to the external reviewers for their contribution.

Personally, we owe a debt of gratitude to a number of people whose help and support with the technical program and the conference organization are unfailing and indispensable. We are deeply indebted to Dr. S. K. Chang, Chair of the Steering Committee, for his constant guidance and support that are essential to pull off SEKE 2020. Our heartfelt appreciation goes to Dr. Angelo Perkusich, Federal University of Campina Grande, Brazil, the Conference Chair, for his help and experience. We also thank Dr. Ji Wu for his excellent keynote.

We would like also to express our great appreciation to all of the conference organization committee members, including the Publicity Chair, Lan Lin, Ball State University, USA and Rong Peng, Wuhan University, China. Moreover, we would like to appreciate and recognize our Conference Liaisons in different regions for their important contributions. They are: Asia Liaison – Hironori Washizaki, Waseda University, Japan; Australasia Liaison – Jing Sun, The University of Auckland, New Zealand; and India Liaison - Swapan Bhattacharya, National Institute of Technology Karnataka, Surathakl.

Last but certainly not the least, we must acknowledge the important contributions that the KSI staff members have made. Their timely and dependable support and assistance throughout the entire process have been truly remarkable. Finally, we wish you have productive discussion, great networking and effective virtual presentation to participate in SEKE 2020.

Raul Garcia Castro, Universidad Politecnica de Madrid, Spain, Program Committee Chair

SEKE 2020

The 32ⁿᵈ International Conference on Software Engineering & Knowledge Engineering

July 9 – 19, 2020

KSIR Virtual Conference Center, Pittsburgh, USA

Conference Organization

CONFERENCE CHAIR

Angelo Perkusich, Federal University of Campina Grande, Brazil

PROGRAM COMMITTEE CHAIR

Raul Garcia Castro, Universidad Politecnica de Madrid, Spain

PROGRAM COMMITTEE CO-CHAIRS

Huiqun Yu, East China University of Science and Technology, China
Kazuhiro Ogata, JAIST, Japan

STEERING COMMITTEE CHAIR

Shi-Kuo Chang, University of Pittsburgh, USA

STEERING COMMITTEE

Vic Basili, University of Maryland, USA
Bruce Buchanan, University of Pittsburgh, USA
C. V. Ramamoorthy, University of California, Berkeley, USA

ADVISORY COMMITTEE

Jerry Gao, San Jose State University, USA
Swapna Gokhale, University of Connecticut, USA
Xudong He, Florida International University, USA
Natalia Juristo, Universidad Politecnica de Madrid, Spain
Taghi Khoshgoftaar, Florida Atlantic University, USA
Guenther Ruhe, University of Calgary, Canada
Masoud Sadjadi, Florida International University, USA
Du Zhang, California State University, USA

PROGRAM COMMITTEE

Silvia Teresita Acuna, Universidad Autonoma de Madrid, Spain
Shadi Alawneh, Oakland University, USA
Vaibhav Anu, North Dakuta State University, USA
Dionysios Athanasopoulos, Queen's University of Belfast, United Kingdom
Doo-Hwan Bae, Korea Advanced Institute of Science and Technology, Korea
Kyungmin Bae, Pohang University of Science and Technology, Korea
Vita Barletta, University of Bari, Italy
Fevzi Belli, University of Paderborn, Germany
Ateet Bhalla, Consultant, India
Swapan Bhattacharya, NITK, Surathakl, India
André Pinz Borges, Federal University of Technology - Parana (UTFPR), Brazil (EEAS)
Ivo Bukovsky, Czech Technical University in Prague, Czech Republic
Guoray Cai, Penn State University, USA
Rafael Cardoso, University of Liverpool, United Kingdom (EEAS)
Raul Garcia Castro, Universidad Politecnica de Madrid, Spain
Keith Chan, Hong Kong Polytechnic University, Hong Kong
Kuang-nan Chang, Eastern Kentucky University, USA
Meiru Che, University of Texas at Austin, USA
Wen-Hui Chen, National Taipei University of Technology, Taiwan
Xiangping Chen, Sun Yat-sen University, China (IPA)
Fabio M. Costa, Universidade Federal de Goias, Brazil
Maria Francesca Costabile, University of Bari, Italy
Andrea De Lucia, University of Salerno, Italy (IPA)
Lin Deng, Towson University, USA
Derek Doran, Wright State University, USA
Weichang Du, University of New Brunswick, Canada
Ashish Kumar Dwivedi, National Instititute of Technology, India (IPA)
Abdelrahman Osman Elfaki, University of Tabuk, Saudi Arabia
Iaakov Exman, Jerusalem College of Engineering, Israel
Fumiyo Fukumoto, University of Yamanashi, Japan (IPA)
Honghao Gao, ShangHai University, China
Olivier Le Goaer, University of Pau, France
Swapna Gokhale, Univ. of Connecticut, USA
Wolfgang Golubski, Zwickau University of Applied Sciences, Germany
Desmond Greer, Queen's University Belfast, United Kingdom
Hao Han, National Institute of Informatics, Japan
Xudong He, Florida International University, USA
Hamdy Ibrahim, University of Calgary, Canada
Bassey Isong, North-West University, South Africa
Clinton Jeffery, University of Idaho, USA
Jason Jung, Chung-Ang University, South Korea
Pankaj Kamthan, Concordia University, Canada
Ananya Kanjilal, B.P. Poddar Institute of Technology and Management, India
Taghi Khoshgoftaar, Florida Atlantic University, USA
Jun Kong, North Dakota State University, USA
Vinay Kulkarni, Tata Consultancy Services, India
Meira Levy, Shenkar College of Engineering and Design, Israel
Bixin Li, Southeast University, China
Xin Li, Google Inc., USA
Yingling Li, Chengdu University of Information Technology, China, (IPA)
Jianhua Lin, Eastern Connecticut State University, USA
Lan Lin, Ball State University, USA
Xiaodong Liu, Edinburgh Napier University, United Kingdom

Luanna Lopes Lobato, Federal University of Goias, Brazil
Jiawei Lu, Zhejiang University of Technology, China (IPA)
Baojun Ma, Shanghai International Studies University, China
Beatriz Marin, Universidad Diego Portales, Chile
Riccardo Martoglia, University of Modena and Reggio Emilia, Italy
Santiago Matalonga, University of the West of Scotland, UK
Andre Menolli, Universidade Estadual do Norte do Parana (UENP), Brazil
Hiroyuki Nakagawa, Osaka University, Japan
Alex Norta, Tallinn University of Technology, Estonia
Kazuhiro Ogata, JAIST, Japan
Edson A. Oliveira Jr., State University of Maringa, Brazil
Carlos Eduardo Pantoja, Federal Center for Technological Education (CEFET-RJ), Brazil (EEAS)
George Angelos Papadopoulos, University of Cyprus, Cyprus
Rong Peng, Wuhan University, China
Oscar Mortagua Pereira, University of Aveiro, Portugal
Rick Rabiser, Johannes Kepler University, Austria
Filip Radulovic, Universidad Politecnica de Madrid, Spain
Claudia Raibulet, University of Milan, Italy
Damith C. Rajapakse, National University of Singapore, Singapore
Rajeev Raje, IUPUI, USA
Marek Reformat, University of Alberta, Canada
Diogo Regateiro, Institute de Telecomunicacoes, Portugal
Stephan Reiff-Marganiec, Leicester University, United Kingdom
Robert Reynolds, Wayne State University, USA
Ignacio Garcia Rodriguez De Guzman, University of Castilla-La Mancha, Spain
Daniel Rodriguez, Universidad de Alcala, Spain
Claudio Sant'Anna, Universidade Federal da Bahia, Brazil
Klaus-Dieter Schewe, SCCH, Austria
Abdelhak-Djamel Seriai, University of Montpellier 2 for Sciences and Technology, France
Michael Shin, Texas Tech University, USA
Vijayan Sugumaran, Oakland University, USA
Jing Sun, University of Auckland, New Zealand
Meng Sun, Peking University, China
Yanchun Sun, Peking University, China
Xiaobing Sun, Yangzhou University, China
Gerson Sunye, University of Nantes, France
Kumiko Tadano, NEC, Japan
Chuanqi Tao, Nanjing University of Science and Technology, China
Jeff Tian, Southern Methodist University, USA
Mark Trakhtenbrot, Holon Institute of Technology, Israel
Peter Troeger, TU Chemnitz, Germany
Christelle Urtado, LGI2P Ecole des Mines d'Ales, France
Sylvain Vauttier, Ecole des mines d'Ales, France
Gleifer Vaz Alves, Federal University of Technology - Parana (UTFPR), Brazil (EEAS)
Silvia Vergilio, Federal University of Parana (UFPR), Brazil
Gennaro Vessio, University of Bari, Italy
Aaron Visaggio, University of Sannio, Italy
José Viterbo, Fluminense Federal University (UFF), Brazil (EEAS)
Jiaojiao Wang, China Communication University of Zhejiang, China (IPA)
Ye Wang, Zhejiang Gongshang University, China
Yong Wang, New Mexico Highlands University, USA
Zhongjie Wang, Harbin Institute of Technology, China
Hironori Washizaki, Waseda University, Japan
Bingyang Wei, Midwestern State University, USA
Guido Wirtz, Bamberg University, Germany
Franz Wotawa, TU Graz, Austria
Peng Wu, Institute of Software, Chinese Academy of Sciences, China

Qing Wu, Hangzhou Dianzi University, China
Frank Weifeng Xu, University of Baltimore, USA
Haiping Xu, University of Massachusetts Dartmouth, USA
Lai Xu, Bournemouth University, UK
Guowei Yang, Texas State University, USA
Yuyu Yin, Hangzhou Dianzi University, China
Dongjin Yu, Hangzhou Dianzi University, China
Huiqun Yu, East China University of Science and Technology, China
Jiang Yue, Fujian Normal University, China
Fiorella Zampetti, University of Sannio, Italy
Du Zhang, Macau University of Science and Technology, China
Pengcheng Zhang, Hohai University, China
Yong Zhang, Tsinghua University, China
Zhenyu Zhang, Institute of Software, Chinese Academy of Sciences, China
Nianjun Zhou, IBM, USA
Huibiao Zhu, East China Normal University, China
Eugenio Zimeo, University of Sannio, Italy
Eugenio Zimeo, University of Sannio, Italy

PUBLICITY CHAIR

Lan Lin, Ball State University, USA
Rong Peng, Wuhan University, China

ASIA LIAISON

Hironori Washizaki, Waseda University, Japan

AUSTRALASIA LIAISON

Jing Sun, The University of Auckland, New Zealand

EUROPE LIAISON

Raul Garcia Castro, Universidad Politecnica de Madrid, Spain

INDIA LIAISON

Swapan Bhattacharya, National Institute of Technology Karnataka, Surathakl, India

Keynote

Looking into the Evidences for Claiming System Quality

Professor Ji Wu
School of Computer Science and Engineering
Beihang University
China

Abstract

IT system is getting complicated and evolving quickly. As IT system plays at more and more scenarios to improve human life experiences, people really care about its quality. In general, the quality of IT system is not a trivial problem. In software engineering domain, people deploy tool chains to collect data from software development activities, which can increase trust of the quality. In this talk, I will present the motivation of applying knowledge engineering to construct model to better extract evidences from software artefacts to claim system quality. And I will also discuss some of the challenges.

About the Speaker

Ji Wu, Dr., Associate Professor from the School of Computer Science and Engineering in Beihang University, which is well-known in China and strong at engineering complex systems, in particular in the domain of avionics and aeronautics. My personal research interests include software modeling, verification and testing. I am interested in industry problem driven research, and collaborate with industry partners in most of the research projects. Recently, I focus on the methodology of testing and assessing the autonomous system such as drone swarm. I teach the course of object-oriented design and construction for bachelor students, we propose the competitive learning model and use the cloud deployed testing services to assure the quality of assignments. I also teach the course of software modeling for graduate students.

Table of Contents

Session SS: Software Specification

Session ESE: Empirical Software Engineering

Session DSML: Distributed Systems and Machine Learning

Session SD: Software Development

Session ST: Software Testing

Session NLP: Natural Language Processing

Session SMM: Social Media Mining

Session RS: Recommender Systems

Session SB: Security and Blockchain

Note: (S) denotes a short paper.

Formalization and Verification of VANET

Ran Li[1], Huibiao Zhu[*1], Lili Xiao[1], Jiaqi Yin[1], Yuan Fei[2], Gang Lu[*1]

[1]Shanghai Key Laboratory of Trustworthy Computing,
East China Normal University, Shanghai, China
[2]School of Information, Mechanical and Electrical Engineering,
Shanghai Normal University, Shanghai, China

Abstract—Vehicular Ad Hoc Network (VANET) is a subclass of Mobile Ad Hoc Network (MANET) types. As a key part of the Intelligent Transportation Systems (ITSs) framework, it can be used not only to provide value added services, but also to guarantee the security of ITS. Since VANET is extensively applied, its security is of great significance.

In this paper, we model the architecture of VANET using process algebra Communicating Sequential Processes (CSP). By utilizing model checker Process Analysis Toolkit (PAT), we verify five properties (deadlock freedom, divergence freedom, data leakage, vehicle faking and RSU faking) of the model and find that the proposed architecture may cause data leakage. Hence, we improve the model by encrypting the messages with receiver's public key to cope with this problem. The new verification results show that our study can guarantee the security of VANET.

Index Terms—VANET; Security; CSP; Modeling; Verification

I. INTRODUCTION

While the rapid growth of driverless vehicles has been fueled by the development of vehicle industry and wireless communication technology, VANET is actually the supporting infrastructure and paves the way for driverless vehicles.

VANET is the application of traditional MANET in the field of intelligent traffic [1]. The basic architecture of VANET is demonstrated in Fig. 1. Each vehicle has an On Board Unit (OBU) and one or more Application Units (AUs). Road Side Unit (RSU) is the device installed along road side and communicates with OBUs using Dedicated Short Range Communication (DSRC) technology. Moreover, RSU can also get in touch with the gateway to the Internet.

Fig. 1 illustrates that communication in VANET can be divided into three domains: in-vehicle domain, ad hoc domain and infrastructure domain [1]. In-vehicle domain contains an OBU and one or more AUs. Infrastructure domain communication is between RSUs and infrastructure. Ad hoc domain is composed of two types of communication. One is Vehicle-to-Vehicle (V2V) communication which means a vehicle can connect with other vehicles, and the other is Vehicle-to-Infrastructure (V2I) communication which represents that a vehicle can exchange information with infrastructure.

Since VANET is an emergent technology, it has massive challenges of security issues. Many studies have been carried out on the topic of communications security in VANET [2]–[4]. However, the previous solutions cannot comfirm the

*Corresponding authors: hbzhu@sei.ecnu.edu.cn (H. Zhu),
glu@cs.ecnu.edu.cn (G. Lu).

confidentiality-integrity-availability (CIA) property [5]. To secure communications in VANET, a newly proposed architecture uses end-to-end authentication to avoid intrusion in VANET and considers VANET as a hierarchical model to decrease the number of message exchanges [5]. However, the proposed architecture is not verified formally.
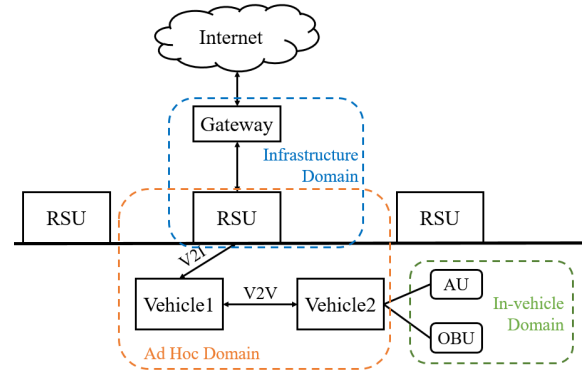
Fig. 1. Basic Architecture of VANET

In this paper, we formalize the model of the proposed architecture of VANET [5] using process algebra CSP [6], [7]. Additionally, we use model checking tool PAT [8], [9] to verify some related properties, including deadlock freedom, divergence freedom, data leakage, vehicle faking and RSU faking. From the verification results, we find that the original proposed architecture [5] is not safe and it may cause data leakage problem. Hence, we modify the original model by means of encrypting messages with receiver's public key to solve this problem. The verification results of the improved model show that the modification is truly effective.

The remainder of this paper is organized as follows. In Section II, we give a brief introduction to VANET which contains the proposed architecture [5] and the flow of communications between entities. Besides, the process algebra CSP is also introduced briefly in this section. Section III presents the formalized model of VANET [5]. Furthermore, Section IV is about the verification results of the original model and we also put forward some improvements of the model. We conclude our work and propose the future work in Section V.

II. BACKGROUND

In this section, we briefly explain the proposed architecture of VANET [5] and focus on the communication flow of it. In addition, an introduction to CSP is given as well.
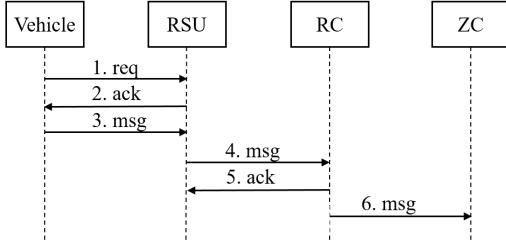
Fig. 2. In Ad Hoc and Infrastructure Domain
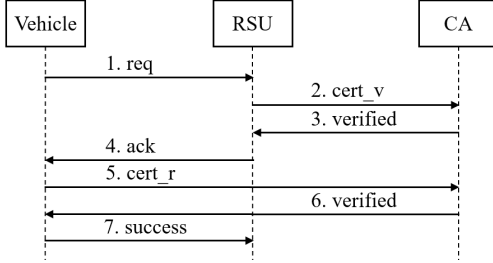
Fig. 3. In-vehicle Domain (Adapted from[5])

A. VANET

In order to ensure communications security, a newly proposed architecture adopts end-to-end authentication to avoid intrusion in VANET and considers VANET as a hierarchical model to decrease the number of message exchanges.

The proposed architecture covers five entities: Vehicle, RSU, RSU Controller, Zone Controller and CA. Vehicle and RSU are the same as the entities in the basic architecture mentioned above, while RSU Controller, Zone Controller and CA are newly introduced entities in VANET.

- **RSU Controller:** It is in charge of the communications in an area with a number of RSUs.
- **Zone Controller:** It controls multiple RSU Controllers.
- **CA (Certification Authority):** It is responsible for distributing the certificates to RSUs, RSU controllers and Zone controllers. Each certificate contains the entity's ID, public key and expiry time.

To ensure security applications in ad hoc domain communication and infrastructure domain communication, the proposed architecture emphasizes end-to-end authentication process using certificates from CA. Fig. 2 shows the flow of this solution.

1. The vehicle sends a request to the RSU.

2. Once a RSU receives this request, it checks the expiry time and the threshold value of the message control. If they are both valid, it returns its ID and the timestamp. If not, it broadcasts an alarm notification.

3. Then, the vehicle executes Deffie-Hellman key exchange [10] with the RSU and uses the shared key to encrypt the message with Elliptic Curve Cryptography [11].

4. Afterwards, the RSU decrypts the message and passes it along with timestamp forward to the RSU controller.

5. If the timestamp is valid, the RSU controller returns its certificate with an ack to the RSU. It also proceeds to encrypt

and sends the message along with its certificate to the Zone controller. Otherwise, the RSU controller broadcasts an alarm notification, i.e., it fails.

6. When the Zone controller receives the message, it checks the timestamp at first. If the timestamp is valid, the Zone controller decrypts the message and sends the message to other Zone controllers and base stations. Otherwise, it fails.

On the other hand, to ensure security in in-vehicle communication, the proposed architecture emphasizes the verification of the certificates. The flow is shown in Fig. 3. Here, RSU in Fig. 3 can also be a service provider. Due to the similarity in flows of them, we list the flow of the communication among the vehicle, the RSU and CA.

1. The OBU of a vehicle sends a service request which includes the certificate of the vehicle.

2. A RSU sends the certificate of the vehicle to CA to check its validity.

3. CA returns the verification result.

4. Once the RSU receives the verification result which shows the certificate is valid, it sends its own certificate to the vehicle along with an ack. If not, it refuses the request.

5. Then, the vehicle sends the received certificate to CA to check its validity.

6. CA returns the verification result.

7. Once the vehicle receives the verification result which shows the certificate is valid, the vehicle can use the service successfully and safely in AU. Otherwise, it rejects the service.

B. CSP

CSP, the abbreviation of Communicating Sequential Processes, is a process algebra proposed by C. A. R. Hoare [6]. The syntax of CSP provides many operators to express the actions of processes and their interactions. We give a brief definition of the syntax used in this paper as below.

$$P, Q ::= Skip \mid a \rightarrow P \mid c?x \rightarrow P \mid c!u \rightarrow P$$
$$\mid P\|Q \mid P\Box Q \mid P \lhd B \rhd Q \mid P[[a \leftarrow b]]$$

- $Skip$ denotes that a process terminates successfully.
- $a \rightarrow P$ describes an object which first engages in the event a and then behaves exactly as described by P.
- $c?x \rightarrow P$ represents that a process receives a message through a channel called c and assigns the value of the message to x, and then behaves like process P.
- $c!u \rightarrow P$ means that a process sends message u through a channel called c and then behaves like process P.
- $P\|Q$ indicates process P executes in parallel with process Q.
- $P\Box Q$ stands for external choice, which means that a process performs like P or Q and the choice is determined by the environment.
- $P \lhd B \rhd Q$ expresses conditional choice. If B is true, then the process behaves like P, otherwise behaves like Q.
- $P[[a \leftarrow b]]$ is the syntax of renaming and signifies an event a is replaced by b in process P.

III. MODELING VANET

In this section, we give the formalized model of VANET. First, we introduce the definitions of sets, messages and channels. On this basis, we formalize the model of VANET using CSP.

A. Sets, Messages and Channels

First, some sets are introduced in our formalized model for convenience. For communicators, we define the set **Vehicles** of the Vehicle components, **RSUs** of the RSUs and **Controllers** of the RSU Controller and the Zone Controller components.

Entity set involves entities mentioned above. Further, each entity has its own certificates, ID and keys. **Cert** is the set of certificates. **Id** is the set of IDs and **Key** is the set of keys. $Key = PUBK \cup PRIK$, where $PUBK$ denotes the public keys and $PRIK$ represents the private keys.

Moreover, we give definitions of **Content** of the content, **T** of the time and **State** of the states which contain true state and false state.

Based on the sets above, the messages are further abstracted as follows.

$$MSG_{cert} =_{df} \{msg_{reqv}.a.b.cert, msg_{rspv}.a.b.state|$$
$$a, b \in Entity \cup CA, cert \in Cert, state \in State\}$$
$$MSG_{prdc} =_{df} \{msg_{prdc}.a.b.E(k, c)|$$
$$a, b \in Entity \cup CA, k \in Key, c \in Content\}$$
$$MSG_{reqb} =_{df} \{msg_{reqb}.a.b.id.t|a, b \in Entity, id \in Id, t \in T\}$$
$$MSG_{reqs} =_{df} \{msg_{reqs}.a.b.rid.vid.cert|$$
$$a, b \in Entity, rid, vid \in Id, cert \in Cert\}$$
$$MSG_{ack} =_{df} \{msg_{ackr}.a.b.rid.ack.cert, msg_{ackrc}.a.b.ack.cert|$$
$$a, b \in Entity, rid \in Id, ack \in State, cert \in Cert\}$$
$$MSG_{data} =_{df} \{msg_{d1}.a.b.E(k, c), msg_{d2}.a.b.E(k, c).cert.t|$$
$$a, b \in Entity, k \in Key, c \in Content, cert \in Cert, t \in T\}$$
$$MSG_{c1} =_{df} MSG_{cert} \cup MSG_{prdc} \cup MSG_{reqb}$$
$$\cup MSG_{data} \cup MSG_{ack}$$
$$MSG_{c2} =_{df} MSG_{cert} \cup MSG_{reqs} \cup MSG_{ack}$$

MSG_{c1} is the set of messages in ad hoc and infrastrusture domain and MSG_{c2} consists of messages communicated in in-vehicle domain.

Besides, we use the symbols **E** and **D** to represent encryption function and decryption function respectively.

- **E**(k,msg) indicates that k is the key which is used to encrypt the message msg.
- **D**(k^{-1},$E(k, msg)$) means that the corresponding decryption key k^{-1} can decrypt the message which is encrypted with k.

Then, we give the definitions of channels.

- channels of processes between legal entities, denoted by **COM_PATH**:
  $ComVC, ComRC, ComRcC, ComZcC,$
  $ComVR, ComRRc, ComRcZc$
- channels of intruders faking/intercepting processes, represented by **INTRUDER_PATH**:
  $FakeVR, InterceptRV, FakeRV, InterceptVR,$
  $FakeRRc, InterceptRcR, FakeRcR, InterceptRRc,$
  $FakeRcZc, InterceptZcRc, FakeZcRc, InterceptRcZc$

- channel of synchronization time: $Time$

The declarations of the channels are as follows.

Channel $COM\_PATH, INTRUDER\_PATH$: **MSG**

B. Overall Modeling

We formalize the whole model as below. $VANET_0$ represents the system without considering intruders, while $VANET$ takes account of the attacks from intruders. The channels of our model are shown in Fig. 4.

$$VANET =_{df} VANET_0 \, [|INTRUDER\_PATH|] \, Intruder$$
$$VANET_0 =_{df} Vehicle \, \|RSU \, \|RC \, \|ZC \, \|CA \, \|Clock$$

$Vehicle$, RSU, RC, ZC and CA describe the performance of vehicles, RSUs, RSU controllers, Zone controllers and CA respectively. The process called $Clock$ is used to realize the synchronization of time. Additionally, we use the process $Intruder$ to simulate intruders' actions, such as intercepting or faking messages.
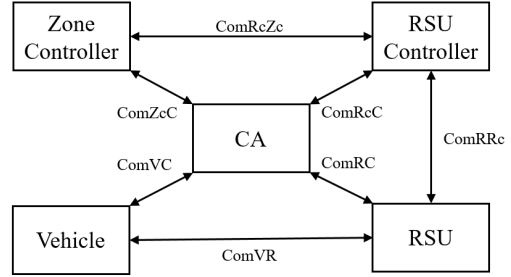
Fig. 4. Channels of VANET

C. Clock Modeling

When entities deliver messages, they need to check whether the timestamp is valid. So we define $Clock$ in order to realize the synchronization of all entities. The $Clock$ process is responsible for recording time and returning the current time whenever some entities want it.

$$Clock(t) =_{df} tick \rightarrow Clock(t + 1) \,\square\, Time!t \rightarrow Clock(t)$$

D. CA Modeling

CA provides the identities and certificates for all RSUs, RSU controllers and Zone controllers. Besides, CA can verify the entities by certificate verification. The model of CA is given as follows.

$$CA =_{df} ComRC!msg_{prdc}.C.R.E(ca_{kpri}, cert_r) \rightarrow CA$$
$$\square ComRcC!msg_{prdc}.C.Rc.E(ca_{kpri}, cert_{rc}) \rightarrow CA$$
$$\square ComZcC!msg_{prdc}.C.Zc.E(ca_{kpri}, cert_{zc}) \rightarrow CA$$
$$\square ComRC?msg_{reqv}.R.C.cert_v \rightarrow$$
$$verified := valid(cert_v) \rightarrow$$
$$ComRC!msg_{rspv}.C.R.cert_v.verified \rightarrow CA$$
$$\square ComVC?msg_{reqv}.V.C.cert_r \rightarrow$$
$$verified := valid(cert_r) \rightarrow$$
$$ComVC!msg_{rspv}.C.V.cert_r.verified \rightarrow CA$$

$valid(cert)$ is a function to verify whether the certificate is valid. If the certificate is valid, it returns $true$ and then the variable $verified$ is set to true.

E. Vehicle Modeling

We formalize the process $Vehicle$ using general choice $\square$ to describe the two types of communication mentioned above. The model of $Vehicle$ is shown as below.

$$Vehicle_0 =_{df} ComVR!msg_{reqb}.V.R.vid.t_{exp} \rightarrow$$
$$ComVR?msg_{reqb}.R.V.rsu_{id}.t_s \rightarrow$$
$$dh\_key\_change \rightarrow$$
$$ComVR!msg_{data1}.V.R.E(k_{dh}, msg) \rightarrow Vehicle_0$$
$$\square \quad ComVR!msg_{reqs}.V.R.rsu_{id}.vid.cert_v \rightarrow$$
$$ComVR?msg_{ackr}.R.V.rsu_{id}.ack.cert_r \rightarrow$$
$$\left(\begin{pmatrix} ComVC!msg_{reqv}.V.C.cert_r \rightarrow \\ ComVC?msg_{rspv}.C.V.cert_r.verified \rightarrow \\ (success \rightarrow Vehicle_0) \\ \lhd(verified == true) \rhd (\ fail \rightarrow Vehicle_0 \ ) \\ \lhd(ack == true) \rhd (\ fail \rightarrow Vehicle_0) \end{pmatrix} \right)$$

In the first half of the model, we describe the behaviors of the vehicle in ad hoc or infrastructure domain and they correspond to Steps 1-3 in Fig. 2. t_{exp} is the expiry timestamp and t_s records when the RSU sends the reply. We also define an event called dh_key_change which denotes that the vehicle executes Deffie-Hellman key exchange with the RSU. After the execution, the vehicle and the RSU have a shared key k_{dh}, which is used to encrypt the message with Elliptic Curve Cryptography. The remaining actions correspond to Steps 1, 4, 5 and 6 in Fig. 3 and represent the actions of the vehicle in in-vehicle communication.

Since we have given the model without intruders, then we need to consider the existence of intruder actions. For example, we should allow intruders to fake or intercept messages. We describe the behavior of the intruder via renaming as follows and the channels that intruders involved are shown in Fig. 5.

$$Vehicle =_{df} Vehicle_0[[$$
$$ComVR!\{|ComVR|\} \leftarrow ComVR!\{|ComVR|\},$$
$$ComVR!\{|ComVR|\} \leftarrow InterceptVR!\{|ComVR|\},$$
$$ComVR?\{|ComVR|\} \leftarrow ComVR?\{|ComVR|\},$$
$$ComVR?\{|ComVR|\} \leftarrow FakeRV?\{|ComVR|\}]]$$
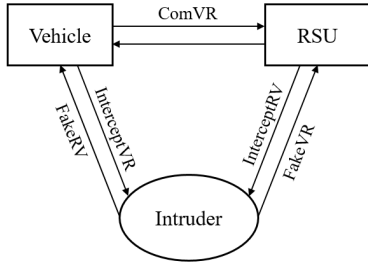
Fig. 5. Channels with an Intruder

$\{|c|\}$ is the symbol which denotes that the set of all communications over the channel c. The first two formulas represent that the process performs either a $ComVR!msg$ or $InterceptVR!msg$ event whenever $Vehicle_0$ performs a corresponding $ComVR!msg$ event. Similarly, the last two formulas imply that when the $ComVR!msg$ event occurs in $Vehicle_0$, the process $Vehicle$ behaves like $ComVR?msg$ or $FakeRV?msg$.

F. RSU Modeling

We formalize RSU process as below.

$$RSU_0 =_{df} ComRC?msg_{prdc}.R.C.E(ca_{kpri}, cert_r) \rightarrow$$
$$ComVR?msg_{reqb}.V.R.vid.t_{exp} \rightarrow Time?t \rightarrow$$
$$\left(\begin{pmatrix} ComVR!msg_{reqb}.R.V.rsu_{id}.t_s \rightarrow \\ \{msg_{cnt} := msg_{cnt} + 1\} \rightarrow dh\_key\_change \rightarrow \\ ComVR?msg_{data1}.V.R.E(k_{dh}, msg) \rightarrow \\ \begin{pmatrix} ComRRc!msg_{data2}.R.Rc. \\ E(r_{kpri}, msg).cert_{rsu}.t_v \rightarrow \\ ComRRc?msg_{ackrc}.Rc.R.ack.cert_{rc} \\ \{msg_{cnt} := msg_{cnt} - 1\} \rightarrow RSU_0 \end{pmatrix} \\ \lhd(D(k, E(k_{dh}, msg)))\rhd \\ (fail\{msg_{cnt} := msg_{cnt} - 1\} \rightarrow RSU_0) \\ \lhd(msg_{cnt} < m_{threshold} \wedge t < t_{exp}) \rhd (fail \rightarrow RSU_0) \end{pmatrix} \right)$$
$$\square \quad ComVR?msg_{reqs}.V.R.rsu_{id}.vid.cert_v \rightarrow$$
$$ComRC!msg_{reqv}.R.C.cert_v \rightarrow$$
$$ComRC?msg_{rspv}.C.R.cert_v.verified \rightarrow$$
$$\left(\begin{pmatrix} ack := true \rightarrow \\ ComVR!msg_{ackr}.R.V.rsu_{id}.ack.cert_r \rightarrow RSU_0 \end{pmatrix} \right)$$
$$\lhd(verified == true) \rhd (fail \rightarrow RSU_0)$$

msg_{cnt} is the current number of messages in this RSU. $m_{threshold}$ denotes the threshold value of message control and t_v indicates the validity time duration. For the RSU, in ad hoc or infrastructure domain, the actions on channels $ComVR$ and $ComRRc$ correspond to Steps 2-5 in Fig. 2. While, in in-vehicle communication, the actions on channels $ComVR$ and $ComRC$ correspond to Steps 1-4 in Fig. 3.

The corresponding process RSU is formalized using renaming which is similar to the process $Vehicle$, so we leave out details here.

G. Controller Modeling

Controllers are only used in ad hoc or infrastructure domain. The actions of the RSU controller on channels $ComRRc$ and $ComRcZc$ correspond to Steps 4-6 in Fig. 2 and the action of the Zone controller on channel $ComRcZc$ corresponds to Step 6 in Fig. 2. The models of $RSUController$ and $ZoneController$ are shown as below.

$$RC_0 =_{df} ComRcC?msg_{prdc}.C.Rc.E(ca_{kpri}, cert_{rc}) \rightarrow$$
$$ComRRc?msg_{data2}.R.Rc.E(r_{kpri}, msg).cert_{rsu}.t_v \rightarrow$$
$$Time?t \rightarrow$$
$$\left(\begin{pmatrix} ComRRc!msg_{ackr}.Rc.R.ack.cert_{rc} \rightarrow \\ ComRcZc!msg_{data1}.Rc.Zc. \\ E(rc_{kpri}, E(r_{kpri}, msg)).t_r \rightarrow RC_0 \end{pmatrix} \right)$$
$$\lhd(t < t_v) \rhd (fail \rightarrow RC_0)$$

$$ZC_0 =_{df} ComZcC?msg_{prdc}.C.Zc.E(ca_{kpri}, cert_{zc}) \rightarrow$$
$$ComRcZc?msg_{data1}.Rc.Zc.$$
$$E(rc_{kpri}, E(r_{kpri}, msg)).t_r \rightarrow Time?t \rightarrow$$
$$\left(\begin{pmatrix} \begin{pmatrix} (success \rightarrow ZC_0) \\ \lhd(D(r_{kpub}, E(r_{kpri}, msg)))\rhd \\ (fail \rightarrow ZC_0) \end{pmatrix} \\ \lhd(D(rc_{kpub}, E(rc_{kpri}, E(r_{kpri}, msg))))\rhd \\ (fail \rightarrow ZC_0) \end{pmatrix} \right)$$
$$\lhd(t < t_r) \rhd (fail \rightarrow ZC_0)$$

Similarly, the processes RC and ZC are built using renaming and due to the space limit, we omit the details here.

H. Intruder Modeling

We formalize the intruder as a process which can carry out attacks such as intercepting and faking messages from valid communications. Firstly, we have a set **Fact**, containing all facts an intruder can learn.

$$Fact =_{df} \{Entity,\ CA\} \cup Cert \cup T \cup \{I_{kpri},\ I_{kpub}\} \cup$$

$$\{k, c | k \in Key, c \in Content\} \cup \{E(k, c) | k \in Key, c \in Content\}$$

Then we define how the intruder deduces new fact f from given fact set F. We use the symbol $F \mapsto f$ to represent that f can be deduced from the set F. The detailed definition is given as follows. The first two rules indicate encryption and decryption. The last rule means that if the intruder can deduce the fact f from the fact set F and F is the subset of F', the f can also be derived from F'.

$$\{k, c\} \mapsto E(k, c) \quad \{k^{-1}, E(k, c)\} \mapsto c$$

$$F \mapsto f \wedge F \subseteq F' \Rightarrow F' \mapsto f$$

We also introduce a function $Info(m)$, which implies the facts learned by the intruders if they intercept messages.

$$Info(msg_{reqv}.a.b.cert) =_{df} \{a, b, cert\}$$

$$Info(msg_{rspv}.a.b.state) =_{df} \{a, b, state\}$$

$$Info(msg_{prdc}.a.b.E(k, c)) =_{df} \{a, b, E(k, c)\}$$

$$Info(msg_{reqb}.a.b.id.t) =_{df} \{a, b, id, t\}$$

$$Info(msg_{d1}.a.b.E(k, c)) =_{df} \{a, b, E(k, c)\}$$

$$Info(msg_{d2}.a.b.E(k, c).cert.t) =_{df} \{a, b, E(k, c), cert, t\}$$

$$Info(msg_{reqs}.a.b.rid.vid.cert) =_{df} \{a, b, rid, vid, cert\}$$

$$Info(msg_{ackr}.a.b.rid.ack.cert) =_{df} \{a, b, rid, ack, cert\}$$

$$Info(msg_{ackrc}.a.b.ack.cert) =_{df} \{a, b, ack, cert\}$$

Then, we add a definition of $Deduce$ channel which is used to deduce new facts.

$$Channel\ Deduce : FACT.P(FACT)$$

Based on this, we give the formalization of $Intruder$ which is parameterized by the facts he knows.

$$Intruder(I) =_{df}$$

$$\square_{msg \in MSG} Intercept.msg \rightarrow Intruder_0(I \cup Info(msg))$$

$$\square\square_{msg \in MSG \cap Info(msg) \in I} Fake.msg \rightarrow Intruder_0(I)$$

$$\square\square_{f \in Fact, f \notin F, F \mapsto f} Init\{data\_leakage\_success := false\}$$

$$\rightarrow Deduce.f.F \rightarrow \left(\begin{array}{c} \left(\begin{array}{c} data\_leakage\_success = true \\ \rightarrow Intruder_0(F \cup \{f\}) \end{array} \right) \\ \triangleleft(f == Data)\triangleright \\ \left(\begin{array}{c} data\_leakage\_success = false \\ \rightarrow Intruder_0(F \cup \{f\}) \end{array} \right) \end{array} \right)$$

If the intruder intercepts the message msg, then it adds $info(msg)$ into his facts. Also, if the intruder knows $info(msg)$, then he can pretend as a legal entity and fake the message msg. Further, as introduced before, the intruder can deduce new facts from the known facts as well.

For $Intruder_0$, we define its parameter as IK, which stands for the intruder's initial knowledge.

$$Intruder =_{df} Intruder_0(IK),\ IK =_{df} \{Entity, I_{kpub}, I_{kpri}\}$$

IV. VERIFICATION AND IMPROVEMENT

In this section, we use model checker PAT to verify the properties of the above constructed model. Moreover, we propose an improved model according to the verification results.

A. Properties Verification

The descriptions and the corresponding assertions of specific security properties are given below.

Property 1: Deadlock Freedom

Deadlock is a situation in which nothing further can happen. This property means that we need to ensure the model cannot get stuck in a deadlock state. In the tool PAT, we use a primitive to describe this situation.

$$\#assert\ VANET\ deadlockfree;$$

Property 2: Divergence Freedom

Divergence is a phenomenon in which a process has an infinite loop or unguarded recursion. To ensure that the model is well defined, we need to check if the model is divergence free. We complete the check by means of a primitive in PAT.

$$\#assert\ VANET\ divergencefree;$$

Property 3: Data Leakage

We also verify whether the intruder can obtain the message successfully since this property is relevant to the security of VANET. The assertion is set to check it.

$$\#define\ Data\_Leakage\_Success$$
$$data\_leakage\_success == true;$$
$$\#assert\ VANET\ reaches\ Data\_Leakage\_Success;$$

Property 4: Vehicle Faking

This property means that the system is unaware that an intruder has succeeded in posing as a legal Vehicle successfully. The assertions are listed as follows, where $vehicle\_fake\_success1$ and $vehicle\_fake\_success2$ are boolean variables defined to verify whether the intruder succeeded in in-vehicle communication and in ad hoc or infrastructure domain respectively.

$$\#define\ Vehicle\_Fake\_Success$$
$$vehicle\_fake\_success1 || vehicle\_fake\_success2 == true$$
$$\#assert\ VANET\ reaches\ Vehicle\_Fake\_Success;$$

Property 5: RSU Faking

Analogously, this property means that an intruder has disguised as a legal RSU successfully. $rsu\_fake\_success1$ and $rsu\_fake\_success2$ are used to check whether the intruder succeeded in in-vehicle communication and in ad hoc or infrastructure domain. The related assertions are shown as below.

$$\#define\ RSU\_Fake\_Success$$
$$rsu\_fake\_success1 || rsu\_fake\_success2 == true;$$
$$\#assert\ VANET\ reaches\ RSU\_Fake\_Success;$$

Verification Results

The verification results are shown in Fig. 6.

- Property 1 and Property 2 are valid, indicating that our model can never run into a deadlock state and is well defined.
- Property 3 is valid. It means the intruder has acquired the message successfully, i.e., data security of the system is not guaranteed. Therefore, we put forward the improvement next.
- Property 4 and Property 5 are invalid, which represents that the intruder can never pretend as a legal vehicle or a legal RSU successfully.

Fig. 6. Verification Results of the Model

Fig. 7. Verification Results of the Improved Model

B. Attack and Improvement

As illustrated in the verification results above, Property 3 is valid. It indicates that the system still has security risk in spite of the usage of digital signature and encryption. After executing Deffie-Hellman key exchange with the RSU, the vehicle sends the message encrypted with the key shared with the RSU. Later, the RSU uses the shared key to decrypt the message. After decrypting, the RSU encrypts the decrypted message with its own private key. Then the RSU sends the encrypted message along with its certificate to the RSU controller. Once the intruder gets the RSU's public key, he can decrypt the message and obtain it. An example which causes data leakage is presented as below.

$$ComVR.msg_{data1}.V.R.E(k_{dh}, msg)$$
$$\rightarrow InterceptRRc.msg_{data2}.R.Rc.E(r_{kpri}, msg).cert_{rsu}.t_v$$
$$\rightarrow FakeRcR.msg_{ackrc}.Rc.R.ack.cert_{rc}$$

First, the vehicle sends the encrypted message to RSU through the channel $ComVR$. Then the intruder intercepts this message through the channel $InterceptRRc$. Since every entity in the system knows the public key of CA, the intruder can decrypt the certificate of the RSU with CA's public key. Also, as the certificate contains the entity's public key, the intruder can access the public key and obtain the message.

In order to overcome the above problem, we modify the architecture by encrypting the original encrypted message with the receiver's public key. Thus, we replace MSG_{data} defined before with the new following definition.

$$MSG_{data} =_{df} \{msg_{data1}.a.b.E(k1, E(k2, c)),$$
$$msg_{data2}.a.b.E(k1, E(k2, c)).cert.t \mid$$
$$a, b \in Entity, k1 \in PUBK, k2 \in PRIK,$$
$$c \in Content, cert \in Cert, t \in T\}$$

In this case, the intruder may intercept msg_{data} without the ability to decrypt it. Since the intruder cannot get the receiver's private key and it cannot decrypt the intercepted message consequently.

C. Improved Model and Verification

We formalize the following model to verify whether the improved model $VANET_{NEW}$ can solve data leakage problem based on the analyses above.

$$VANET\_NEW =_{df}$$
$$VANET\_NEW1 \, [|INTRUDER\_PATH|] \, Intruder$$
$$VANET\_NEW1 =_{df} Vehicle||RSU||RC||ZC||CA||Clock$$

The new verification results are shown in Fig. 7. Property 3 is invalid, which indicates that the intruder cannot get the message, i.e., data security is ensured in our new model.

V. CONCLUSION AND FUTUREWORK

Kumar et al. proposed a new architecture to confirm the security of VANET using end-to-end authentication and hierarchical structure [5]. In this paper, we have modeled this proposed architecture of VANET using CSP. With the aid of model checking tool PAT, we have verified five properties of this model, including *deadlock freedom, divergence freedom, data leakage, vehicle faking and RSU faking*. The verification results show that the proposed architecture may cause data leakage. Aiming to handle this problem, we improved the above proposed model by encrypting the messages with receiver's public key. The new verification results indicate that the improved model is truly secure. We will dive into more security issues over VANET and explore how to verify other security properties with formal methods in the future.

REFERENCES

[1] Saif Al-Sultan, Moath M. Al-Doori, Ali H. Al-Bayatti, Hussein Zedan: A comprehensive survey on vehicular Ad Hoc network. J. Network and Computer Applications 37: 380-392 (2014)

[2] H. Hasrouny, C. Bassil, A.E. Samhat, A. Laouiti, "Security Risk Analysis of a Trust model for Secure Group Leader-based communication in VANET", Ad-hoc Networks for Smart Cities Book IWVSC Malaysia, pp. 71-83, 2016.

[3] H. Hasrouny, A.E. Samhat, C. Bassil, A. Laouiti, "VANET Security Challenges and Solutions: A Survey" in Vehicular Communications journal, Elsevier, vol. 7, pp. 7-20, January 2017.

[4] Hamssa Hasrouny, Abed Ellatif Samhat, Carole Bassil, Anis Laouiti: VANet security challenges and solutions: A survey. Vehicular Communications 7: 7-20 (2017)

[5] Gulshan Kumar, Rahul Saha, Mritunjay Kumar Rai, Tai-Hoon Kim: Multidimensional Security Provision for Secure Communication in Vehicular Ad Hoc Networks Using Hierarchical Structure and End-to-End Authentication. IEEE Access 6: 46558-46567 (2018)

[6] C. A. R. Hoare: Communicating Sequential Processes. Commun. ACM 21(8): 666-677 (1978)

[7] Gavin Lowe, A. W. Roscoe: Using CSP to Detect Errors in the TMN Protocol. IEEE Trans. Software Eng. 23(10): 659-669 (1997)

[8] Ho T. Dung, Thang H. Bui, Tho T. Quan: Model Checking Control Flow Petri Nets Using PAT. ICCSA (6) 2013: 124-129

[9] Zhipeng Shao, HanYong Hao, Yuanyuan Ma, Chen Wang, Jiaxuan Fei: Modeling and Verifying Intelligent Unit Transmission Protocol Using CSP Model Checker PAT. QRS Companion 2016: 244-251

[10] Whitfield Diffie, Martin E. Hellman: New directions in cryptography. IEEE Trans. Information Theory 22(6): 644-654 (1976)

[11] Victor S. Miller: Use of Elliptic Curves in Cryptography. CRYPTO 1985: 417-426

The Prediction of Delay Time at Intersection and Route Planning for Autonomous Vehicles

Genwang Gou[1], Yongxin Zhao*[1,2], Jiawei Jiang[1], Ling Shi[3]

[1]Shanghai Key Laboratory of Trustworthy Computing, East China Normal University, Shanghai, China
[2]Shanghai Trusted Industrial Control Platform Co., Ltd., China
[3]School of Information Systems, Singapore Management University

Abstract—**Intelligent Intersections (roundabout and cross-roads) management is considered as one of the challenges to significantly improve urban traffic efficiency. Recent researches in artificial intelligence suggest that autonomous vehicles have the possibility of forming intelligent intersection management, and likely to occupy the leading role in future urban traffic. If route planning method can be used for route decision of autonomous vehicle, the urban traffic efficiency can be further improved. In this paper, we propose an Intelligent Intersection Control Protocol (IICP) for controlling autonomous vehicles cross intersection, and recommend route for autonomous vehicles to reduce travel time and improve urban traffic efficiency. Firstly, we run IICP to obtain the original data, use SMOTE algorithm to synthesize balance data, and use RF, GBDT algorithms to predict delay time. Secondly, we use the iEigenAnt algorithm to find multiple short routes in traffic network. Finally, we recommend route for autonomous vehicles based on the minimum of driving time on the route and all delay time at each intersection to improve urban traffic efficiency.**

Index Terms—**intersection management, autonomous vehicle, SMOTE algorithm, route planning.**

I. Introduction

Over the past half-century, road intersections are managed by traffic lights or traffic control signals, these control mechanisms used to be efficient. However, as a growing number of vehicles flooding into the urban traffic flow, the shortcomings of these management mechanisms begin to emerge. In many cases, vehicles are required to stop even if there is no vehicles inside the intersection, resulting in traffic congestion at intersection, the trip time is also increased. The study of American cities shows that the congestion problem caused urban Americans to travel an extra 8.8 billion hours and purchases an extra 3.3 billion gallons of fuel for a congestion cost of $166 billion [1].

In order to solve these problems and improve intersection condition, some works [5][12] focus on regulating traffic flow and optimizing signal control, the study [11] tries to explore new intersection management design. Some researchers focus on proposing intelligent transportation system and develop autonomous vehicles to improve intersection efficiency, and they have made many remarkable achievements. On one hand, various autonomous vehicles have been developed and tested at intersection. On the other hand, many intelligent transportation system have been demonstrated efficient, such as

work [3] which manages the efficient passage of autonomous vehicles through intersections, increase the throughput of the intersections by up to 96.24% compared to common signalized intersections.

In this paper, we propose an IICP protocol to manage autonomous vehicles cross intersection. There are two kinds of objects in IICP, which are Intelligent Control Center (ICC) and autonomous vehicle, ICC makes two important calculations to manage autonomous vehicles cross intersection. First, it analyzes the transmitted message by autonomous vehicles and generates a priority sequence of all vehicles near the intersection by First Come, First Serve strategy. Second, it detects potential collision among vehicles in order of priority sequence, and uses two speed adjustment mechanisms to calculates the safety speed for each autonomous vehicle. The simulation result shows that IICP can greatly reduce delay time for each autonomous vehicle crossing intersection.

Afterwards, we apply machine learning algorithms on IICP to predict delay time at intersection under different traffic flow and parameter configurations. We run IICP to generate the original data set, whereas the original data set is imbalanced. In an imbalanced dataset, the training instance of minority data is obviously less than that of other data, as a result, these examples are more likely to be mispredicted. Hence many researchers have proposed numbers of algorithms to solve the consequences of imbalanced data. These algorithms can be categorized as three mainstream types, which are algorithm, sampling and oversampling, Among them, oversampling technology can avoid losing data information[8], and synthetic minority samples to form balance data, one of the most popular method of oversampling level is the synthetic minority oversampling technique (SMOTE) in [7]. Experiments on imbalanced UCI data reveal that using SMOTE algorithm can effectively improve the performance compared with sampling level [10], so we use SMOTE algorithm to preprocess data set to form balance data, and use RF and GBDT algorithms fit these balance data to predict delay time.

Finally, we use ant colony algorithm to solve route planning problem. In our previous work, we have proposed an iEigenAnt algorithm [9], which can find short route between the source node and destination node in reticular structure. We apply iEigenAnt algorithm to find multiple short routes in traffic network, and recommend route for autonomous vehicles according to the sum of driving time on the road and all delay

*Corresponding author: yxzhao@sei.ecnu.edu.cn

7

time at each intersection.

The main contributions of this paper includes:

- **Prediction of Delay Time**. We use SMOTE algorithm to preprocess the original data to form the balanced synthetic data, and use RF algorithm to fit the synthetic data to predict delay time under different traffic flow and parameters configuration, which get a higher prediction accuracy.
- **Route Planning**. We use iEigenAnt algorithm to find multiple short routes between the source intersection and the destination intersection, and recommend route for autonomous vehicle based on the minimum driving time, which improve the urban traffic efficiency.

The remainder of this paper is organized as follow. Section II presents the methods used in this paper. Section III describes IICP in detail. Section IV elaborates the process of the prediction of delay time. Section V includes the route planning for autonomous vehicles. Finally, this paper is concluded in section VI.

II. PRELIMINARIES

In this section, we describe some basic knowledge about techniques and models. We use SMOTE algorithm to synthesize minority samples to generate balance data. Afterwards, we use RF and GBDT to train the balance training data to predict the value of delay time.

A. SMOTE Algorithm

The paper [6] suggested that the SMOTE algorithm can avoid the risk of overfitting by randomly duplicates minority class instance. The core idea of SMOTE algorithm is to analyze the minority class samples and synthetic new samples according to these minority class samples, and add new samples to the dataset. There are three steps for SMOTE algorithm generates a new sample. Fig.1 shows the principle of SMOTE algorithm.

- For each sample x in minority class samples S, calculates the distance from x and all samples in S according to the Euclidean distance, and then obtain its k-nearest neighbors.
- Determine the sampling ration n by the sample imbalance proportion. For each sample x, randomly choose several samples from its k-nearest neighbors, suppose as x_n.
- For each randomly selected neighbor x_n, use the equation (1) to synthetic a new sample x_{new}.

$$x_{new} = x + rand(0, 1) * |x - x_n| \qquad (1)$$

B. Random Forest (RF)

Random Forest is one of the most successful general-purpose algorithms in modern times. As an integrated training method, RF generates multiple prediction models and summarizes the results of the model to improve the accuracy of the prediction model. Figure 2 shows three main steps for RF to get the final prediction result or classification result.
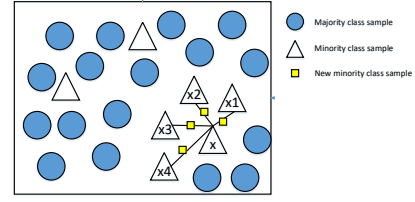
Fig. 1. Schematic diagram of synthetic data in SMOTE algorithm

- Get the training sets $S_1, S_2, ..., S_n$, (n represents the number of regression trees) according to the bootstrap [4] mechanism randomly with replacement.
- Training decision tree $T_1, T_2, ..., T_n$ based on training sets.
- The results of all regression trees are integrated to generate prediction value.
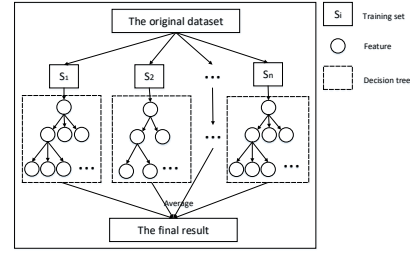
Fig. 2. Process of random forest algorithm

C. Gradient Boosting Decision Tree

Gradient Boosting Decision Tree is an iterative decision tree algorithm, which is composed of multiple decision trees, and the results of all trees are accumulated to make the final result. There are three steps for GBDT generates the final classification tree.

- Initial a weak tree with one root node, which can minimize the loss function.
- Calculate the negative gradient of loos function in current model, and take it as an estimate of the residuals. The core idea of Gradient Boost is to build a new model in the gradient direction of residual reduction to eliminate the previous residual, which is quite different from the traditional boost in weighting the correct and wrong samples.
- Estimate the regression leaf node area to fit the approximate residual value. Using linear search to estimate the value of leaf node region, minimizing the loss function, and update decision tree.

III. INTELLIGENT INTERSECTION CONTROL PROTOCOL

In this section, we present the intelligent protocol IICP, which aims at increasing the throughput of autonomous vehicles at intersection. Fig.3 shows the cross process of a new coming vehicle, it needs to go through four zones to pass intersection.
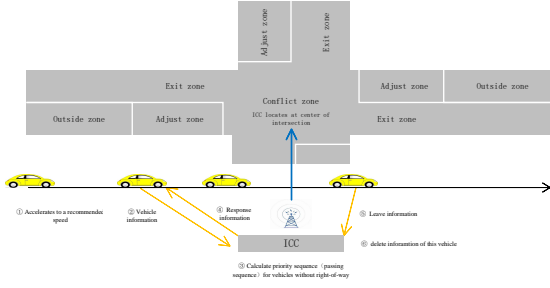
Fig. 3. The intersection layout and cross process of a new coming vehicle

In outside zone, all vehicles will accelerate to a recommend speed (as shown in Fig.4) to reduce travel time. In adjust zone, vehicles need to adjust their speed according to the instruction of ICC. In conflict zone, these vehicles will keep a constant speed until they cross the intersection. When autonomous vehicles arrival at the exit zone, which means vehicles have crossed the intersection, they can accelerate to the maximum speed to travel. Notice that all autonomous vehicles need to follow car-following strategy in these four zones, that is to say, the vehicles in the back must follow these in front.
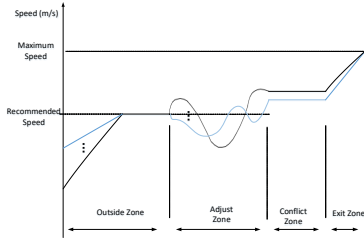
Fig. 4. Speed curve in IICP

A. Assumptions

In IICP, the human driving is replaced by autonomous driving, the whole control system should be rethought, so we introduce the following assumptions:

Intersection Assumptions: We model the intersection as a grid which consists of multiple cells, each cell has a unique identification number and can accommodate an autonomous vehicle. The intersection is controlled by ICC rather than the traffic light mechanism, and ICC is equipped with wireless communication device, powerful calculation device, etc...

Vehicle Assumptions: We assume all vehicles are autonomous vehicles, they have similar shape and physical, and they are equipped with sensor, positioning system, wireless communication device, so they can perceive nearby obstacles, obtain their position, interactive information with ICC.

B. Actions

ICC Actions:

1) ICC calculates the priority sequence and safe speed at fixed intervals by *priority determination* and *speed adjust-*

ment mechanism, afterwards, it sends speed adjustment message to each vehicle accordingly.
2) ICC will ignore some vehicles when it is notified that these vehicles have crossed the intersection.

Vehicle Actions:

1) All vehicles cannot enter the intersection without receiving speed adjustment message from ICC.
2) Whatever vehicles are in conflict zone or adjust zone, they should periodically send vehicle driving information to ICC, including vehicle location, speed, etc...
3) If ICC has replied driving information to each vehicle, they must change their speed accordingly.
4) If some vehicles have crossed the intersection, they should immediately send message to notify ICC.

C. Priority Determination

IICP is a sequence-based protocol, ICC needs to generate a *priority sequence* for the adjust zone vehicles, lay the foundation for the subsequent speed adjustment mechanism.

Now, we define the notations that will be used later.
- (r, r): Intersection cell size.
- $CList$: Records the vehicles which inside conflict zone.
- $AList$: Records the vehicles which inside adjust zone.
- $D_{X,e}$: The distance between vehicle X and the entrance of intersection.
- CL: The cell list which vehicle needs to use to cross the intersection. CL_i means the ith cell of CL.
- $count_n$: Number of cells before vehicle X enters cell n.
- T_n: The occupy time of cell n.
- $T_{in_n}^X$: Time of vehicle X entering cell n.
- V_X: Speed of vehicle X.
- V_X^{new}: The desired speed of vehicle X to avoid collision.
- P: The priority sequence of all AList vehicle, P_X refers to the priority of vehicle X.
- HP_i: Record AList vehicles which have higher priority than ith highest priority vehicle.

With these notations, we now have: the time of vehicle X enters cell n consists of the time when vehicle X reaches the edge of intersection and the time when vehicle X passes through each cell before cell n (Equation (2)). Notice that the priority of AList vehicles is lower than that of CList vehicles, that is to say, the AList vehicles must avoid the CList vehicles.

$$T_{in_n}^X = \frac{D_{X,e}}{V_X} + \sum_{i=1}^{n-1} T_{CL_i} \qquad (2)$$

D. Speed Adjustment Mechanism

When the priority sequence of AList vehicles is determined, ICC uses speed adjustment mechanism to adjust the speed of these vehicles. Assume there exist a vehicle X and a higher priority vehicle $X1$, if there's no collision between vehicles X and $X1$, ICC will accelerates vehicle X. The best situation is that vehicle $X1$ has just left a cell n and vehicle X enters it, so V_B^{new} can be calculated from Equations (3) and (4).

Algorithm 1: Speed Adjustment Mechanism

---

1 Assume there are m vehicles in CList and HP_i.
2 **while** *exists vehicles in adjust zone* **do**
3   Calculate the time of each vehicle arrival at intersection, record vehicles in priority sequence P by FCFS strategy.
4   Calculate $T_{in_n}^X$ for each CList vehicles and AList vehicles through Equation (2).
5   **for** *i=0, $i <$ the length of P* **do**
6   Choose the ith highest priority vehicle i from P, check whether there exist potential collision among vehicle i and m vehicles.
7   **if** *There is no conflict* **then**
8   **for** *j=0, $j < m$* **do**
9   Calculate the max allowed speed for vehicle i with vehicle j by Equations (3) and (4), suppose as V_i^j.
10   Choose the $\min(V_i^1, V_i^2, ..., V_i^m)$ as the safety speed vehicle i can accelerates to.
11   **else**
12   **for** *j=0, $j < m$* **do**
13   Calculate the max allowed speed for vehicle i with vehicle j by Equations (5) and (6), suppose as V_i^j.
14   Choose the $\min(V_i^1, V_i^2, ..., V_i^m)$ as the safety speed vehicle i can decelerates to.
15   Add vehicle i to HP_i, $m = m + 1$.
16   Reply specific driving information to AList vehicles.

$$\frac{(V_X + V_X^{new}) * T_{out_n}^{X1}}{2} = D_{X,e} + r * count_n \qquad (3)$$

So we have:

$$V_X^{new} = \frac{2 * (D_{X,e} + r * count_n)}{T_{out_n}^{X1}} - V_X \qquad (4)$$

In contrast, if there exists potential collision between vehicles X and $X1$, ICC will decelerates vehicle X. The way to calculate V_B^{new} is shown in Equations (5) and (6).

$$\frac{(V_X - V_X^{new}) * T_{out_n}^{X1}}{2} = D_{X,e} + r * count_n \qquad (5)$$

So we have:

$$V_X^{new} = V_X - \frac{2 * (D_{X,e} + r * count_n)}{T_{out_n}^{X1}} \qquad (6)$$

On the whole, the process of speed adjustment mechanism is to traverse each vehicle in order of priority sequence P and determine the safe driving speed. Assume that there are n vehicles in AList, so there will be n cycles in total. At the ith cycle, ICC gets the ith highest priority vehicle i from P, and check whether there will be potential collision among vehicle i, CList vehicles and HP_i vehicles. Note that potential collision means if these vehicles drive at current speed, there will be collision among them. There are two cases, which are *conflict* and *no conflict*.

Case 1. No conflict: In this case, ICC will accelerates vehicle i while ensuring safety. Assume the number of vehicles in CList and HP_i is m, ICC needs to calculate the maximum allowed speed with each vehicle in m vehicles according to

equations (3) and (4), suppose as $V_i^1, V_i^2, ..., V_i^m$. Afterwards, ICC chooses the $\min(V_i^1, V_i^2, ..., V_i^m)$ as the safe driving speed that vehicle i can accelerate to.

Case 2. Conflict: In this case, ICC will decelerates vehicle i to avoid potential collision. Assume the number of vehicles in CList and HP_i is m, ICC needs to calculate the maximum allowed speed with each vehicle in m vehicles according to equations (5) and (6), suppose as $V_i^1, V_i^2, ..., V_i^m$. Afterwards, ICC chooses the $\min(V_i^1, V_i^2, ..., V_i^m)$ as the safe driving speed that vehicle i can decelerate to.

IV. PREDICTION OF DELAY TIME

In this section, architecture of prediction of delay time is explained. Fig. 4 shows the process of the prediction of delay time. We run IICP to generate the original data set, and use SMOTE algorithm to balance the imbalanced data, then use RF and GBDT algorithms to predict the value of delay time under different traffic flow and algorithm parameters.
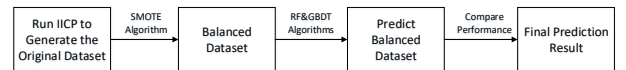
Fig. 5. Architecture of prediction of delay time

A. SMOTE Algorithm for Data Preprocess

We obtain the original data by running IICP on SUMO [2] firstly. There are five important parameters in this protocol, which are vNumber, acc, dec, minGap, maxSpeed, the meaning and the configuration of these parameters are shown in Table I. We set these features to the given value in Table I respectively. However, we find the original data is imbalanced

---

Algorithm 2: Prediction of delay time

---

1 Input: train dataset (x_i, y_i), test dataset (x_j, y_j), where $i = 1, 2, ..., m$, $j = m + 1, m + 2, ..., m + n$.
2 Output: the appropriate classification algorithm and fitted models.
3 Step 1: Divide training dataset into N binary subsets considering all classes.
4 Step 2: Use SMOTE algorithm synthesize balance data.
5 **for** $c = 1, c \leq N$ **do**
6   Apply SMOTE algorithm on ith class data.

7 Combine all classes to form balance data.
8 Step 3: Apply RF and GBDT algorithms on the balanced data, select the better algorithm according to the evaluation method.
9 Return the appropriate prediction algorithm and fitted models.

(as shown in Fig.6), most data is in the range of 0 to 6, so we need to use SMOTE algorithm preprocess the original data.

TABLE I
KEY FEATURES IN IICP

Name	Meaning	Configuration
vNumber	the number of vehicles cross intersection in given time	$[25, 1000]$
acc	max acceleration of vehicle	$[2, 2.4]$, m/s^2
dec	max deceleration of vehicle	$[2, 2.4]$, m/s^2
minGap	min gap between continuous vehicles	$[10, 14]$, m
maxSpeed	max speed allowed on road	$[20, 29]$, m/s

Firstly, we segment the original data into 7 classes by the range of delay time. Secondly, we execute 7 SMOTE operations on the original dataset. On the ith SMOTE operation, we choose the ith class as the minority data and synthesize data according to the imbalance rate, and then we combine all classes to form the balance data. Finally, the synthetic balanced data will be the entry data of RF and GBDT algorithms, the appropriate prediction algorithm will be obtained by the evaluation methods. We have transform the data processing part as algorithm 2.
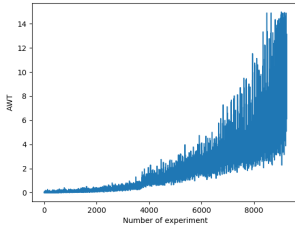
Fig. 6. Delay time under different traffic flow

B. Evaluation Methods

Performance evaluation metrics play an important role in assessing the prediction performance. We use Mean_Absolute_Error (MAE), Mean_Square_Error (MSE), R^2 Score (R^2) as the evaluation methods to evaluate the

performance RF and GBDT algorithms in the original data and the synthetic data. y_i^t is the true value of y_i, y_i^p is the predicted value of y_i.

MAE: It illustrates the difference between the predicted value and the real value, the smaller, the better.

$$MAE = \frac{1}{n} \sum_{i=1}^{n} |y_i^p - y_i^t| \tag{7}$$

MSE: It illustrates the square error between the predicted value and the actual value, the smaller, the better.

$$MSE = \frac{1}{n} \sum_{i=1}^{n} (y_i^p - y_i^t)^2 \tag{8}$$

R^2: It illustrates the fitting degree of the prediction model and the real data, The best value is 1.

$$R^2 = 1 - \frac{\sum_{i=1}^{n} (y_i^t - y_i^p)^2}{\sum_{i=1}^{n} (y_i^t - \frac{1}{n} \sum_{j=1}^{n} y_j^t)^2} \tag{9}$$

C. Experiment Result

We apply RF and GBDT algorithms on the original data and synthetic data sets respectively, to evaluate the performance of using smote algorithm synthesizes data. The experiment result shows that using smote algorithm can significantly improve the prediction performance, and RF has a better prediction result than GBDT algorithm (as shown in Table II).

TABLE II
EXPERIMENT RESULT OF RF, GBDT ALGORITHMS ON THE ORIGINAL
DATA AND THE SYNTHETIC DATA

Algorithm	Evaluation Methods	Original Data	Synthetic Data
RF	MAE	0.406	0.162
	MSE	0.681	0.146
	R^2	0.876	0.971
GBDT	MAE	0.384	0.268
	MSE	0.599	0.214
	R^2	0.891	0.956

V. ROUTE PLANNING

In this section, we describe the way to provide route planning services for autonomous vehicles in detail. When human driving vehicles are replaced by autonomous vehicles, it's important to provide route planning services for autonomous vehicles.

A. iEigenAnt Algorithm

In recent years, many researchers have put forward route planning algorithms, among them, intelligent bionics algorithm is rather useful when dealing with the problem of route planning under the condition of complex dynamic environment. In our previous work, we have proposed an intelligent bionic algorithm which called improved EigenAnt (iEigenAnt) algorithm, it can find short route between multiple points according to the way of both positive and negative feedback, we have successfully applied iEigenAnt algorithm on TSP problem. In this section, we use iEigenAnt algorithm to provide route planning service for autonomous vehicles.

B. Experiment Design and Result

In this section, we recommend route for autonomous vehicles according to the minimum of driving time on the route and all delay time at each intersection. For driving time on the route, we use iEigenAnt algorithm to find multiple short routes between source intersection and destination intersection. For delay time we randomly allocate traffic flow and parameter setting for each intersection to simulate the real scene, and use the model trained in section IV to predict the value of delay time. Finally, we choose the route which takes the least time as the recommend route.

We model a $10 * 10$ intersection network, where vehicles start from the source intersection (intersection 1) to the destination intersection (intersection 100), the horizontal and vertical distance between each two adjacent intersections is about 500 meters. Firstly, we use iEigenAnt algorithm find multiple short routes, and choose 8 routes as candidate recommended route, the length of each route is 8309.3, 8318.1, 8404.5, 8412.5, 8486.0, 8542.1, 8625.3, 8720.5 meters respectively. Secondly, we calculate the sum of delay time on each route, which is 32.43, 24.57, 41.53, 34.73, 55.47, 37.13, 44.26, 45.95 seconds respectively. Assume vehicles drive at 30km/h, the total travel time of 8 routes is shown in Fig.7. It's obviously, the route which takes the least travel time is route 2 instead of the shortest route 1.

VI. CONCLUSION

Autonomous driving can significantly decrease delay time for autonomous vehicles crossing intersection, and it will be the heart of urban transportation in the near future. So we propose an IICP to manage the intersection area to solve traffic congestion problem and seek the global benefit by dynamically allocating a safe time-space passage for each vehicle. Afterwards, we use iEigenAnt algorithm to recommend route for autonomous vehicles according to the minimum of driving

(a) The recommended route   (b) Total travel time of each route
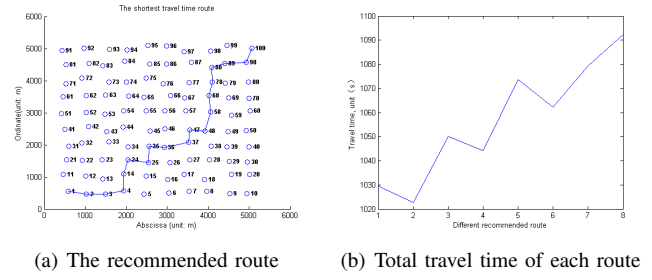
Fig. 7. Applying iEigenAnt algorithm on route planning problem

time on route and all delay time at intersection, the experiment results show our idea is effective.

VII. ACKNOWLEDGEMENT

REFERENCES

[1] 2019 urban mobility report and appendices. https://mobility.tamu.edu/umr/report/.
[2] simulation of urban mobility. http://sumo.sourceforge.net/.
[3] Reza Azimi, Gaurav Bhatia, Ragunathan Rajkumar, and Priyantha Mudalige. Ballroom intersection protocol: Synchronous autonomous driving at intersections. In *2015 IEEE 21st International Conference on Embedded and Real-Time Computing Systems and Applications*, pages 167–175. IEEE, 2015.
[4] Leo Breiman. Bagging predictors. *Machine learning*, 24(2):123–140, 1996.
[5] Shujian Bu, Zhang Tong, and Du Li. Approximations and simulation of the optimal change interval for roundabout. In *International Workshop on Database Technology & Applications*, 2010.
[6] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer. Smote: Synthetic minority over-sampling technique. *Journal of Artificial Intelligence Research*, 16(1):321–357, 2011.
[7] Nitesh V Chawla, Kevin W Bowyer, Lawrence O Hall, and W Philip Kegelmeyer. Smote: synthetic minority over-sampling technique. *Journal of artificial intelligence research*, 16:321–357, 2002.
[8] Douzas Georgios, Bacao Fernando, and Last Felix. Improving imbalanced learning through a heuristic oversampling method based on k-means and smote. *Information Sciences*, pages S0020025518304997–.
[9] Genwang Gou, Yongxin Zhao, Qin Li, and Qiwen Xu. A mathematical analysis of improved eigenant algorithm. *Journal of Experimental & Theoretical Artificial Intelligence*, 31(3):1–19, 2018.
[10] Li Ma and Suohai Fan. Cure-smote algorithm and hybrid algorithm for feature selection and parameter optimization based on random forests. *Bmc Bioinformatics*, 18(1):169.
[11] Tomaz Tollazzi, Marko Rencelj, and Saso Turnsek. Slovenian experiences with alternative types of roundabouts-" turbo" and" flower" roundabouts. In *Environmental Engineering. Proceedings of the International Conference on Environmental Engineering. ICEE*, volume 8, page 1220. Vilnius Gediminas Technical University, Department of Construction Economics ¡, 2011.
[12] Bai Yu and Kun Xue. Association of signal-controlled method at roundabout and stop rate. In *International Conference on Intelligent Computation Technology & Automation*, 2011.

DCBlock : Efficient Module for Unpaired Image to Image Translation Using GANs

Jin Yong Kim, Myeong Oh Lee, Geun Sik Jo[*]
Department of Computer Science
Inha University
nastynas9004@gmail.com, eremo2002@naver.com, gsjo@inha.ac.kr

Abstract — Recently, as various image-to-image translation studies have been progressed, it is possible to generate high-quality images. In particular, generation models using unpaired data produce meaningful results even building data at a low cost. However, these studies, which are based on Generative Adversarial Networks (GANs), is composed a very heavy architecture. Unlike the commonly used other deep learning models, generally the GANs model consists of two or more in a particular case deep architecture, which has a large computational cost. To solve this limitation, this paper proposes an efficient generator module called DCBlock (Depthwise separable Channel Attention Block). DCBlock consists of a depthwise separable convolution with a relatively low computational cost to replace the standard convolution commonly used in the image to image translation, and channel attention to compensate for information loss caused by depthwise separable convolution. DCBlock showed similar performance to the existing original model while reducing the number of parameters that represents the amount of computation by up to 91.6%. Besides, we experiment with the proposed method for various novel researches and prove that the problem is solved.

Keywords-component Generative Adversarial Networks , Unpaired Image-to-Image translation, Efficient model architecture, deep learning

I. Introduction

Recently, image-to-image translation studies using Generative Adversarial Networks (GANs) [1] produce plausible results. GANs can translate the style of the image to another domain [2-5] or generate new high-quality images with high resolution [6,7]. However, GANs are very expensive to compute because of standard convolutional layers, such as convolutional neural networks using very deep architectures (e.g. VGG [8], ResNet [9], AlexNet [10]). Therefore, the number of parameters representing the model's complexity will appear dramatically higher. A large number of parameters have a significant impact on training and inference time and requires high memory resources which is the major limitation for many Image-to-Image translation applications to be applied in real world.

To solve the aforementioned problems, this paper introduces the "Depthwise-separable Channel attention Block (DCBlock)" which replaces standard convolution with depthwise separable convolution and applies channel attention for an efficient unpaired image to image translation. DCBlock dramatically reduces number of trainable parameters that enables use of GANs in applications with limited resources. When we first tried to reduce the number of parameters, we replace standard convolution with depthwise separable convolutions. However, depthwise separable convolution is known to cause information loss [11,12]. Information loss causes poor quality image generation in the GAN model. At this point, we considered how to generate the image as natural as the existing other methods and were inspired by Zhang et al [13] who using channel attention in the residual block. Applying the channel attention focuses on the important parts in feature and make up for information loss, thus ensuring the quality of image. Therefore, we applied the techniques mentioned earlier to create a module called DCBlock. DCBlock is a replacement for "Resblock" [9] which is usually used in GAN architectures [2-5] for image-to-image translation.

Overall, our contributions are as follows:

- We propose a DCBlock that reduces number of parameters and generate almost similar quality images as existing image-to-image translation models
- We have demonstrated how to use channel attention to avoid information loss in depthwise separable convolution.
- We provide experimental results, including quantitative and qualitative assessments of our results with existing models and ablation study on the effect of channel attention on our model
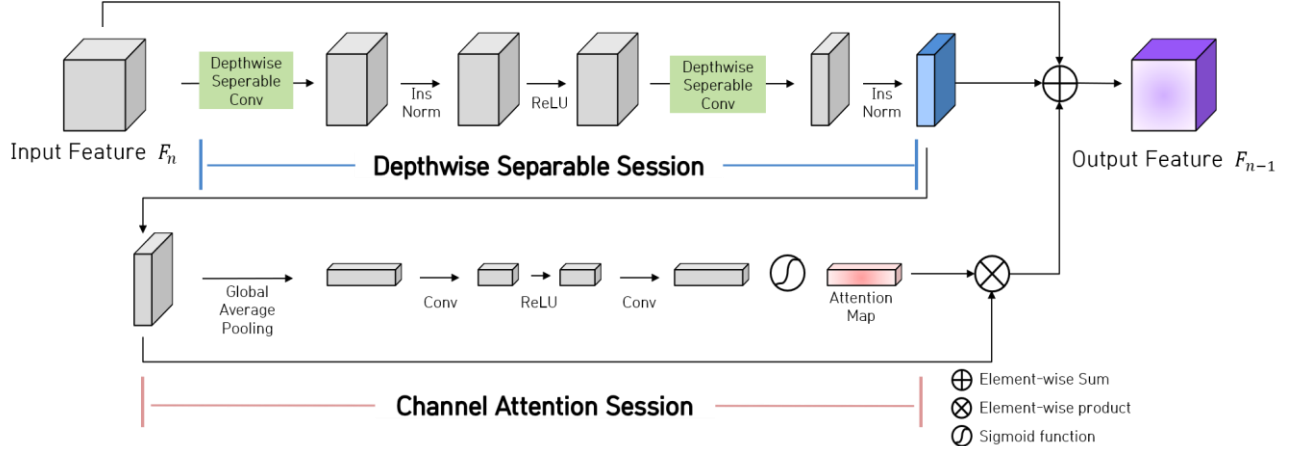
Fig. 1 . DCBlock Architecture

In conclusion, our method can generate the same quality image even though we reduce the parameters of existing baseline models.

II. **Related Work**

Generative Adversarial Networks. Generative Adversarial Networks (GANs) have shown great performance in image generation and image translation [2-5]. Inspecting image generation mechanism of GANs, generator tries to produce fake images that are indistinguishable from the real ones, while a discriminator tries to distinguish the real image from the fake or generated images. Since two networks are opposing, what each network learns is called "adversarial loss" which is a key point in GAN. In the basic GAN [1] model, there is one generator and one discriminator, but nowadays there are many models with multiple generators and discriminators depending on purpose.

Unpaired Image-to-Image Translation. Image translation that proceeds with paired data is often difficult to apply because data is rarely paired in the real world. On the contrary, using unpaired training data is suitable for real-world application, Consequently, there are various GAN methods presented. CycleGAN [2] learns the cycle consistency loss by mapping the two domains separately in two generators. Among the methods using attention guided, AttentionGAN [4] is a model that adds attention mechanism to CycleGAN. It can make the important part changes via the built-in attention mechanism without the need for additional labeled data or models. The case of multimodal is more efficient than the above models when generating a diversity of images. StarGAN [3] consists of one generator and several discriminators, which efficiently generate various images to increase efficiency and quality. MUNIT [5] creates multimodal images without any guides. MUNIT is composed of contents encoder and style encoder for recombine random noise with input contents at style space.

Efficient CNN Architecture. Many networks using depthwise separable convolution have been studied for efficient neural networks. At first, Xception [11], an architecture inspired by inception and depthwise separable convolution, proposed an extreme version inception module that does 1x1 conv first and then performs spatial correlation mapping on all output channels individually. MobileNet [12] also use depthwise separable convolution and additionally proposed shrinking hyper-parameter consisting of a width multiplier to control the input and output channels and a resolution multiplier to adjust the size of the input image. ShuffleNet [14] highlights that pointwise convolution is still a high cost area. To solve this problem, ShuffleNet designed channel sparse without connecting all weights, and shuffled groups to prevent the problem of getting only information flow for a specific area as input.

As we have seen, efficient CNN networks are being actively researched and real-world applications using them are actively being developed. Therefore, we will introduce a module to be used in the GAN method for efficient image-to-image translation.

III. DCBlock

To address the heavyweight model that unpaired image-to-image translation with GAN has, we proposed DCBlock (Depthwise separable Channel attention Block). At first, we applied depthwise separable convolution to reduce number of parameters. However, as can be seen in Xception [11], it causes information loss. Xception bridges this gap with residual connection but in image-to-image translation did not alleviate it, causing poor generation. Since depthwise separable convolution is performed for each channel, the loss of the feature appearing in the whole part is inevitable.

Accordingly, to tackle an optimal balance between the quality of output and computational cost, we had to add a technique to compensate for information loss. Therefore, we applied an attention module to keep the information we need as much as possible and not lose it even in deep architectures. Our proposed module DCBlock is shown in **Fig 1**. It consists
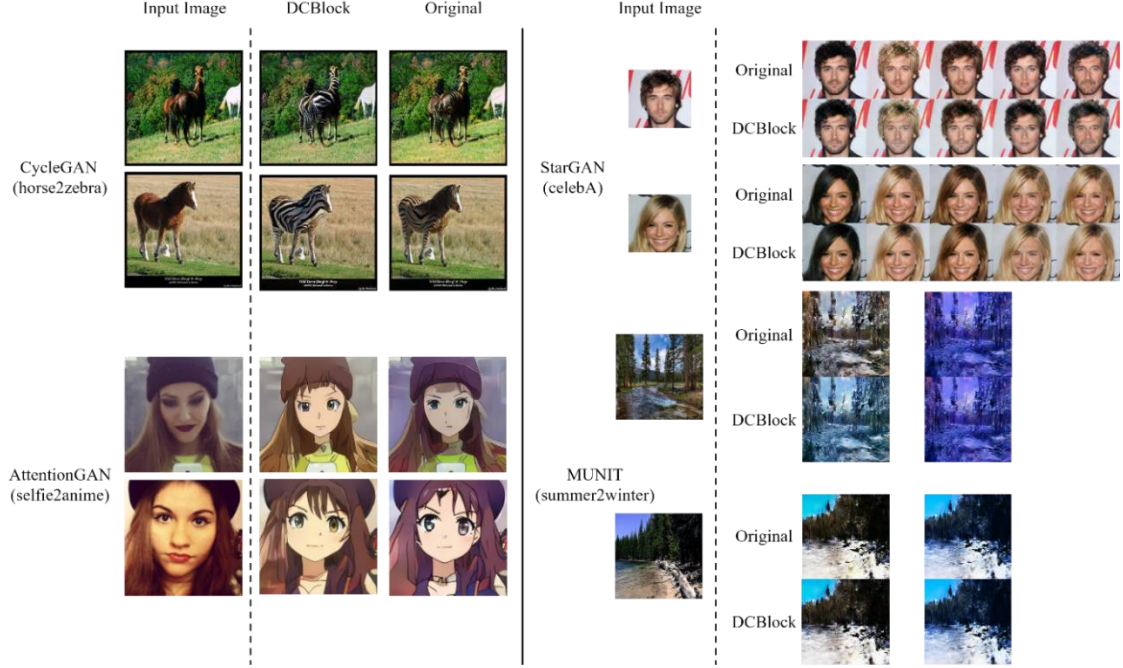
Fig. 2. Output Comparison of The Two Methods

of two sessions, Depthwise separable session and Channel attention session. In DCBlock, the input feature and output feature will be feed to each block proceeding residual learning. For input feature F_n, DCBlock can be formulated as follows:

$$DC(F_n) = F_n \oplus S_{depth}(F_n) \oplus CA(S_{depth}(F_n)), \tag{3}$$

where $DC(\cdot)$ denote DCBlock module and $S_{depth}(\cdot)$, $CA(\cdot)$ denote depthwise separable session and channel attention session. Depthwise separable session significantly reduces the number of parameters than the standard convolution of the existing resblock. And channel attention that denoted $CA(S_{depth}(F_n))$ is formulated as,

$$CA\left(S_{depth}(F_n)\right) = S_{depth}(F_n) \otimes S_{att}\left(S_{depth}(F_n)\right), \tag{4}$$

where $S_{att}(F_n)$ is attention map extracted by channel attention. This is a statistic of the channel obtained through the gating mechanism [15], which can prevent the information loss. In summary, we present DCBlock which a method that is efficient and produces quality similar to existing models. In the next part, we describe a detailed description of the DCBlock configuration.

Depthwise Separable Convolution. As explained in the previous part, to reduce number of parameters, we used depthwise separable convolution, which composed a combination of depthwise convolution and pointwise convolution. In depthwise convolution, there are filters for the number of channels to extract spatial features, which is why the number of input and output channels is the same. Depthwise convolution is can be written as,

$$\widehat{G}_{k,l,m} = \sum_{i,j} \widehat{K}_{i,j,m} \times F_{k+i-1,l+j-1,m}, \tag{5}$$

where $\widehat{K}$ denote kernel size of depthwise convolution, i,j,m denote width, height, input channel and F denote feature map. And pointwise convolution is a 1x1 convolution, and the size of the filter is fixed to 1x1. In contrast to the depthwise convolution, pointwise convolution is performed only on the channel without dealing with spatial features. This helps to greatly reduce the amount of computation in DCBlock. **Table I** shows the differences between the parameters and computational costs of the two convolutions.

Table I
COMPARISON OF TWO KIND OF CONVOLUTIONS

Method	Standard Convolution	Depthwise Separable Convolution
# of Param	$K^2 \times C \times M$	$C \times (K^2 + M)$
Computational cost	$K^2 \times C \times M \times H \times W$	$C \times H \times W \times (K^2 + M)$

where K denote kernel size, C, M denote input and output channel size and H, W denote input height, width. As **Table I** shown, Standard convolution has a computational cost of $K^2 \times C \times M \times H \times W$ while depthwise separable convolution has $C \times H \times W \times (K^2 + M)$. Dividing the two costs to see the difference shows that the cost has reduced by $\frac{1}{M} + \frac{1}{K^2}$. This has great effect in reducing proportionally increasing parameters in GANs where relatively deep networks are used.

TABLE II
QUANTITATIVE EVALUATION

Method		Generator Million Parameter	LPIPS↑	FID ↓	SSIM↑	IS↑
CycleGAN	ResBlock	11.06	Evaluate only multimodal	210.48	0.7898	1.3771
	DCBlock (Ours)	0.89		195.04	0.8495	1.4354
AttnetionGAN	ResBlock	11.82		221.09	0.4392	1.5045
	DCBlock	3.62		226.16	0.4166	1.4620
StarGAN	ResBlock	8.43	0.114	17.61	0.8221	3.2183
	DCBlock	2.56	0.109	19.36	0.8252	3.1949
MUNIT	ResBlock	15.02	0.047	105.94	0.3344	1.8457
	DCBlock	7.22	0.044	105.81	0.3348	1.8322

Channel Attention Mechanism. Channel attention, proposed by Zhang et al [13], is one of the attention's variations that leverages the interdependencies between the channel to focus on informative feature. It uses global average pooling to compress channel information and restore the feature through the convolution layer. Subsequently, the channel statistics are extracted via the gating mechanism [15]. This effect enhances and restores the feature for the focused part of the network. Consequently, we use channel attention to sustain as many features as possible even in deep architectures and to address information loss caused by depthwise separable convolution. We provided more details about Channel Attention Mechanism usage in the next section and show how it affects to generated images in section 4.

IV. Experiments

To explore the suitability of proposed model, we evaluated DCBlock quantitatively and qualitatively on various datasets, comparing different models. The method of experiments is replacement of the "Resblock" [9] with a "DCBlock" on other novel models as we mentioned at section II.

Baseline Models. As baseline models, we adopt CycleGAN[2], AttentionGAN [4], StarGAN [3] and MUNIT [5]. Since they include resblock in their models, they are suitable models for evaluation. For comparison, we apply DCBlock to aforementioned models, and compared the performance and number of parameters. **CycleGAN** consists of two generators and two discriminators, where the generator typically uses nine resblocks, which could be six or U-Net [16] depending on the resolution of images dataset. We used horse2zebra datasets [2] for this task. **AttentionGAN** has a similar architecture to CycleGAN. It has also nine or six resblocks and additionally produce attention mask via a built-in attention mechanism. For the experiment, we used the selfie2anime dataset created in Kim et al [17] for the AttentionGAN. **StarGAN** consists of one

generator and N discriminators, where N is datasets number of class. In addition, StarGAN generator consists of six resblocks. For StarGAN experiments we used celebA dataset. **MUNIT** consists of a content encoder and a style encoder as described in the paper. In the MUNIT model, resblocks were applied only for the content encoder. For the experiments, we use summer2winter yosemete dataset [2] with MUNIT model.

Evaluation Metrics. In quantitative evaluation, we measured the quality and diversity of the image with four metrics along the baseline papers. **FID** [18] uses the inception-v3[22] model to extract features and measure the distance between the distribution of real and fake images. The lower value of FID is more similar with real image. **LPIPS** [19] is perceptual metric about patch images. LPIPS evaluate the distance between images patches. Although It indicated that higher is different, and lower is more similar, we use it as a measure of the diversity of the image. **SSIM** [20] is a metric that handles the structure of an image. It evaluate three factors that affect perceptual quality: average brightness, contrast, and structure. SSIM indicates higher is similar. **Inception Score (IS)** [21] is similar to FID in that it uses Inception network [22]. However, they differ in the way they use features, and IS evaluates how diverse the image is and how well it can be determined. In qualitative evaluation was conducted perceptual study according to Kim et al [17]. We conducted a user study in which users voted on their preference image.

Quantitative Evaluation. As seen in **TABLE II**, we applied DCBlock to adopted models. In CycleGAN, replacing the generator's resblock with a DCBlock shows that the FID, SSIM, and IS are similar or better, despite a 91.6% reduction in the number of parameters from 11.06M to 0.87M. In AttentionGAN, SSIM and IS were lower than original model when applied, but FID was higher, and the number of parameters decreased by 69.3% from 11.25M to 3.3M. In
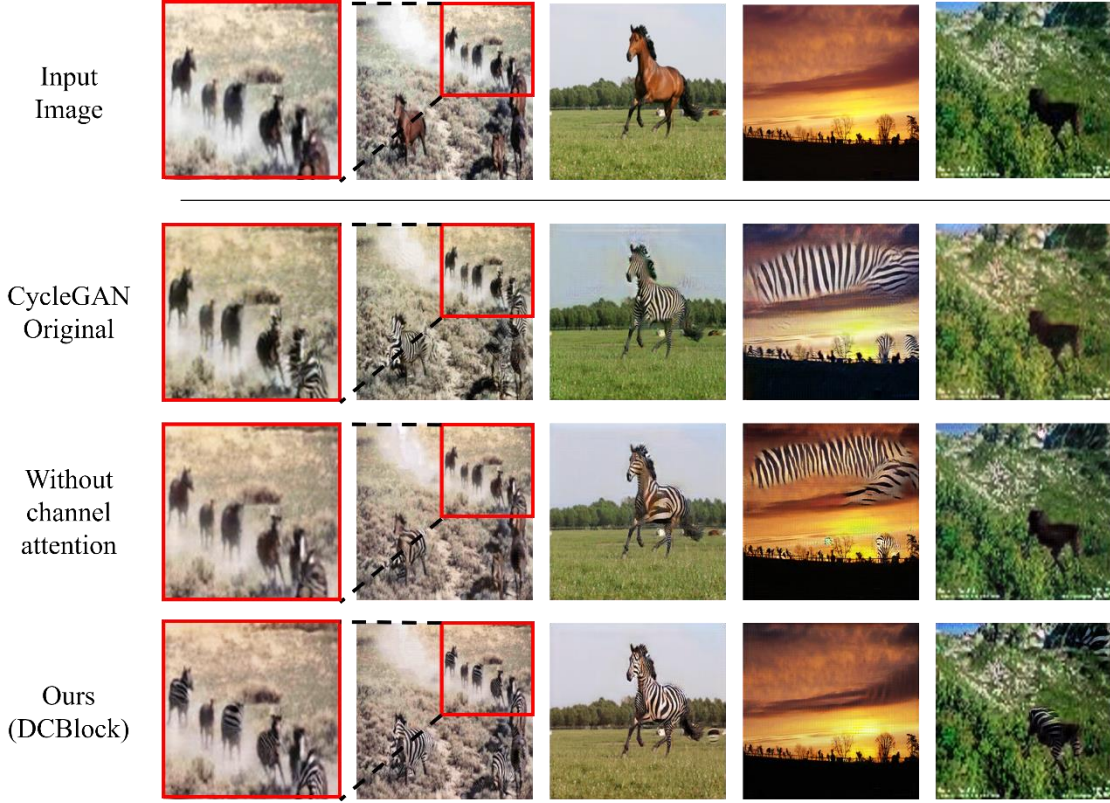
Fig. 3. Ablation Study Comparison

StarGAN, number of parameters reduced by 70.4% from 8.43 to 2.5, and other metric show that original and DCBlock generate images of similar quality. And in result of MUNIT, the number of parameters decreased by 51.9% from 15.02 to 7.22, and similar figures in other metrics as well.

Analyzing the results of quantitative evaluation, we can see that DCBlock shows similar performance as resblock, a method of each model, while dramatically reducing the number of parameters.

Table III
PREFERENCE PERCENTAGE OF USER SCORE

Model	Original (Resblock)	DCBlock(Ours)
CycleGAN (horse2zebra)	51.4(257)	48.6(243)
StarGAN (CelebA)	54.2(271)	45.8(229)

Qualitative Evaluation. We provide 10 randomly sampled images from CycleGAN [2] (horse2zebra) and StarGAN [4] (CelebA) with DCBlock and the original model pairs to 50 participants, and participants evaluate the fake images to choose what they think is more natural. We inform the participant that only the domain of fake image and the original image. The results of the user study are shown in **Table III**. As we aimed for, both methods showed nearly similar preference percentages. First, the two methods of

CycleGAN have a difference of 2.8% p (14 votes), which is higher than our method. And StarGAN showed the original with an 8.4% p (42 votes) high preference.

As a result, as shown in **Fig. 2**, our module produces a very similar level of image quality, although it is slightly less in terms of preference than the original module.

Ablation Study. As discussed in section 3, channel attention [13] is an important contribution to image quality in this paper. Our ablation study examines how channel attention affects our module and contribute to generating images. To verify the impact of channel attention, we compared CycleGAN's original model, the "without-channel-attention" model using only depthwise separable convolution, and DCBlock. In terms of number of parameters, CycleGAN, as provided **TABLE IV**, has 11.06 million parameters, without-channel-attention has 0.66 million parameters and ours has 0.89 million. In **Fig 3**, several failure cases are shown in CycleGAN original model and without channel attention module. Second (The enlarged images are the first column) and fifth column images were not translated to zebra when horses in blurry form were used with CycleGAN and without-channel-attention model. However, when we use our module, we can see that the blurry horses are also translated to zebra. And as shown in fourth column, input image is difficult to recognize the horse on the hill with human perception. Despite CycleGAN and Without-channel-attention failure that translate wrong part (i.e. sky), our

method overcomes this issue. As a result, adding channel attention increases a small number of parameters, however, contributed to generate the similar quality as the existing model or improve to better results. This is clear to say that channel attention contributes to extracting the righter focus even in the images that are difficult to distinguish.

Table IV
COMPARISON OF ABLATION STUDY

Model	Millions of parameters
CycleGAN original	11.06 M
Without-CA	0.66 M
Ours (DCBlock)	0.89 M

V. Conclusion

In this research, we proposed DCBlock that solves high computational cost problem in the unpaired image-to-image translation with GANs. DCBlock overcome the large number of parameters and high requirements of memory resources while ensuring the quality of the image. Experimental results show that when our method is applied to the baseline method, it generates images of similar quality or more natural to the existing method while reducing the number of parameters. Since DCBlock is a lightweight network, it is thought to be easier to use in a real world with limited resources.

Acknowledgements

Reference

[1] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D.Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. In Advances in Neural Information Processing Systems (NIPS), 2014, pages 2672–2680.

[2] J.-Y. Zhu, T. Park, P. Isola, and A. A. Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. In Proceedings of the IEEE International Conference on Computer Vision (ICCV), 2017.

[3] Y. Choi, M. Choi, M. Kim, J.-W. Ha, S. Kim, and J. Choo, "Stargan: Unified generative adversarial networks for multi-domain image-to-image translation," in IEEE Conference on Computer Vision and Pattern Recognition, 2018, pp. 8789–8797 .

[4] H. Tang, H. Liu, D. Xu, P. Torr, N. Sebe. AttentionGAN: Unpaired Image-to-Image Translation using Attention-Guided generative Adversarial Networks. arxiv preprint. Arxiv : 1911.11897, 2019

[5] X. Huang, M.-Y. Liu, S. Belongie, and J. Kautz, "Multimodal unsupervised image-to-image translation," in European Conference on Computer Vision (ECCV), 2018, pp. 172–189.

[6] C. Ledig, L. Theis, F. Husz´ar, J. Caballero, A. Cunningham, A. Acosta, A. Aitken, A. Tejani, J. Totz, Z. Wang, et al.,

"Photorealistic single image super-resolution using a generative adversarial network," in IEEE Conference on Computer Vision and Pattern Recognition, 2017, pp. 4681–4690.

[7] X. Wang, K. Yu, S. Wu, J. Gu, Y. Liu, C. Dong, Y. Qiao, and C. Change Loy, "Esrgan: Enhanced super-resolution generative adversarial networks," in European Conference on Computer Vision, 2018, pp. 63–79.

[8] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556, 2014.

[9] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. arXiv preprint arXiv:1512.03385, 2015.

[10] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In Advances in neural information processing systems, 2012, pages 1097–1105.

[11] Chollet, F. Xception: Deep learning with depthwise separable convolutions. In 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2017, 1800–1807

[12] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko,W.Wang, T. Weyand, M. Andreetto, and H. Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. arXiv preprint arXiv:1704.04861, 2017.

[13] Y. Zhang, K. Li, K. Li, L. Wang, B. Zhong, and Y. Fu, "Image super-resolution using very deep residual channel attention networks," in European Computer Vision Conference (ECCV), 2018.

[14] X. Zhang, X. Zhou, M. Lin, and J. Sun, "Shufflenet: An extremely efficient convolutional neural network for mobile devices," in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2018, pp. 6848–6856.

[15] Hu, J., Shen, L., Sun, G.: Squeeze-and-excitation networks. arXiv preprint arXiv :1709.01507 (2017)

[16] Ronneberger, O., Fischer, P., Brox, T.: U-net: Convolutional networks for biomedical image segmentation. In: International Conference on Medical image computing and computer-assisted intervention. pp. 234{241. Springer (2015)

[17] J. Kim, M. Kim, H. Kang, and K. Lee, "U-GAT-IT: Unsupervised generative attentional networks with adaptive layer-instance normalization for image-to-image translation," arXiv preprint arXiv: 1907.10830, 2019.

[18] M. Heusel, H. Ramsauer, T. Unterthiner, B. Nessler, and S. Hochreiter, "Gans trained by a two time-scale update rule converge to a local nash equilibrium," in NIPS, 2017.

[19] R. Zhang, P. Isola, A. A. Efros, E. Shechtman, and O. Wang. The unreasonable effectiveness of deep features as a perceptual metric. In CVPR, 2018.

[20] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli, "Image quality assessment: From error visibility to structural similarity," IEEE Transactions on Image Processing, vol. 13, 2004.

[21] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen, "Improved techniques for training gans," in Neural Information Processing Systems, pp. 2234–2242, 2016.

[22] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in IEEE Conference on Computer Vision and Pattern Recognition, pp. 2818–2826, 2016.

A Detect-and-Modify Region-based Classifier to Defend Evasion Attacks

Jiawei Jiang[1], Yongxin Zhao[*1,2], Xi Wu[3] and Genwang Gou[1]
[1] Shanghai Key Laboratory of Trustworthy Computing,
[2] National Trusted Embedded Software Engineering Technology Research Center,
East China Normal University, Shanghai 20062, China
[3] The University of Sydney, Australia

Abstract.

Deep Neural Networks (DNNs) are powerful models that have achieved impressive results on image classifications. However, they are vulnerable to be attacked by adversarial examples, which are crafted to cause prediction errors in DNNs. In order to make the networks more robust and reliable, in this paper, we present an improved region-based classification to mitigate the evasion attack, which is well known to attack DNNs via generating adversarial examples. Specifically, in our framework, an image is considered as a matrix of Markov Chains and we detect possible adversarial examples according to the Image Transition Probabilities (ITPs) in Markov Chains. Furthermore, we modify the original ITPs of the detected adversarial examples by using the saliency map of ITPs, and we employ our improved region-based classification on these updated adversarial examples to get a better output prediction. Finally, our experiments illustrate that our approach reduces the test errors imposed by adversarial examples on MNIST datasets and CIFAR-10 datasets.

1 Introduction

In recent years, deep learning has achieved remarkable results in image classification [5], malware detecting [3]. Though deep networks have exhibited a very good performance in classifications, researchers from the machine learning area have found a fatal weak point that Deep Neural Networks (DNNs) are extremely vulnerable to be attacked by adversarial examples [14]. In 2014, Szegedy et al. [14] firstly proposed the concept of adversarial example, which is a kind of images crafted by adding tiny perturbations into normal images (i.e., normal examples). For instance, an attacker may add a small noise into a test example such that it can deceive the state-of-the-art classifiers into giving an incorrect classification, which is also called the evasion attack. Recently, many algorithms, including FGSM [4], JSMA [12], have been proposed to generate adversarial examples, which make the rate of wrong predictions given by DNNs classifiers much higher. Moreover, because of transferability [11], an adversarial example generated by one DNN model is also able to mislead other DNN models which may have different network structures and physical attacks [6]. Thus, adversarial examples significantly limit the use of deep learning, especially in safety critical applications such as self-driving cars [9]. It is important for us to develop an approach to defend against adversarial examples and to mitigate the evasion attack from DNNs.

* Corresponding Author: yxzhao@sei.ecnu.edu.cn

To defend against evasion attacks, some defense methods have been proposed such as detecting adversarial example, adversarial training and distillation defense [7]. The method of detecting adversarial example is quite straightforward that the detection model determines whether an input is legal or not according to the differences between adversarial examples and normal examples. If the input is illegal, an exception handling strategy will be executed. However, it is usually difficult to design a proper exception handling strategy. The state-of-the-art detecting adversarial example method, which considers an image as a Markov Process, was proposed by Zhou et al [15]. Additionally, we can also mitigate evasion attacks by enhancing the robustness of networks themselves. Goodfellow et al. [4] used adversarial examples to train DNNs models, which is well known as adversarial training. Besides, Papernot et al. [13] proposed a distillation method to make DNNs robust against adversarial attacks, which uses the knowledge of the network to improve its own robustness. However, all these methods above sacrifice the classification accuracy of normal examples.

In 2018, Cao et al. [1] proposed the region-based classification algorithm, which samples some examples from a hypercube area centered by a test example and selects the label which appears most as the prediction result of the test example after predicting labels of all these sample examples via a trained DNNs model. Even though this algorithm not only maintains a high accuracy of DNNs classifiers on normal examples, but also increases the robustness of DNNs classifiers against adversarial examples, it in fact heavily relies on the surroundings of test examples. Specifically, the most sample examples from the surroundings of the test example can be classified by the DNNs model, the more reliable and robust of the prediction result of the algorithm on the existing evasion attacks. In the case shown in Figure 1, the region-based classification algorithm will behave badly because most sample examples from the surroundings (a hypercube area in blue) of the test example x' are outside the classification boundary (a curve in red) and they cannot be classified by the DNNs model. In this paper, we propose a new algorithm to improve the region-based classification algorithm for image classifications. An image is regarded as a matrix of Markov Processes, in which each Markov Process corresponds to one row of this image. Firstly, we define the Image Transition Probability (ITP) to represent the transformation probabilities of all image pixel values, and we perform 8000 MNIST data with adversarial examples and normal examples to illustrate that ITP can remarkably distinguish adversarial examples from normal examples. Secondly, we use ITP to detect
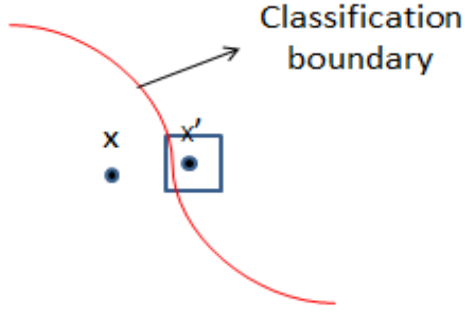
Figure 1. A case where the region-based classification behaves badly. Here, x is a normal example whereas x' is an adversarial example.

adversarial examples in an image, and decrease the original ITP of detected adversarial examples in order to make the examples locate at a proper location and to make the process more reliable. Finally, we observe that it is easy to sample some examples with much noise when sample examples from hypercube centered by the test example. Thus, we newly create an area centered by the test example which is different from the one used in the region-based classification.

In summary, the main contributions of our work include:

- We propose a new algorithm to address dependency on the surrounding of the test example through modifying the detected possible adversarial examples. As far as we know, it is the first work to combine the region-based classification with detecting method.

- We give a new region centered by the test example to sample examples which are different from region-based classification. Our region is anisotropic and the region given by region-based classification is isotropic.

The structure of the remaining paper is given as follows. Section 2 shows some preliminaries of this paper including a brief introduction on evasion attacks and some defense methods. Section 3 describes our improved region-based classification approach. Our experiments are in Section 4, which demonstrate the effectiveness of our algorithm. In Section 5 we conclude our work and outlook future works.

2 Preliminaries

We firstly introduce the following notations:

- Let X be the normal example without perturbation.
- Let X' represent the adversarial example which is designed by attacker to make classifiers wrong.
- Let $\nabla J_X(X, \theta, y)$ denote the gradient of cost function in DNNs. Where, X is the input, y is the true class and θ is the parameter of neural network.
- We use $sign(m)$ to represent sign function which returns the sign of a certain number m (positive or negative). If m is metric, it will return a matrix which consists of -1, 1 and 0.
- Let ϵ represents a small numerical.

Now, we will give a brief introduction on several typical attacks and some major defense methods which have been used in our experiments.

Evasion Attacks. Evasion attacks are well-known attacks in machine learning area which generate adversarial examples to attack machine learning models. There are two types of evasion attacks, including target evasion attacks and no-target evasion attacks. In target attack, an adversarial example is designed to make classifiers to give a wrong prediction. In contrast, an adversarial example is only generated to make classifiers wrong in the no-target attacks. In this paper, we only discuss no-target attacks.

1) **Fast Gradient Sign Method (FGSM)**: It is one of the simplest methods to get adversarial examples by calculating the gradient of cost function with respect to the input X, which is motivated by the linear nature of DNNs models. Goodfellow et al. [4] efficiently get the adversarial example X' via the following equation:

$$X' = X + \epsilon \cdot sign(\nabla J_X(X, \theta, y))$$

2) **Jacobian-based Saliency Map Attack (JSMA)**: JSMA is a method to create an adversarial example by restricting the l_0-norm of the perturbations, proposed by Papernot et al. [12]. It computes a saliency map by extracting some important pixels from the input image in which a small change can make the output various, and then modifies these pixels iteratively until getting the output variously.

3) **Carlini Wagner (CW)**: CW [2] is an efficient algorithm which generates successful adversarial examples with small noise. It shows that an adversarial example generated by the CW method is also effective in defensive distillation networks. Considering three constraints l_0, l_2 and l_∞, CW attacks can mainly divided into three attacks as well, i.e., l_0 attack, l_2 attack and l_∞ attack. In our paper, we mainly use l_∞ attack in the experiments.

4) **DeepFool**: DeepFool is a no-targeted attack proposed by Moosavi-Dezfooli et al [10]. The key idea of DeepFool is to generate adversarial examples iteratively, that is each step searches a decision boundary direction to modify examples. It can generate adversarial examples with minimum noise.

Defense Methods Against Evasion Attacks. Adversarial examples have brought great threats to security applications in deep learning area. In order to defend against adversarial examples, various defense methods have been advised.

1) **Adversarial training**: Firstly introduced by Goodfellow et al., it is an effective defense method which trains the model via augmenting the training datasets with adversarial examples. Specifically, adversarial training can be considered as model regularization by adding the loss function of adversarial examples to DNNs models. Since adversarial training achieves a great effect in dealing with adversarial examples, several variants of it have been proposed with respect to different adversarial attack algorithms. Although adversarial training has the state-of-the-art defending performance, it still has a limitation. Adversarial training is found that it decreases the classification accuracy on normal examples. For instance, the DNNs model without adversarial training performs greater than that with adversarial training on CIFAR-10.

2) **Region-based classification**: Although many defense methods have been proposed, they have the same issue that they sacrifice the accuracy on normal examples. The region-based classification achieves a high accuracy on normal examples and also enhances the robustness of DNNs models. The key idea in region-based classification is sampling some examples which can help DNNs models predict the results on test examples. When testing an example, it samples some examples from a small region centered at this example and uses a trained DNNs classifier to test these examples. The suggested class of this example is the most voted by the classifier. However, region-based classification cannot behave well on FGSM attacks, which only gets about 10 percent accuracy. It is considered that this method is easy to be attacked by high-distortion adversarial examples. In this paper, we propose an improved region-based method to solve this problem.

3 Detect-and-modify region-based classifiers

The region-based classification proposed by Cao et al. [1] depends on the region centered by the test example. If a test example locates at the area where most of examples cannot be classified correctly by the DNNs model, the region-based classification then behaves badly. In this section we mainly present our detect-and-modify region-based classifiers to mitigate evasion attacks. There are three main processes
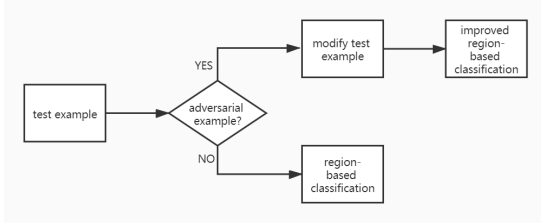
Figure 2. Flow of our approach

in our method, including detecting adversarial examples, modifying adversarial examples and an improved region-based classification. Figure 2. shows the whole process of our approach.

When testing an example, we firstly employ ITP, which can significantly distinguish adversarial examples from normal examples, to determine whether this test example is an adversarial example or not. Specially, if the ITP of this example is greater than the threshold value derived from training datasets, this example is supposed to be an adversarial example.

Secondly, if this example is an adversarial example, we use the saliency map of ITP to modify its original ITP. Thus the updated example facilitates to decrease its ITP and enhance the reliability.

Thirdly, we use our improved region-based classification to predict the label of the test example. However, if the test example is a normal example, we simply use the traditional region-based classification to predict its label.

The pseudo code of our algorithm is listed in Algorithm 1. There are four main procedures used by our approach. Procedure **detect**(x) determines whether x is an adversarial example or not. Procedure **modify**(x) carries out the modification of x if x is detected as an adversarial example and returns x'. Procedure **IRC**(x') proposed by us, gives the label of x. Procedure **RC**(x) predicts the label for x.

Algorithm 1 Detect-and-modify region-based classification

Input: A test example x
Output: The prediction label y_x of x
1: Flag = **detect**(x)
2: **if** Flag == true **then**
3:     $x' = $ **modify**(x)
4:     $y_x = $ **IRC**(x')
5: **else**
6:     $y_x = $**RC**($x$)
7: **end if**
8: **return** y_x

3.1 Detecting adversarial examples

Firstly, we introduce our detecting process. In an image, there are many pixels and a pixel is usually related to its adjacent pixel in the same row. In this paper, we consider a single relation that the next pixel is related to its former pixel. For an image, we consider one image row as a Markov Chain and the pixel in the same row is equal to the discrete state of the Markov Chain. Therefore, the image can be considered as a structure with many Markov Chains. The space of state in a Markov Chain, which is consist of the value of pixels, ranges from 0 to 255. The ITP which we firstly proposed represents the transformation probability of all image pixel values related to adjacent pixels in the same row. The ITP value can be calculated by the following formula:

$$ITP = \sum_{h=1}^{H} P_h(x_2|x_1) \times P_h(x_3|x_2) \times \cdots \times P_h(x_w|x_{w-1}),$$

where $P_h(x_{i+1}|x_i)$ is the probability of adjacent pixels in h-th image row and w represents the number of pixels in one image row. $P_h(x_{i+1}|x_i)$ can be gotten in state transition which is calculated by the normal examples. Then, we calculate the ITP by adding all Markov chain transitions in an image. We put a single-channel image as an example.

Suppose a single-channel image has H rows and W columns, which means it includes H Markov Chains and W discrete states in each chain. Each row of image can be converted into one dimensional vector X with W pixels. So an image is composed by a set of $X_1, X_2, ..., X_m$. For a X_m, x_m^n represents n-th pixel in X_m. Let i represent the value of x_m^n range from 0 to 255. The x_m^{n+1} is a pixel which is next to x_m^n and its value denotes j. While $P(i,j)$ represents the transition probability from state value i to state value j. In other words, $P_{i,j}$ is the probability of appearance of value pair(i,j) at two adjacent pixels. So, the transition of probability matrix P can be described as follows:

$$P = \begin{bmatrix} P_{0,0} & \cdots & P_{0,255} \\ \vdots & \ddots & \vdots \\ P_{255,0} & \cdots & P_{255,255} \end{bmatrix}$$

We can calculate the transition of probability matrix P based on training sets as below:

$$P_{i,j} = \frac{\sum_{n=1}^{N} \sum_{h=1}^{H} \sum_{t=1}^{W-1} h_{i,j}(x_t, x_{t+1})}{\sum_{n=1}^{N} \sum_{j=0}^{255} \sum_{h=1}^{H} \sum_{t=1}^{W-1} h_{i,j}(x_t, x_{t+1})}$$

Here, $h_{i,j}(x_t, x_{t+1})$ will be 1 only if the value of two adjacent pixels is (i, j). Otherwise, it will be 0. H denotes the number of image rows and N is the number of examples in the training set. Then,

we can calculate one image row which is considered as a markov chain transition probability ($IRTP$) value by the following formula:

$$IRTP_h = P_h(x_1, x_2) \times P_h(x_2, x_3) \times \cdots \times P_h(x_{W-1}, x_W)$$

where $P_h(x_i, x_{i+1})$ is the probability of adjacent pixels which the first pixel is i-th position pixel in h-th row. However, the value of P is so small that we use log in $IRTP$.

$$IRTP_h = \log P_h(x_1, x_2) + \log P_h(x_2, x_3) + \cdots + \log P_h(x_{W-1}, x_W)$$

The ITP is the sum of all $IRTP$ value.

$$ITP = \sum_{h=1}^{H} IRTP_h$$

Because of adding small perturbation, an adversarial example breaks Markov process nature for image row pixels. Thus, the ITP of adversarial examples is bigger than the one of normal examples. Through experiment results showed in Figure 3, we can find that the ITP efficiently distinguishes between adversarial examples and normal examples.
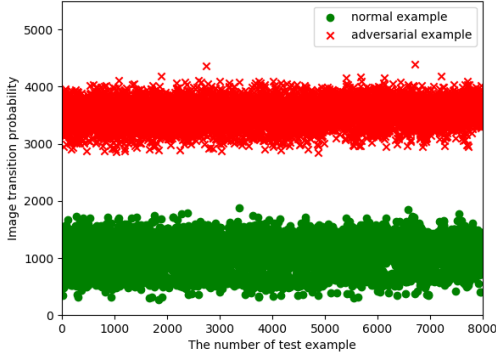
Figure 3. ITP on mnist datasets, red color represents adversarial examples and green color represents normal examples

Thus, the ITP is a sum of all markov chain transition probabilities which we consider a image row as a markov chain. Of course, image transition probability also can be got by considering one image column as a markov chain. So, ITP_r represents image transition probability which consider image row as markov chain. ITP_c represents image transition probability which consider image column as markov chain. In our work, image transition probability we have proposed denotes ITP_r. During detecting period, we compute the image transition probability of test example. If ITP of this example more than a the threshold ITP_m which can be got from training dataset, this example maybe adversarial example and need to be modify. Especially, we transform it into one channel image if this image is multi-channel image. The process of detecting is showed in algorithm 2.

3.2 Modifying adversarial examples

Next, we give the process of modifying image after detecting test example which is a possible adversarial example. It is difficult to sample some examples which can be correctly predicted for a adversarial

---

Algorithm 2 Procedure **Detect**

Input: A test example x and a threshold value ITP_m
Output: Boolean value
1: compute the image transition probability of x as ITP_x
2: **if** $ITP_m \leq ITP_x$ **then**
3:     return true
4: **else**
5:     return false
6: **end if**
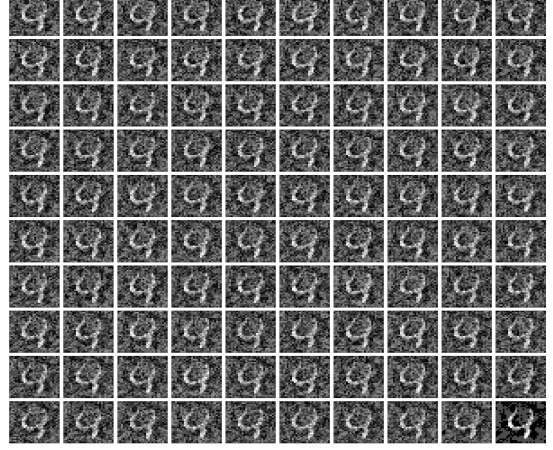
Figure 4. The figure shows the samplers from small region centered at adversarial examples, At last in the figure is adversarial examples which generated by FGSM

example. As figure 4 showed, the examples sampled around adversarial example is extremely unclear and obscure which has a large number of noise and can not recognized by us. So it is necessary to modify image to make the sampled examples with small noise. And we found that the image transition probability of adversarial example is bigger than normal example's. The image transition probability describes the difference on image noise. If the image has much noise, The image transition probability value will be big. Thus, we can modify image by making its image transition probability smaller before region-based classification.

The key idea of modify image is how to choose suitable pixels to change and how to select suitable values to represent for chose pixels so that make the image transition probability decrease. To address this problem, we firstly use saliency map to describe the impact of each pixel in an image. The saliency map can be got by following equation

$$S_{r,m,i,n} = ITP_{r,n} - ITP_{r,o}$$
$$S_{c,m,i,n} = ITP_{c,n} - ITP_{c,o}$$

where m represents m-th row or column. r denotes row. i is i-th pixel in one markov chain. The n denotes pixel value. So $ITP_{r,n}$ is the value of image transition probability which consider image row as a markov chain if current pixel value is n. ITP_o represents the value of image transition probability without modifying. $S_{r,m,i,n}$ denotes the change of image transition when i-th pixel value turn into n in m-th row. The $S_{c,m,i,n}$ represents the change of ITP_c that consider image column as a markov chain. Then, we choose pixels to modify with min $S_{r,m,i,n}$ in order to decrease ITP_r quickly. However the ITP_r of edge pixels which we do not need to modify are obviously bigger than the others. In other word, only when the all

$s_{r,m,i,n}$ is not exceed threshold ITP_t, we should choose suitable value to represent original value. this threshold ITP_t can be got in training datasets by search min value of all edge pixels. In our experiments, we take $ITP_t -2$ for MNIST datasets. Another challenge is choose suitable pixel n for chose pixel. we search n from 0 to 255 and choose n which satisfy min $S_{r,m,i,n}$ and $S_{c,m,i,n}$ is not more than 0.

3.3 Improving region-based classification

At last, we give a method called improved region-based classification after modifying detected adversarial examples. Region-based classification randomly uniformly sample some examples from the hypercube of test examples x and help test example predict its label. this hypercube can formally defined: $B(x,l) = \{y|y_i \in [0,1], |y_i - x_i| \le l, \forall i = 1, 2, ..., n\}$, where y_i and x_i are the i-th dimensions of x and y. However, as figure 4 showed, the examples sampled centered at adversarial example is extremely unclear and can not be recognized by us.

So we need to create a new area which can promise the example with small noise when test example is detected as adversarial example. we denote the new area $A(x,r)$ which is centered at test example x which we consider it as a matrix and different $x_{i,j}$ has different length $r_{i,j}$. Formally, $A(x,r) = \{y|y_{i,j} \in [0,1], |y_{i,j} - x_{i,j}| \le r_{i,j}, \forall j = 1, 2, ..., n\}$, where i are the i-th row of metric x, metric y, metric r and j are the j-th column of metric x, metric y, metric r. Choosing area centered at test example is equal to determine the metric r on test example x. Specifically, we learn the length l in r for every dimension of metric x though a search process described in algorithm 4. The key idea in search process is selected the max length l such that the image transition probability of example sampled from $A(x,r)$ is all smaller than the test example's. In our work, it will be modified to be x' if it is detected as adversarial example for a test example x. Initially, we set $r_{i,j}$ which is corresponding to $x_{i,j}$ be a small value. Then we increase $r_{i,j}$ until image transition probability is bigger than x.

Algorithm 3 Learning metric r by Searching

Input: A text example x with H rows and W columns, ITP value ITP_x of x and an updated image x' after modifying the ITP

1: $ITP_{x',i,j,z}$ denotes the image transition probability of x' when $x'_{i,j}$ becomes z.
2: $l = r_0, i = 0, j = 0$
3: **while** $i \le H$ **do**
4:     **while** $j \le W$ **do**
5:         $z1 = x'_{i,j}$, $z2 = x'_{i,j}$
6:         **while** $ITP_{x',i,j,z1} \le ITP_x$ and $ITP_{x',i,j,z2} \le ITP_x$ **do**
7:             $r_{i,j} = r_{i,j} + 1$
8:             $z1 = x'_{i,j} + r_{i,j}$, $z2 = x'_{i,j} - ri,j$
9:         **end while**
10:         $j = j + 1$
11:     **end while**
12:     $i = i + 1$
13: **end while**

---

4 Experiments

This section introduces our experiments on two common image datasets with our algorithm and compared methods.

4.1 Experiment setup

Datasets: we test our algorithm on two standard image datasets : MNIST and CIFAR-10. MNIST datasets includes a handwritten numbers images with 50000 training examples and 10000 testing examples which size is 28*28. CIFAR-10 has a total of 60,000 color images. These images are 32*32, divided into 10 categories, each category has 6000 images.

Nerual Networks: In our experiments, we use a common convolutional neural network described as following:

Layer type
Convolution + Relu
Max pooling
Convolution + Relu
Max Pooling
Convolution + Relu
Max Pooling
Fully connected
Softmax

compared methods: We use four algorithms to generate adversarial example, such as JSMA, DeepFool, FGSM and CW. Especially, we take ϵ 0.2 in FGSM algorithm. And we use l_∞ to generate adversarial examples in CW algorithm. Then we Compare our method with the following classifiers in same adversarial examples.

- **adversarial training**: First, we use adversarial training to learn a DNN model for each datasets. Recently, A popular adversarial training algorithm which was proposed by Madry et al [8] use PGD algorithm to generate adversarial examples and learn the DNN classifier with these adversarial examples and normal examples. However, These adversarial examples have so much noise that sacrifices classification accuracy significantly. Thus, we adopt DeepFool to get adversarial examples in adversarial training.
- **region-based classification**: For each datasets, we train a DNN classifier which structure is the same as adversarial training's. According to Cao et al work. we respectively set r which determine the size of region centered at test example to be 0.3 in MNIST datasets and take r 0.02 for CIFAR-10 datasets. In addition, we sample 100 examples in region-based classification for each testing example.
- **our methods**: The architecture of nerual networks is same with compared method's. we set ipt_m is 1800 in MNIST datasets and 2000 for CIFAR-10 during detecting adversarial examples. During region-based classification, we sample 100 examples for each testing example.

4.2 Results

First, we perform a experiment which test normal example on our methods and compared methods which test 5000 examples from MNIST testing datasets and CIFAR-10 datasets. The result is showed in Table 1 which show classification accuracy on normal examples. From the table 1, we can find that our methods get better accuracy on normal examples than adversarial training and achieve the same accuracy with standard networks. Next, we test totally 4000 adversarial examples of MNIST datatsets generated on testing datasets for region-based classification and our methods. These adversarial examples is generated by FGSM, JSMA, CW and DeepFool which each methods generated 1000 adversarial examples. The result is showed

Table 1. Classification accuracy on normal examples

	MNIST	CIFAR-10
standard CNNs	99.1%	89.8%
Adversarial training	98.3%	87%
region-based classification	99.1 %	89.8%
our methods	99.1 %	89.8%

in table 2. It showed that our methods perform better than region-based classification on mnist datasets. Especially in CW attack, our methods can evasion this attack.

Table 2. Classification accuracy on adversarial example

Method	MNIST datasets		CIFAR-10 datasets	
	region-based classification	our method	region-based classification	our method
FGSM	16.2%	54.5%	13.4%	52.0%
DeepFool	5.1 %	59.8%	10.6 %	50.3%
JSMA	7.3 %	46.2%	6.0 %	56.7 %
CW-L$_\infty$	23.1 %	79.8%	16.5 %	73.7%

In the end, we compare region-based classification and our method on CIFAR-10 datasets. CIFAR-10 datasets is a three channel image set. First, we use CIFAR-10 trianing datasets train a CNNs and randomly sample 1000 examples from CIFAR-10 testing datasets as testing examples. Then we use FGSM , CW and DeepFool methods to generate adversarial examples based on testing examples. Especially, we transform three channel image into one channel when carry out our detecting methods. Though experiments, as table 2 showed, our methods also achieve good accuracy on adversarial examples.

5 Conclusion and Future work

In this work, we propose a detect-and-modify region-based classifier to mitigate evasion attacks in DNNs. Firstly, we observe that region-based classification is limited by the surroundings of examples (especially the adversarial examples). Thus we use ITP, which can remarkably distinguish adversarial examples from normal examples, to detect whether the test example is an adversarial example or not. In order to improve the accuracy on adversarial examples, we modify the ITP of the test example which may be an adversarial example by decreasing its original ITP. Then we use our improved region-based classification to predict results on test examples.

We apply our approach on experiments with different adversarial examples generated by different methods and different datasets. The experimental results show that our method behave better than the traditional region-based classification. As far as we know, our work is the first work to improve the region-based classification by combining it with detecting methods. The improved method finally solves the issue that the region-based classification is limited by the surroundings of test examples. In the future, we will continue improving our method to make it better to modify images with a large perturbation.

6 Acknowledgement

REFERENCES

[1] Xiaoyu Cao and Neil Zhenqiang Gong, 'Mitigating evasion attacks to deep neural networks via region-based classification', in *Proceedings of the 33rd Annual Computer Security Applications Conference, Orlando, FL, USA, December 4-8, 2017*, pp. 278–287, (2017).

[2] Nicholas Carlini and David A. Wagner, 'Towards evaluating the robustness of neural networks', in *2017 IEEE Symposium on Security and Privacy, SP 2017, San Jose, CA, USA, May 22-26, 2017*, pp. 39–57, (2017).

[3] Bingcai Chen, Zhongru Ren, Chao Yu, Iftikhar Hussain, and Jintao Liu, 'Adversarial examples for cnn-based malware detectors', *IEEE Access*, **7**, 54360–54371, (2019).

[4] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy, 'Explaining and harnessing adversarial examples', in *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, (2015).

[5] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton, 'Imagenet classification with deep convolutional neural networks', in *Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems 2012. Proceedings of a meeting held December 3-6, 2012, Lake Tahoe, Nevada, United States*, pp. 1106–1114, (2012).

[6] Alexey Kurakin, Ian J. Goodfellow, and Samy Bengio, 'Adversarial examples in the physical world', in *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Workshop Track Proceedings*, (2017).

[7] Qiang Liu, Pan Li, Wentao Zhao, Wei Cai, Shui Yu, and Victor C. M. Leung, 'A survey on security threats and defensive techniques of machine learning: A data driven view', *IEEE Access*, **6**, 12103–12117, (2018).

[8] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu, 'Towards deep learning models resistant to adversarial attacks', in *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*, (2018).

[9] Roland Meier, Thomas Holterbach, Stephan Keck, Matthias Stähli, Vincent Lenders, Ankit Singla, and Laurent Vanbever, '(self) driving under the influence: Intoxicating adversarial network inputs', in *Proceedings of the 18th ACM Workshop on Hot Topics in Networks, HotNets 2019, Princeton, NJ, USA, November 13-15, 2019*, pp. 34–42, (2019).

[10] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, and Pascal Frossard, 'Deepfool: A simple and accurate method to fool deep neural networks', in *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, pp. 2574–2582, (2016).

[11] Nicolas Papernot, Patrick D. McDaniel, and Ian J. Goodfellow, 'Transferability in machine learning: from phenomena to black-box attacks using adversarial samples', *CoRR*, **abs/1605.07277**, (2016).

[12] Nicolas Papernot, Patrick D. McDaniel, Somesh Jha, Matt Fredrikson, Z. Berkay Celik, and Ananthram Swami, 'The limitations of deep learning in adversarial settings', in *IEEE European Symposium on Security and Privacy, EuroS&P 2016, Saarbrücken, Germany, March 21-24, 2016*, pp. 372–387, (2016).

[13] Nicolas Papernot, Patrick D. McDaniel, Xi Wu, Somesh Jha, and Ananthram Swami, 'Distillation as a defense to adversarial perturbations against deep neural networks', in *IEEE Symposium on Security and Privacy, SP 2016, San Jose, CA, USA, May 22-26, 2016*, pp. 582–597, (2016).

[14] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian J. Goodfellow, and Rob Fergus, 'Intriguing properties of neural networks', in *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*, (2014).

[15] Yue Zhou, Xiaofang Hu, Lidan Wang, Shukai Duan, and Yiran Chen, 'Markov chain based efficient defense against adversarial examples in computer vision', *IEEE Access*, **7**, 5695–5706, (2019).

Inspect Characteristics of Rice via Machine Learning Method

Xin Ma[1*] , **Mingliang Li**[1*] , **Jinxi Kong**[3] , **Siming Zhao**[4] , **Wei Li**[2†] , **Xiaohui Cui** [2†]

[1]School of Sofware, Tsinghua University
[2]School of Cyber Science and Engineering, Wuhan University
[3]GREE Electric Appliances
[4]College of Food Science and Technology, Huazhong Agricultural University
mx19@mails.tsinghua.edu.cn, lml19@mails.tsinghua.edu.cn, kongjinxi18@gmail.com,
zxmjx@mail.hzau.edu.cn, auto_weili@whu.edu.cn, xcui@whu.edu.cn

Abstract-**For more than half of humanity rice is life. Therefore, assessing the quality of rice in fast, accurate and objective methods has attracted a lot of attention of rice producers and processors. Unfortunately, the current inspection methods that focusing on the computer vision to inspect the characteristics of rice are either cost expensive (e.g., it needs extra sensors to assist photography) or need to be significantly improved in practice (e.g., framed object incorrectly). In this paper, we make an in-depth study of the characteristics of rice and explore an alternative direction to use machine learning methods to inspect them through photos taken by cell phones. To be exact, we develop a new mathematical variation formula and a new area calculation formula, combining clustering methods, to inspect the main characteristics of rice both statically and dynamically. The effectiveness of our approach is very visible no matter what the type of rice it is, which is shown by comprehensive experiments on four typical types of rice datasets. Moreover, We cooperate with one of the world's largest home appliance manufacturers, applying the rice characteristics extraction approach to produce smart electric rice cookers, thus improving the quality of life for millions of people.**

Keywords— **characteristics of rice; machine learning; digital image processing; IoT**

1 Introduction

Among the cereals, rice is the most important foodstuff in the world and nearly half the population of the world takes rice as a staple food. Rice is often damaged by the agricultural machinery during past-harvest handling and processing, such damaged could affect the quality and appearance. When consuming or processing rice, people prefer to obtain sound products with less fissure and breakage. Furthermore, people have a higher pursuit of the quality of rice cooking. For rice with different characteristics, we should adopt different strategies to cook to achieve the best taste. With the rapid development of Internet of Things (IoT), smart electric rice cookers are becoming much more popular. If we can identify the characteristics of rice conveniently and accurately and apply it to the rice cooker, we can improve the quality of life for many people by choosing a specific cooking strategy dynamically when cooking rice. In this way, inspecting and understanding the quality of rice is necessary.

In general, the methods of inspecting the characteristics of rice are categorized into two aspects. The first one is to use a special machine, the rice detector, to inspect the characteristics of rice. It scans each rice in a special closed space (the space can shield the natural light and just remains a single light-source to scan the rice like a scanner), and calculates the characteristics with specific machine-borne software. Although it can produce very precise results, the cost is very expensive, the size of such a machine is too huge to be a home appliance. The second one is to develop a computer vision algorithm to inspect the characteristics of rice, and it is a thriving direction currently, because of its potential flexibility and the extra cost is unnecessary. This is because one can use portable devices such as mobile phones[7] to take images and validate their algorithm on such images.

Traditional rice property testing is mainly based on large analytical instruments in laboratories and the result is used in agricultural research. Before the rapid development of Internet of Things (IoT), lightweight rice characteristic analysis was difficult to apply to production or life. However, with the popularity of smart rice cookers and the establishment of IoT, the demand for rice analysis that relies solely on pictures through computer vision and machine learning has increased rapidly. This also puts forward requirements on how we apply machine learning and computer vision technology.

Currently, directly using the traditional computer vision methods to detect the appearance of rice basically focuses on judging whether the rice is intact or broken with edge detection [4]. Such a case always assumes that there is only rice in an image, this is not often the case where the impurities and connected rice usually arise. The traditional computer vision method cannot handle this challenge.
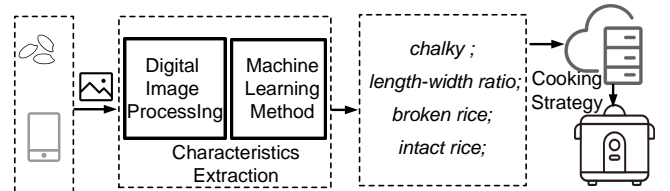
Figure 1: Application of rice characteristics extraction in smart rice cooker

In this study, we, instead, explore an alternative direction, which is to use machine learning to inspect the characteristics of rice. The five characteristics of rice can be grouped into two sets. The first group is chalky, it includes the percentage of chalky (The rice with chalkiness / All rice) and the chalkiness (Amount of chalky pixels / Amount of all pixels). The chalkiness indicates the untransparent region with a grain of rice. Although the chalkiness can measure the quality of the rice, few computer vision studies focus on it. The

second group consists of the length-width ratio, the percentage of broken rice (Broken rice / All rice) and intact rice (Intact rice / All rice).

For calculating the chalkiness, we develop a new mathematical variation formula to set a threshold to separate the chalkiness from other components, given that the chalkiness is the untransparent region and other components are transparent. All components within a grain of rice can be grouped into two aspects, the brightness and the darkness. The former one indicates the chalkiness in our study while the last one indicates other components. We can dynamically calculate the difference between the brightness and the darkness with our new formula. More details are shown in Section III.

For the second group of characteristics, we calculate them by using the clustering methods (e.g., K-Means or Spectral-Clustering [10]), which have been widely used in image classification[9]. To make a reasonable calculation, we consider the impurities and the connected rice. we develop a new area calculation formula and make it worked with the clustering method to cluster the four characteristics. We can observe more details in Section III. Our proposed method does not need an extra machine, and there is no need for people to learn professional knowledge.

Our characteristics extraction technology has been adopted by the world 's largest home appliance manufacturer to manufacture smart rice cookers. As shown in Fig.1, when rice pictures taken by mobile phones are uploaded, digital image processing and machine learning methods designed by us are used for characteristics extraction. Upload characteristics to the cloud service to request cooking strategies to achieve the effect of intelligent cooking. By the way, cooking strategies are provided by cooperative agricultural researchers.

In summary, the major innovations and contributions of this paper are as follows:

- This paper casts light on applying machine learning to inspect the rice characteristics. In particular, this paper introduces the clustering method to the impurities, broken rice, intact rice and connected rice.

- To the best of our knowledge, this paper proposes for the first time to develop a new mathematical variation formula to dynamically inspect the chalkiness.

- Through comprehensive experiments on inspecting the characteristics of rice, we demonstrate the effectiveness of the proposed approach. Meanwhile, Our characteristics extraction technology has been adopted by the world 's largest home appliance manufacturer to manufacture smart rice cookers, thus improving the quality of life for millions of people.

This paper is organized as follows. In Section II we discuss some related work. In Section III, We will present our main idea. In Section IV we will show our experiment results, and we conclude in Section V.

2 Related Work

Many image acquisition instruments and image processing algorithms are based on the computer vision method, and a lot of researchers try to use these instruments and algorithms to inspect the characteristics of foodstuff. Typically, we categorize them into two aspects: image acquisition systems and image processing methods.

Image acquisition systems for food. When researchers want to analyze the characteristics of food, the first step is to collect enough available images. In current, although there are many instruments, such as the USB-based camera, scanner, ultrasound, X-ray and near-infrared spectroscopy, to be used to collect images in practice, it still needs to design new machine, given that the food has different shapes and colors. Peter et al. [16] design a line-scan camera to take

the photo to obtain precise information (foreign body) when the food products pass through the camera, given that the food products can always mix the foreign body during processing. The camera reaches to 2000 times per minute. Besides, since some food products exist special characteristics (e.g., core or bones), researchers try to use the X-ray radiography to generate available images to analyze. Kim et al. [11] use a two-dimensional (2-D) X-ray radiography to detect whether an apple contains watercore or not, especially in the early stages. Although the machines tremendously facilitate people to improve the efficiency of agriculture product inspecting and handling, those machines belong to professional equipment and the cost is expansive.

Image processing methods for food. Image processing is the principal core for computer vision because the results from image processing are directly related to the goal of a task. Such steps can be categorized into two sub-steps, the pre-processing and processing. In some cases, the pre-pocessing step has been adopted to reduce the noise in the image to improve the quality of the image for inspection, with the method of enhancing the important features of interest [3]. In the process of quality inspection, Dissing et al. [3] use a rapid multispectral imaging device to quantify the degree of spoilage for pork, given different qualities of pork display different chromatic aberrations in spectral. It can classify 76.13% of the meat samples correctly. Since different foods hold different colors, Eddins et al. [8] use the histogram to calculate the threshold value of different foods to classify. Those methods belong to low-level image processing. As for high-level image processing, it involves objection recognition and interpretation, and always requires more complicated models. Ying et al. [17] use the Artificial Neural Network (ANN) [2] to classify the Huanghua pears.

Instead of directly inspecting the characteristics of rice using the computer vision method, we apply machine learning to inspect rice characteristics. In next section, we would introduce the specific technical details of our approach.

3 The proposed method

In this section, we will introduce how to apply the machine learning method to inspect the characteristics of rice. Specifically, we will discuss two important challenges: a new mathematical variation formula for getting chalkiness of rice, and use the machine learning method to get the quality measurements of rice(the broken rice ratio and the intact rice ratio). Since our focus is on calculating the characteristics of each rice, the first step is to extract rice from the picture.

3.1 Preprocessing

Although we can easily observe the rice in an image with the naked-eyes, it is a non-trivial thing for a computer since there are a lot of observations(e.g., noise, broken rice and overlapping rice) in an image, and we need to filter out what we expect. A picture with a lot of rice is shown in Fig.2.

In Fig.2, the rice shown in sub-figure (a) is taken by the cell-phone, while that shown in sub-figure (b) is from the camera with high resolution. The red ellipse marks the chalky grain (this rice is from the sub-figure (a). For illustrating the chalkiness we use a specific camera to take the photo). Before framing the rice shown in sub-figure (a) correctly, we need to know the position of each rice. Here we adopt the edge detection method. If we can detect the edge of each rice, we naturally find each rice. The clearer the edge is, the better the inspection is. Since the sub-figure (a) is taken in the natural scene, two major kinds of noise (the dust and the blot) are inevitable. The edge detection method also detects the edge of noise, which would affect the precision of framing rice. More details are shown in Section experiment. In this way, we need to filter out the

(a)   (b)
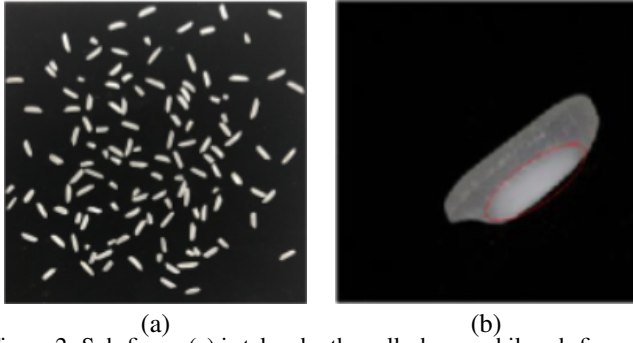
Figure 2: Sub-figure (a) is taken by the cell-phone, while sub-figure (b) is from the camera with high resolution. The untransparent region marked by the red ellipse indicates the chalkiness.

noise within this image before calculating the characteristics of rice. Because we do not want our algorithm mistakes the noise for rice. In our study, we adopt the Bilateral filter method to filter out the noise, because the Bilateral filter can preserve the edge of anobject,which is important for our future calculation. Although other filters can filter out more noise in some cases, they blurry the edge of the expected object. The remaining noise can be separated in the next step by clustering. We would use an example to demonstrate our point. We choose different filters and apply them to the image shown in sub-figure (a) of Fig.2, and the results are shown in Fig.3.

(a)   (b)   (c)   (d)   (e)

Figure 3: (a) indicates the original sample. (b) indicates the filtered sample with Gaussian filter [1].(c) indicates the filtered sample with Bilateral filter [15]. The filtered sample in sub-figure (d) is from the Average filter while that in sub-figure (e) is from the Median filter. From the results, we can see that the denoised result from Bilateral filter is better than others.

In Fig.3, we can see that the sub-figures (b), (d) and (e) blurry the edge of the rice even though they have denoised the noise. The blurry edge would cause the edge detection method to detect the wrong edge of rice. If this happened, the rice cannot be framed correctly, and we cannot calculate the true characteristic of rice. Despite the performance of denoise shown in sub-figure (c) is weaker than sub-figures (b), (d) and (e), the bilateral filter still eliminates most of the noise and keeps edges as complete as possible. Given that in most real cases the noise cannot be filtered out completely(even if we use other filters), we must process the remaining noise in future steps. So we choose the bilateral filter as the denoise filter. Besides, some rice may be connected, which also hurts the calculation of the characteristics but still do not get any treatment. Next, we will explore how to process the rice image further.

3.2   Frame the rice

First, we find out each rice edge and box it using the rectangle. We use the Bilateral filter to denoise the noise, and we adopt edge detection with the Canny operator to detect the edge of rice for framing. The results are shown in Fig.4.

From sub-figure (b) of Fig.4, we can see that the outlines of those rice are plotted correctly, even for noise. What we need to do is to track and find the smallest circumscribed rectangle of the individual edges. We continue to frame the contour of rice using the traditional computer vision methods (e.g., topological structural analysis method [6] [12]). This method uses the encoding method to give different integer values for different edges, the input image is a binary image, and it uses a function $f(i,j)$ to indicate each pixel value. The

value of $f(i,j)$ would be updated when the method scans the image. We frame each closed edge with the smallest circumscribed rectangle. The results, after applying the topological structural analysis method to the rice, are shown in sub-figure (c) of Fig.4.

Although those rice are framed correctly, there remain several challenges needing to be addressed. 1). The noise has also been framed. 2). Several rice grains overlapped are extracted by one frame, for the framed algorithm thinks that the connection area is integration. These two challenges affect the precision of calculating the characteristics.

Our method is based on the edge-detection (See the sub-figure (b) of Fig.4) because detecting the edge is necessary for understanding the shape of one object. Since the traditional method holds the low performance, we would introduce our model to improve the performance in the next subsection.

3.3   Using clustering method to frame the rice

Although noise hurts the performance of characteristics inspection of rice products, few studies focus on it. Researchers are just interested in how to find out the broken rice in a noise-free environment and try to disperse the rice grains to each other to avoid overlap. Sansomboonsuk et al. [14] use these features such as area, perimeter, circularity and shape compactness as criteria to classify the broken rice and intact rice. They also utilize the Fuzzy logic method to organize and classify the class of each kernel. Comparing with human inspection, the proposed method reaches to 90% accuracy and saves 70% of the time. However, there still exit challenges.

First, the required time may not be suitable for real-time processing operations. Second, the proposed method could not separate the line formed by touching kernels, which indicates that there may exist errors [18].

When we use the common cell-phone to take a photo of rice under the nature scene, the noise and the connected rice is inevitable. Since the four characteristics (broken rice, intact rice, dust, and connected rice) are different from each other in appearance, we try to address those issues using the appearance feature. To this end, we combine mathematical calculations with the clustering method to inspect the noise and the rice. Our goal is to categorize the circumscribed rectangles of the individual edges (including noise) into four categories (noise, broken rice, intact rice, and connected rice). We make use of the clustering method (K-Means over the case) to cluster these samples. We expect that a specific sample would be clustered into the corresponding category (e.g., the noise would be clustered around the noise category while the connected rice would be clustered around the connected category). Because the areas of noise are much smaller than that of intact rice, and the areas of connected rice are much larger than that of normal rice, the performance of the cluster can be guaranteed. More details are shown in section experiments.

Next, we would use an example to demonstrate our clustering method. Assuming there is a rice dataset $(X_1, X_2, ..., X_m)$, m indicates the number of rice. The function of X_i is shown as follows.

$$X_i = \begin{bmatrix} H_i \\ W_i \end{bmatrix} \tag{1}$$

In Eq.(1), H_i indicates the length of rice and W_i indicates the width of rice. In the initial step, we randomly initialize four categorical centers, which are marked as $\alpha_1, \alpha_2, \alpha_3, \alpha_4$, respectively. For each X_i, we label it as α_j in which X_i has the shortest distance from α_j. The function is shown in Eq.(2).

$$label_i = \arg_{1 \leq j \leq k} min(\sqrt{\sum_{i=1}^{n}(X_i - \alpha_j)^2}) \tag{2}$$

(a)   (b)   (c)
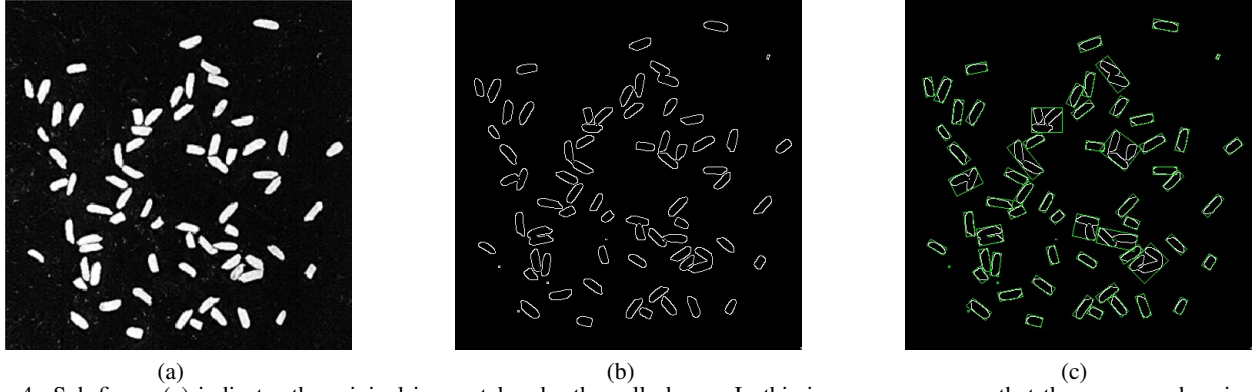
Figure 4: Sub-figure (a) indicates the original image taken by the cell-phone. In this image, we can see that there are much noise and connected rice. (b) indicates that we adopt edge detection with the Canny operator to plot the outline of rice after using the Bilateral filter. Although the Bilateral filter filters out most noises, there still remains noise. Sub-figure (c) indicates that we adopt the traditional method (e.g., [18]) to frame the outline of the rice.
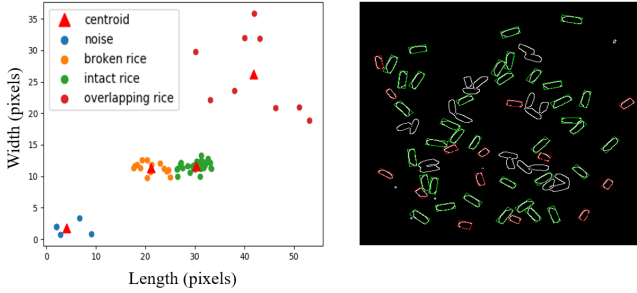
Figure 5: In the left sub-figure, the four red triangles indicate the categorical centers, and the four red eclipses indicate the four clusters (, broken rice, intact rice and overlapping samples). In the right sub-figure, the green rectangle frames the intact rice, while the red rectangle frames the broken rice. The other touched or connected rice is not framed.

In Eq.(2), k indicates the number of cluster (here $k = 4$). We proceed to perform the clustering method, the samples belonging to the categorical center α_j would be updated. The updated function is shown in Eq.(3).

$$\alpha_j = \frac{1}{N(c_j)} \sum_{i \in c_j} X_i \qquad (3)$$

In Eq.(3), c_j is such sample that holds the shortest distance from α_j, while $N(c_j)$ indicates the number of c_j. We repeat the Eq.(2) and the Eq.(3) until the change rate of the categorical center is less than 0.0001 (the number can be randomly set. In this study, we found that 0.0001 is enough to categorize all samples. The clustering results are shown in Section experiment. In this way, we can filter out the impurities and the connected rice and just keep the broken rice and intact rice. If we can frame the rice correctly, we can calculate the characteristics of length-width ratio and percentage of broken rice as well as that of intact rice.

3.4 Designing new variation formula to inspect the chalkiness

The chalkiness is an important criterion to measure the quality of rice, because it can affect the quality of appearance and the economy of rice. The chalkiness is caused by the insufficient accumulation of albumen starch and protein granules. The larger region the chalkiness is, the less the nutrition of rice is. The image of chalkiness is shown in sub-figure (b) of Fig.2. Fig.2 implicitly indicates that inspecting the chalkiness of one rice can rely on the color change.

An intact white-rice is actually translucent (the colored rice is not translucent so it has no chalkiness) when we observe it with naked-eyes. The chalky grain is the opaque region (See the marked eclipse of sub-figure (b) of Fig.2) and it stops the light transmission, while other regions allow the transmission of scattered light. Based on these characteristics, Fang et al. [4] hypothesize the chalky grain is brighter than other parts within a grain of rice, and they set a threshold to identify the chalkiness. However, there still exist some issues. 1). The threshold in [4] is fixed, and there are no more details to demonstrate how to find out such threshold. If the threshold is fixed, it is hard to distinguish the chalkiness under different chromatic aberrations which could affect the performance of the computer vision algorithm. 2). Although the threshold is related to the gray-level, the number of levels is hard to define because the fewer levels cannot distinguish the chalkiness and the larger levels could cause the loss of chalkiness. 3) The color of chalkiness also holds differences in practice. Also, the different shooting scenes could cause different color distributions in gray-level(especially shooting scenes by cell-phone). Thus, we propose our idea to inspect the chalkiness.

We use the chromatic aberration strategy to calculate the chalkiness instead of using a fixed threshold. The difference between our idea and traditional method [4] is that we first dynamically divide the rice pixels into two aspects (brightness and darkness), and then find out whether the difference between the two parts is significant, given that the shooting scenes are different. Even though the rice belongs to white rice, the white color is also different (See sub-figure (a) of Fig.2). Therefore, we need to determine the boundary dynamically.

Moreover, when the rice has been divided into two parts (brightness and darkness), the brightness cannot guarantee that it is real chalkiness (because when we divide the rice into two parts, it does not care about whether it exists the chalkiness or not. The brightness is only relative to the darkness, it may just a little lighter than darkness). The method shown in [4] does not care this detail. Here we make use of the maximum inter-cluster variance (the larger the inter-cluster variance, the more the difference) and the minimum intra-cluster variance (the smaller the intra-cluster variance, the closer the color of intra-cluster) to find out the optimal Boundary Value (BV) to distinguish the two parts. The proof of formulation is as follows.

$$u = w_0 \times u_0 + w_1 \times u_1 \qquad (4)$$

In Eq.(4), u indicates the number of pixels in an image, w_0 indicates the bright pixels proportion to whole image and w_1 indicates the dark pixels proportion to whole image. u_0 indicates the average of bright pixels while u_1 indicates the average of dark pixels. The

variation of two parts is shown in Eq.(5).

$$V = w_0 \times (u - u_0)^2 + w_1 \times (u - u_1)^2 \qquad (5)$$

In Eq.(5), u indicates the average gray of rice. We integrate the two equations into a new formula, which is shown in Eq.(6).

$$V = w_0 w_1 (u_0 - u_1)^2 \qquad (6)$$

The mean square deviation of bright pixels is shown in Eq.(7) while that of dark pixels is shown in Eq.(8).

$$V_0 = \frac{1}{w_0} \sum_{0 \le i \le BV} (i - u_0)^2 p_i \qquad (7)$$

$$V_1 = \frac{1}{w_1} \sum_{BV \le i \le 255} (i - u_1)^2 p_i \qquad (8)$$

In Eq.(7) and Eq.(8), the p_i indicates the frequency of ith gray level. Thus, the boundary value can be formulated as follows.

$$BV = \arg \max_{0 \le BV \le 255} \frac{V}{V_0 V_1} \qquad (9)$$

According to the boundary value (BV), we can divide the rice into two parts (brightness and darkness) no matter what kind of rice is or scene is. Then, we calculate the average value of gray levels of the two parts. When the value reaches to a certain range (Here we use the term T to indicate this range so T can be viewed as a threshold), we can point out that the color difference between the two parts is significant and the two parts can be distinguished. Note it is feasible to use T as a fixed value here We define the brightness part as the chalkiness (that is to say, $|u_0 - u_1| > T$). We empirically recommend that $T \in [20, 30]$ is a good value for distinguishing the chalkiness of the rice, the larger one or the smaller one cannot distinguish the same thing. We can validate our idea in Section experiment.

4 Experiments

To validate our approach, we choose four types of rice, which are GangTeYou37 (red-brown rice), purple glutinous rice, red glutinous rice, and Thai fragrant rice (white rice), respectively. We take the traditional algorithms (e.g., [4], [18], [13]) as the baselines and compare the baselines with our idea.

4.1 Inspecting broken rice and intact rice with K-Means

	length-width ratio	height	width	area
mean	2.2119	30.4385	14.8607	486.4525
std	0.5262	7.8109	6.8117	343.2865
min	1.0104	16.9706	9.8	191.9999
25%	1.7415	26.0325	11.303	286.6873
50%	2.4289	30.0744	11.7041	355.3846
75%	2.6057	32.9344	12.6196	405.2769
max	2.9706	52.6264	35.1187	1498.3458

Table 1: The four features (length-width ratio, height, width, area) indicate the features of rectangle which is used to frame the rice. All framed samples are 60. The mean, std, min, max indicate the mean-value, variation, minimum value and maximum value of those 60 samples, while the 25%, 50% and 75% indicate the quantile [5].

From the visual perspective, we find that the connected rice has been viewed as an intact object to be framed and the framed region is larger than that of single intact rice, while the region of noise is smaller than that of intact rice. To validate our hypothesis, we calculate each framed region and the corresponding height and width. The results are shown in Table.1.

In Table.1, the three quantiles indicate that they have similar values on height, width, and area, while the values of min or max display the abnormally small or large value, which could be noise or connected samples in the original image. Therefore, we think that if we set a categorical center for noise or connected samples, and the small values or larger values are clustered into this center, we can filter out the noise or connected samples. In this way, we adopt the K-Means to cluster these samples. K-Means is one of the simplest clustering strategies. If we can get good performance on K-Means, we can also get good performance on other clustering strategies.

In K-Means, the number of clusters is set to 4 (k=4), which indicates the noise, the broken rice, the intact rice, and the connected rice, respectively. From Table.1, we can see that the minimum value and maximum value are very different from the three quantiles. The center point of each cluster is randomly set. Because the samples within the four clusters hold different region values, K-Means can guarantee that holding the same or similar region value would be grouped into the same cluster after iteration. The parameters of K-Means if default and the clustering processes are as follows.

- We view each frame as an object, and the length and width of this object would be regarded as the features.

- We randomly initialize the central values of k.

- The rice would be grouped into a certain cluster with the shortest distance.

- We use the Eq.(6) to repeat the last step until the change rate of each categorical center is less than 0.0001, given that the four categories have their special features on the shape (e.g., different lengths and widths).

The results are shown in Fig.5. In sub-figure (a) of Fig.5, the left-bottom cluster, and the right-top cluster indicate the noise and the connected samples, respectively. We then abandon the two abnormal clusters using our proposed method, the results are shown in sub-figure (b).

In comparison with sub-figure (c) of Fig.4, Fig.5 shows the excellent effectiveness. We can see that the broken rice and the intact rice are framed correctly with different colors, while the and connected rice are filtered out. Because we abandon them during clustering, they have not framed by the rectangle. In this way, we can calculate the length-width ratio, the percentage of broken rice and intact rice.

4.2 Inspecting the chalkiness

We continue to inspect the chalkiness. In general, the chalky is the opaque part of the rice, and we can observe it with our naked-rice (See Fig.2). For inspecting the chalkiness, we need to pick up each framed rice. Here we use the binarization method to collect the framed rice from an image. After that, we continue to separate the opaque chalky from the transparent region using Eq.(9). Here we set T as 20 (we also test other values (from 21 to 30) of T, the results are similar. Larger T or less T would cause that we cannot distinguish the chalky), and the separated results are shown in Fig.6.

In Fig.6, the sub-figure (a) indicates the original image. We set T to 20 and we use Eq.(9) to separate the chalky from the intact rice, the sub-figure (b) shows the separated result.

4.3 The results of characteristics

After that, We use our methods to calculate the five characteristics (Length-Width Ratio (LWR), percentage of Intact Rice (IR),

Table 2: The traditional algorithms are viewed as baselines, and the results produced by the machine (Automated seeds and grains analyzer) are viewed as the benchmark. The results show that our algorithm approaches the benchmark, which indicates the effectiveness of our approach.

	LWR	IR (%)	BR (%)	Cky (%)	Cki
Thai fragrant rice (machine)	3.41	96.13	3.87	18.32	3.33
Thai fragrant rice (our approach)	3.21	90.32	9.6	15.16	2.11
Thai fragrant rice (traditional approach)	2.84	75	25	10.92	0.5
Purple glutinous rice (machine)	2.632	90.42	9.58	0	0
Purple glutinous rice (our approach)	2.82	89.36	10.63	0	0
Purple glutinous rice (traditional approach)	3.52	96.88	3.12	0	0
Red glutinous rice (machine)	1.88	99.03	0.97	0	0
Red glutinous rice (our approach)	1.86	95.35	4.6	0	0
Red glutinous rice (traditional approach)	2.05	90.3	9.7	0	0
GangTeYou (machine)	2.811	93.89	6.11	0	0
GangTeYou (our approach)	2.79	91.63	8.36	0	0
GangTeYou (traditional approach)	3.06	92.04	7.95	0	0

Figure 6: The left sub-figure shows the original image, while the right sub-figure shows the separated chalky with Eq.(9).

percentage of Broken Rice (BR), percentage of Chalky (Cky) and Chalkiness (Cki)) on four types of rice (red-brown rice, purple glutinous rice, red glutinous rice, and Thai fragrant rice), and the results are shown in Table

In Table.2, the results from the traditional approaches ([4], [18], [13]) are viewed as the baselines, and the results generated by the machine (Automated seeds and grains analyzer) are regarded as the benchmark. The results show that our algorithm approaches to the benchmark and outperforms traditional algorithms, which shows the effectiveness of our approach.

5   Conclusion

In this study, we've created a lightweight and high-precision method to inspect the major characteristics of rice (length-width ratio, chalkiness, the percentage of chalky and the percentage of broken rice and intact rice). We have achieved our goal by using a combination of digital image processing and machine learning algorithms. Our work can effectively extract the characteristics of rice from the photos taken by mobile phones. In the era of the IoT, we cooperate with one of the world's largest home appliance manufacturers to help intelligent rice cookers make more intelligent decisions based on the rice characteristics extraction technology.

References

[1] Volker Aurich and Jörg Weule. Non-linear gaussian filters performing edge preserving diffusion. In *Mustererkennung 1995*, pages 538–545. Springer, 1995.

[2] Judith E Dayhoff and James M Deleo. Artificial neural networks. *Cancer*, 91(S8):1615–1635, 2001.

[3] Bjørn Skovlund Dissing, Olga S Papadopoulou, Chrysoula Tassou, Bjarne Kjær Ersbøll, Jens Michael Carstensen, Efstathios Z Panagou, and George-John Nychas. Using multispectral imaging for spoilage detection of pork meat. *Food and Bioprocess Technology*, 6(9):2268–2279, 2013.

[4] Changyun Fang, Xianqiao Hu, Chengxiao Sun, Binwu Duan, Lihong Xie, and Ping Zhou. Simultaneous determination of multi rice quality parameters using image analysis method. *Food analytical methods*, 8(1):70–78, 2015.

[5] Michael Frigge, David C Hoaglin, and Boris Iglewicz. Some implementations of the boxplot. *The American Statistician*, 43(1):50–54, 1989.

[6] M. B. Fuchs. Topological structural analysis. *Structural Optimization*, 13(2-3):104–111, 1997.

[7] Wenwen Gong, Lianyong Qi, and Yanwei Xu. Privacy-aware multidimensional mobile service quality prediction and recommendation in distributed fog environment. *Wireless Communications and Mobile Computing*, 2018, 2018.

[8] Woods Gonzalez and Richard E Woods. Eddins, digital image processing using matlab. *Third New Jersey: Prentice Hall*, 2004.

[9] Yi Hong and Weiping Zhu. Spatial co-training for semi-supervised image classification. *Pattern Recognition Letters*, 63(oct.1):59–65, 2015.

[10] Yi Hong and Weiping Zhu. Learning visual codebooks for image classification using spectral clustering. *Soft Computing*, 22(6):1–10, 2017.

[11] S Kim and TF Schatzki. Apple watercore sorting system using x-ray imagery: I. algorithm development. *Transactions of the ASAE*, 43(6):1695, 2000.

[12] Wei Li, Xiao Liu, Jin Liu, Ping Chen, Shaohua Wan, and Xiaohui Cui. On improving the accuracy with auto-encoder on conjunctivitis. *Applied Soft Computing*, 81:105489, 2019.

[13] Xu Li. Research on measurement of rice plumule ratio by machine vision [j]. *JOURNAL OF JIANGSU UNIVERSITY OF SCIENCE AND TECHNOLOGY*, 6:001, 1997.

[14] Siriluk Sansomboonsuk and Nitin Afzulpurkar. The appropriate algorithms of image analysis for rice kernel quality evalution. In *20th conference of mechanical engineering network of Thailand, Bangkok, Thailand*, pages 18–20, 2006.

[15] Carlo Tomasi and Roberto Manduchi. Bilateral filtering for gray and color images. In *Computer Vision, 1998. Sixth International Conference on*, pages 839–846. IEEE, 1998.

[16] Peter Wallin and Peter Haycock. *Foreign body prevention, detection and control*. Blackie Academic & Professional, 1998.

[17] Y Ying, H Jing, Y Tao, and N Zhang. Detecting stem and shape of pears using fourier transformation and an artificial neural network. *Transactions of the ASAE*, 46(1):157, 2003.

[18] Hemad Zareiforoush, Saeid Minaei, Mohammad Reza Alizadeh, and Ahmad Banakar. Potential applications of computer vision in quality inspection of rice: A review. *Food Engineering Reviews*, 7(3):321–345, 2015.

Modeling and Verifying
NDN-based IoV Using CSP

Ningning Chen[1], Huibiao Zhu[1,*], Jiaqi Yin[1], Lili Xiao[1], Yuan Fei[2,*]

[1]Shanghai Key Laboratory of Trustworthy Computing,
East China Normal University, Shanghai, China
[2] College of Information, Mechanical and Electrical Engineering,
Shanghai Normal University, Shanghai, China

Abstract—As a crucial component of intelligent transportation system, Internet of Vehicles (IoV) plays an important role in the smart and intelligent cities. However, current Internet architectures cannot guarantee efficient data delivery and adequate data security for IoV. Therefore, Named Data Networking (NDN), a leading architecture of Information-Centric Networking (ICN), is introduced into IoV. Although problems about data distribution can be resolved effectively, the combination of NDN and IoV causes some new security issues.

In this paper, we apply Communicating Sequential Processes (CSP) to formalize NDN-based IoV. We mainly focus on its data access mechanism and model this mechanism in detail. By feeding the formalized model into the model checker Process Analysis Toolkit (PAT), we verify four vital properties including deadlock freedom, data availability, PIT deletion faking and CS caching pollution. According to verification results, the model cannot ensure the security of data with the appearance of intruders. To solve these problems, we adopt a method derived from Blockchain in our improvement. Through the analysis of the improved model, we can truly guarantee the security of NDN-based IoV.

Index Terms—NDN, IoV, CSP, Blockchain, Modeling Verification

I. INTRODUCTION

Internet of Vehicles (IoV) [1] arouses wide public concern in both industry and academia sectors files. However, the current Internet is a point-to-point communication and channel-based security model. In IoV, the current Internet architectures cannot ensure high-efficiency data distribution and sufficient data security. To resolve this problem, Name Data Networking (NDN) [2] is introduced into IoV. NDN is a crucial architecture of Information-Centric Networking (ICN) [3]. NDN uses data names instead of IP addresses to retrieve and identify data. Due to this characteristic, NDN can better meet IoV's demand for big data processing.

There are some work on the related files of IoV. Abbas et al. proposed an road-aware estimation model for path duration in IoV [4]. In order to develop software-defined wireless network in IoV, Chien et al. created a SFC-based access point switching mechanism [5]. However, problems about data distribution and data security still exist. Therefore, NDN was applied to IoV in some researches. Su et al. presented a novel framework of a content-centric vehicular network (CCVN) [6]. Chowdhury et al. created a novel forwarding strategy CCLF [7]. Kalogeiton et al. defined a geographical aware routing protocol using

directional antennas [8]. From these existing work, we find that they mainly focus on routing and forwarding. Unfortunately, there are still many security issues in NDN-based IoV.

In this paper, formal methods are used to verify and analyze NDN-based IoV. This paper uses Communicating Sequential Processes (CSP) [9] to formalize the system. Using Process Analysis Toolkit (PAT) [10], we verify four properties (deadlock freedom, data availability, PIT deletion faking and CS caching pollution). Under the interference of intruders, the last three verification properties are invalid for the model. This paper adopts a method based on Blockchain in the improvement. All properties are satisfied for the improved model.

The rest of this paper is organized as follows. Section II presents an overview of NDN-based IoV and Blockchain. In Section III, we formalize the model of NDN-based IoV. In Section IV, we verify four properties and give improvement to the model. Finally, conclusion and future work are described in Section V.

II. BACKGROUND

This section gives a brief introduction of NDN-based IoV and Blockchain, especially combinations of them. After that, we also describe process algebra CSP.

A. NDN-based IoV

Some studies apply NDN to IoV to improve its security and performance. To further improve its performance, a vehicle may obtain much data by sending an interest packet. So we must take separation of interest packets and aggregation of data packets into consideration. Vehicles could be consumers or it could be producers of data. This scheme contains the following entities :

- **Consumer and Producer:** Consumers are vehicles who request and consume data packets. As a producer, the vehicle produces and provides data packets.
- **RSU (Road Side Unit):** An RSU forwards interest packets and data packets for vehicles. An RSUC and an RSUP represent the behavior of an RSU which communicates with a consumer and a producer separately.
- **Router:** Routers manage the access processes.

When a consumer wants to get a data packet, the following sequence of actions occurs:

- A consumer sends an interest packet containing the required data names to an RUSC.
- The RSUC transmits the interest packet to a router.
- The router addresses the interest packet with process *Interest Processing*. If there is a data packet matching this interest packet in its CS, the router returns this data packet. Otherwise, this process goes to next step.
- A router obtains the corresponding data through network propagation.
- Then it deals with received data packets through process *Data Processing* and may forward a data packet made up of all received data to corresponding RSUC.
- The consumer gets the data packet from the RSUC and verifies the data packet.

The process of producing data is defined similarly, the following sequence of actions occurs:

- A router gets an interest packet from the network and disposes it with process *Interest Processing*.
- A special case is that the matching data packet can be got from a producer managed by this router.
- The router sends the interest packet to an RSUP.
- The RSUP obtains the data packet from a producer.
- The RSUP returns it to the router.
- The router addresses this data packet with process *Data Processing*.
- According to the verification results of this data packet, it might be injected into the network.

As shown in Fig.1, the interest packets and data packets forwarding processes are described as follows:

Interest Processing: Whenever a router receives an interest packet, it queries CS (Content Store) to find out a data packet matching received interest packet. If so, the data packet is returned to the corresponding request entities and the interest packet is discarded. Otherwise, the router finds matching entries in PIT (Pending Interest Table). If it finds an entry, it just adds the incoming interface of the interest packet to that entry. If not, the router queries its FIB (Forwarding Information Base) to find the outgoing interface for each data name. If the router finds out all interfaces, it adds a new entry to PIT including the data names and incoming interface of the interest packet. Then it puts the data names of the same forwarding interface into a new interest packet and forwards those new interest packets. Otherwise, this received interest packet is thrown away.

Data Processing: When a router receives a data packet, it traverses PIT with packet's name firstly. If there are interest packets waiting for the data packet, the router saves the packet into CS. If the router receives all the data required for a PIT entry, the entry is deleted. The data packet containing all data is emitted from the interfaces that the entry records. The data packet is cached into the CS.

B. Blockchain

We bring Blockchain into NDN-based IoV in this paper. The blockchain generation process contains two parts. On the one hand, transactions are produced, forwarded and verified.
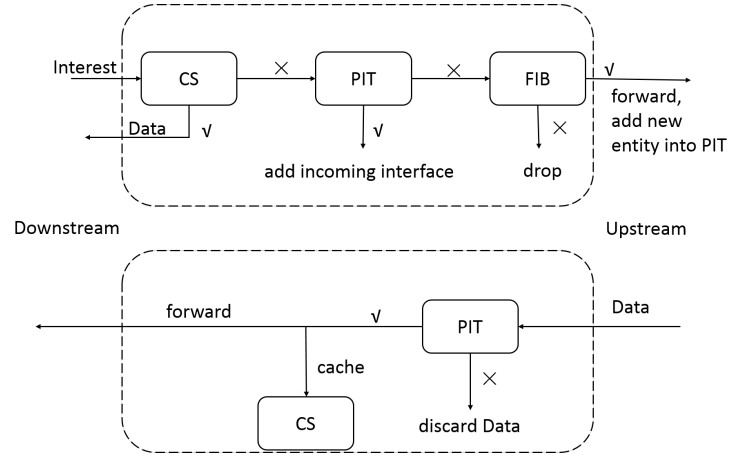
Fig. 1. Processing of interest and data packets in NDN [2]
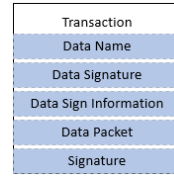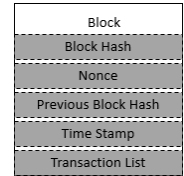
Fig. 2. Transaction   Fig. 3. Block

On the other hand, entities produce, forward, verify and store blocks. In Section IV, we give a detailed description for these two parts. The structures of transactions and blocks are shown in Fig.2 and Fig.3 respectively. *Data Packet* represents the result of our repeated hash calculation of hash values (*Data Name*, *Data Signature* and *Data Sign Information*). A producer signs these hash values using its private key. Those components make up a transaction. A blockchain is composed of blocks. A block contains *Block Hash*, *Nonce*, *Previous Block Hash*, *Time Stamp* and *Transaction List*. *Block hash* is the unique identifier distinguishing a block. All transactions of this block are stored in *Transaction List*.

In the improvement of NDN-based IoV, the system undergoes the following main changes:

- Producers create a transaction for each data packet and transmit the transaction to other entities.
- If a transaction passes verification, routers store and forward it.
- A router creates a block containing all transactions what it has saved so far. Then the router adds the block into its blockchain and forwards this block.
- According to verification results of a received block, entities add this block into their blockchains and delete their transactions that are repeated in this block.
- When an entity receives a data packet, it verifies received data packet using its blockchain firstly.

C. CSP

This section is used to introduce CSP (Communication Sequential Process). We give part of CSP syntax as follows:

$$P, Q \ ::= \ SKIP \mid STOP \mid a \to P \mid c?x \to P \mid c!u \to P \mid P; Q \mid$$
$$\mid P||Q \mid P \square Q \mid P \triangleleft B \triangleright Q \mid P[[a \leftarrow b]] \mid P[|c|]Q$$
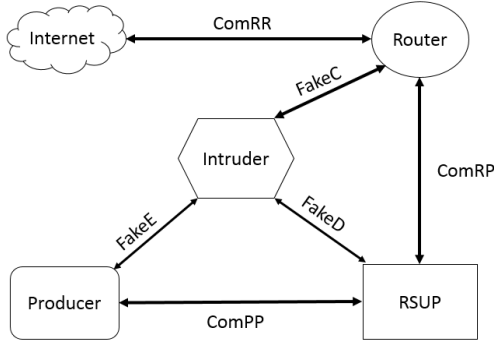
Fig. 4. Producer Modeling

Fig. 5. Consumer Modeling

- $SKIP$ represents that a process terminates successfully.
- $STOP$ indicates that a process runs into a deadlock state.
- $a \to P$ denotes that a process P executes after event a.
- $c?x \to P$ represents that a process receives a value and assigns it to the variable x before executing process P.
- $c!x \to P$ describes that a process sends a value v through channel c, then process P is executed.
- $P; Q$ represents that process Q is executed after process P terminates successfully.
- $P||Q$ denotes that process Q and process P are executed in parallel.
- $P\square Q$ stands for general choice. Selecting process P or process Q depends on external environment.
- $P \triangleleft B \triangleright Q$ is a conditional choice. If boolean expression B is true, process P will be executed. Otherwise, process Q is executed.
- $P[[a \leftarrow b]]$ indicates that a process changes event a for event b.
- $P[|c|]Q$ represents that processes P and Q execute the concurrent events on the set c of channels.

III. MODELING NDN-BASED IOV MODEL

In this section, we model NDN-based IoV by using CSP. This model is formalized based on Section II.

A. Sets, Messages and Channels

For convenience, this section gives some crucial information about sets, messages and channels models used in our formulation. Six sets are defined. *Entity* set contains entities including *Consumer*, *Producer*, *RSUC*, *RUSP* and *Router*. *Name* set denotes the names of data. **PRKey** set is composed of entities' public keys. **PUKey** set includes entities' private keys. The set *SigInfo* indicates signature information. *Content* represents other message contents.

In order to describe message packets transmitted between entities, this section defines some messages based on those definitions. *E(k,c)* indicates that key k encrypts content c. Each message includes a tag from the set $\{msg_{int}, msg_{data1}, msg_{data2}\}$. The messages are transmitted among entities as follows:

$$MSG_{int} = \{msg_{int}.a.b.n \mid a,b \in Entity, n \in Name\}$$
$$MSG_{data1} = \{msg_{data1}.a.b.n.E(K1^{-1},c).sigin \mid$$
$$a,b \in Entity, K1^{-1} \in PRKey,$$
$$n \in Name, c \in Content, sigin \in SigInfo\}$$
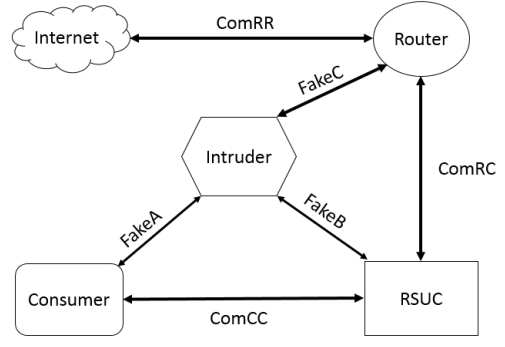
$$MSG_{data2} = \{msg_{data2}.a.b.n1.n2.E(K1^{-1},c1).$$
$$E(K2^{-1},c2).sigin1.sigin2 \mid$$
$$a,b \in Entity, K1^{-1}, K2^{-1} \in PRKey,$$
$$n1,n2 \in Name, c1,c2 \in Content,$$
$$sigin1, sinin2 \in SigInfo\}$$
$$MSG_{data} = MSG_{data1} \cup MSG_{data2}$$
$$MSG = MSG_{int} \cup MSG_{data}$$

MSG_{data} and MSG_{int} represent the messages of data packets and interest packets respectively. MSG denotes the messages between all entities.

To model the communication among components, we give the definitions of channels.

- channels between consumers, RSUC, routers and RSUP described by *COM_PATH*:
$$ComCC, ComRS, ComRR, ComRP, ComPP$$
- channels of intruders intercepting consumers, RSUC, routers and RSUP constituted by *INTRUDER_ PATH*:
$$FakeA, FakeB, FakeC, FakeD, FakeE$$

The declarations of the channels are as follows:

$$Channel\ \ COM\_PATH, INTRUDER\_PATH : MSG$$

B. Overall Modeling

We define a CSP model *System0* without intruders. *System0* is composed of *SystemC* and *SystemP*. *SystemC* and *SystemP* indicate a system which consumes and produces data respectively. The *SystemC* model is made up of three main entities including *Consumer*, *RSUC* and *Router*. *SystemP* model contains *Producer*, *RSUP* and *Router*. To take intruders into consideration, we create *SYSTEM* based on *System0*.

$$SystemC =_{df}$$
$$Consumer[|COM\_PATH|]RSUC$$
$$[|COM\_PATH|]Router$$
$$SystemP =_{df}$$
$$Producer[|COM\_PATH|]RSUP$$
$$[|COM\_PATH|]Router$$
$$System0 =_{df}$$
$$SystemC[|COM\_PATH|]SystemP$$
$$SYSTEM =_{df}$$
$$System0[|INTRUDER\_PATH|]Intruder$$

Consumer, *Producer* and *Router* denote the behavior of the consumers, producers and routers respectively. *RSUC* and

RSUP describe the actions of an RSU which communicates with a consumer and a product respectively. Considering the existence of intruders, this part defines *Intruder* which intercepts and fakes the messages. Fig.4 and Fig.5 describe interprocesses communication between processes.

C. Router Modeling

In NDN-based IoV, a router is responsible for forwarding interest packets and getting data packets. A router incisions and distributes an interest packet on the grounds of its routing information. For simplicity, each interest packet contains a maximum of two data names in our formalization. Then the router aggregates and forwards received data packets on the basis of verification results. This section formalizes a process $Router_{io}$ without intruders.

$Router_{io}(CSTable_i, PITTable_i, FIBTable_i) =_{df}$
$Initia\{inFIB = false; inCS = false; inPIT = false;$
$inFIB1 = false; inFIB2 = false; delPIT = false;$
$addcs = false; ininterface = 0; outface1 = 0; outface2 = 0\}$
$\rightarrow ComRS_i?B.C.dn1.dn2 \rightarrow ininterfece := i \rightarrow$

$$\begin{pmatrix} (inCS = true) \\ \lhd(\exists entry \in CSTable_i \bullet entry.data1name == dn1 \\ \land entry.data2name == dn2) \rhd (inCS = false) \end{pmatrix};$$

$$\begin{pmatrix} (ComRS_{interface}!C.B.dn1.dn2.CSTable_i[CSindex][2] \\ .CSTable_i[CSindex][3].CSTable_i[CSindex][4]. \\ CSTable_i[CSindex][5].CSTable_i[CSindex][6]) \\ \lhd(inCS == true) \rhd (NOCS_i) \end{pmatrix};$$

$Router_{i0}$

There are several variables appeared in the process. *i* is the ID of a router and its channels. A router knows its CS table $CSTable_i$, PIT table $PITTable_i$ and FIB table $FIBTable_i$. *inFIB*, *inFIB2* and *inFIB2* represent querying results for FIB. Similarly, *inCS*, *inPIT* and *inPIT1* denote searching results for CS and PIT respectively. *delPIT* indicates the router deletes PIT entries. *addcs* represents the router adds a data packet into CS. *ininterface* records the incoming interfaces of interest packets. *outinterface1* and *outinterface2* are the outgoing interfaces of interest packets.

First, the router receives an interest packet including the names of required data and records the incoming interface for the interest packet. Then the router checks its CS to find if there is a data packet matching the interest packet. If the result is positive, the data packet is returned to the corresponding request nodes and the interest packet is discarded. Otherwise, we define process $NOCS_i$ to address the situation in which the desired data packet cannot be obtained from CS.

$NOCS_i =_{df}$

$$\begin{pmatrix} \begin{pmatrix} inPIT = true; \\ AddPIT(dn1, dn2, entry.index, \\ interface, PITTable_i) \end{pmatrix} \\ \lhd(\exists entry \in PITTable_i \bullet entry.data1name == dn1 \\ \land entry.data2name == dn2) \rhd \\ (inPIT = false; NeedForward_i) \end{pmatrix};$$

$NOCS_i$ traversals $PITTable_i$ according to the names in the interest packet. If $NOCS_i$ finds a matching entry, the router adds incoming interface of the interest packet into the entry and discards the interest packet. If not, we give process $NeedForward_i$ to forward the interest packet.

$NeedForward_i =_{df}$

$$\begin{pmatrix} (inFIB1 = true; outface1 = entry2.outface) \\ \lhd(\exists entry2 \in FIBTable_i \bullet entry2.dataname == dn1) \rhd \\ (inFIB1 = false;) \end{pmatrix};$$

$$\begin{pmatrix} (inFIB2 = true; outface2 = entry2.outface) \\ \lhd(\exists entry2 \in FIBTable_i \bullet entry2.dataname == dn1) \rhd \\ (inFIB2 = false) \end{pmatrix};$$

$$\begin{pmatrix} (SKIP) \lhd (!(inFIB1 \land inFIB2)) \rhd \\ \begin{pmatrix} (AddPIT1(dn1, dn2, PITlength, \\ interface, PITTable); Forward_i \end{pmatrix} \end{pmatrix}; SKIP$$

$NeedForward_i$ queries $FIBTable_i$ with the names of the interest packet. If process $NeedForward_i$ finds an outgoing interface for each name, $NeedForward_i$ adds a new entry into $PITTable_i$ including the data names and incoming interface of the interest packet. Then we define process $Forward_i$ to forward the interest packet according to query results for each name. If $NeedForward_i$ cannot find all outgoing interfaces, the interest packet is dropped.

$Forward_i =_{df}$

$$\begin{pmatrix} \begin{pmatrix} ComRR_{outface1}!msg_{int}.C.D.dn1.dn2 \rightarrow \\ ComRR_{outface1}?msg_{data}D.C.dn1.dn2. \\ E(prk1\_Key, data1).E(prk2\_Key, data2). \\ siginfo1.siginfo2 \rightarrow Datapacketreceive_i \end{pmatrix} \\ \lhd(outface1 == outface2) \rhd \\ \begin{pmatrix} (ComRR_{outface1}!msg_{int}.C.D.dn1 \rightarrow \\ ComRR_{outface1}?msg_{data}D.C.dn1. \\ E(prk1\_Key, data1).siginfo1 \rightarrow \\ ComRR_{outface2}!msg_{int}.C.D.dn2 \rightarrow \\ ComRR_{outface2}?msg_{data}D.C.dn2. \\ E(prk2\_Key, data2).siginfo2 \rightarrow \\ Datareceive_i \end{pmatrix} \end{pmatrix}; SKIP$$

If all data can be got by one outgoing interface, $Forward_i$ transmits the interest packet from this interface and waits a matching data packet. If not, $Forward_i$ produces new interest packets by putting the data names of the same outgoing interface into a new interest packet. Then $Forward_i$ forwards each interest packet from corresponding outgoing interfaces and waits data packets. $Datareceive_i$ deals with received data packets.

$Datareceive_i =_{df}$

$$\begin{pmatrix} \begin{pmatrix} inPIT = true; Datasend_i(entry3.index); \\ DelPIT(dn1, dn2, PITTable_i); \\ AddCS(dn1, dn2, E(prk1\_Key, data1), \\ E(prk2\_Key, data2), \\ siginfo1, siginfo2, CSTable_i) \\ delPIT = true; addcs = true \end{pmatrix} \\ \lhd(\exists entry3 \in PITTable_i \bullet entry3.data1name == dn1 \\ \land entry3.data2name == dn2) \rhd \\ (inPIT = false; delPIT = false; addcs = false) \end{pmatrix};$$

Before introducing process $Datareceive_i$, we define two functions. $DelPIT$ function denotes that the router deletes entries from the $PITTable_i$ according to the input data names. $AddCS$ function represents that the router adds a data packet into its CS. When $Datareceive_i$ receives data packets, $Datareceive_i$ checks $PITTable_i$. If there is an interest packet waiting for these data packets, the entry is deleted and these data packets are emitted through $Datasend_i(index)$. Otherwise, the received data packets are abandoned.

$Datasend_i(index) =_{df}$
$Intail\{interfacelist = PITTbale[index].interfcelist;$
$x = 0; y = \#interfacelist; \} \rightarrow$

$(0 \leq x < y)*$

$$\begin{pmatrix} ComRS_{interfacelist[x]}!msg_{data}C.B.dn1.dn2. \\ E(prk1\_Key, data1).E(prk2\_Key, data2). \\ siginfo1.siginfo2 \rightarrow x = x + 1 \end{pmatrix} ; SKIP$$

$Datasend_i(index)$ gets the entry of $PTITable_i$ with index *index*. Then it records the incoming interface list *interfacelist* of the entry. Variables *x* and *y* represent entries' index variables and lengths of *interfacelist* respectively. $Datasend_i(index)$ queries *interfacelist* and sends a data packet, containing all data, from the incoming interfaces recorded in *interfacelist*.

$$Router_i =_{df}$$
$$Router_{io}[[$$
$$ComRS_i!\{|ComRS_i|\} \leftarrow ComRS_i!\{|ComRS_i|\},$$
$$ComRS_i!\{|ComRS_i|\} \leftarrow FakeC_i!\{|ComRS_i|\},$$
$$ComRS_i?\{|ComRS_i|\} \leftarrow ComRS_i?\{|ComRS_i|\},$$
$$ComRS_i?\{|ComRS_i|\} \leftarrow FakeC_i?\{|ComRS_i|\},$$
$$ComRR_i!\{|ComRR_i|\} \leftarrow ComRR_i!\{|ComRR_i|\},$$
$$ComRR_i!\{|ComRR_i|\} \leftarrow FakeC_i!\{|ComRR_i|\},$$
$$ComRR_i?\{|ComRR_i|\} \leftarrow ComRR_i?\{|ComRR_i|\},$$
$$ComRR_i?\{|ComRR_i|\} \leftarrow FakeC_i?\{|ComRR_i|\}]]$$

$\{|c|\}$ denotes the set of all communications over channel *c*. Whenever $Reader_{io}$ does an action on channel *ComRS*, $Reader_{io}$ will execute actions on channel *ComRS* or channel *FakeC*. $Reader_{io}$ carries out either actions on channel *ComRR* or *FakeC* when $Reader_{io}$ performs actions on channel *ComRR*. Besides, $Reader_i$ performs the same actions as $Reader_{i0}$.

D. Intruder Modeling

In order to take intruders into consideration, this subsection builds *Intruder* process. It intercepts or fakes messages in the communication via channel *ComCC, ComRS, ComRR, ComRP* and *ComPP*.

At first, we define the set of facts which intruders might learn.
$$Fact =_{df} Entity \cup MSG_{out} \cup Name$$
$$\cup \{E(key, c)|key \in PRKey, c \in Content\}$$

Intruder can derive new facts from the set of facts which intruders have learned. Symbol $F \mapsto f$ is used to indicate that the fact f can be deduced from the set *F* of facts.

$$\{K, E(K^{-1}, c)\} \rightarrow c$$
$$\{K^{-1}, c\} \rightarrow E(K^{-1}, c)$$
$$F \mapsto f \wedge F \subseteq F' \implies F' \mapsto f$$

The first two rules denote decryption and encryption respectively. The final rule represents that if a intruder can derive the fact *f* from the known fact *F*, the intruder can also deduce fact *f* from a larger set F'.

We define *Info* function to describe that intruders can obtain facts from messages, shown as follows

$$Info(msg_{int}.a.b.n) =_{df} \{a, b, n\}$$
$$Info(msg_{data}.a.b.n.E(K^{-1}, c).sig) =_{df} \{a, b, E(K^{-1}, c), n, sig\}$$
$$Info(msg_{data}.a.b.c) =_{df} \{a, b, c\}$$

where $a, b \in Entity,\ n \in Name,\ K^{-1} \in PRKey,$ $c \in Content, sig \in SigInfo$

We give a channel *Deduce* for intruders. Intruders can deduce new facts via channel *Deduce*, shown as follows :

Channel $Deduce : Fact.P(Fact)$

A intruder overheads all messages transmitted between entities. New facts can be deduced from the intruder's known facts. If a intruder gets all sub messages, it can fake some messages and send those messages to other entities. We formalize $Intruder_0$ as below:

$$Intruder_0(F) =_{df}$$
$$_{m \in MSG_{out}}Fake?m \rightarrow Intruder_0(F \cup Info(m))$$
$$\Box\Box_{m \in MSG_{out} \cap Info(m) \subseteq F}Fake!m \rightarrow Intruder_0(F)$$
$$\Box\Box_{f \in Fact, f \notin F, F \mapsto f}Deduce.f.F \rightarrow Intruder_0(F \cup \{f\})$$

This subsection uses set *Fake* to represent all channels of $INTRUDER\_PATH$. In the first part, a intruder gets messages via a channel of *Fake*. Then the intruder adds those messages to its knowledge. In the second part, a intruder fakes some messages according to its knowledge and sends faking messages to other entities. In the third part, a intruder can deduce some new facts from its knowledge via channel *Deduce*. Then the intruder adds the speculative results to its knowledge. We define *IK* to denote the initial knowledge of the intruder:

$$Intruder =_{df} Intruder_0(IK)$$
$$IK =_{df} \{A, B, C, D, E, Ipk\_Key, Irk\_Key\}$$

IV. VERIFICATION AND IMPROVEMENT

In this section, we use model checker PAT to verify four properties, including deadlock freedom, data availability, PIT deletion faking and CS caching pollution. According to verification results, we improve the system by referencing Blockchain.

A. Properties Verification

We use $System()$ to denote the original model. This subsection uses Linear Temporal Logic (LTL) formulas to describe four security properties. By using LTL formulas in PAT code, we give some assertions to help our verification.

Property 1: Deadlock Freedom

$$\#assert\ System()\ deadlockfree;$$

$System()$ should not run into a deadlock state. In PAT, there is a primitive to describe this situation.

Property 2: Data Availability

$$\#define\ Data\_Arival\_Success\ dataarive == true;$$
$$\#assert\ System()\ reaches\ Data\_Arive\_Success;$$

In NDN-based IoV, each entity should get the desired data. In order to guarantee this situation, we give this assertion.

Property 3: PIT Deletion Faking

$$\#define\ PITdelete\_Faking$$
$$getdatachange == true \wedge delPIT == true;$$
$$\#assert\ System()\ |= []!\ PIT\_Deletion\_Faking;$$

If a intruder intercepts an interest packet, it returns a faking data packet. When a router receives this faking packet, the router deletes corresponding PIT entries. Even though the router gets the legitimate data packet, it throws the data packet away. Using *always* operator [], we define this assertion to guarantee that routers are not vulnerable to such attacks.

Property 4: CS Caching Pollution

$$\#define\ CSCaching\_Pollution$$
$$getdatachange == true \wedge addccs == true;$$
$$\#assert\ System()\ |= []!\ CS\_Caching\_Pollution;$$

Fig. 6. Verification Result of Formalized Model

If a router receives a faking data packet, it may store the packet. When the router obtains an interest packet which contains the name of the faking packet, it returns the faking packet directly. This assertion is given to ensure that a router cannot store a faking data packet.

As shown in Fig.6, Property 1 (deadlock freedom) is valid. In other words, model cannot run into a deadlock state. Property 2 (data availability) is not satisfied for the system. This means that entities cannot get desired data with intruders intervention. Property 3 (PIT deletion faking) and Property 4 (CS caching pollution) are invalid. When a router obtains a faking data packet, it may delete its PIT entities and add the packet into its CS. Then the router can no longer obtain or provide a legal data packet which has the same name as the faking data packet.

B. Improvement

In order to ensure NDN-based IoV model to satisfy *Property 2*, *Property 3* and *Property 4*, we improve this model by using a method which is similar to Blockchain.

$$MSG_{data11} = \{msg_{data}.a.b.block, msg_{data}.a.b.trans \mid$$
$$a, b \in Entity, block \in Block, trans \in Transcation\}$$

A new kind of message MSG_{data11} is defined to describe the transmission of blocks and transactions. After a producer creates a data packet, it constructs and transmits a transaction which includes some information about this data packet to an *RSUP*. Then the *RUSP* sends the transaction to a router. If the verification result of the transaction is positive, the router stores and forwards it. Meanwhile, the router produces a block which contains all transactions stored in it so far. Then the router stores the block and forwards it to other entities. After verifying this block, an entity adds it into its blockchain. Then the entity deletes duplicate transactions and forwards this block to others. Gradually all entities' blockchains become synchronous. When an entity obtains a data packets, its verifies the packet with its blockchain firstly as follows:

$Checkhash_i(BlockTable_i, dnhash, sighash, siginhash) =$

$_{df} Intail\{check = fasle; x = 0; y = \#BlockTable_i;$

$packethash = Hash(dnamehash, sighash, siginhash)\}$

$\rightarrow (0 \leq x < y)*$

$$\left(\begin{array}{l} translist = BlockTable_i[x].transcationlist; \\ \left(\begin{array}{l} (check = true) \lhd (\exists transaction \in translist \bullet \\ transaction.datanamehash == dnamehash \land \\ transaction.packethash == packethash) \rhd \\ (checkpacket = false) \end{array} \right) \end{array} \right);$$

When the entity gets the data packet, it computes some hash values of this data packet. It queries its blockchain $BlockTable_i$ and gains some information about this data packet. It verifies the received data packet by comparing hash values obtained by calculation and results of querying. As we can see from Fig.7, the verification results of all properties are both valid for this model.

Fig. 7. Verification Result of Improved Model

V. CONCLUSION AND FUTURE WORK

NDN-based IoV is built by applying NDN into Internet of Vehicles. In this paper, we have formalized NDN-based IoV using CSP. Feeding the formalized model into PAT, we have verified four properties (deadlock freedom, data availability, PIT deletion faking and CS caching pollution). The last three properties are invalid. It means that the security of data has been compromised, once intruders appeared. In order to solve these problems, we improved the model with a method based on Blockchain. Then we verified the improved model. According to the verification results, the improved model can prevent intruders from invading the system.

In our future work, we will take the performance of NDN-based IoV model into consideration. To ensure the correctness and preciseness of the research results, formal methods will be used again in future work. Some related algorithms will be explored to improve the efficiency of this model.

REFERENCES

[1] Juan Contreras-Castillo, Sherali Zeadally, Juan Antonio Guerrero Ibáñez: Internet of Vehicles: Architecture, Protocols, and Security. IEEE Internet of Things Journal 5(5): 3701-3709 (2018)

[2] C. Yi, A. Afanasyev, L. Wang, B. Zhang, and L. Zhang, Adaptive forwarding in named data networking, Computer Communication Review, vol. 42, no. 3, pp. 62C67, 2012

[3] Bengt Ahlgren, Christian Dannewitz, Claudio Imbrenda, Dirk Kutscher,Börje Ohlman: A survey of information-centric networking. IEEE Communications Magazine 50(7): 26-36 (2012)

[4] Muhammad Tahir Abbas, Muhammad Afaq, Wang-Cheol Song: Road-Aware Estimation Model for Path Duration in Internet of Vehicles (IoV). Wireless Personal Communications 109(2): 715-738 (2019)

[5] Wei-Che Chien, Hung-Yen Weng, Chin-Feng Lai, Zhang Fan, Han-Chieh Chao, Ying Hu: A SFC-based access point switching mechanism for Software-Defined Wireless Network in IoV. Future Gener. Comput. Syst. 98: 577-585 (2019)

[6] Zhou Su, Yilong Hui, Qing Yang: The Next Generation Vehicular Networks: A Content-Centric Framework. IEEE Wireless Commun. 24(1): 60-66 (2017).

[7] Muktadir Chowdhury, Junaid Ahmed Khan, Lan Wang: Smart Forwarding in NDN VANET. ICN 2019: 153-154

[8] Eirini Kalogeiton, Domenico Iapello, Torsten Braun: A Geographical Aware Routing Protocol Using Directional Antennas for NDN-VANETs. LCN 2019: 133-136

[9] C. A. R. Hoare: Communicating Sequential Processes. Prentice-Hall 1985, ISBN 0-13-153271-5

[10] PAT, PAT: Process analysis toolkit. [Online]. Available: http://pat.comp.nus.edu.sg/

Controller Synthesis for ROS-based Multi-Robot Collaboration

Xudong Zhao[1], Rui Li[1], Wanwei Liu[1], Hao Shi[1], Shaoxian Shu[2], Wei Dong[1]

[1]*College of Computer Science, National University of Defense Technology, Changsha, China*
[2]*Hunan Institute of Traffic Engineering, Changsha, China*
{zhaoxudong13, lirui18, wwliu, shihao14}@nudt.edu.cn, shushaoxian@163.com, wdong@nudt.edu.cn

Abstract—**Given a multi-robot system and the high-level tasks for the robots. How to ensure the correct behavior of robots to complete their tasks is critical. In this paper, we design a framework that can automatically generate correct-by-construction controllers for multi-robot system. In this framework, we propose a multi-robot specification based on the Temporal Logic Synthesis Format (TLSF) to guarantee the robots' behavior. And two execution algorithms were proposed to abstract the synthesized automata into high-level controllers. These controllers were integrated into Robot Operating System (ROS) to control actions and movements of robots in Gazebo, which is an open-source 3D robotics simulator. Based on this framework, we developed a toolkit, which allows users to achieve system-level controller synthesis and simulation capable to be used for research such as mission planning, motion planning, automatic obstacle avoidance and so on.**

Index Terms—**multi-robot system, robot operating system, temporal logic synthesis format, controller synthesis**

I. INTRODUCTION

In the past few years, the study of autonomous robots has become increasingly appealing. Robots have been employed in many application domains. Especially in the fight against the novel coronavirus pneumonia that broke out in early 2020, robots are widely used in various applications include food delivery, medicine delivery, temperature measurement, disinfection, etc. Robots effectively replace humans for operation, reducing the possibility of cross-infection. People increasingly realize the importance and convenience of robots in human life.

Multi-robot systems (MRS) is defined as a group of robots coordinated to perform some complex tasks that cannot be completed by a single robot [1]. Therefore, in some particular scenarios, we usually need MRS to complete tasks. However, in practice, most application scenarios are uncertain and dynamic, robots may behave unexpectedly in these scenarios. How to ensure the correctness of robot behavior in these dynamic environments is critical.

Based on these considerations, the researchers used formal methods such as model-checking [2] and synthesis to ensure the correctness of the robot's behavior. One important approach is to use Linear Temporal Logic (LTL) [3] as the

specification to generate controllers of robots [4] [5]. Extensive research has been carried out on how to translate the high-level specifications into robot controllers. Finucane et al. developed a toolkit called LTLMoP [6]. It can translate the user-written LTL specification into a robot controller, and the controller can be executed to simulate the behavior of the robot in a two-dimension area. Their work successfully bridges the gap between high-level specifications and low-level robot controllers.

However, this tool only provides controller synthesis and simulation for a single robot. To apply these studies in multi-robot scenarios, we extend their work to MRS. In this paper, we introduce a framework that automatically translates the multi-robot specification based on the Temporal Logic Synthesis Format (TLSF) [7] into high-level controllers for the multi-robot system. These controllers can be integrated into ROS [8], which is an opensource software architecture that contains a variety of libraries and packages suitable for robots. Then these controllers were executed by appropriate packages and tools based on ROS. To better demonstrate the experimental results, the behavior of robots is simulated in Gazebo [9].

In order to prove the practicality of this framework, we developed a toolkit for designing, executing, and simulating multi-robot controllers generated automatically from the multi-robot specification. Our contributions include the following three aspects:

- First, the framework we proposed provides a complete development process for multi-robot collaborative tasks, including automata synthesis, generation and execution of controllers.
- Second, we propose two algorithms that can more efficiently abstract the synthesized automata into high-level controllers.
- Third, we achieve system-level simulation in Gazebo with ROS, which makes it easier to experiment with physical robots.

The rest part of this paper is structured as follows: Section II summarizes the theoretical basis of this paper; In section III, we elaborate on the various components of this framework; Then we present an example of collaborative task in section IV, which briefly introduce how to use our toolkit; We conclude this paper in section V.

II. PRELIMINARIES

A. Linear Temporal Logic

Linear Temporal Logic (LTL) is a modal temporal logic which has been widely used to model the change of a reactive system over time.

LTL Syntax. Let AP be a set of atomic propostions with temporal logic X (next) and U (until), where $p \in AP$ is a Boolean variable. LTL formulas are defined according to the follwing grammar:

$$\varphi ::= p \mid \neg\varphi \mid \varphi \vee \varphi \mid X\varphi \mid \varphi U\varphi$$

LTL Semantics. Semantics of an LTL formula φ are defined on an infinite sequences $\pi = \pi_1 \pi_2 \cdots$ of truth assignment to the atomic propostions $p \in AP$, where $\pi(i)$ denote the i-th element of π and $\pi(i) \in 2^{AP}$. The satisfaction relationship $\models$ between π, i and a LTL formula φ is defined as follows:

$$
\begin{aligned}
\pi, i &\models p & iff \quad & p \in \pi(i) \\
\pi, i &\models \neg\varphi & iff \quad & \pi, i \nvDash \varphi \\
\pi, i &\models \varphi_1 \vee \varphi_2 & iff \quad & \pi, i \models \varphi_1 \ or \ \pi, i \models \varphi_2 \\
\pi, i &\models X\varphi & iff \quad & \pi, i+1 \models \varphi \\
\pi, i &\models \varphi_1 U\varphi_2 & iff \quad & \exists k \geqslant i \ with \ \pi, k \models \varphi_2 \ and \\
& & & \forall i \leqslant j < k \ with \ \pi, j \models \varphi_1)
\end{aligned}
$$

The formula $X\varphi$ means that φ is true in the next position of the sequence. $\varphi_1 U\varphi_2$ indicates that φ_2 will be true somewhere in the future, and φ_1 must be maintained as true until φ_2 is true.

The sequence π satisfies formula φ if $\pi, 0 \models \varphi$. The temporal operators of LTL G(always), F(eventually):

- $F\varphi \equiv \text{true}U\varphi$;
- $G\varphi \equiv \neg F\neg\varphi$;

Where $G\varphi$ with *always* and $F\varphi$ with *eventually* express the properties that φ will always hold true in every position of the sequence and φ will be true at some position of the sequence in the future respectively. Further, $GF\varphi$ indicates that φ is true infinitely often.

B. General Reactivity(GR(1))

LTL formulas are particularly suited to model the evolution of a reactive system where atomic propositions can be divided into two parts: system input (environment) and output (system). However, the realizability of LTL is 2-EXPTIME-complete, which increases computational overhead.

To reduce the computational complexity into an acceptable range, we consider a special class of temporal logic formulas [10]. The GR(1) fragment of LTL specification consists of environment assumptions and system guarantees. A GR(1) synthesis problem is defined as a game between a system player and an environment player. We expect the system wins the game. In other words, we can always synthesize controllers that generate behavior strategies satisfying given specifications. A GR(1) game structure is organized as follows:

- $\mathcal{X}$ is the set of input variables controlled by environment, $\mathcal{X}'$ is the value of $\mathcal{X}$ in the next state;
- $\mathcal{Y}$ is the set of output variables controlled by system, $\mathcal{Y}'$ is the value of $\mathcal{Y}$ in the next state;
- θ^e is an assertion over $\mathcal{X}$ characterizing the initial states of the environment;
- θ^s characterizes an assertion over $\mathcal{X} \cup \mathcal{Y}$ characterizing the initial states of the system;
- ρ^e characterizes transition relation of the environment over $\mathcal{X} \cup \mathcal{Y} \cup \mathcal{X}'$;
- ρ^s characterizes transition relation of the system over $\mathcal{X} \cup \mathcal{Y} \cup \mathcal{X}' \cup \mathcal{Y}'$;
- $\mathcal{J}^e_{i \in 1..m}$ is a set of justice requirements of the environment;
- $\mathcal{J}^s_{j \in 1..n}$ is a set of justice requirements of the system;

The acceptance condition is finally defined as:

$$(\theta^e \wedge G\rho^e \wedge GF\mathcal{J}^e) \rightarrow (\theta^s \wedge G\rho^s \wedge GF\mathcal{J}^s)$$

where $G\rho^e$ and $G\rho^s$ are safety conditions over the environment and the system while $GF\mathcal{J}^e$ and $GF\mathcal{J}^s$ are liveness properties over the environment and the system.

C. Robot Operating System (ROS)

ROS [8] is a framework for robotics research and development. The core of ROS is communication mechanism. ROS is a peer-to-peer network of nodes that communicate with each other using custom ROS messages that are based on TCP/IP. Each node can be used to control the behavior of the robot, process the information obtained by the sensors, etc. In an MRS, we regard each robot as a node, and all nodes are connected to a ROS master. Through the master, each node can locate and communicate with other nodes by three different methods:

(1) Topic: For real-time and periodic messages, the topic is the best choice. The node that subscribes to messages from a topic is called the topic's subscriber, while the node that publishes messages to a topic is called the topic's publisher.

(2) Service: Service communication is two-way. It can send messages and return feedback. The service consists of two parts: the requester (Client) and the responder/service provider (Server). The client sends a request, waits for the server to process it and returns a reply. The entire service communication is completed through a "request-response" mechanism.

(3) Actionlib: Actionlib is used to execute a long-term communication process. The actionlib communication process can be viewed at any time, and the request can be terminated. Actionlib works in client-server mode and is a two-way communication mode.

The ROS ecosystem includes some tools to analyze and simulate robot behavior. Gazebo [9] is a three-dimensional physics simulation platform with a powerful physics engine, high-quality graphics rendering, convenient programming, and graphical interfaces. Gazebo can add the physical properties of the robot and the surrounding environment to the model, such as mass, coefficient of friction, coefficient of elasticity,
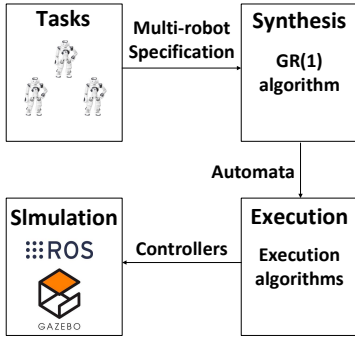
Fig. 1: Overview of the framework

etc. Therefore, we can simulate physical phenomena in the real world and show them as much as possible in this simulation environment.

III. CONTROLLER SYNTHESIS AND SIMULATION FOR MRS

Given a multi-robot system and tasks to be completed by the robots, the objective of our framework is to automatically generate controllers for each robot and simulate the behavior of robots in Gazebo. The framework consists of four parts, the relationship between them is illustrated in Figure 1 and the functions of them are introduced as follows.

A. Multi-robot specification

TLSF [7] is a high-level format for the specification of synthesis problems. Compared with LTL, it is more readable, therefore, users can easily write and read expressive specifications. Another advantage of TLSF is that it's easy to support by synthesis tools. After writing the specification, the $Synthesis\ Format\ Conversion\ Tool$ (SyFCo) can compile TLSF specifications into LTL specifications.

In a multi-robot system, a robot regards other robots as part of the environment. A robot's environment propositions can be sensed by its sensors and obtained by communicating with other robots. To express the robot's perception ability, execution ability and communication relationship with other robots, a tuple $\mathcal{R} = \langle S, A, C \rangle$ for each robot in MRS was defined:

- S is the environment variables gained by robot;
- A is the action variables performed by the robot's actuators;
- C is the communication node that are used to communicate with each other through network.

Based on the advantages of TLSF and the tuple we difined, we propose a multi-robot specification. Take two robots as an example, where:

$R_1 = \langle \{R2.act2\}, \{act1\}, node_1 \rangle$

$R_2 = \langle \{R1.act1\}, \{act2, act3\}, node_2 \rangle$

The specification for these two robots are shown in Figure 2. Keywords env (environment), sys (system) are set of input variables and output variables respectively and keywords asm (assumption), gar (guarantee) characterize initial conditions,

transition relations and justice requirements for environment and system respectively.

```
main R1 {
  env { R2.act2;}
  sys { act1;}
  asm { GF(R2.act2);}
  gar { G(R2.act2->X(act1));}}

main R2 {
  env { R1.act1;}
  sys { act2,act3;}
  asm { GF(R1.act1);}
  gar { G(R1.act1->X(act3));}}
```

Fig. 2: An example of multi-robot specification

The users could write the corresponding specification for each robot in MRS according to the multi-robot specification format we proposed. Based on the capabilities of the robot and the collaborative task to be completed, the continuous behavior of the robot should be abstracted into a finite set of propositions by a discrete formalism (TLSF). These propositions consist of the sensor information the robot perceives, the actions to be performed. The specification also includes the topological information of the robot's task area. Same as a single robot, in an MRS, each robot has a corresponding specification and its automaton that synthesized from these specifications.

B. Specification to Automaton

Take two robots as an example, Figure 3 illustrates the process of synthesizing, as mentioned in section II, we use GR(1) as the synthesis algorithm and automatically synthesize automaton from the multi-robot specification by a tool called JTLV [11].
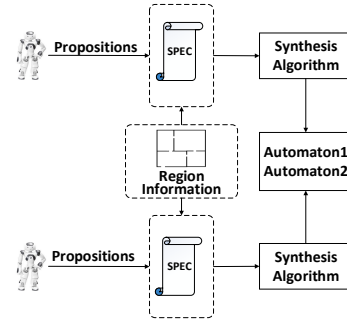
Fig. 3: The synthesis of automaton

To more intuitively explain the process of synthesizing specification into an automaton, the following briefly introduces the synthesis algorithm in [10].

As mentioned before, the synthesis algorithm was used to solve the game between the robot and the environment. Consider a game structure G: $\langle \mathcal{X}, \mathcal{Y}, \theta^e, \theta^s, \rho^e, \rho^s, \mathcal{J}^e, \mathcal{J}^s \rangle$, the initial state of robot and environment is $s_{\mathcal{X} \cup \mathcal{Y}}$ where $s_{\mathcal{X} \cup \mathcal{Y}} \models \theta_e \wedge \theta_s$. Then from the initial state, both the robot and the environment make decisions that determine their

next states, the environment choose an input $s_{\mathcal{X}'}$ such that $(s, s_{\mathcal{X}'}) \models \rho^e$ and the system choose an output $s_{\mathcal{Y}'}$ such that $(s, s_{\mathcal{X}'}, s_{\mathcal{Y}'}) \models \rho^s$. The winning condition for the game is given as a GR(1) formula $\phi = (\texttt{GF}\mathcal{J}^e \rightarrow \texttt{GF}\mathcal{J}^s)$, the implication between justice goals J^e of the environment and J^s of the robot. In other words, no matter what the environment does, the robot can always find a way to proceed and satisfy the GR(1) formula ϕ, we say that the robot is winning and an automaton can be synthesized from the specification. Otherwise, we say that the environment is winning and the specification is unrealizable. Once the task specification of the robot is realizable, the synthesis algorithm is to find a winning strategy that the robot should follow to complete the desired task.

The strategy synthesised by the algorithm can be viewd as an automaton $\mathcal{A} = (\mathcal{X}, \mathcal{Y}, \mathcal{Q}, Q_0, \gamma, \delta)$:

- $\mathcal{X}$ is the set of input (environment) propositions,
- $\mathcal{Y}$ is the set of output (robot) propositions,
- $\mathcal{Q}$ is the set of states,
- $\mathcal{Q}_0 \subset \mathcal{Q}$ is the set of initial states,
- $\gamma : \mathcal{Q} \rightarrow 2^{\mathcal{X} \cup \mathcal{Y}}$ is the state labeling function where $\gamma(q)$ is the set of robot propositions and input propositions that are true in state q, i.e., states hold the environment inputs.
- $\delta : \mathcal{Q} \rightarrow 2^{\mathcal{Q}}$ is the transition relation. If current state is q and at next point environment inputs is $s_{\mathcal{X}}$, then q' is the successor state of q if and only if $s_{\mathcal{X}} \models \gamma(q')$.

A run of a strategy is sequence $s = s_{\mathcal{X}}^0, s_{\mathcal{Y}}^0, s_{\mathcal{X}}^1, s_{\mathcal{Y}}^1, \cdots$, s.t. $\forall i : \left((q_i, s_{\mathcal{X}}^i, q_{i+1}) \models \delta \right) \wedge \left(s_{\mathcal{Y}}^i = \gamma(q_i) \right) \wedge \left(s_{\mathcal{Y}}^{i+1} = \gamma(q_{i+1}) \right)$. Based on this sequence, the discrete path of the robot can be acquired which guides the robot to choose a region to go or activate/deactivate the different robot actions.

C. Automaton to Controller
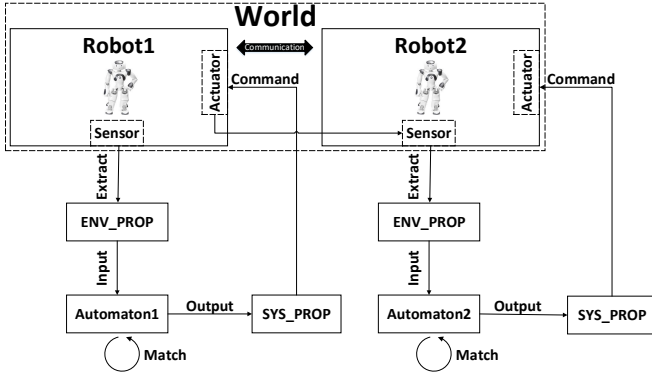
Fig. 4: Turn automatons into controllers

We introduce that the continuous behavior of the robot is abstracted into a discrete specification and then synthesized into an automaton. This part introduces how to turn these discrete automata into continuous controllers.

As shown in figure 4, an automaton is actually a finite state machine (FSM). Different states include the robot's perception of dynamic environments and the actions that the robot should perform. We propose an algorithm to deal with environment propositions, which are gained by the sensors of robots and the communication between robots. Furthermore, based on the Boolean value of these propositions, this algorithm can match the corresponding state in the automaton.

Algorithm 1: Discrete automaton to continuous controllers

Input: Automaton $\mathcal{A}$
$CurrState \leftarrow q_0 \in Q_0$
$SuccStateSet \leftarrow \delta(q_0)$
$Actions \leftarrow \{a_1, a_2, \cdots\} \in \gamma(q_0)$
$Execute(Actions)$
while *True* **do**
    $InputVal \leftarrow SenseOrInformed();$
    $FoundState \leftarrow False;$
    foreach $q_i \in SuccStateSet$ **do**
        if $InputVal \models \gamma(q_i)$ **then**
            $NextState \leftarrow q_i;$
            $Execution(CurrState, NextState);$
            $CurrState \leftarrow NextState;$
            $SuccStateSet \leftarrow \delta(CurrState);$
            $FoundState \leftarrow True;$
            BREAK;
        end
    end
    if $Foundstate\ is\ False$ **then**
        ERROR('Invalid Input')
    end
end

As shown in Algorithm 1, in the beginning, the robot is in its initial state, which is defined as the current state. According to this state, the robot executes current actions and get the successor states set. In each step, the robot obtains its environmental information and determines input values through its sensors or by communicating with other robots. Based on these inputs and current state, we can get the $NextState$ in the successor states set. The robot then executes the actions and complete the state transition, the execution algorithm is illustrated in Algorithm 2. When the next state is not found in the set of successor states, which means that the environment violated its assumptions, the execution will report an error and stop running.

When it comes to the cross-regional problem, we propose some simple controllers. Generally, we define those control robot's actuators as execute controllers, which can complete execution in a short time, while those control robot's motion are navigation controllers. Compared with the former, navigation controllers take a longer time to complete execution. The process of executing these controllers is shown in Algorithm 2. Take a transition between states as an example, when the robot is in different regions between the current state and the next state, the navigation controller should be executed first, and then the execute controller. That is, the robot will only execute the action when it reaches the designated region.

Robot movement is achieved through the navigation package in ROS. When both states are in the same region, the action is executed directly.

Algorithm 2: Execution

Input: Current state q_i and Next State q_{i+1}
$CurrState \leftarrow q_i$
$CurrRegion \leftarrow r_i \in \gamma(q_i)$
$NextState \leftarrow q_{i+1}$
$NextRegion \leftarrow r_i \in \gamma(q_{i+1})$
$Actions \leftarrow \{a_1, a_2, \cdots\} \in \gamma(q_{i+1})$
if $NextRegion \neq CurrRegion$ **then**
  $Navigation\_Controller(NextRegion);$
  $Execute\_Controller(Actions);$
else
  $Execute\_Controller(Actions);$
end

D. Simulation

To simulate multi-robot missions in Gazebo, the experimental environment must be configured in advance. The various components include the world model, map information, navigation packages, and communication method for configuration are elaborated as follows. To minimize the user's workload as much as possible, the corresponding navigation packages files and the program for communication between robots will be generated automatically according to the number of robots.

1) World: As shown in figure 4, the environment that the robot simulates in Gazebo is called world. As the name indicates, the world file is to simulate a real physical world. It can simulate physical parameters, such as gravity, friction coefficient, elastic coefficient, etc. For the tasks to be completed by the MRS, the user could build a corresponding world model using Gazebo's building editor, which allows the user to import a floor plan and add walls, stairs, and doors relatively easily.

2) Map: After establishing the world model, we control the robot to move in the world and use laser radar or depth camera to generate laser data and convert it into the map. This is the troublesome part of the simulation process. Then we manually extract the topological information of the generated map.

3) Navigation: Navigation and localization are important parts of the robot's tasks. The robot performs navigation and localization according to the established map. There are two packages in ROS that can be used directly.

- *move_base*: It can plan the global path according to a given target position, and also plan a local path based on nearby environments to avoid obstacles.
- *amcl*: It is a probabilistic localization system for a robot moving in two-dimensional area. It implements the adaptive Monte Carlo localization approach [12], which can get the position of the robot by using a particle filter against a known map.

IV. IMPLEMENTATION AND EXAMPLE

The toolkit we developed is modular and includes three modules. For different collaborative tasks, users can configure the number and propositions of robots to write multi-robot specification. Then the realizable specification can be synthesized into automata, finally, turn these automata into high-level controllers and simulate in Gazebo. For the sake of clarity, we introduce a robot collaboration experiment based on the home service scenario. As shown in figure 5, we design a workspace for robots. Then according to this sketch, we create a world model in Gazebo.
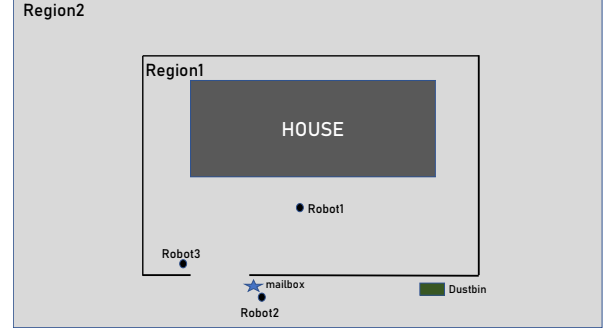
Fig. 5: Example diagram

In this example, $region_1$ is inside the yard while $region_2$ is the outside of the yard. Three home service robots work in these two areas, Among them, $Robot_1$ is mainly responsible for patrolling the $region_1$, and is also responsible for handling some chores in family life. For example, when there is mail in the mailbox, it will fetch it, and when waste is detected on the ground, it will pick it up and throw it into the dustbin in $region_2$. $Robot_2$ is responsible for delivering the mail. After delivering the mail, $Robot_1$ will be informed. $Robot_3$ is responsible for security tasks. When it detects that a thief is trying to break into the house, it will send a warning and inform $Robot_1$ to come and defend. At the same time, $robot_2$ is never allowed to enter $region_1$.

As defined in section III, we formalize this multi-robot system based on the capabilities of robots and the collaborative tasks, where:

$R_1 : \langle \{waste, R2.mail, R3.thief\}, \{pick, fetch, catch\}, node_1 \rangle$
$R_2 : \langle \{mailbox\}, \{deliver, informR1\_mail\}, node_2 \rangle$
$R_3 : \langle \{thief\}, \{catch, informR1\_thief\}, node_3 \rangle$

The multi-robot specification for robots is given in figure 6, where $R2.mail$ means R1 gets mail proposition by communicating with R2, so is $R3.thief$. After synthesizing these automata from the specification, we turn them into continuous control commands and simulate in Gazebo. There are six screenshots shown in figure 7.

V. CONCLUSION AND FUTURE WORK

In this paper, we propose a framework that can automatically generate correct-by-construction controllers for multi-robot system. To simulate the behavior of robots, these generated controllers were executed in Gazebo with ROS,

which provides state-of-the-art algorithms for several known problems in robotics. We develop a toolkit to implement this framework. Because ROS targets at real robots and not only 3D simulation, it is easier to embed these controllers for simulation into real robots. Compared to experiments using real robots, it is low-cost, convenient and extensible for research.

In future work, we plan to provide a user-friendly interface and experiment with real robots. Besides, we will combine methods in the field of robot control to extend the functionality of this framework as much as possible.

```
main R1{
  env {
    Waste;
    R2.Mail;
    R3.Thief;}
  sys {
    Pick;
    Catch;
    Patrol;
    Fetch;}
  asm {
    !Waste;
    !R2.Mail;
    !R3.Thief;}
  gar {
    !Pick;
    !CatchThief;
    !Patrol;
    !Fetch;
    GF Patrol;
    G(X Waste -> X Pick);
    G(X R2.Mail -> X Fetch);
    G(X R3.Thief -> X Catch);}}

main R2{
  env {
    Mailbox;}
  sys {
    Deliver;
    InformR1_mail;}
  asm {
    !Mailbox;}
  gar {
    !Deliver;
    !InformR1_mail;
    G(X Mailbox -> X Deliver);
    G(Deliver -> X InformR1_mail);
    G !Region1;}}

main R3{
  env {
    Thief;}
  sys {
    InformR1_Thief;
    CatchThief;}
  asm {
    !Thief;
    GF Thief;}
  gar {
    !InformR1_Thief;
    !CatchThief;
    G(X Thief -> X InformR1_Thief);
    G(X Thief -> X CatchThief);}}
```

Fig. 6: Specification for home service robots

(a) Mail delivery

(b) Fetch mail

(c) Waste detected

(d) Pick up and throw away

(e) Thief detected

(f) Catch the thief

Fig. 7: Experimental run of the service robots scenario

REFERENCES

[1] A. Gautam and S. Mohan, "A review of research in multi-robot systems," in *2012 IEEE 7th International Conference on Industrial and Information Systems (ICIIS)*, Aug 2012, pp. 1–5.

[2] E. M. Clarke and B. Schlingloff, "Model checking," in *Handbook of Automated Reasoning (in 2 volumes)*, 2001, pp. 1635–1790.

[3] A. Pnueli, "The temporal logic of programs," in *18th Annual Symposium on Foundations of Computer Science, Providence, Rhode Island, USA, 31 October - 1 November 1977*, pp. 46–57.

[4] I. Filippidis, S. Dathathri, S. C. Livingston, N. Ozay, and R. M. Murray, "Control design for hybrid systems with tulip: The temporal logic planning toolbox," in *2016 IEEE Conference on Control Applications (CCA)*, Sep. 2016, pp. 1030–1041.

[5] H. Kress-Gazit, "Robot challenges: Toward development of verication and synthesis techniques [errata]," *IEEE Robot. Automat. Mag.*, vol. 18, no. 4, pp. 108–109, 2011.

[6] C. Finucane, G. Jing, and H. Kress-Gazit, "Ltlmop: Experimenting with language, temporal logic and robot control," in *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems, October 18-22, 2010, Taipei, Taiwan*, pp. 1988–1993.

[7] S. Jacobs, F. Klein, and S. Schirmer, "A high-level LTL synthesis format: TLSF v1.1," in *Proceedings Fifth Workshop on Synthesis, SYNT@CAV 2016, Toronto, Canada, July 17-18, 2016*, pp. 112–132.

[8] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Ng, "Ros: an open-source robot operating system," vol. 3, 01 2009.

[9] N. P. Koenig and A. Howard, "Design and use paradigms for gazebo, an open-source multi-robot simulator," in *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems, Sendai, Japan, September 28 - October 2, 2004*, pp. 2149–2154.

[10] N. Piterman, A. Pnueli, and Y. Sa'ar, "Synthesis of reactive(1) designs," in *Verification, Model Checking, and Abstract Interpretation, 7th International Conference, VMCAI 2006, Charleston, SC, USA, January 8-10, 2006, Proceedings*, pp. 364–380.

[11] A. Pnueli, Y. Sa'ar, and L. D. Zuck, "Jtlv: A framework for developing verification algorithms," in *Computer Aided Verification, 22nd International Conference, CAV 2010, Edinburgh, UK, July 15-19, 2010. Proceedings*, pp. 171–174.

[12] F. Dellaert, D. Fox, W. Burgard, and S. Thrun, "Monte carlo localization for mobile robots," in *1999 IEEE International Conference on Robotics and Automation, Marriott Hotel, Renaissance Center, Detroit, Michigan, USA, May 10-15, 1999, Proceedings*, pp. 1322–1328.

Understanding Travel Patterns of Commuting Private Cars
using Big Data of Electronic Registration Identification of Vehicles

Junchao Lv[1], Linjiang Zheng[1], Yuhui Ye[1], Chenglin Ye[1]

[1]College of Computer Science, Chongqing University, Chongqing, China

(cqu_ljc@163.com, zlj_cqu@cqu.edu.cn, yeyuhui@cqu.edu.cn, cqu_yechenglin@163.com)

Abstract—**Commuting private cars are the most important component of urban road traffic in the morning and the evening rush hours. Identifying commuters from all private cars and mining their travel pattern can provide promising solutions to solve the urban road traffic congestion. However, current research rarely involves private cars due to the difficulty in obtaining the travel data of private cars. Electronic Registration Identification (ERI) based on Radio Frequency Identification (RFID), is an emerging vehicle identification technology to collect travel data of individual vehicles. In this paper, we focus on investigating travel patterns of commuting private cars using ERI big data. A method of identifying commuters from all private cars is presented based on the spatiotemporal similarity measurement. Then regular travel behaviors are mined by spatial and temporal clustering and the individual vehicle with regular travel behaviors is identified as commuting private car. In the experiments, real-world ERI big data from Chongqing is employed into the proposed method. The group of commuting private cars is discovered. We analyzed travel time distribution and hot spot areas of commuting private cars. The results show that the proposed method can accurately identify commuting private cars, and the spatiotemporal characteristics of commuting private cars can reflect commuting time and the residence-workplace relationship.**

Keywords—ERI data, Private cars, Commuter, Spatiotemporal similarity, Regular travel behavior

I. INTRODUCTION

Urbanization's rapid progress has modernized many people's lives but also engendered big issues, such as traffic congestion, energy consumption and pollution [1]. Therefore, many scholars have studied travel patterns using various types of data, such as GPS data [3]–[8], smart card data [9]–[14]. However, the research on these types of data has limitations due to the characteristics of the data. Most of the researches on GPS data are on taxis, although taxis only account for a small part of urban traffic. So their travel characteristics do not represent the travel characteristics of other vehicles, such as private cars, making the draw conclusions sometimes inconsistent with the actual traffic situation [15]. Smart card data is for public transport, but not all vehicles.

Among many studies on travel patterns, few of them focus on the travel patterns of private cars, although private cars are the most important part of urban road traffic (Taking China as an example, as of the end of 2019, the car ownership of China has reached 260 million, including 207 million private cars, accounting for 79.62% of the total number of cars). This is mainly because the travel data of private cars is difficult to collect. This problem can be solved by Electronic Registration Identification of the motor vehicle (ERI), which is an emerging vehicle identification and tracking technology based on Radio Frequency Identification (RFID) and can identify all

vehicles with high accuracy. In December 2017,China issued the national standard on ERI, which was formally implemented in July 2018. This indicates the broad application prospects of ERI technology in China.

As a product of Urbanization, commuting is the process of going there and back between home and workplace. It not only reflects the long-term travel behavior of people [9], but also relates to the home and workplace of individual commuters. Commuting also engendered a series of problems, such as traffic congestion and long commuting time. In addition, commuting mainly occurs in the morning and the evening rush hours, and the proportion of travel volume of private cars during rush hours in the total travel volume of private cars is very high (Taking Chongqing as an example, 53.03%). Although it cannot be simply considered that the private cars that travel during rush hours are all commuting private cars, this can reflect that commuting private cars are an important part of private cars. Therefore, understanding travel patterns of commuting private cars from traffic data can not only take targeted measures for the long-term travel behavior to alleviate traffic problems during rush hours, but also provide useful information for urban planning by reflecting the spatial distribution of commuters' home and workplaces.

This paper utilizes ERI data to study the travel patterns of commuting private cars. To address this issue, this paper first proposes a novel method to identify commuting private cars. Then, according to the Origin-Destination information of commuting private cars to analyze travel patterns. The main contributions of our work are summarized as follows:

- We designed a method to identify commuting private cars by combining trip information. This method determines whether private cars have regular travel behavior by measuring the spatiotemporal similarity of travel behaviors, and identifies private cars with regular travel behaviors as commuting private cars.

- We analyzed the travel pattern of commuting private cars, including four aspects of departure time, arrival time, origin and destination. We verified the nature of land use at origin and destination at different times according to the land use property map of Chongqing. The results show that the travel patterns of commuting private cars identified by our proposed method can reflect the commuting time and the residence-workplace relationship.

The remainder of this paper is organized as follows. In Section II, we discuss the related work. Section III introduces some preparations. In Section IV, we present the details of our approach. Analysis of spatiotemporal characteristics is given in Section V, Finally, we conclude our work in Section VI.

Field name	Sample of field value	Remarks
RID	R228	Identification of the RFID reader
EID	838326	Identification of the vehicle
passtime	2016-03-06 15:15:58	Timestamp
carType	K33	Vehicle Type., "K33" means a small car
plateType	02	Plate type, "02" means a compact car
useProperty	A	Usage of the vehicle, "A" means the vehicle is non-operating.

II.   DATA AND PREPARATION

In this section, we present the ERI data that we used for our study, clarify basic definitions and do regional partition.

A.   ERI principle

ERI is based on RFID. RFID is a non-contact information transmission technology using radio frequency signal through spatial coupling (alternating magnetic field or electron magnetic field), and automatically identifies the object through the information transmitted. It has many advantages, such as long recognition distance, high recognition accuracy, more information stored, fast reading speed, etc. The characteristics make it very suitable for urban traffic information collection. The RFID-based traffic information collection is called Electronic Registration Identification of the vehicles (ERI) in relevant international standards. In ERI system, the collection of ERI data mainly relies on two devices: RFID tags attached to the vehicle windshield and RFID readers deployed on key urban road sections. The tag stores the vehicle registration information, such as unique electronic identification (i.e. EID), vehicle type, usage, etc. When a vehicle passes through an RFID reader, the information in the vehicle's RFID tag is read and a travel record is generated.

B.   ERI data

The content of ERI data is shown in Table I. We can determine the range of vehicle, such as private cars, using "carType", "plateType", and "useProperty".

C.   ERI data in Chongqing

Chongqing is the earliest and only city in China that requires all motor vehicles to be equipped with RFID tags. It has realized that all legal motor vehicles have RFID tags, and RFID readers are deployed in key sections of the city. Combined with the real ERI data generated by 1198519 cars from February 29, 2016 to March 6, 2016 in Chongqing (Note: The electronic registration identification of private vehicles has been masked or obfuscated to avoid leakage of privacy), we have conducted classification statistics of motor vehicles. The result shows that the number of private cars is 1082991, accounting for 90.36% of the total number of motor vehicles.

For the convenience of the following discussion, we give the related definitions based on ERI data.

Definition 1 (ERI Record). *A record is a three-tuple consisting of (EID, RID, Passtime), called R.*

Definition 2 (ERI Segment). *A segment is composed of two adjacent Rs with the same EID. It is a six-tuple consisting of (EID, Ot, Dt, ORID, DRID, Interval), called Seg.*

Definition 3 (ERI Trajectory). *The trajectory of the car is consisting of all its Rs or all Segs, called Tra. That means a car has at most one Tra, Tra can be obtained from (1).*

$$Tra = \{R_i \mid R_i \square EID = eid\} \qquad (1)$$

Definition 4 (ERI Trip). *The Tra of the car can be divided into multiple Trips after trajectory segmentation [2]. The composition of each Trip is the same as that of Tra, as shown in (2), called Trip, Trip ⊆ Tra.*

$$Trip = \{R_i \mid R_i \square EID = eid\} \qquad (2)$$

According to the above definition, the relationship between *R*, *Seg*, *Tra* and *Trip* is shown in Fig. 1.

Definition 5 (Travel behavior). *The elements of a trip are different in importance to individuals. Some elements are essential and other are not. We define the essential part of each a trip as travel behavior, called Tb,*

$$Tb = \{R_i \mid R_i \square EID = eid\} \qquad (3)$$

The relationship between *Tb* and *Trip* is shown in Fig. 2. There are two different *Trip*s with same *Tb*. In terms of RID alone, the two *Trip*s are {A, B1, D, E1, F, G, H} and {A, B2, C, E2, F, G, H}, and *Tb* is composed of areas circled by red circles.

D.   Regional partition

As of April 2016, 688 RFID readers had been deployed in Chongqing, but many RFID readers are adjacent in space, which will generate *Trip*s similar to Fig. 2. This causes some deviation in extracting *Tb*s from *Trip*s. In order to reduce the deviation, we used the DBSCAN algorithm to partition regions. The reason we choose the DBSCAN algorithm is that the DBSCAN algorithm is able to automatically infer the
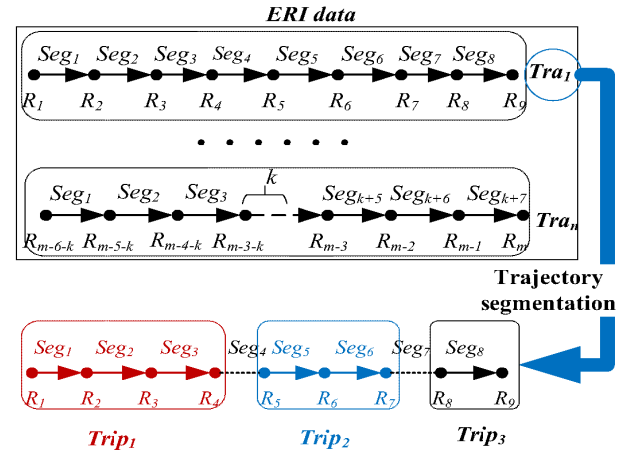
Figure 1.   The relationship between *R*, *Seg*, *Tra* and *Trip*.
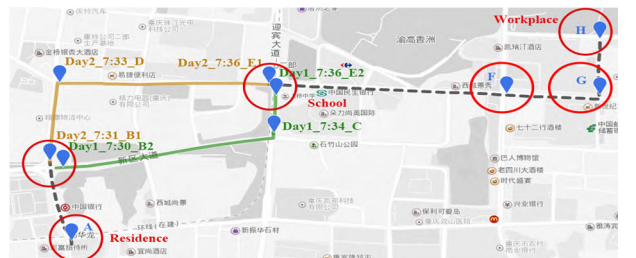
Figure 2.   The relationship between *Tb* and *Trip*.

number of clusters. We used the Silhouette Coefficient as the evaluation index of cluster results, it was determined that the cluster effect was best when *eps*=0.190km and *minPts*=1. Finally, 688 RFID readers were clustered into 320 regions. We renumbered the clustered region as the new RID and updated ERI data with new RIDs.

III. IDENTIFYING COMMUTING PRIVATE CARS BY CLUSTERING IN SPATIAL AND TEMPORAL DIMENSIONS

Our main research goal is investigating travel patterns of commuting private cars, so first, we need to determine which cars are commuting private cars. In this section, we first give a general review of identifying commuting private cars and then elaborate the details.

A. Overview

Commuting is the long-term travel process of going there and back between home and workplace. The phrase "long-term" embodies regularity. That means commuters have regular travel behavior in the process of going there and back between home and workplace. Regular travel behavior should have spatiotemporal similarity, and the spatiotemporal similarity can be quantified by the repeatability of *Tb*s, or more precisely, the spatiotemporal similarity of *Tb*s. So how to measure the spatiotemporal similarity of *Tb*s is very critical. In order to solve the problem, for two *Tb*s, here recorded as Tb_1 and Tb_2, if the difference between $Tb_1(RID)$ and $Tb_2(RID)$ is small and (4) is satisfied, we consider them to be similar. The small difference represents spatial similarity, and when (4) is satisfied, it represents temporal similarity.

$$\forall i, Tb_1 \square R_i \square RID = Tb_2 \square R_i \square RID,$$
$$Tb_2 \square R_i \square Passtime - \alpha \leq \qquad (4)$$
$$Tb_1 \square R_i \square Passtime \leq Tb_2 \square R_i \square Passtime + \alpha$$

Here we set α to 15min.

Therefore, considering most of commuting behaviors take place at rush hours, we proposed a method to identify private cars by clustering in spatial and temporal dimensions during rush hours. The method includes two processes: (1) Extract *Tb*s from *Trip*s; (2) Measure spatiotemporal similarity (i.e. regularity) of *Tb*s. The detailed procedures are as follows:

1) Extract *Tb*s according to individual *Trip*s.

2) Divide *Tb*s into different **TW**s.

3) Measure the spatial similarity of *Tb*s in each **TW**. If there is spatial similarity, execute step 4).

4) Measure the temporal similarity of *Tb*s in each **TW**. If there is temporal similarity, the individual is considered to have regular behavior within the **TW**.

5) Identify private cars with regular travel behaviors as commuting private cars.

The process is based on the fact that over a long period of working days, commuters using private cars will determine the travel route according to their own travel habits combined with new travel needs. Travel habits play a major role in this process. The travel habit here is regular travel behavior.

B. Detailed Algorithms

In this section, we demonstrate in detail 1), 2), 3), and 4) of the above process. And all steps are individual oriented.

TABLE II. NOTATIONS.

Notations	Description
Dnum(RID)	The number of days an individual has passes a RID
Dnum(EID)	The number of days an individual drives a private car
Trip(RID)	The set of RIDs contained in *Trip*s of an individual
Tb(RID)	The set of RIDs contained in *Tb*s of an individual
S(RID)	Whether RID is often passed through, 0 means not often while 1 means often.
S_{Seg}	The set of *Seg*
S_{Tb}	The set of *Tb*
DT(Trip)	The departure time of a *Trip*, *Passtime* of the first *R*
DT(Tb)	The departure time of a *Tb*, *Passtime* of the first *R*
AT(Tb)	The arrival time of a *Tb*, *Passtime* of the last *R*
Tb_{min}	*Tb* have the smallest value of *DT(Tb)* in S_{Tb}
TW	Time window
L_{TW}	The number of *Tb* in a *TW*, $1 \leq L_{TW} \leq Dnum(EID)$
B_{TW}	The beginning position of *TW*
E_{TW}	The end position of *TW*
eps	One of the parameters of DBSCAN algorithm
minPts	One of the parameters of DBSCAN algorithm
RoS	The result of spatial clustering

1) Extract Tbs.

Step 1: Extract spatial information of *Tb*s (i.e. **Tb(RID)**).

According to **Definition 5**, *Tb* is an essential part of *Trip*. So **Tb(RID)** must be set of RIDs individual passed through frequently. We defined **S(RID)** to indicate whether a RID is passed through frequently. **S(RID)** can be obtained from (5),

$$S(RID) = \begin{cases} 0, & \dfrac{Dnum(RID)}{Dnum(EID)} < 0.6 \\ 1, & \dfrac{Dnum(RID)}{Dnum(EID)} \geq 0.6 \end{cases} \qquad (5)$$

Where $RID \in$ **Trip(RID)**. Then we can use (6) to get **Tb(RID)**,

$$\textbf{Tb(RID)} = \{RID \mid \textbf{S(RID)} = 1, RID \in \textbf{Trip(RID)}\} \quad (6)$$

If the number of RIDs in **Tb(RID)** is less than 20% of the number of RIDs in **Trip(RID)**, we consider that the individual does not have regular travel behavior. The algorithm will be terminated.

Step 2: Extract temporal information of *Tb*s.

For each *Trip*, we kept *R* whose RID part belongs to **Tb(RID)**, called *nR*, and *R*s in *nR* was still an ascending sequence. We used these *R*s to reconstitute *Seg*s.

After processing all *Trip*s, we got S_{Seg}. We proposed the similar concept of two *Seg*s here. When two *Seg*s satisfied $Seg_i \square ORID = Seg_j \square ORID, Seg_i \square DRID = Seg_j \square DRID;$

$$Seg_i, Seg_j \in S_{Seg}, i \neq j$$

, we call these two *Seg*s similar. According to the similar concept of *Seg*, we divided S_{Seg} into multiple subsets composed of similar *Seg*, and calculated the average value of Interval for each subset. Next, for each *nR*, we changed the Passtime of the first *R* in *nR* to the **DT(Trip)** of its corresponding *Trip*, and then modified the Passtime of all *R*s based on the average of the Interval and the relationship between *R* and *Seg*. After that, we got *Tb*.

*2) Divide Tbs into different **TW**s.*

In order to measure the spatial similarity of *Tb*s, we needed to divide *Tb*s that occurred at the same time on different days into a *TW*. We judged whether they occurred at the same time according to the **DT(Tb)** of each *Tb*.

Step 1: Adjust time format.

We ignored the date part of time information in *Tb*s, kept only the hour and minute part, and sorted *Tb*s into ascending order according to **DT(Tb)**. Then we divided 24 hours into 48 (0-47) time slices at an interval of $2*\alpha$ (30 min), and used 0-47 to adjust the time format of the sorting results.

Step 2: Determine S_{Tb} during rush hours.

According to the definition of commuting, we determined rush hours (06: 00–10: 00, 16: 00–20: 00). Then we selected *Tb* whose **DT(Tb)** was at the rush hours to form S_{Tb}.

Step 3: Divide *Tb*s in S_{Tb} into different *TW*s .

Initialized a *TW* as an empty set, added Tb_{min} sorted first in S_{Tb} to the *TW*, and deleted it in S_{Tb}. Then, for Tb_i in S_{Tb}, added Tb_i to the *TW* if *Tb* satisfied (7), then deleted it in S_{Tb}.

$$DT(Tb_i) - DT(Tb_{min}) < 2 \tag{7}$$

Equation (7) means we believe that *Tb* which differs from Tb_{min} by a time interval $(2*\alpha)$ belongs to the same time with Tb_{min}.

Step 4: Determine B_{TW} and E_{TW} of *TW*.

If there was no Tb_i in S_{Tb} that can satisfy (7) or if the number of *Tb* in *TW* equaled to **Dnum(EID)**, a *TW* was determined. We took the minimum value of **DT(Tb)** in *TW* as B_{TW} and the maximum value of **AT(Tb)** in *TW* as E_{TW}.

Step 5: Repeat **Step 3** and **Step 4** until S_{Tb} is empty.

Step 6: Convert *Tb*s to a binary sequence.

Initialized a binary sequence whose number of bits was determined by the number of RIDs in the **Tb(RID)**, i.e. each bit corresponded to one RID. All bit values were initialized to bit 0. The transformation rules are as follows: (1) For *TW* satisfying (8), L_{TW} groups of binary sequences are initialized according to its B_{TW}, E_{TW} and L_{TW}, each group has $(E_{TW} - B_{TW})$ binary sequences, and each sequence is numbered according to a number in 0-47 according to B_{TW} and E_{TW}; (2) A *Tb* corresponds to a group of binary sequences. Set the bit value of corresponding position of the binary sequence to bit 1 based on the RID and Passtime of *R* in the current *Tb*.

$$L_{TW} \geq Dnum(EID) * 0.6 \tag{8}$$

3) Spatial similarity measurement.

After 1) and 2), we got binary serialized *Tb*s, and each *Tb* belonged to different *TW*s. The difference between the set of

RIDs in different *Tb*s is reflected in the binary sequence, i.e. whether the values of corresponding bits are equal.

Next, we measure the spatial similarity of *Tb*s in *TW*.

Step 1: Get the spatial information of binary serialized *Tb*s.

We performed bitwise OR operation on $(E_{TW} - B_{TW})$ binary sequences corresponding to each *Tb* in a *TW*, thereby obtaining the spatial information after binary serialization. After OR operation, for each *TW*, we got L_{TW} binary sequences. We denoted each binary sequence as **Bs** and the set of **Bs** as S_{Bs}. Then we measured the spatial similarity between L_{TW} **Bs**s based on Hamming distance, as shown in (9).

$$Hamming\_dist(Bs_i, Bs_j) \leq threshold \tag{9}$$

$Bs_i \in S_{Bs}$, $Bs_j \in S_{Bs}$, i≠j. The threshold is given in next step.

Step 2: Using the DBSCAN algorithm to measure the spatial similarity of *Tb* in each *TW*.

We chose Hamming distance as the distance measure of DBSCAN algorithm. And set **eps** as **Tb(RID)***0.3, **minPts** as L_{TW}*0.6, then conducted clustering on S_{Bs}. Since the data cycle is five days, there are only two cases of clustering results: (1) All **Bs**s are identified as outliers. In this case, we believe that the individual does not have regular travel behavior and does not perform subsequent steps; (2) Part or all **Bs**s are clustered into a cluster. In this case, execute the next step.

Step 3: Adjust the clustering results as input to the temporal similarity measurement.

Firstly, the number of **Bs**s in the cluster was recorded, denoted as **L**. Then the corresponding $(E_{TW} - B_{TW})$ **Bs** were determined according to the clustering result. The $(E_{TW} - B_{TW})$ **Bs**s were denoted as **B**, and the set of **B** was denoted as **RoS**. Took **RoS** as input to the temporal similarity measurement.

4) Temporal similarity measurement.

Here we adjusted according to (4). We first determined the time, and then determined whether the performance of *Tb*s in RIDs was consistent in a specific time slice, i.e. whether the corresponding bit values were equal.

Temporal similarity measurement was similar to 3). We set **eps** as **Tb(RID)***0.3 and **minPts** as **L***0.6. Unlike (9), we did not directly use Hamming distance as the distance measure, but (10) as the distance measure.

$$dist(B_i, B_j) = \max_{B_i, B_j \in RoS, i \neq j} \{Hamming\_dist(B_{il}, B_{jl})\} \tag{10}$$

Here. $B_{il} \in B_i$, $B_{jl} \in B_j$, $B_{TW} \leq l < E_{TW}$.

There are also only two clustering results: (1) no clustering. In this case, we believe that the individual does not have regular travel behavior; (2) generate a cluster. In this case, we believe that the *Tb*s corresponding to **B**s of the cluster is regular travel behavior of the individual and identify the private car used by individual as commuting private cars.

IV. ANALYSIS OF THE CHARACTERISTICS OF SPATIOTEMPORAL

After experiment, our method identified 215716 commuting private cars from 1082991 private cars. We analyze the travel patterns from the perspectives of individuals and groups.

A. Individuals

For individuals, if they have regular travel behaviors during rush hours, they are commuters. According to this principle, we finally identify 215716 commuting private cars. Next, we will show travel patterns of one commuter identified by our method. The 5-day trajectory of the commuter during the workday is shown in Fig. 3, the red marks in Fig. 3 are the RFID readers it passes by, and the red lines are the travel trajectories. It can be seen that the travel trajectory is almost identical. After extracting the travel behavior, we get the regular travel behavior (No1.08:04a--No2.08:08am--No3.08:13am--No4.08:18am--No5.08:27am--No6.08:32am). Combined with the actual scene, the commuter departures from vicinity of the Sigongli interchange at about 08:00 every morning, and arrivals vicinity of the Banan interchange after half an hour's drive, while the neighborhood of the Sigongli interchange is mostly residential area and the Banan interchange is close to the Banan District government where are many commercial offices nearby, meeting the condition of from home to work in the definition of commuting.

B. Groups

According to the result, 1082991 private cars were divided into two groups: commuting private cars (CPC) and non-commuting private cars (NCPC). We first conduct their statistical analysis on travel duration and travel distance, then analyze the temporal characteristics of departure time and arrival time. Finally, we analyze the spatial characteristics of CPC at origin and destination, and verify their nature of land use.

1) Statistical analysis

According to the *Trips of CPC and NCPC*, we conducted statistical analysis from three aspects: average travel time, average travel distance and average travel speed, the results are shown in Table III. It can be seen that CPC has a faster average travel speed due to the strong purpose of travel behavior.

2) Temporal characteristics analysis

a) Commuting private cars (CPC).

As can be seen from Fig. 4, CPC occur in two concentrated periods in a day, i.e. morning rush and evening rush. 69.02% of CPC occurred in rush hours, while 53.03% of *Trips* of private cars occurred in rush hours. Compared with the evening rush, CPC occurring in the morning rush are more concentrated. In addition, the distribution of departure time and arrival time is consistent with the normal distribution during rush hours, and the distribution of them is similar.

b) Non-commuting private cars (NCPC).
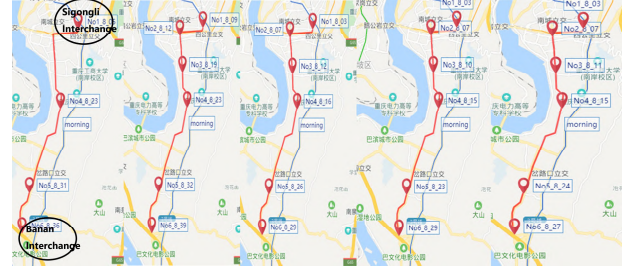
Figure 3.   The 5-day trajectory of one commuter during workday..

TABLE III.   THE STATISTICAL INFORMATION OF THE CPC AND NCPC.

Category	Number of Trips	Average travel time (min)	Average travel distance (km)	Average travel speed (km/h)
CPC	2293342 (34.74%)	28.7	12.67	26.49
NCPC	4307712 (65.26%)	38.7	15.90	24.65
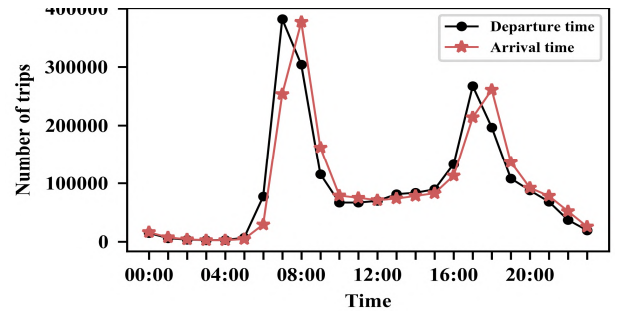
Figure 4.   Distributions of departure times and arrival times of CPC.
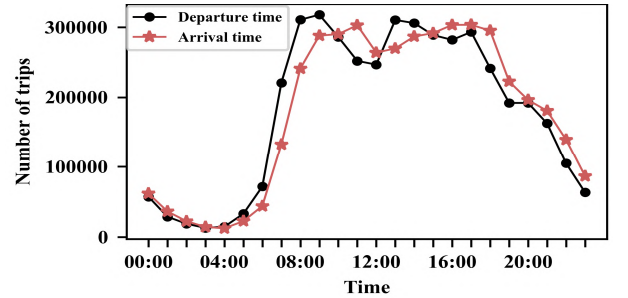
Figure 5.   Distributions of departure times and arrival times of NCPC

As shown in Fig. 5, NCPC have a large number of *Trip*s in the rest of the time except for 00: 00 to 06: 00. The departure time and arrival time of NCPC are not regular, and there is a big difference in distribution between them.

3) Temporal characteristics analysis
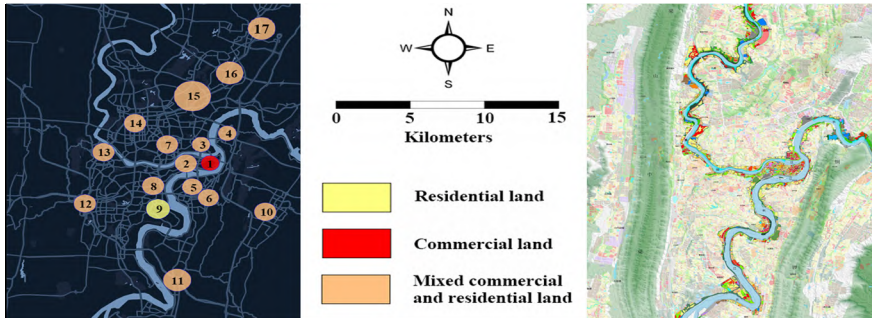
Figure 6.   The typical origins and destinations and the land use property map of Chongqing.

<div style="text-align:center">(a)Morning rush   (b)Evening rush</div>
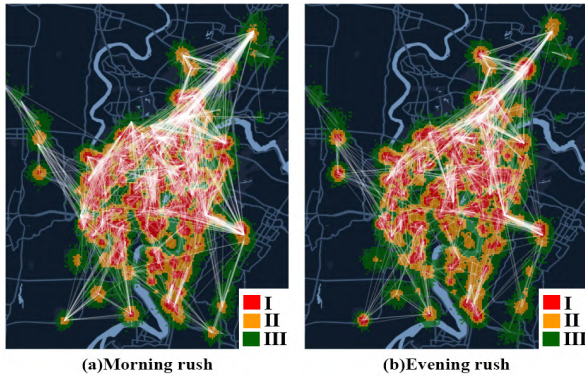
Figure 7. The hot spot areas distribution at the origins of CPC in rush hours and traffic transfer information in rush hours.

Fig. 7 shows the hot spot areas distribution at the origins of CPC and the traffic transfer information of CPC, during rush hours. The visualization method of the hot spot areas of origins combines the location information of the parking lot and RFID readers, and divides the hot spot areas into three grades, I, II, III according to the heat degree from high to low. The traffic transfer information is the white line part in figures, in which the intersection of white lines is the end of the traffic transfer and also the destination of *Trip*s. Combined with the definition of commuting, we believe that morning rush travels take the residence as the origin and the workplace as the destination, while evening rush travels take the opposite, so we focus on the analysis of level I hot spot areas and end points of flow transfer, and observe their spatial distribution characteristics.

As shown in Fig. 7, combined with the typical origins and destinations in Fig. 6, we find that region *2, 3, 4, 5, 6, 7, 8, 10, 11, 12, 13, 14, 15, 16* belong to the level I both in Fig. 7a and Fig. 7b, region *9* belongs to level I only in Fig. 7a while region *1* belongs to level I only in Fig. 7b. In Fig. 7a, end points of traffic transfer are mainly located at *1, 2, 3, 5, 7, 8, 10, 11, 13, 14, 15, 16, 17*, and in Fig. 7b, end points of traffic transfer are mainly located at *2, 3, 5, 6, 7, 8, 9, 11, 13, 14, 15, 16, 17*. The reason may be that region *2, 3, 4, 5, 6, 7, 8, 10, 11, 12, 13, 14, 15, 16, 17* belong to mixed commercial and residential land, while region *1* is mainly commercial land and region *9* is mainly residential land.

In order to verify the conjecture, we combine the land use property map of Chongqing in Fig. 6, and find that *2, 3, 4, 5, 7, 8, 10, 13* are located in the business district with a high proportion of mixed commercial and residential land, while *6, 11, 14, 15* are located near the business district, which is the transportation hub connecting residential land and commercial land such as Sigongli interchange. Region *1* is the area from Qixinggang to Chaotianmen whose land use is mainly for commercial land; Region *9* is near the main street of Yangjiaping whose land use is mainly for residential land; Region *17* is Yubei airport, with factories and houses nearby.

V. CONCLUSION

In summary, we identify commuting private cars based on ERI data. ERI data is a new type of intelligent traffic data. In view of the concentrated distribution of RFID readers, we use the DBSCAN algorithm to cluster RFID readers into different regions. On this basis, we propose a novel method to measure the spatiotemporal similarity of *Tb*s to mine regular travel behavior, then identify private cars with regular travel behavior as commuting private cars. According to the results

of experiment, we analyze the temporal characteristics of CPC and NCPC at departure time and arrival time. Finally, we verify the nature of land use of CPC at origin and destination during rush hours, and it proves that the travel patterns of commuting private cars identified by our proposed method can reflect the residence-workplace relationship. It should be acknowledged that Origin-Destination (OD) information is important in commuting behavior, so in the future, based on the existing work, we will give different weights to OD points and other points in the route to optimize the identification of commuting private cars.

REFERENCES

[1] S. Colak, A. Lima, and M. C. Gonzalez, "Understanding congested travel in urban areas," Nature Common, vol. 7, no. 10793, Mar 2016.

[2] L. Zheng, D. Xia, L. Chen, and D. Sun, "Understanding Citywide Resident Mobility Using Big Data of Electronic Registration Identification of Vehicles," IEEE Transactions on Intelligent Transportation Systems, pp. 1-15, 2019.

[3] B. Jiang, J. Yin, and S. Zhao, "Characterizing Human Mobility Patterns in a Large Street Network," Physical Review E, vol. 80, pp. 1711-1715, 2008.

[4] L. Liu, C. Andris, and C. Ratti, "Uncovering cabdrivers' behavior patterns from their digital traces," Computers, Environment and Urban Systems, vol. 34, no. 6, pp. 541-548, Nov 2010.

[5] G. Pan, G. Qi, Z. Wu, D. Zhang, and S. Li, "Land-Use Classification Using Taxi GPS Traces," IEEE Transactions on Intelligent Transportation Systems, vol. 14, no. 1, pp. 113-123, 2013.

[6] L. Zheng, D. xia, X. Zhao, L. Tan, H. Li and L. Chen, "Spatial–temporal travel pattern mining using massive taxi trajectory data," Physica A: Statistical Mechanics and its Applications, vol. 501, pp. 24-41, Jul 2018.

[7] C. Wan, Y. Zhu, J. Yu, and Y. Shen, "SMOPAT: Mining semantic mobility patterns from trajectories of private vehicles," Information Sciences, vol. 429, pp. 12-25, Mar 2018.

[8] Y. Huang, Z. Xiao, D. Wang, H. Jiang, and D. Wu, "Exploring Individual Travel Patterns Across Private Car Trajectory Data," IEEE Transactions on Intelligent Transportation Systems, pp. 1-15, 2019.

[9] X. Ma, C. Liu, H. Wen, Y. Wang, and Y.-J. Wu, "Understanding commuting patterns using transit smart card data," Journal of Transport Geography, vol. 58, pp. 135-145, Jan 2017.

[10] Y. Long and J.-C. Thill, "Combining smart card data and household travel survey to analyze jobs–housing relationships in Beijing," Computers, Environment and Urban Systems, vol. 53, pp. 19-35, Sep 2015.

[11] L. M. Kieu, A. Bhaskar, and E. Chung, "Passenger Segmentation Using Smart Card Data," IEEE Transactions on Intelligent Transportation Systems, vol. 16, no. 3, pp. 1537-1548, 2015.

[12] X. Ma, Y.-J. Wu, Y. Wang, F. Chen, and J. Liu, "Mining smart card data for transit riders' travel patterns," Transportation Research Part C-Emerging Technologies, vol. 36, pp. 1-12, Nov 2013.

[13] N. Lathia, C. Smith, J. Froehlich, and L. Capra, "Individuals among commuters: Building personalised transport information services from fare collection systems," Pervasive and Mobile Computing, vol. 9, no. 5, pp. 643-664, Oct 2013.

[14] G. Goulet-Langlois, H. N. Koutsopoulos, Z. Zhao, and J. Zhao, "Measuring Regularity of Individual Travel Patterns," IEEE Transactions on Intelligent Transportation Systems, vol. 19, no. 5, pp. 1583-1592, 2018.

[15] Y. Zheng, L. Capra, O. wolfson, and H. Yang, "Urban computing: Concepts, methodologies, and applications," ACM Transactions on Intelligent Systems & Technology, vol. 5, no. 38, pp. 1-55, Sep 2014

TIMESIGHT: Discovering Time-driven Insights Automatically and Fairly

Yohan Bae, Suyeong Lee, Yeonghun Nam*

Samsung Research, Samsung Electronics

yhan.bae@samsung.com, sy710.lee@samsung.com, yeonghun.nam@samsung.com

Abstract— **Exploratory data analysis (EDA) on time-series data is an indispensable and important process for not only data analysts but also non-expert users. It helps them make data-driven decisions by discovering important patterns of a certain phenomenon. However, it poses 2 challenges for data analysts and decision-makers. First, although a lot of business intelligence tools have been introduced that can help explore the data, they require repeated analytic procedures and most of the procedures rely on users' intuition, knowledge, and efforts. Second, even though there have been several attempts to quantify insights to automatically detect interesting patterns, they do not consider score fairness among detected patterns. Therefore, they are not suitable when data has the heterogeneity of insight types, attributes scales, and time intervals. We attack these challenges by introducing our new proposed system *Timesight*, which explores data through all possible time units and all attributes automatically. *Timesight* evaluates various types of time-driven insight, matching the fairness among each type of insight, each attribute, and each time interval. We verify our system using an internal application log dataset. Our experiment with data analysts working on the same dataset shows that *Timesight* alleviates tedious works and is effective in discovering insight.**

Keywords-component; Data exploration; Insight discovery; Data mining; Time-series data.

I. INTRODUCTION

Nowadays, efforts to make data-driven decisions are continuously increasing in various industries [11] to reduce costs or improve productivity. For instance, a manufacturer who wants to increase productivity can adjust the time on the assembly line, observing delays calculated from the data. After introducing a solution to the bottleneck, the decision-maker would measure the time of the process again to see if the change results in time-saving. Thus, exploratory data analysis (EDA) is an indispensable and important process to discover potential meaning or important patterns (called *insight*) from a data [12]. Based on the EDA process, data analysts are able to understand the data to predict or prevent certain phenomenon via building statistical methods or machine learning models. It is also important for non-experts users who are in an important position and should make a critical decision from the data without any data scientific knowledge such as statistics and probability theory.

TABLE I. CRIME DATASET

timestamp	district	# of victims	crime category
2019-09-29 06:39:00	A3	12	firearm
2018-12-31 18:42:00	B2	24	knife
2017-03-05 13:15:00	C1	5	vehicle

Among many kinds of datasets, time-series data that contains time information about certain events is more valuable than other simple multi-dimensional data. Because, it is important to discover striking patterns of specific phenomena or events over various time units (e.g., year, month, weekday, etc.) in many domains such as business, manufacturing, healthcare, economy, sociology, and even government.

Suppose that we have a crime in a city dataset with the schema (timestamp, district, number of victims, crime category) as shown in Table 1. Figure 1 presents some examples of insights. It can be very helpful for people who are looking for hidden insight in the data if we get the information that the number of firearm accident has seasonality for each quarter (as Figure 1(a)) or the number of victims is most pronounced in the second day of the week as shown in Figure 1(b).
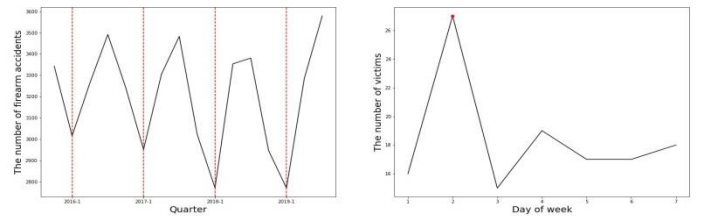
(a) # of firearm accidents for each quarter   (b) # of victims by day of week

Figure 1. Example of insights

EDA Challenge. Although lots of business intelligence tools like Tableau [9] and Qlik [10] have been introduced that can help explore the data, they require repeated analytic procedures and most of the procedures rely on users' intuition, knowledge, and efforts. Users have to repeat trial-and-error procedures: building their own hypothesis, selecting appropriate attributes and possible time units, entering the formula, plotting results with suitable visualization, and exploring remarkable patterns (e.g., trend, spike point). It has

*Corresponding author

the advantage of offering a high degree of freedom for advanced users. However, it is mentally and physically tedious and exhausting for both non-expert users and professionals in data analysis. If the data has more than thousands of attributes and millions of records, it becomes impossible to evaluate all the assumptions, regardless of users' expertise.

Fairness Challenge. Even though there have been several attempts [1] [2] to quantify insights to automatically detect interesting patterns, they do not consider score fairness among detected patterns. It is difficult to match the fairness because each type of insight needs different score function to calculate interestingness, most real-world datasets have different scales and units among attributes, and each time unit might have different intervals. Thus, it is challenging to provide fair scores that users can easily and reasonably accept.

We attack these challenges by introducing our new proposed system *Timesight*, which explores data through all possible time units and all attributes automatically. *Timesight* defines and evaluates various types of time-driven insight, matching the fairness among each type of insight, each attribute, and each time interval. Therefore, the contributions of this paper are:

- We demonstrate the way that automatically prepares the time-series data to well extract hidden insights.

- We propose the normalization technique to make them fairly comparable among diverse time intervals and attribute scales.

- We define 4 types of time-driven insight and unified formulation of each type to assess the magnitude of interestingness.

The rest of the paper is organized as follows: Section 2 presents an overview of our data modeling procedure. Section 3 provides 4 types of insight and score functions and Section 4 describes the pseudo-code of *Timesight* and optimization techniques. Section 5 demonstrates our experiment using a real-world dataset. Section 6 discusses related work, followed by the conclusion and future work in Section 7.

II. DATA MODELING

In this section, we present the data preparation procedures to calculate data insights. It is assumed that a multi-dimensional dataset D is given as a tabular format consisting of a series of rows, and each row is represented by a set of attributes (columns). We assume that D contains 3 types of attribute sets T, N and C: $T=\{t_1,t_2,...,t_\alpha\}$ is a timestamp attribute set, $N=\{n_1,n_2,...,n_\beta\}$ is a numerical attribute set, and $C=\{c_1,c_2,...,c_\gamma\}$ is a categorical attribute set where α, β, and γ are the number of timestamp, numerical, and categorical attributes, respectively. In this paper, the data modeling process is divided into two phases: *timestamp decomposition* and *data aggregation and normalization*.

Timestamp Decomposition. Suppose that each timestamp attribute has 'yyyy-MM-dd HH:mm:ss' format. We define a set $O=\{o_1,o_2,\cdots,o_\delta\}$ that includes extracted time units according to

the analytics objective (in this section, it is assumed that there is one timestamp attribute in D for the convenience of derivation). We illustrate an example in Figure 2 where the original timestamp attribute, and the extracted attributes are in Figure 2(a) and Figure 2(b), respectively. In this paper, we define 7 (i.e., $\delta=7$) extracted attributes for each timestamp attribute, i.e., O={'yyyy', 'MM', 'dd', 'HH', 'yyyy-qq (year-quarter)', 'yyyy-MM', 'yyyy-MM-dd'}. The elements of O depend on the analytics objective and can be declared dynamically (e.g., 'yyyy HH', 'HH:mm', etc.).

timestamp
2019-09-29 06:39:00
2018-12-31 18:42:00
2017-03-05 13:15:00

(a)

Year	month	day	hour	year-quarter	year-month	year-month-day
2019	09	29	06	2019-3	2019-09	2019-09-29
2018	12	31	18	2018-4	2018-12	2018-12-31
2017	03	05	13	2017-2	2017-03	2017-03-05

(b)

Figure 2. A timestamp decomposition example of (a) the original attribute, and (b) the extracted attributes from (a).

Data Aggregation and Normalization. The data aggregation and normalization techniques are applied to the dataset to make it fairly comparable among diverse time units and types of insight. In this paper, numerical as well as categorical attributes are used for insight scoring (which is different in that related works only consider numerical attributes) because the pattern of categorical attributes can contain important meaning after appropriate aggregations. For the available aggregation functions agg $\in$ {SUM, AVG, COUNT, $\cdots$}, we consider agg for the categorical and the numerical attributes separately, because the available aggregations for the categorical and numerical attributes are different. For example, COUNT and PERCENTAGE are for categorical, while SUM and AVG are for numerical attributes. For the arbitrary time unit element o_l in O ($1\leq l\leq \delta$), the aggregated datasets can be obtained based on two cases.

Case1: For numerical attributes, the function $GN(o_l,n_j)$ groups D by o_l with certain aggregation on the attribute n_j, which is presented as follows:

$$GN(o_l,n_j) \approx \text{SELECT } agg(n_j) \text{ FROM D GROUP BY } o_l$$

Case2: For categorical attributes, for the arbitrary ith categorical attribute, let E_i denote the set of distinct elements of the c_i, assuming that $|E_i|\geq 1$ and $e_{i,m}$ the arbitrary mth element of E_i ($1\leq m\leq|E_i|$). The function $GC(o_l,e_{i,m})$, which filters D with value $e_{i,m}$ and groups D by o_l with certain aggregation on the attribute c_j can be presented as follows:

$$GC(o_l,e_{i,m})\approx \text{SELECT } agg(c_j) \text{ FROM D WHERE } c_j= e_{i,m}$$
$$\text{GROUP BY } o_l$$

Thus, from the given GN(o_l,n_j) and GC(o_l,$e_{i,m}$), the result set X can be derived as:

$$X = \begin{cases} GN(o_l, n_j) & \text{(for the numeric attributes)} \\ GC(o_l, e_{i,m}) & \text{(for the categorical attributes)} \end{cases} \text{ on D}$$

We obtain an aggregated result set X={$x_1, x_2, \cdots, x_n$} from each categorical and numerical attribute considering multiple time unit elements. Next, we normalize all values in the X using min-max normalization [3]. It maps all values to the range [0, 1], and helps us focus on the relative ratio, improving the balance among other result sets that are measured by different units. It also enhances fairness with different insight types. There is another well-known normalization method called standardization (or Z-score normalization) [3], but it creates new data not bounded to a certain interval. Consequently, we get a normalized result set X_{norm} from X. An example of normalization is illustrated in Figure 3 where Figure 3(a) is the original result set X and Figure 3(b) is the normalized result set X_{norm}.

$$X_{norm} = \left\{ x_{norm,i} : \frac{x_i - x_{min}}{x_{max} - x_{max}}, i = 1, 2, 3, \ldots \right\}$$

5	21	-1	32	-11	13	0	-7	19	29

(a)

0.37	0.74	0.23	1.0	0.0	0.56	0.26	0.09	0.70	0.93

(b)

Figure 3. A normalization example of (a) the original result set, and (b) the normalized result set from (a) which is bounded in range [0, 1].

III. INSIGHT SCORE FUNCTIOIN

We define 4 types of time-driven insight: *spike point*, *change point*, *seasonality,* and *trend*. We want to score and rank interestingness of each insight based on the *p*-value to discover important insight from the entire result set. In statistics, the *p*-value is the probability of the observation from the null hypothesis and commonly used to determine whether an observation is statistically significant [4]. We use different kinds of null hypotheses for different types of insight to calculate the appropriate *p*-values. In this paper, we use the Gaussian distributions $N(\mu,\sigma^2)$ [5], where μ and σ^2 are constant parameters to model the distribution of observational values of each insight type. We use $\mu=0$ and $\sigma^2=1$ for all types of insights for convenience, but each can be replaced with the appropriate parameters in future studies.

Spike point. The Spike point means that a certain value in X_{norm} represents a notable difference from others. Data analysts who are exploring a data are attracted by the significant deviations from predictable patterns. The spike point can be discovered by measuring how the certain point is noticeable over other values. The distribution of various domains, such as physics, biology, economics, social science, and other numerous man-made phenomena often follow power-law distribution [6]. Therefore, we set the null hypothesis of spike point as:

H_0: X_{norm} follows power-law distribution.

Let X_{norm}={$x_{norm,1}, x_{norm,2}, \ldots, x_{norm,n}$} be the set of values in the result of aggregation as mentioned previous section. We sort X_{norm} in the descending order and get the maximum value $x_{norm,max}$. Then, from the $x_{norm,max}$, we evaluate the magnitude of interestingness against the hypothesis H_0 is true. Hence, we fit values in X_{norm} \{ $x_{norm,max}$ } to the power-law distribution (i.e., $a \cdot i^{-b}$). Next, from the $x_{norm,max}$'s prediction error $e_{max} = x_{norm.max} - \hat{X}_{norm,max}$, we calculate *p*-value $p_{spike} = P(e > e_{max} | e \sim N(\mu,\sigma^2))$. Consequently, the score of spike point is $1 - p_{spike}$

$$score_{spike} = 1 - p_{spike}$$

(a) Change point   (b) Trend
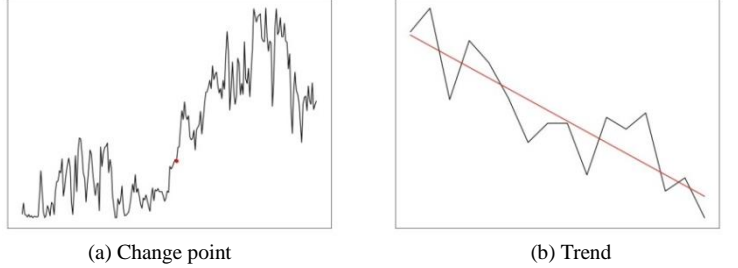
Figure 4. Examples of change point and trend.

Change point. A change point generally indicates an abrupt variation of values between the previous and subsequent intervals [17]. Figure 4(a) shows an example of change point. We use mean value on the window size K to obtain representative value. Thus, the null hypothesis of change point is:

H_0: Difference of mean between before and after $x_{norm,i} \approx 0$.

Let X_{norm}={$x_{norm,1}, x_{norm,2}, \ldots, x_{norm,n}$} be the time series of values and assume length of window size, K < n/2. By shifting the window along the values, we calculate the mean for values in left and right window of size *K* respectively. And their difference d_i is as follows:

$$\text{mean}_{left,i} = \frac{1}{K} \sum_{j=i}^{i+K-1} x_{norm,j},$$
$$\text{mean}_{right,i} = \frac{1}{K} \sum_{j=i+K}^{i+2K-1} x_{norm,j},$$

$$d_i = |mean_{left,i} - mean_{right,i}|, 1 \leq i \leq n - 2K + 1.$$

Let d_{max} denote the maximum difference value. Then, we calculate *p*-value $p_{change} = P(d > d_{max} | d \sim N(\mu,\sigma^2))$. Thus, the score of change point is

$$score_{change} = 1 - p_{change}$$

Seasonality. If the data show repetitive patterns or fluctuation over a specific period, we can say it has seasonality. We use the autocorrelation function (ACF) because it is commonly used to determine whether the data has a dependency on its past [7]. If the strongest correlation appears at a particular period *p* for a given o_l (e.g., 4 for 'yyyy-qq', 12 for 'yyyy-mm', etc.), we determine it has seasonality. Thus, we set the null hypothesis as follows:

H_0: $a \in$ ACF(X_{norm}) has maximum value a_{max} at *p*.

Then, the *p*-value is $p_{seasonality} = P(a > a_{max} \mid a \sim N(\mu, \sigma^2))$. As a result, the score of seasonality is $1 - p_{seasonality}$

$$score_{seasonality} = 1 - p_{seasonality}$$

Trend. It indicates that the data show continuously rising or falling movement over time, like in Figure 4(b). Those who want to discover explainable patterns in the data are well obsessed when it has a very different slope from 0. Thus, we set the null hypothesis as:

H_0: Slope of the values in X_{norm} over the entire time ≈ 0.

First, we fit X_{norm} to a line by linear regression as shown in Figure 4(b). We also normalize the x-axis values using min-max normalization, for fairness with another result set. As a result, we can concentrate on the change of values, not the length (time interval) of the data. Then we calculate its slope s^* and coefficient of determination, r^2. The r^2 represents how well the line fits the data [8]. Also, we compute the *p*-value as $p_{trend} = P(s > \mid s^* \mid \mid s \sim N(\mu, \sigma^2))$. Finally, we can obtain the trend score $r^2 * (1 - p_{trend})$ where r^2 is used to reflect the accuracy of the regression.

$$score_{trend} = r^2 * (1 - p_{trend})$$

IV. FRAMEWORK

In this section, we describe the full procedure of *Timesight* using pseudo-code first, and then discuss the pruning-based optimization techniques that can reduce search space and running time, improving the overall performance of *Timesight*.

Algorithm 1 InsightDiscovery(T, N, C)

1: max-heap $\mathbb{H} \leftarrow \{\}$

2: O ← extract all possible time units from T

3: **for** o_l *in* O **do**

4:    **for** $\underline{n_j}$ in N **do**

5:      $X_{norm} \leftarrow$ *normalize*(GN(o_l, n_j))

6:      CalculateInsights(X_{norm}, $\mathbb{H}$)

7:    **for** c_i in C **do**

8:      **for** $e_{i,m}$ in c_i **do**

9:        $X_{norm} \leftarrow$ *normalize*(GC($o_i, e_{i,m}$))

10:       CalculateInsights(X_{norm}, $\mathbb{H}$)

11: return $\mathbb{H}$

Function: *CalculateInsights*(X, $\mathbb{H}$)

12: **for** each insight type I **do**

13:   $score_I \leftarrow calculate_I(X)$

14:   insert ($score_I$, X) to $\mathbb{H}$

A. Psuedo Code

Algorithm 1 presents the full procedure of our insight discovery system. We assume that there is one timestamp attribute in D for the convenience of derivation.

First, we initiate the max heap $\mathbb{H}$ to store insights in descending order (Line 1) and extract all possible time units (Line 2). Then iterating over all time units (Line 3), we repeatedly generate X_{norm} to score insights for all numerical attributes N (Lines 4, 5). At the same time, we generate X_{norm} for all categorical attributes C (Lines 7-9). Using the generated result set X_{norm}, function '*CalculateInsights*' calculates scores of all insight types and updates $\mathbb{H}$ (Lines 12-14).

B. Pruning-Based Optimization Technique

Searching and computing all possible time units and all attributes take a lot of time and degrade performance. Therefore, we suggest the three pruning methods that can save time performance.

1) We pass calculating the score if multiple X_{norm} sets are identical for different time units. For instance, if the data is only for 2019, the result sets of '*yyyy-mm*' and '*mm*' have the same values. This can be applied equally on '*yyyy-qq*' with '*qq*', '*yyyy-MM-dd*' with '*dd*' and so on.

2) If the length of X_{norm} is too short, the data cannot represent a particular pattern properly. As a result, we set the minimum length ζ (e.g., $\zeta = 4$, because '*qq*' can have a maximum length of 4.) and if the length of X_{norm} is shorter than ζ, then we do not calculate all insight score.

3) If $|E_i|$ is too large, the search space grows exponentially and the performance is degraded. Also, if $|E_i| = 1$, it may not be meaningful to apply aggregation on c_i. Consequently, we set the minimum and maximum length θ (e.g., 70) and calculate score only if $1 < |E_i| < \theta$.

V. EXPERIMENT

TABLE II.   SUMMARY OF EXPERIMENT DATA

Range of date	2009.06.26 00:29:12 ~ 2019.07.12 23:50:02
# of rows	100000
# of numerical attributes	10
# of categorical attributes	49

In this section, we apply the real-world time-series data to evaluate the effectiveness of *Timesight*. This data is an internal application log dataset that is de-identified for research purposes. The summary of the dataset is shown in Table 2. The data has a timestamp attribute that represents the access date for each user from 2009 to 2019. And 10 numerical attributes and 49 categorical attributes contain a variety of information.
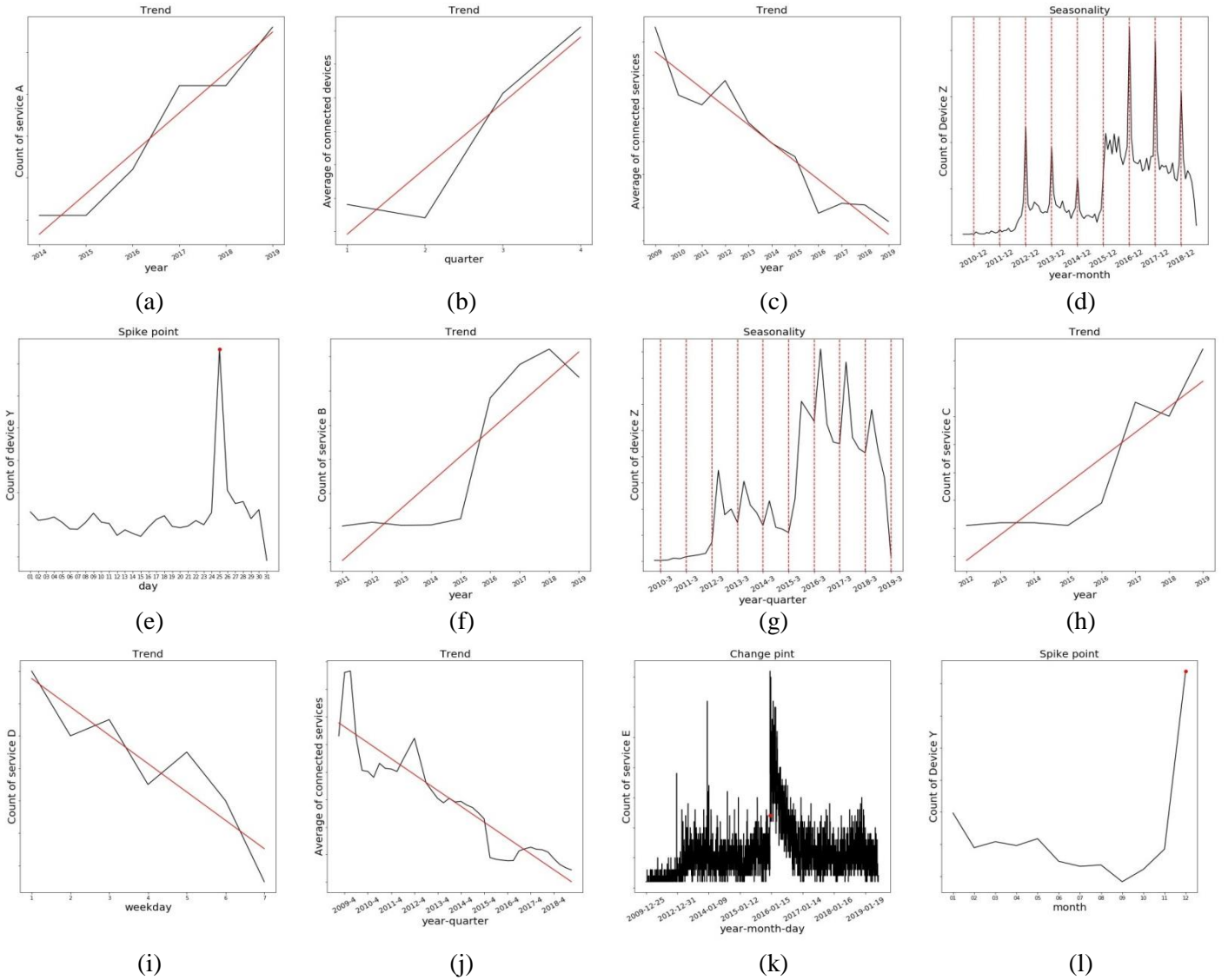
Figure 5. Result of our experiments. The x-axis is each time unit and the y-axis is each aggregated result set.
The black lines indicate actual result set, and the red lines indicate fitted trends or periods of seasonality. The red points indicate spike points or change points.

Figure 5 presents 12 result insights from *Timesight*. *Timesight* analyzes time stamp attributes by decomposing it in many ways such as *year*, *quarter*, *year-quarter*, *month*, *year-month*, *day*, *weekday*, *year-month-day,* and so on. As a result, users can examine the data from various time perspectives. Each of A, B, C, D, and E is a service name in the application, and each Y, Z is a device name that the user used to access the application.

Figure 5(d) and Figure 5(g) show that the number of people who accessed the application using device Z had a clear seasonality for *year-month* and *year-quarter*. Also, we can see that most people accessed the application using device Y on the 25th of every month and every December, as shown in Figure 5(e) and Figure 5(l), respectively. Furthermore, Figure 5(k) represents that the number of people who signed in through service E had a significant change point before and after 2016-01-15. The rest of Figure 5 shows us the clear tends of each aggregated attribute for *year*, *quarter*, *year-quarter*, *weekday*.

It is very convenient for analysts and decision-makers in the aspect of getting these insights from the data without the effort to explore it in person. Furthermore, they can make immediate decisions like: 1) From Figure 5(d) and Figure 5(g) they can try to find out the reason that the usage of device Z has seasonality and is most prominent in December and look for a solution that can balance monthly usage to improve overall usage. 2) From Figure 5(k), they can investigate why service E shows dramatic changes in usage around 2016-01-15 and apply that factor on other services. 3) From Figure 5(i), the decision-maker can promote service D on weekends so that the usage of service D does not decrease on weekends. Analysts who are working on the same dataset observed the effectiveness of results in alleviating their tedious works and discovering meaningful insights.

VI. RELATED WORK

In this section, we discuss prior works from multiple areas related to our research. We review the related works and describe how they differ from *Timesight*.

Business Intelligence Tools. Business intelligence tools, such as Tableau [9], Qilk [10], and Sisense [13] have recently improved capabilities in EDA and gained in popularity. These tools allow data analysts and decision-makers who lack programming skills to easily select attributes and build visualizations based on their abilities. However, they require repeated analytic procedures and most of the procedures rely on users' intuition, knowledge, and efforts. Furthermore, if the data has more than thousands of attributes, it becomes impossible to evaluate all the assumptions regardless of users' expertise. On the contrary, *Timesight* discovers insight automatically, exploring all attributes and all time units based on unified formulations of various insight types. Therefore, *Timesight* alleviates the tedious trial-and-error process of users.

Automated Exploratory Analysis. There have been several kinds of research that attempt to quantify insights to automatically detect interesting patterns. The SeeDB [14], which is the visualization recommendation system, identified charts that are largely deviated from a given reference, and considers them as insight. But, it is difficult to use for non-expert users or service managers because they have to select and put queries by themselves. Whereas *Timesight* extracts insight automatically using unified formulations of various insight types. In Foresight [1], the authors defined about 6 insight types and their score functions to facilitate the rapid discovery of insights from large, high-dimensional datasets. However, first, because they tried to define insights into the general attribute domain, they did not consider time-driven insights such as trends, change points, seasonality. Also, they did not consider the difference between each attribute`s scales that might affect a huge effect on calculated insight scores. QuickInsight [2] [15] which is the most recent research, is automatic insight discovering system released in Microsoft Power BI [16]. QuickInsight proposed a unified formulation of important patterns and introduced an insight mining framework to automatically mine insight from given data. However, it also did not consider fairness among detected patterns caused by the heterogeneity of attribute scales, time intervals, and formulations. On the other hand, *Timesight* normalizes attributes and time intervals to provide fair scores that users can easily and reasonably accommodate.

VII. CONCLUSION AND FUTURE WORK

We introduce a novel approach to automatically discover interesting insight from multi-dimensional time-series data to offer invaluable hidden information to both data analysts and decision-makers. We decompose a timestamp attribute in several ways to examine the data at various time perspectives and use both numerical attributes and categorical attributes as targets by applying appropriate aggregations. And we normalize values in the result set to obtain fair scores between each insight type, each attribute, and even each time interval. And then, we define 4 types of time-driven insight and unified score functions to assess the interestingness of each result set. Furthermore, we demonstrate *Timesight* using pseudo-code and propose several pruning techniques to improve the performance of *Timesight*. Lastly, we present our experimental result based on an internal service log dataset that helps data analysts to discover hidden insight easily.

We want to advance this research through some direction of future work. First, we will develop and supplement additional types of insight such as the correlation between different Xs. Second, we use uniformed μ and σ^2 for distributions of all insight types in the paper for convenience. But we will investigate lots of datasets to find appropriate μ and σ^2 for each type of insight. Lastly, the limitation of our system is that as the number of rows and attributes in the dataset increase, the search space is extended together. This can increase the time and space it takes to calculate the scores. We need an advanced optimization method to solve these problems.

REFERENCES

[1] Demiralp. C., Haas. P.J., Parthasarathy. S., and Pedapati. T., "Foresight: Recommending visual insight", Proceedings of the VLDB Endowment, Vol. 10, No. 12, pp. 1937-1940, 2017.

[2] Tang. B, Han. S., Yiu. M.L., Ding. R., and Zhang. D., "Extracting Top-K Insights from multi-dimensional Data", In: Proceeding of 2017 ACM International Conference on Management of Data (SIGMOD `17), pp. 1509-1524. ACM, New York, NY, USA, 2017.

[3] Patro. S.G.K., and Sahu. K.K., " Normalization: A Preprocessing Stage", International Advanced Research Journal in Science, Engineering and Technology, Vol. 2, No. 3, pp. 20-22, 2015.

[4] Krzywinski. M., Altman. N., "Points of significance: Significance, p values and t-tests." Nature methods, Vol. 10, pp. 1041-1042, 2015.

[5] Lyon, A., "Why are Normal Distributions Normal?." The British Journal for the Philosophy of Science, Vol. 65, No. 3, pp. 621-649, 2014.

[6] Newman M. E. J., "Power laws, Pareto distributions and Zipf's law." Contemporary Physics, Vol.46, No.5, pp323-351, 2005.

[7] Nopia. Z.M., Lennie. A., Abdullah S., Nuawi. M.Z., Nuryazmin. A.Z., and Baharin. M.N., "The use of autocorrelation function in the seasonality analysis for fatigue strain data." Journal of Asian Scientific Research, Vol. 2, No. 11, pp. 782-788, 2012.

[8] Hamilton. D.F., Ghert. M. and Simpson. A.H., "Interpreting regression models in clinical outcome studies." Bone Joint Res, Vol. 4, No. 9, pp. 152-153, 2015.

[9] Tableau Homepage, https://www.tableau.com/, last accessed 2019/11/14.

[10] Qlik Homepage, https://www.qlik.com, last accessed 2019/11/14.

[11] Brynjolfsson. E., McElheran. K., "The Rapid Adoption of Data-Driven Decision-Making." American Economic Review, Vol. 106, No. 5, pp. 133-139, 2016.

[12] Yu, C.H., "Exploratory data analysis." Methods 2, 2017, pp. 131-160.

[13] Sisense Homepage, https://www.sisense.com/, last accessed 2019/11/14.

[14] Vartak. M., and Rahman. S., Madden. S., "SeeDB: efficient data-driven visualization recommendations to support visual analytics." Proceedings of the VLDB Endowment, Vol. 8, No. 13, pp. 2182-2193, 2015.

[15] Ding. R., Han. S., Xu. Y., Zhang. H., and Zhang. D., "QuickInsights: Quick and Automatic Discovery of Insights from Multi-Dimensional Data." In: Proceeding of the 2019 International Conference on Management of Data (SIGMOD `19), pp. 317-332. ACM, New York, NY, USA, 2019.

[16] Microsoft Power BI Homepage, https://powerbi.microsoft.com/, last accessed 2019/11/14.

[17] Aminikhanghahi. S., and Cook D.J., "A Survey of Methods for Time Series Change Point Detection." Knowledge and information systems, Vol. 51, No. 2, pp. 339- 367, 2017.

Physical Artifacts for Agents in a Cyber-Physical System: A Case Study in Oil & Gas Scenario (EEAS)

1st Fabian Cesar Pereira Brandão Manoel
Federal Center for Technological Education (CEFET/RJ)
Rio de Janeiro, Brasil
0000-0003-0614-0592

2nd Carlos Eduardo Pantoja
Federal Center for Technological Education (CEFET/RJ)
Rio de Janeiro, Brasil
0000-0002-7099-4974

3rd Leandro Marques Samyn
Federal Center for Technological Education (CEFET/RJ)
Rio de Janeiro, Brasil
0000-0002-0733-4172

4rd Vinicius Souza de Jesus
Federal Center for Technological Education (CEFET/RJ)
Rio de Janeiro, Brasil
souza.vdj@gmail.com

Abstract—**Physical devices have been integrated with artificial intelligence to create Cyber-Physical Systems (CPS). Multi-Agent Systems (MAS) can provide pro-activity and autonomy using agents, social organizations, and environment modeling by means of artifacts. Usually, some works that use MAS for interfacing physical environments employ agents accessing directly all the available data of the environment, which could overload this agent. This issue could be avoided if there were tools to facilitate the integration of sensors and actuators as artifacts into the physical environment. Therefore, the objective of this work is to create physical artifacts capable of accessing hardware devices from a physical environment to be used by agents in a MAS. As the Oil & Gas industry demands robustness in its equipment and an ability to do predictive maintenance, a case study including MAS and CPS was developed and some tests were carried out to validate the functioning of physical artifacts.**

Index Terms—**Physical Artifact, Physical environment, Oil & Gas Industry**

I. INTRODUCTION

In the last years, the agent approach has been switching from simulated to physical applications where Multi-Agent Systems (MAS) have been used to interact and control devices working in dynamic environments [1] [2] [3] [4]. In general, some approaches define four dimensions that guide a MAS implementation: agency, environmental, organisational [5], and interaction [6]. Agents interact in an environment according to their implemented beliefs, desire, and intentions (BDI); Artifacts provide operational functions and observable properties for agents, and they represent non-cognitive entities situated in workspaces; organizational dimension models the society notion and the collective norms of the agent's behavior; interaction dimension models the interaction between the three dimensions (agent, environment, and organization). In parallel, when connecting computing elements to physical elements, such as embedded computers connected in a network, it maintains a system known as Cyber-Physical Systems (CPS) [7].

When considering physical environments, rarely they are explored considering other dimensions aside from the agent one. In an agent application in the oil domain, only the agency dimension is considered [8]. The agent performance depends directly on the amount of information that an environment has to offer. There is an approach called ARGO that allows agents to collect data directly from sensors and process them as beliefs in their Belief-Desire-Intention (BDI) reasoning cycle [9]. This process requires reading all the sensors during every cycle execution even if the data are not necessary for the agent, at that moment. Some filtering techniques are available, but they can only be applied after the data has been collected [10].

Initial laboratory experiments for BDI agents in a Web-of-Cell context [11] and a proposed model of many resources of the factory following the A&A [1] are works that consider physical environments using the notion of artifacts. However, both implementations are domain-specific. Artifact is a suitable notion for agents to interact with physical objects in a CPS. When MAS employs artifacts, agents are able to access the physical environment according to their need. It avoids the agents to collect unnecessary data. However, traditional agent-oriented programming languages do not provide direct approaches to access physical environment and they are limited to a particular application domain.

Some initiatives, like the Predictive Maintenance Program (PMP) reveal the importance of collecting data from sensors in the environment to perform predictive maintenance [12]. This importance can also be seen in the Oil & Gas industry because predictive maintenance can minimize economic and environmental losses from poor preventive maintenance.

The objective of this work is to provide physical artifacts for interfacing hardware devices from a physical environment to be accessed by MAS in a CPS. In order to develop theses Physical Artifacts, it will be created an extension of

CArtAgO artifacts that communicates with microcontrollers using serial interfaces. A case study will be presented in a scenario considering a physical engine as an artifact in the Oil & Gas field.

This paper is organized as follows: Section 2 presents the theoretical background to understand the idea; Section 3 shows the methodology used to implement Physical Artifacts, Centralized Layer, and the scenario of study; Section 4 presents the related works and the Section 5 concludes this work.

II. Theoretical Background

Multi-Agent technologies provide tools for distributed control, decentralization, adaptation, and openness. These characteristics can be found in four MAS domains: (i) agent-oriented programming languages, (ii) interaction languages and protocols, (iii) environment frameworks, architectures and infrastructure, and (iv) organizational systems. These perspectives lead MAS to the four dimensions of development, such as described by JaCaMo approach [13] and complementary works [6]: *organization*, where rules and missions are defined to ensure the society behavior; *agent*, where BDI agents are implemented; and *environmental*, responsible to integrate the external environment and agents using artifacts with operational functions; *integration*, that represents program languages responsible to ensure integration between agents, artifacts and organization rules.

In the Multi-Agent field, artifacts are Activity Theory and Distributed Cognition-based computational devices existing in environments and capable of performing a particular function or service that agents can explore. Regarding the agent/artifact relationship, there are two different types of external objectives attached to an artifact: (i) *use value*, where external goals head the artifact selection by agents; and (ii) *use*, which is associated with agent's internal goals [14]. Therefore, three distinct aspects characterize the relationship between agents and artifacts: agents can *select*, *use*, and *construct/manipulate* artifacts, where the latter occurs when the artifact does not exist and needs to be created.

Artifacts are composed of four elements [13]: *User Interface (UI)*, *Operating Instructions (OI)*, *Function*, and *Structure and Behavior*. *User Interface (UI)* is a set of operations that agents can call to use the artifact; *Operating Instructions (OI)* describe how the artifact should be used to access its functionality; *Function* is the purpose of the artifact's existence; and *Structure and Behavior* are the internal characteristics of artifacts that define how it is implemented [15].

For programming the environmental dimension for agents, there is the CArtAgO framework, which is based on three main pillars. The (i) Agent Body is the part of an agent where artifacts represent some behaviors that it can access and control but it is not part of their internal reasoning; (ii) Artifacts are the components identified in a Workspace that agents or any part of their body can interact with; A (iii) Workspace is used to define the desktop topology. Artifacts and Agent Bodies are stored in these Workspaces, where the relation between them is established. Then, artifacts must be within a specific Workspace so that agents can use. Consequently, events generated by these artifacts can only be seen by agents living in the same Workspace [16].

Using artifacts that are only accessible within their workspaces may not represent the best approach to be employed in dynamic scenarios since it restricts agents that are not originally from these workspaces to access the environment's resources. In dynamic scenarios agents can come and go freely and they can compete for each available component. Moreover, the environment should be open for any entity that intends to enter it. However, even CArtAgO, and other languages and frameworks that consider the development of artifacts do not provide a distributed and open characteristics for environments.

III. Methodology

In CPS, the use of environmental objects by computational entities is an essential factor that helps these entities to adapt to environments with dynamic characteristics. Besides, these environments are increasingly demanding automation, pro-activity, and cognition. While the agents layer promotes computational intelligence and the Organization layer promotes social rules, the Artifacts layer encourages the modeling of objects from the external environment. Although the environment layer is ideal for representing objects from the external MAS environment, there are approaches that still transfer this responsibility to agents. Therefore, this work presents a solution to apply MAS in physical, dynamic, and intelligent environments using Physical Artifacts to connect MAS artifacts to ATMEGA microcontrollers. A scenario will be presented with instrumented engines in the Oil & Gas industry with a focus on predictive maintenance implemented in MAS with Physical Artifacts.

A. Oil & Gas Engine Scenario

When it comes to equipment maintenance, the natural approach is prevention, which aims to replace defective components or parts from time to time. However, this type of maintenance can be costly from a financial and environmental point of view. From an economic point of view, the periodic replacement of a specific component can make the process more expensive; from the environmental point of view, the equipment may present failure situations before the replacement period and cause accidents to the environment. On the financial side, prediction is better than prevention because predicting that the equipment's life cycle will be longer than usual can avoid spending on unnecessary maintenance. On the environmental side, predicting that equipment is being damaged can result in support before it is damaged. Following this idea, the Oil & Gas industry benefits in the economic and environmental fields with predictive maintenance.

The Strategic Petroleum Reserve (SPR) - that is an Oil emergency fuel storage unit - is composed of several engines that supply power to the pumps that move a large amount of oil in the unit. As the SPR does not have a continuous operation, the motors do not remain connected at all times,

which hinders the temporal precision that is necessary to carry out preventive maintenance. Therefore, prediction techniques on engines such as vibration analysis, thermography, and oil analysis can be useful to reduce maintenance costs and prevent accidents. In addition to sensors for analysis, the motors have actuators that define their operation and can also be activated intelligently to minimize the risk of equipment degradation.

As a motivation to use prediction as an approach, the Predictive Maintenance Program (PMP) proposed in 1994 sets targets for reducing maintenance costs by 20% by the third year of operation of this PMP [12]. With PMP, it is possible to offer accuracy to equipment operators as to when intervention should occur. In this case, expenses with unnecessary maintenance and the risk of accidents would be reduced.

B. The Physical Artifacts

A Physical Artifact is an extension of the standard MAS Artifact capable of integrating with a physical Device in the environment to collect its sensor data or send actuation commands to actuators. For Physical Artifacts, a Device is an object in the physical environment composed of a microcontroller with sensors or actuators. Besides, a Device must have communication functions between the microcontroller and another external computational entity to provide readings on its sensors and receive commands for its actuators. Therefore, to become a Physical Artifact, an object in the physical environment must assume the characteristics of a Device. In this case, the Operation Functions of this Artifact can be implemented to read the sensors and operate directly on the actuators of this Device. For example, in the scenario of engines in the Oil & Gas industry, it is necessary a microcontroller in them that sends the data from the vibration sensors, thermography and oil to this Artifact.

To create Physical Artifacts, the Artifacts implementation of CArtAgO framework was employed. We chose CArtAgO because it is used to create the environmental dimension of the JaCaMo framework for Jason and because both CArtAgO and Jason are widely used in the academia. In the hierarchical structure of CArtAgO, the Physical Artifact is a child class of the Artifact class. Therefore, physical artifacts can also implement Observable Properties and Operations. It is expected with this integration to allow MAS integration with CPS without overloading agents.

Once incorporated as CArtAgO Artifacts, Physical Artifacts must be able to communicate with Devices in the physical environment. For this, the serial interface Javino [17] was employed, which is a library that implements a protocol for exchanging messages between low-level hardware (microcontrollers) and high-level software (Java). The choice for Javino is justified because it is a serial communication library that handles error detection, unlike libraries based on serial ports, such as RxTx and JavaComm. The messages exchanged between hardware and software follow a format composed of 3 fields: 2 bytes of a pre-scope that is used to identify the beginning of the message, 1 byte to represent the size of the

main content of the message, and finally, 256 bytes containing the content of the message to be passed. The loss or collision of information from the past message is verified through the pre-scope field and the size field: the receiver validates the content in the pre-scope; if the preamble is correct, the size field helps to verify that the message arrived at the correct size. If all verification is validated, the message is used; otherwise, the message is discarded. Javino offers three modes of operation: **Send**, **Request**, and **Listen** modes. The **Send** mode provides simplex message transmission from software to hardware; the **Request** mode offers half-duplex message transmission (the hardware responds to the message sent); the **Listen** mode allows the transmission of simplex messages from hardware to software. Another factor that justifies the choice of Javino is the possibility that it is designed to be multi-platform and can be used in ATMEGA, PIC, or Intel Families microcontrollers.

The use of Javino as a connection bridge must be analyzed both on the side of the abstraction (Physical Artifact) and the embedded hardware (Device). On the Physical Artifact side, the Javino implementation class for high-level software is added as an attribute to the *PhysicalArtifact* class and instantiated directly in the constructor. All child classes of *PhysicalArtifact* must define, via abstract method, the following values: **Serial Port** that will be used to connect the artifact to the microcontroller (method *String definePort()*), a **Number of Attempts** to send a message (method *int defineAttemptsAfterFailure()*), and **Timeout** in milliseconds between one attempt to send a message and another *int defineWaitTimeout()*). In addition, the class *PhysicalArtifact* has the implementation of the *String read()* method, which performs reading from a physical device in Javino **Listen** mode; and also has implementation of the *void send (String message)* method, which sends messages in **Send** mode to the microcontroller. Figure 1 shows the architecture of Physical Artifacts in a MAS communicating with a physical environment.
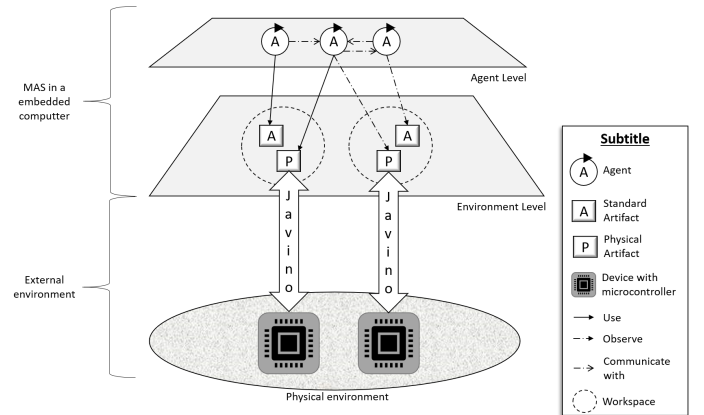
Fig. 1. The architecture of a MAS integrated with a physical environment showing only the Agent level and the Environment level. At the Environment level, Standard Artifacts are together to Physical Artifacts that connects to a Device in the physical environment using Javino middleware.

C. Engine Scenario Prototype

To represent the engine scenario in the Oil & Gas industry and test the Physical Artifacts approach, a prototype of an instrumented engine with a sensor was created and connected to a MAS that will control it, as shown in the Figure 3. The physical prototype consists of a fan to represent the motor actuator, a temperature sensor, and LEDs that indicate the state of operation of the motor. An Arduino Mega was used as a microcontroller that contains all the sensors and actuators of the prototype. Besides, Arduino Mega is responsible for exchanging messages with MAS. This physical configuration configures the physical prototype as a Device that can be used by a Physical Artifact.

The engine designed in the prototype has the following operations: turn on, off, block use, unlock use. In particular, the blocking operation is used by operators when the engine is in an abnormal condition and should not be operated. In this prototype, the motor has three possible states: **Ready to be Operated**, represented in the first lower frame of the Figure 3, where the prototype is turned off and unlocked; **On**, represented in the second lower frame of the Figure 3 (represented by the connection of two of the three LEDs); **Blocked**, represented in the third lower frame of the Figure 3, where the motor is blocked for use (indicated by the red led).

On the Arduino side, the Javino library is imported and used as a support in sending and receiving messages to the Physical Artifact. The Arduino was programmed to send data from the temperature sensor whenever a message arrives from the Artifact that requests it. In addition, the Arduino operates the engine whenever the Physical Artifact requests one of the available forms of operation.

On the MAS side, the Motor Artifact described in Figure 2 was created that extend a Physical Artifact. In this Motor Artifact, operations are implemented to read the temperature sensor, turn on, off, lock, and unlock the motor. In addition, an Agent Manager was created in MAS to control operations. For this, this Agent creates the Artifact Motor and starts a basic cycle of activities to test all the operations provided by Artifact. In the upper left corner of Figure 3, the running agent log is displayed.

When modeling the class diagram in Figure 2, a representation of the engine for the system with the respective registration information can be seen. Besides, a model of sensors and sensor measurements was created to record the data in a MySQL database. With this, an application was developed to allow monitoring at the level of the engine operator so that it can visually diagnose the engine situation. In the upper right corner of the Figure 4, is showed a graph with temperature measurements of the environment where the prototype is located. These measures is in degrees Celsius unit and are provided by the Physical Artifact that reads the engine. The variation in the graph can be analyzed by an operator in the field of work and serve as a variable in the generation of a prediction diagnosis, for example, associating that the increase in temperature crossed with other data, means loss of engine

life.
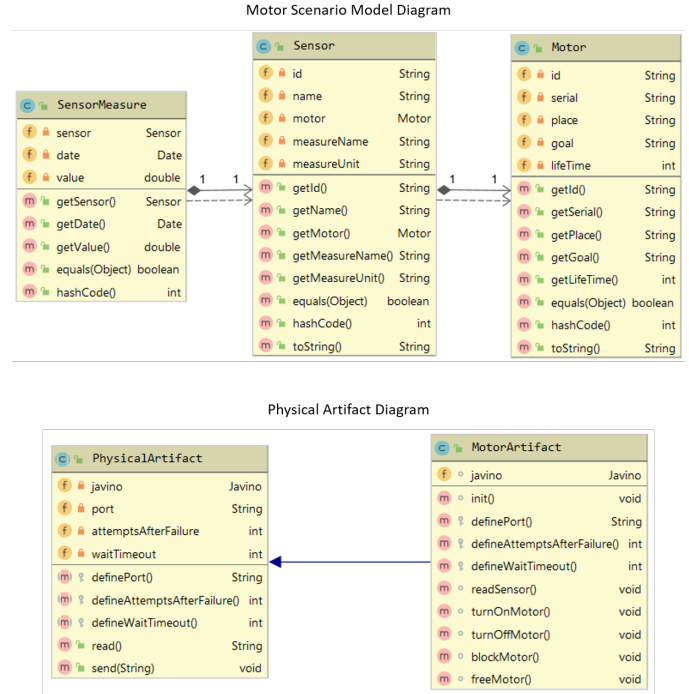
Fig. 2. Class diagram of the model made to represent the Engine scenario.
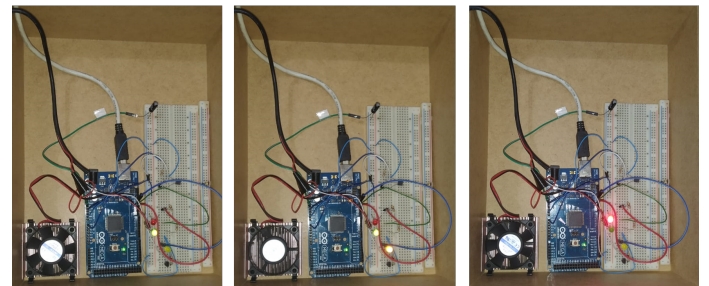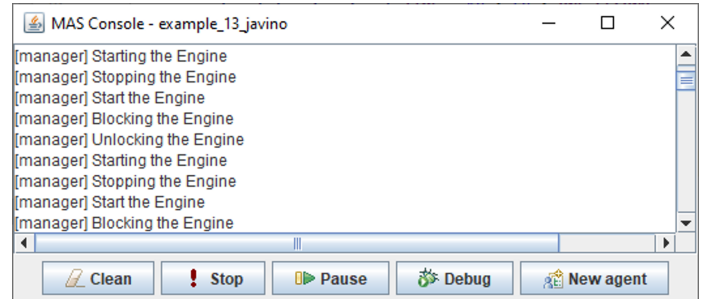
Fig. 3. Engine scenario in the Oil & Gas industry in execution: Agent Manager performing control and monitoring of the Physical Artifact, and prototype of the engine connected to the MAS.

D. Experimental Evaluation

From the elaborated scenario, tests were done to validate the functioning of Physical Artifacts in the physical environment. For this, the requirements of the framework were raised to support the experiments: (i) the Physical Artifact must be able
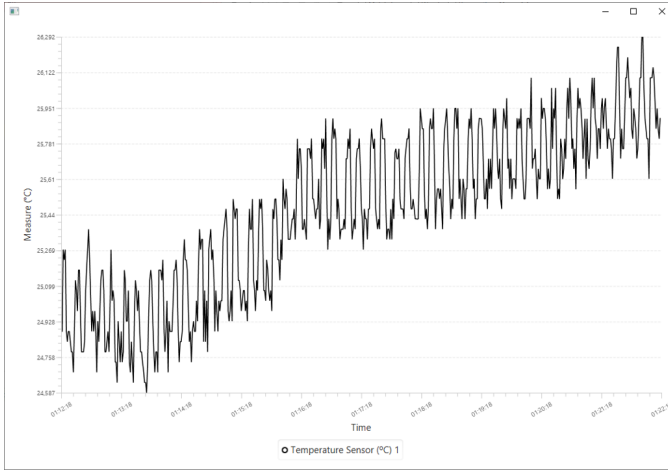
Fig. 4. An application that displays a line graph with temperature measurements of the environment in which the prototype is based on time, in degrees Celsius, provided by the Physical Artifact. The X-axis is expressed in hours, minutes, and seconds. The temperature measurement is an example, which could be replaced by measurements from other sensors.

TABLE I
THE CASE STUDY DESIGN

Design	Description
Objective	Analyze the functioning of the MAS Physical Artifact in a physical setting.
Case	The Physical Artifact will be connected to an oil and gas engine that must be monitored and controlled to perform predictive maintenance.
Questions	Is the Physical Artifact capable of sending and receiving information through Javino? Do physical Artifacts respond to agents' requests in up to one second? Do Physical Artifacts stay running for a minimum of 24 hours?
Method	Observation method with a low degree of researcher interaction.

to send and receive information using Javino; (ii) the Physical Artifact must be able to respond to agents' requests within one second, which is considered acceptable within the high-level programming field; (iii) the Physical Artifact must be able to function in a 24-hour period in the worst case.

Based on these premises, routine tests were carried out, where the agent requested the operations to start, stop, restart, lock, and unlock the engine. At each operation, the agent requests data from sensors ten times. It was concluded that the commands from the Artifact work normally. In addition, a throughput test was carried out between the agent's command and the execution of the Artifact, and it was observed that the waiting time is below one second. Finally, the Agent Manager was kept in operation for 24 hours, where it was observed that the Artifact continues to respond with a failure rate of 0%. Table I shows the case study analysis of this scenario and Table II shows the results from tests.

IV. RELATED WORKS

Physical environments have been demanding computational systems more proactive, autonomous, and adaptable to solve

TABLE II
EXPERIMENTAL EVALUATION RESULTS

Test	Description	Result
The connection between Physical Artifact and Microcontroller	Percentage of success (%) when exchanging data with the microcontroller	100%
Physical Artifact Responses to the Agent	Maximum time (milliseconds) that an Agent takes to receive data from the Artifact	1000ms
Physical Artifact execution time	Checks whether the Physical Artifact remains running for 24 hours	Yes

increasingly complex problems. The community has been developing some works using MAS in industry as an attempt to increase pro-activity and autonomy in the production chain. There is a work in the Oil & Gas industry which uses BDI agents to filter alarms that are generated by different conditions [8]. This filtering considers that an operator is not able to observe a broad set of alarms and act on them. Besides, excessive alarms can hide an important occurrence, and therefore there must be an intelligent system capable of filtering this data. For this, an alarm management system was developed using agents able of reading sensors and act on devices. However, agents were programmed directly connected to environments — in case of agents responsible for only one sensor or only one actuator — without using the notion of artifacts. As a result, agents could face bottlenecks in their reasoning due to the need to be continuously collecting data without necessarily using it.

ARGO [9] is a customized *Jason* agent's architecture that allows interactions with physical devices such as sensors and actuators. For this, a serial interface between microcontrollers and *Java* programming language was developed to collect all data from the environment to be added to the agent's belief base. The generated data flow overloads ARGO agents and filtering techniques [10] can be employed to select which perceptions the agent has to focus on. However, the sensors and actuators are available only for a specific MAS and they are not shareable. Besides, ARGO may experience a decrease in computational performance as the amount of information to be perceived increases. Both works could benefit from an approach that exposes sensors and actuators as shareable resources in the IoT.

Given the overload on the agents and aiming to take advantage of the MAS environmental modeling resources, some works developed solutions applied to physical environments. In the energy sector, a Web-of-Cell (WoC) approach [11] uses MAS to help design and test distributed solutions. For this, the Jason framework is used to develop BDI agents; environment modeling is done using the CArtAgO framework, which allows creating a bridge between the agent layer and the environment layer. The communication between the modeled environment and the physical environment was done by the communication infrastructure of the intelligent and configurable network laboratory (SYSLAB). However, this communication bridge is

strongly linked to the SYSLAB structure, which still does not help in the mission to facilitate implementations with MAS that involve environmental modeling.

MAS heterogeneity has been increasingly required in challenges that integrate CPS with environmental resources. In the Industry 4.0 concept, equipment and sensors must be integrated into the same system using the most diverse communication protocols. Camel Artifact [1] is a component that uses Java-based message routing and mediation technology (Apache Camel) in artifacts. A CamelArtifact makes it possible to transform physical devices into Artifacts in a more generic way than the WoC approach because several communication protocols can be used to create the bridge between the physical and the computational environment. For this, routing is done that directs the messages from a device to the specific artifact. However, although this work does not depend on a particular protocol of communication between physical devices and MAS, there is still a strong dependence on Apache Camel technology that guarantees message routing. Perhaps, an approach that integrates artifacts with microcontrollers can offer even more heterogeneity because it will allow configurations of these devices more directly and at a low level. If these artifacts were shareable between different MAS, the collected data would become resources of the environment that agents from any MAS could exploit.

V. FINAL CONSIDERATIONS

Normally, MAS applications using physical environments for CPS overloads agents with data coming from sensors and actuators. Besides, when they are not overloaded, the connections to these kind of artefacts are bounded to the provided solution. Based on that, this work presented an extension of CArtAgO for providing Physical Artifacts without generating overload to agents using a serial interface for communicating with heterogeneous microcontrollers.

In order to create Physical Artifacts, several technologies were employed such as Jason and CArtAgO frameworks, the Serial Interface Javino, and microcontrollers. CArtAgO's Artifact was extended to allow Javino to interact with sensors and actuators connected to microcontrollers. The proposed extension was tested in an engine scenario for Oil & Gas domain. The results showed that our approach is suitable for designing CPS using MAS and Physical Artifacts.

A future issue to be considered is that the fact that artifacts to be accessible only within their workspaces can make it challenging to implement in dynamic scenarios because this restricts agents that are not from this MAS. If artifacts could be accessed by agents from another MAS, it could be possible to create a multi-purpose layer of physical artifacts to be consumed by different agents. As future works, we intend to create a shareable layer of artifacts to be used along with the Internet of Things. Agents from different MAS, or any other technology, could compete for Physical Artifacts. Besides, we will extend the scenario of motors for Oil & Gas to allow a middle layer capable of managing plans and rules for some situations when using those motors.

REFERENCES

[1] C. Amaral, S. Cranefield, J. Hübner, and M. Roloff, "Giving camel to artifacts for industry 4.0 integration challenges," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 11523 LNAI, 2019.

[2] T. Sanislav, G. Mois, S. Folea, L. Miclea, G. Gambardella, and P. Prinetto, "A cloud-based cyber-physical system for environmental monitoring," in *2014 3rd Mediterranean Conference on Embedded Computing (MECO)*, pp. 6–9, IEEE, 2014.

[3] S. Wang, J. Wan, D. Zhang, D. Li, and C. Zhang, "Towards smart factory for industry 4.0: a self-organized multi-agent system with big data based feedback and coordination," *Computer Networks*, vol. 101, pp. 158–168, 2016.

[4] B. Vogel-Heuser, C. Diedrich, D. Pantförder, and P. Göhner, "Coupling heterogeneous production systems by a multi-agent based cyber-physical production system," in *2014 12th IEEE International Conference on Industrial Informatics (INDIN)*, pp. 713–719, IEEE, 2014.

[5] O. Boissier, R. H. Bordini, J. F. Hübner, A. Ricci, and A. Santi, "Multi-agent oriented programming with jacamo," *Science of Computer Programming*, vol. 78, no. 6, pp. 747–761, 2013.

[6] M. R. Zatelli and J. F. Hübner, "The interaction as an integration component for the jacamo platform," in *International Workshop on Engineering Multi-Agent Systems*, pp. 431–450, Springer, 2014.

[7] E. A. Lee, "Cyber physical systems: Design challenges," in *2008 11th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC)*, pp. 363–369, IEEE, 2008.

[8] N. Sanchez-Pi, L. Leme, and A. Garcia, "Intelligent agents for alarm management in petroleum ambient," *Journal of Intelligent and Fuzzy Systems*, vol. 28, no. 1, pp. 43–53, 2015.

[9] C. E. Pantoja, M. F. Stabile Jr, N. M. Lazarin, and J. S. Sichman, "Argo: A customized jason architecture for programming embedded robotic agents," *Fourth International Workshop on Engineering Multi-Agent Systems (EMAS 2016)*, 2016.

[10] M. F. S. Jr, C. E. Pantoja, and J. S. Sichman, "Experimental analysis of the effect of filtering perceptions in bdi agents," *International Journal of Agent-Oriented Software Engineering*, vol. 6, no. 3-4, pp. 329–368, 2018.

[11] D. Issicaba, M. Rosa, A. Prostejovsky, and H. Bindner, "Experimental validation of BDI agents for distributed control of electric power grids," in *2017 IEEE PES Innovative Smart Grid Technologies Conference Europe, ISGT-Europe 2017 - Proceedings*, vol. 2018-January, pp. 1–6, 2018.

[12] R. J. Murry and B. F. Mitchell, "Cost savings from a practical predictive-maintenance program," in *Proceedings of Annual Reliability and Maintainability Symposium (RAMS)*, pp. 206–209, IEEE, 1994.

[13] A. Ricci, M. Viroli, and A. Omicini, "Programming mas with artifacts," in *International Workshop on Programming Multi-Agent Systems*, pp. 206–221, Springer, 2005.

[14] R. Conte, C. Castelfranchi, *et al.*, *Cognitive and social action*. Garland Science, 2016.

[15] A. Omicini, A. Ricci, and M. Viroli, "Coordination artifacts as first-class abstractions for mas engineering: State of the research," in *International Workshop on Software Engineering for Large-Scale Multi-agent Systems*, pp. 71–90, Springer, 2005.

[16] A. Ricci, M. Viroli, and A. Omicini, "Cartago: A framework for prototyping artifact-based environments in mas," in *International Workshop on Environments for Multi-Agent Systems*, pp. 67–86, Springer, 2006.

[17] N. M. Lazarin and C. E. Pantoja, "A robotic-agent platform for embedding software agents using raspberry pi and arduino boards," in 9th *Software Agents, Environments and Applications School*, 2015.

Classifying Common Security Vulnerabilities by Software Type

Onyeka Ezenwoye[1], Yi Liu[2], and William Patten[1]
[1] Augusta University, Augusta, GA, USA, `oezenwoye,wpatten@augusta.edu`
[2] University of Massachusetts Dartmouth, Dartmouth, MA, USA, `yliu11@umassd.edu`

Abstract—The National Vulnerability Database does not identify a type for the software that is impacted by a specified weakness. To gain some insight into the security vulnerability landscape, we classify by software type a total of 51,110 vulnerability entries from 2015 to 2019. The software types are operating system, browser, middleware, utility, web application, framework, and server. This classification shows the pattern of prevalence of software weaknesses and the persistence of weaknesses as they pertain to each software type.

Keywords: Software, Security, Vulnerability, Weakness, Taxonomy.

I. INTRODUCTION

Since 2002, the National Vulnerability Database (NVD) [2] has maintained a list of exploitable security vulnerabilities that exist in software. Each vulnerability entry in the list has multiple attributes, one of which is the software weakness (E.g., buffer error). An aggregate of common weaknesses over time is available [3]. The vulnerability entries span many thousands of products which range from the recognizable to the very obscure. The database however does not specify a software type for these products. Identifying the software type is important in understanding the vulnerability landscape as it pertains to each software type [9].

With the ubiquity of software comes the associated developer-driven software weaknesses. Knowing the weakness characteristics of the software type is important in devising targeted fault avoidance and detection mechanisms which include threat modeling, architecture review, risk analysis, training, and policies for evolution and maintenance [5], [12]. To this end, we are analysing vulnerability database entries. Part of this effort includes classifying the vulnerabilities by software type. Here, we report our findings thus far of classifying the most recent five years of vulnerability data. With this, we seek to provide answers to two questions:

1) Is the occurrence of the most common weaknesses consistent across software types?
2) How well do weaknesses persist over years for the same product?

With these answers, we provide some additional insight into vulnerability weaknesses that isn't available in existing literature. The rest of this paper is structured as follows, our approach is described in Section II. We present results in

Section III. Related work can be found in Section IV with Conclusion in Section V.

II. APPROACH

The NVD catalogs security vulnerabilities in a list known as Common Vulnerabilities and Exposures (CVE). Each entry in the list contains attributes such as a unique identifier, the product's vendor, product name, description, severity score, etc. Each CVE entry also contains a Common Weakness Enumeration (CWE) name [1]. The CWE name identifies the specific vulnerability type (E.g., Improper Authentication, and Buffer Error). From here on, we refer to each CVE entry as vulnerability and each vulnerability type as weakness.
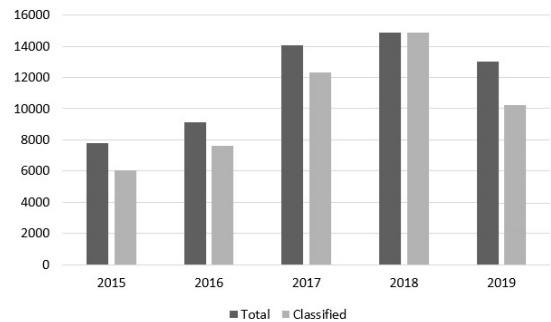
Fig. 1. Total number of vulnerabilities reported Vs number of vulnerabilities classified by software type

Each vulnerability contains the product's vendor and name (E.g., Microsoft and Windows 10) but not the product type (E.g., Operating System). To answer the questions from Section I, we decided to review all vulnerabilities from 2015 to 2019 and classify each vulnerability by a product type. We classified each vulnerability into one of seven product types. We use a database (of vendor, product name, and product type groupings) to map each vulnerability to a product type. We continue to update this database (with vendor, product name, and product type groupings) in order to achieve 100% classification for all years. Currently only 2018 is complete (Figure 1). The product types are Web Application, Utility, Server, Operating System, Browser (Web), Framework, and Middleware. The types are loosely based on the taxonomy proposed by Forward et al. [9]. We briefly describe each type:

- Web Application: web-based (or cloud-based) software such as content management systems, information management systems, transaction processing systems, etc.

- Utility: standalone applications such as productivity, creativity, antivirus, scripts, non-web clients, etc.
- Server: system servers (including cloud-based) such as database, email, proxy, web, FTP, DNS, load balancers, network monitors, etc.
- Operating System: firmware, device drivers, virtual machines, and all types of operating systems.
- Browser: all types of web browsers.
- Framework: software components such as libraries, plugins, and extensions.
- Middleware: enterprise transaction platforms such as message queuing, object storage, and identity management systems.

Of the 58,781 unique vulnerabilities over that period, we have classified 51,110 (87%), including 100% of all vulnerabilities from 2018. Figure 1 shows a comparison of the number of vulnerabilities from each year and the number of those that have been successfully classified by product type, so far. We are confident that at this point, the results from the number that have been classified should be fairly representative of the total. To support this argument, Figure 2 shows a comparison distribution by software product between 2018, which has been 100% classified, and all other years. The chart shows a similarity in distribution across product type. It also shows that operating systems and utilities account for about half of all vulnerabilities.
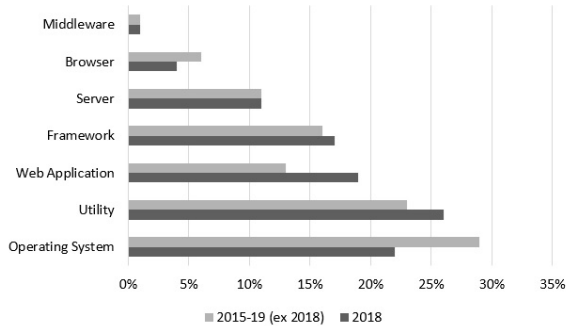
Fig. 2. Relative vulnerability count by software product type

From our classification, we found that there are a total of 169 distinct weaknesses across all product types. Figure 3 shows the most common of these weaknesses. By most common we mean that each weakness has a count that is at least 3% of all vulnerabilities classified (51,110). Collectively, these 7 weaknesses account for just over 53% of all vulnerabilities, the other 162 weaknesses account for the rest. The chart shows that buffer errors and cross-site scripting are the two most individually occurring security weaknesses in general.

III. Results

In this section we offer some answers to the questions discussed in Section I.
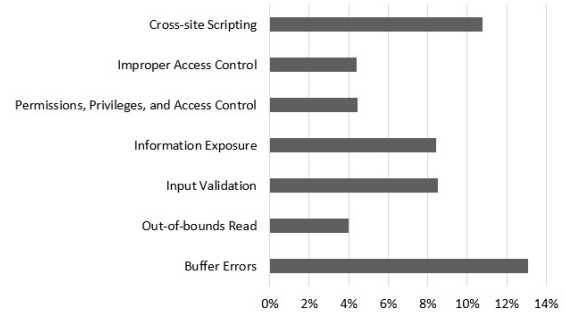
Fig. 3. Most common weaknesses across all vulnerabilities (2015-2019)

A. Is the occurrence of the most common weaknesses consistent across software types?

To help answer this question, we present a breakdown of the most common weaknesses by each product type. Figure 4 shows the most common weaknesses for the Browser type. For this type, there were 74 different weaknesses across a total of 2,897 vulnerabilities. These 8 weaknesses account for 75.5% of all vulnerabilities. Buffer error is the most occurring weakness, which is consistent with Figure 3.
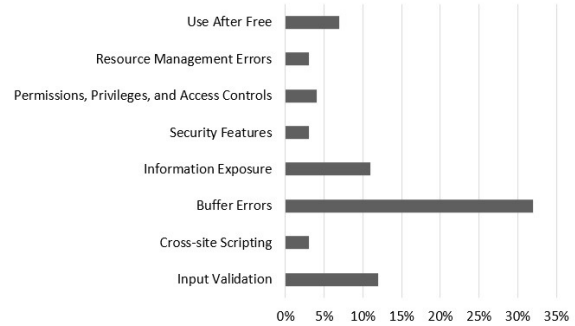
Fig. 4. Most common weaknesses for the Browser type

Figure 5 shows the most common weaknesses for the type classified as Framework. There are a total of 129 weaknesses across 8,444 vulnerabilities. These 9 weaknesses account for 59.6% of all vulnerabilities. The two most common weaknesses are buffer errors and cross-site scripting. This could be explained by the fact that many of the products that fall into this type are frameworks for web applications and other downloadable program libraries. The two most common here is somewhat consistent with Figure 3.

Figure 6 shows the most common weaknesses for the Middleware product type. There are a total of 59 weaknesses across 441 vulnerabilities. These 9 weaknesses account for 54.6% of all vulnerabilities. Although present, buffer errors, a typically common weakness (Figure 3), don't feature prominently here. Figure 7 shows the most common weaknesses for Operating systems. There are a total of 148 weaknesses across 13,670 vulnerabilities for this product type. These 7 weaknesses account for 54.1% of all vulnerabilities. All of these weaknesses are the same that appear as the most common
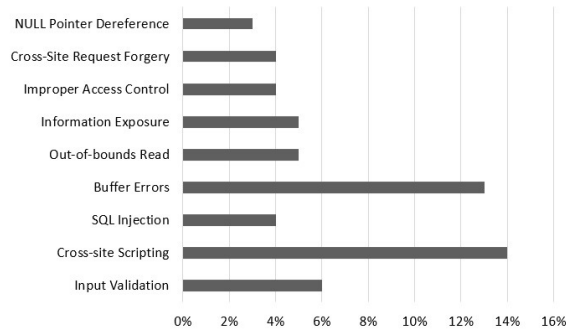
Fig. 5. Most common weaknesses for the Framework type

in Figure 3. This is partly due to prominence of the Operating System type in the classification (Figure 2).
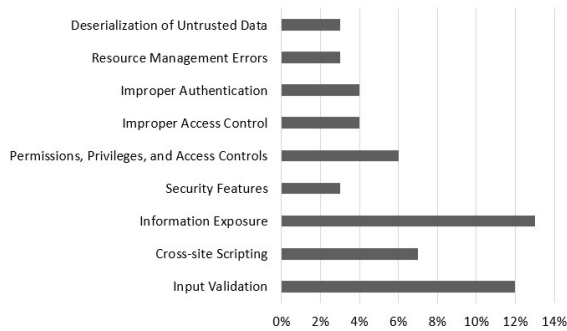
Fig. 6. Most common weaknesses for the Middleware type

Fig. 7. Most common weaknesses for the Operating System type

Figure 8 shows the most common weaknesses for the Server product type. There are a total of 118 weaknesses across 5,840 vulnerabilities for this type. These 9 weaknesses account for 59.8% of all vulnerabilities. All of the most common weaknesses (Figure 3) feature prominently here, in addition to the SQL Injection and Path Traversal weaknesses. Figure 9 shows the most common weaknesses for the product type classified as Utility. There are a total of 124 weaknesses across 8,878 vulnerabilities for utilities. These 9 weaknesses account for 61% of all vulnerabilities.

Figure 10 shows the most common weaknesses for Web applications. For this product type, there are 105 weaknesses

Fig. 8. Most common weaknesses for the Server type

Fig. 9. Most common weaknesses for the Utility type

across 7,595 vulnerabilities. These 8 weaknesses account for 72% of all vulnerabilities for Web applications. We note that the most common weaknesses for the Web application type are similar to those of the Server type (Figure 8), allbeit at different degrees. The results show that the most common weaknesses (Figure 3) do not appear consistently across all software product types. Of the 7 weaknesses, only Input Validation, Cross-site Scripting, and Information Exposure occur at a high enough rate (3%) for every product type. The Buffer Error weakness met this threshold all types but Middleware. Buffer Error was recorded at 1.6% of all vulnerabilities for the Middleware type. Also, the rate at which each weakness occurs does vary greatly across product types.
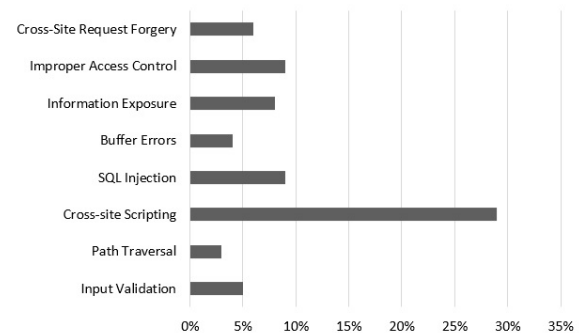
Fig. 10. Most common weaknesses for the Web Application type

B. How well do weaknesses persist over years for the same product?

We reviewed the data to determine whether the same weakness (E.g., Buffer Error) for a given product (E.g., Microsoft Windows 10) occurs again in subsequent years for the same product. Our results show that only a small percentage (less than 5%) of weaknesses repeat for the same product and the rate at which they reoccur does decrease over time (Figure 11). A reasonable assumption here is that a majority of the weaknesses get repaired. Also, over time, some products get discontinued or evolve into a different product. Products that have evolved into a different product (name) would not show up in the data as repeating. Figure 12 shows that Web applications account for the most occurrences of repeating weakness. It is important to note that the Operating System type which accounts for a high number of vulnerabilities (Figure 2) does not have as high a rate of repeating weaknesses, relatively. The results here highlight the importance of updating existing software installations.
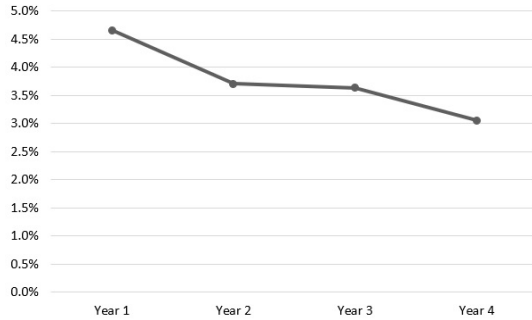
Fig. 11. Number of weaknesses that repeat for the same product over time
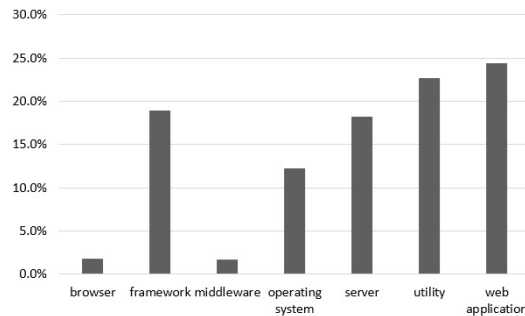
Fig. 12. Distribution of repeating weaknesses by software type

IV. RELATED WORK

Some existing works have looked at analyzing vulnerability data from the NVD. None that we know of classify software weaknesses by product type. Santos et al. [8] devised a Common Architectural Weakness Enumeration as a means to catalog the common types of architectural weaknesses that generally exist in software. Na et al. [13] discuss a technique for analyzing vulnerability entries that do not have identified weaknesses. Their techniques attempts to identify the

weakness using existing information in the vulnerability entry. Neuhaus et al. [14] analyzed vulnerability entries to identify trends in topics such as PHP and Format Strings that appear in the entries. Chang et al. [7] analyzes vulnerability entries from trends in the frequency and severity of vulnerability types. Others [4], [6], [10], [11], [15], [16] take a similar approach by analyzing existing texts for vulnerability trends.

V. CONCLUSION

To gain further insight into reported software security vulnerabilities, we analyzed 51,110 vulnerabilities from 2015 to 2019. We classify these vulnerabilities by software type. Our analysis shows that the occurrence of weaknesses across software types does vary greatly both in the type of weakness and rate of occurrence for each software type. We also show that the rate at which identified weaknesses repeat for the same product is significantly reduced over time. As we continue to analyze the data, we believe that our findings here will help inform approaches to the avoidance and detection of software weaknesses as well as inform strategies for software evolution and maintenance.

REFERENCES

[1] The common weakness enumeration. https://nvd.nist.gov/vuln/categories, Retrieved February, 2020.
[2] National vulnerability database. https://nvd.nist.gov/, Retrieved Febraury, 2020.
[3] Relative vulnerability type totals by year. https://nvd.nist.gov/vuln/visualizations/cwe-over-time, Retrieved February, 2020.
[4] S. Alqahtani and J. Rilling. Semantic modeling approach for software vulnerabilities data sources. In *Proceedings of the 17th International Conference on Privacy, Security and Trust*. IEEE, 2019.
[5] E. Amoroso. Recent progress in software security. *IEEE Software*, 35(2), 2018.
[6] F. Bulut, H. Altunel, PMP, and A. Tosun. Predicting software vulnerabilities using topic modeling with issues. In *Proceedings of the 4th International Conference on Computer Science and Engineering*, 2019.
[7] Y.-Y. Chang, P. Zavarsky, R. Ruhl, and D. Lindskog. Trend analysis of the cve for software vulnerability management. In *Proceedings of the IEEE Third International Conference on Social Computing*, Oct 2011.
[8] J. C. da Silva Santos, K. Tarrit, and M. Mirakhorli. A catalog of security architecture weaknesses. In *Proceedings of the 2017 IEEE International Conference on Software Architecture Workshops*, 2017.
[9] A. Forward and T. Lethbridge. A taxonomy of software types to facilitate search and evidence-based software engineering. In *Proceedings of the 2008 Conference of the Center for Advanced Studies*, 2008.
[10] D. Gonzalez, H. Hastings, and M. Mirakhorli. Automated characterization of software vulnerabilities. In *Proceedings of the IEEE International Conference on Software Maintenance and Evolution*, 2019.
[11] T. H. M. Le, B. Sabir, and M. A. Babar. Automated software vulnerability assessment with concept drift. In *Proceedings of the IEEE/ACM 16th International Conference on Mining Software Repositories*, 2019.
[12] G. McGraw. Silver bullet talks with Ksenia Dmitrieva-Peguero. *IEEE Computing Edge*, February 2019.
[13] S. Na, T. Kim, and H. Kim. A study on the classification of common vulnerabilities and exposures using naïve bayes. In *Proceedings of Advances on Broad-Band Wireless Computing, Communication and Applications*. Springer, 2016.
[14] S. Neuhaus and T. Zimmermann. Security trend analysis with cve topic models. In *Proceedings of the 21st IEEE International Symposium on Software Reliability Engineering*. IEEE, November 2010.
[15] M. Williams, R. Camacho Barranco, S. M. Naim, S. Dey, M. Hossain, and M. Akbar. A vulnerability analysis and prediction framework. *Computers Security*, February 2020.
[16] X. Wu, W. Zheng, X. Chen, F. Wang, and D. Mu. CVE-assisted large-scale security bug report dataset construction method. *Journal of Systems and Software*, 160, 11 2019.

Modeling Relation Path for Knowledge Graph via Dynamic Projection

Hongming Zhu[†], Yizhi Jiang[†], Xiaowen Wang[†], Hongfei Fan[†], Qin Liu[*†], and Bowen Du[*‡]

[†]School of Software Engineering, Tongji University
[‡]Department of Computer Science, University of Warwick
Email: {zhu_hongming, 1931566, 1931533, fanhongfei, qin.liu}@tongji.edu.cn, B.Du@warwick.ac.uk

Abstract— The application of representation learning in knowledge graphs has been a hot topic in recent years. Using representation learning methods, the semantic information contained in knowledge graphs can be embedded into low-dimensional dense vector spaces to achieve the purpose of efficiently processing knowledge graphs. A large number of research results have proved the advantage of the representation learning model represented by the translation model in processing knowledge graph related tasks. However, most translation models focus on the direct relation between entities and ignore the multi-hop relation between entities in the knowledge graph. In this paper, the relation path between entities in the knowledge graph is modeled. Considering the diversity of entities and relations in the knowledge graph, we embed entities and relations into different semantic spaces, and project the embedding results to the same space dynamically, while maintaining the consistency of the relation path between entities. We use benchmark datasets to evaluate the performance of the proposed model on the task of knowledge completion. The experiment shows that the model proposed in this paper is of great significance to solve the problem of knowledge completion in the knowledge graph.

Index Terms—knowledge graph, dynamic projection, relation path

I. INTRODUCTION

The knowledge base is a systematic and structured embodiment of human knowledge and is an important basic technology for intelligent information service applications such as intelligent search, intelligent question answering, and intelligent recommendation. Major search engines and organizations have also established multiple large knowledge bases to serve their products. Common English knowledge bases include Wikipedia, Probase, language knowledge base WordNet [1], and world knowledge base Freebase [2]. Chinese knowledge bases include Baidu Encyclopedia, Sogou Encyclopedia. Knowledge graph is a way to sort out and store information. It was first proposed by Google in 2012. Its essence is a knowledge base of the semantic network. Knowledge graphs have strong semantic expression capabilities, flexible modeling, a human-recognizable, machine-friendly way of expressing knowledge. It's the mainstream form of knowledge base. However, in the form of network representation, people need to design a special graph algorithm to store and utilize the knowledge base, which has the disadvantage of being time-consuming and laborious and is plagued by the problem of data sparseness.

Faced with this problem, representation learning in the field of deep learning has attracted people's attention. Representation learning is to represent the semantic information of the studied object as a low-dimensional dense real-valued vector, and in this space, the two objects with higher semantic similarity are closer. In the field of knowledge graphs, researchers can use representation learning to embed the entities in the knowledge graph and the relations between entities into a low-dimensional dense space, while retaining the semantic relations in the knowledge graph as much as possible. This method can improve the utilization efficiency of graph data and alleviate the problem of data sparseness.

By using representational learning to model the knowledge graph, people can easily achieve the task of completing the knowledge graph and discover the implicit relations among entities to expand the knowledge graph. However, most of the existing models cannot effectively use the multi-hop relation in the knowledge graph, and to some extent, the information hidden in the data is ignored.

This paper explores the application of representation learning in knowledge graphs, focusing on the effects of translation models on knowledge graph completion, and proposes a new model of the relation path between entities in the knowledge graph. The main contributions include:

1) This paper studied and summarized common translation models, and compared the advantages and disadvantages of different models.
2) We combine the construction ideas of PTransE [3] and TransD [4] to model the relation paths in the knowledge graph and proposed a new translation model, PTransD.
3) By evaluating the result of the knowledge completion task with the PTransD model in the benchmark data set, we verify the effectiveness of the model.

The rest of the paper is organized as follows. Related work is presented in Section II. In Section III, we detail our approach. The experiments and results of the proposed model will be introduced in Section IV. The conclusion we draw and feature work is presented in Section V.

65

II. RELATED WORK

In the field of knowledge graphs, translation models using representation learning are mainly used to solve the problems of knowledge representation and reasoning. The translation model mainly learns the structural features of the knowledge graph, namely (head entities, relations, and tail entities) triples, embeds entities and relations into low-dimensional dense spaces, and uses vectors to represent entities and relations. Since the TransE model [5] was proposed in 2013, a series of models have been produced to improve and supplement the TransE model, such as TransH [6], TransR [7], TransD [4], PTransE [3] and so on. This section mainly introduces these models.

A. TransE

In the TransE model, triples in the knowledge graph are denoted by (h, r, t). Correspondingly, their column vector are denoted by $\mathbf{h}, \mathbf{r}$ and $\mathbf{t}$. The mean idea of TransE is that the relation $\mathbf{r}$ is considered as the translation from $\mathbf{h}$ to $\mathbf{t}$. Therefore, the goal of the TransE model is to make $\mathbf{t} - \mathbf{h}$ equal to $\mathbf{r}$ as much as possible. The score function is defined as

$$f_r(\mathbf{h}, \mathbf{t}) = \|\mathbf{h} + \mathbf{r} - \mathbf{t}\|_{L_1/L_2} \tag{1}$$

where L_1/L_2 represents the 1-norm or 2-norm.

However, the TransE model embeds entities and relations into the same space, and for the same relation, different head and tail vectors may be close in distance. Therefore, the TransE model encounters difficulties when dealing with complex relation modeling.

B. TransH

The TransH model [6] overcomes the shortcomings of the TransE model's insufficient processing capacity for complex relations and makes the same entity vector have different representations under different relations.

The TransH model assumes that there is a corresponding hyperplane for each relation r, and the relation r falls on the hyperplane. Each entity can be projected onto the hyperplane where the relation r is located. Then the translation process similar to the TransE model will be performed on this hyperplane.

Let $\mathbf{h}_\perp$ and $\mathbf{t}_\perp$ represent the projected vector of head entity and tail entity respectively. The score function of the TransH model is defined as

$$f_r(\mathbf{h}, \mathbf{t}) = \|\mathbf{h}_\perp + \mathbf{r} - \mathbf{t}_\perp\|_{L_1/L_2} \tag{2}$$

Although the TransH model makes the same entity have different representations through projection under different relations, the model assumes that entities and relations are in the same semantic space, which limits its representation ability to a certain extent.

C. TransR

Both the TransE and TransH models assume that entities and relations are vectors in the same semantic space so that similar entities will be in similar positions in space. The TransR model believes that each entity can have many aspects, and different relations focus on different aspects of the entity, so different relations should have different semantic spaces. For each relation r, a transition matrix $\mathbf{M}_r$ is set. Entity vectors will be projected to the relation space with these matrices. The score function of TransR is

$$f_r(\mathbf{h}, \mathbf{t}) = \|\mathbf{M}_r\mathbf{h} + \mathbf{r} - \mathbf{M}_r\mathbf{t}\|_{L_1/L_2} \tag{3}$$

The TransR model separates the original single semantic space into entity space and relation space, which improves the model's representation ability. However, the transition matrix is only relevant to the relation, and the matrix multiplication increases the complexity of operations.

D. TransD

By using the dynamic mapping matrix, TransD [4] overcomes the above shortcomings of the TransR model to some extent. It uses different mapping methods to project entity vectors to the relation space. Besides the embedding vector, TransD constructs a projection vector for each entity or relation to build the dynamic mapping matrix. When the dimension of entity space and relation space is set to be the same, the score function of TransD can be simplified as

$$f_r(\mathbf{h}, \mathbf{t}) = \left\|\mathbf{h} + \mathbf{r} + (\mathbf{h}_p^T\mathbf{h} - \mathbf{t}_p^T\mathbf{t})\mathbf{r}_p - \mathbf{t}\right\|_{L_1/L_2} \tag{4}$$

where subscript p marks the projection vectors.

E. PTransE

The PTransE model [3] believes that in addition to direct relations in knowledge graphs, indirect relations reached between entities through other entities should also be of great significance for completing completion tasks in the knowledge graph. Therefore, PTransE models the relation paths between entities and gives quantitative calculations for the reliability of different relation paths. Using the relation paths, the PTransE model uses ideas similar to the TransE model to perform semantic relations in the knowledge graph Learn and complete the embedding of the knowledge graph.

PTransE solves two important challenges of using relational paths: i) the reliability calculation of relational paths, and ii) the semantic representation of relational paths. The PTransE model proposes a PCRA (Path-constraint Resource Allocation) algorithm based on the resource allocation algorithm in the network [8] and calculates the reliability of the relational path. The basic idea of the algorithm is that on a subgraph with entity h as the starting point and entity t as the ending point, it is assumed that a certain amount of resources flow out from h through the relation path, and the number of resources that can finally reach t reflects the reliability of the relation path.
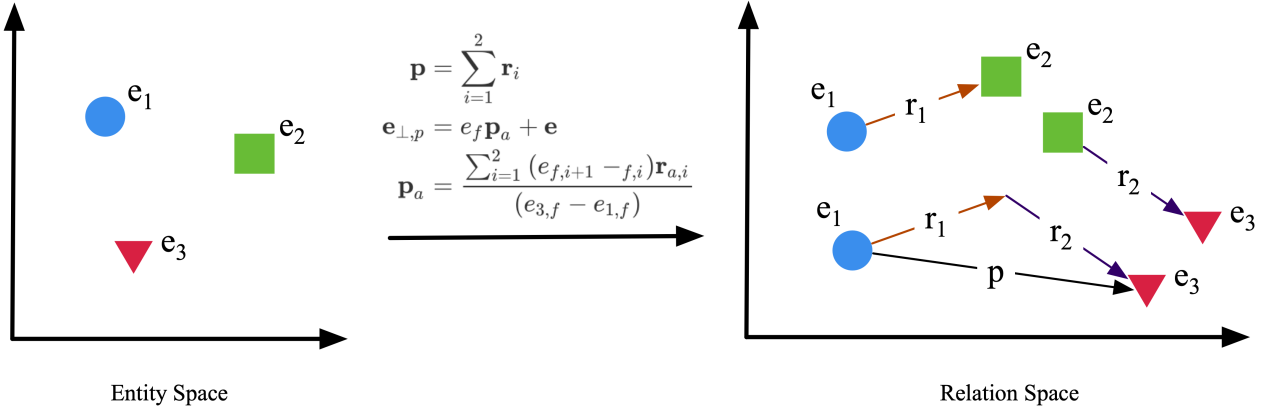
Fig. 1. Simple illustration of PTransD. Each shape represents an entity. There exists a relation path p between entity e_1 and entity e_3, which is denoted as $p = (r_1, r_2)$. With the auxiliary vector $\mathbf{p}_a$ of p, entities corresponding to p are projected from entity space to relation space. Notice that projections of e_2 changes according to its corresponding relation r_1 and r_2.

III. OUR METHOD

In the TransD model, the entities and relations of the knowledge graph are embedded in different semantic spaces, and the entity vector is projected into the relation space by using a projection matrix. In the relation space, the equation $\mathbf{h} + \mathbf{r} = \mathbf{t}$ holds approximately. Since the projection matrices corresponding to the head entity and the tail entity are related to the entity itself and the relation, and the mathematical operation of the projection operation can eliminate the matrix multiplication operation, the TransD model becomes a more advanced model in the translation model.

The TransD model only considers the direct relation between entities when using knowledge graph data. The PTransE model is the first to propose the use of multi-step relations between entities in the knowledge graph. By setting the credibility of the relation path between entities, the PTransE model has also shown its importance in many translation models.

We draw on the advantages of the above two models and propose the PTransD model.

A. Model Description

For knowledge graph G, the semantic space to which the entity is mapped is $\mathbb{E}^k$, the semantic space to which the relation is mapped is $\mathbb{R}^k$, where k represents the spatial dimension. There are entity semantic vectors $\mathbf{h}, \mathbf{t} \in \mathbb{R}^k$ and relation vectors $\mathbf{r} \in \mathbb{R}^k$. To project the entity vector to the semantic space where the relation vector is located, and make use of the information on the structure of the knowledge graph, similar to the TransD model, PTransD model set auxiliary vectors $\mathbf{h}_a, \mathbf{t}_a$, and $\mathbf{r}_a$ for each semantic vector to construct a mapping matrix.

First we consider the case where the length of the relation path is 1, that is, the relation path between the head vector $\mathbf{h}$ and the tail vector $\mathbf{t}$ is the direct relation r between them. We will deduce this to a more general case.

Let $x_f = \mathbf{x}_a^T \mathbf{x}$ be the feature value corresponding to each entity. The PTransD model constructs a mapping matrix under the relation r for each head entity and tail entity by

$$\begin{aligned} \mathbf{M}_{rh} &= \mathbf{r}_a \mathbf{h}_a^T + \mathbf{I} \\ \mathbf{M}_{rt} &= \mathbf{r}_a \mathbf{t}_a^T + \mathbf{I} \end{aligned} \quad (5)$$

where $\mathbf{I}$ represents the identity matrix. Then using a mapping matrix, the projection of the head and tail entities in the semantic space where the relation r is located is:

$$\begin{aligned} \mathbf{h}_{\perp,r} &= \mathbf{M}_{rh}\mathbf{h} = h_f \mathbf{r}_a + \mathbf{h} \\ \mathbf{t}_{\perp,r} &= \mathbf{M}_{rt}\mathbf{t} = h_f \mathbf{r}_a + \mathbf{t} \end{aligned} \quad (6)$$

The goal of PTransD is to make the equation $\mathbf{h}_{\perp,r} + \mathbf{r} = \mathbf{t}_{\perp,r}$ approximately true. When the Equation (5) and Equation (6) holds, we have that

$$\mathbf{r} = (t_f - h_f)\mathbf{r}_a + \mathbf{t} - \mathbf{h} \quad (7)$$

More generally, for the triplets in knowledge graph $(h, r_0, x_1), (x_1, r_1, x_2), \dots, (x_l, r_l, t)$, the relation path from h to t is marked as $p_r = (r_0, r_1, \dots, r_l)$. Let $x_0 = h, x_{l+1} = t$, the entity path from h to t is marked as $p_e = (x_0, x_1, \dots, x_{l+1})$. The vector of relation path is $\mathbf{p} = \mathbf{r}_0 \circ \mathbf{r}_1 \circ \cdots \circ \mathbf{r}_l$, where $\circ$ is a binary operator. In the PTransD model, we use the plus sign of vector as the binary operator. Under these conditions, the relation path vector is

$$\mathbf{p} = \sum_{i=1}^{l} \mathbf{r}_i \quad (8)$$

With Equation (7) and Equation (8), we can infer that

$$\begin{aligned} \mathbf{p} = \sum_{i=1}^{l} \mathbf{r}_i &= \sum_{i=1}^{l} [(x_{f,i+1} - x_{f,i})\mathbf{r}_{a,i} + \mathbf{x}_{i+1} - \mathbf{x}_i] \\ &= \sum_{i=1}^{l} (x_{f,i+1} - x_{f,i})\mathbf{r}_{a,i} + \mathbf{t} - \mathbf{h} \\ &= (t_f - h_f)\mathbf{p}_a + \mathbf{t} - \mathbf{h} \end{aligned} \quad (9)$$

That is

$$\mathbf{p}_a = \frac{1}{(t_f - h_f)} \sum_{i=1}^{l} (x_{f,i+1} - x_{f,i})\mathbf{r}_{a,i} \qquad (10)$$

The general goal of PTransD is to make the equation $\mathbf{h}_{\perp,p} + \mathbf{p} = \mathbf{t}_{\perp,p}$ hold among different entities.

An important idea in the PTransE model is that different relation paths have different degrees of reliability. The PTransD model follows this idea. For a triplet (h, p, t) given by the relation path $p = (r_1, r_2, \ldots, r_l)$, the possible path from h to t is $S_0 \xrightarrow{r_1} S_1 \xrightarrow{r_2} \ldots \xrightarrow{r_l} S_l$, where $S_0 = \{h\}$ and $t \in S_l$. For arbitrary entity $m \in S_i$, the set of the direct predecessor entities in S_{i-1} linked by r_i is noted as $S_{i-1}(\cdot, m)$. For arbitrary entity $n \in S_{i-1}$, the set of the direct successor entities in S_i linked by r_i is noted as $S_i(n, \cdot)$. The reliability degree is calculated with

$$R_p(m) = \sum_{n \in S_{i-1}(\cdot, m)} \frac{1}{|S_i(n, \cdot)|} R_p(n) \qquad (11)$$

Let $R_p(h) = 1$, then $R_p(t)$ will shows the reliability of path p, noted as $R(p|h, t)$. An example for calculating the reliability of the relation path is shown in Fig 2. Then, the score function of PTransD is defined as

$$f_p(\mathbf{h}, \mathbf{t}) = \|\mathbf{h}_{\perp,p} + \mathbf{p} - \mathbf{h}_{\perp,t}\|_{L_1/L_2} \qquad (12)$$

The set of golden path triplets is noted as Δ_p, which means that for any triplet $(h, p, t) \in \Delta_p$, there exists a relation path p from h to t, and the set of corrupt path triplets is $\Delta'_p = \{(h, p, t')|(h, p, t') \notin \Delta_p\} \cup \{(h', p, t)|(h', p, t) \notin \Delta_p\}$. We define the loss function of PTransD as

$$\mathcal{L} = \sum_{\mathbf{h},\mathbf{p},\mathbf{t} \in \Delta_p} \sum_{\mathbf{h'},\mathbf{p'},\mathbf{t'} \in \Delta'_p} C_p[f_p(\mathbf{h}, \mathbf{t}) - f_{p'}(\mathbf{h'}, \mathbf{t'}) + \gamma]_+ \qquad (13)$$

where C_p is the confidence of relation path p. Let $P(h, t)$ be the set of all relation path from h to t, C_p is calculated by

$$C_p = \frac{R(p|h, t)}{\sum_{p \in P(h,t)} R(p|h, t)} \qquad (14)$$

B. Relations among PTransD, TransD, and PTransE

It can be seen from the construction process of the PTransD model that when the length of the relation path in the PTransD model is limited to 1 and the reliability of the relation path is ignored, the PTransD model degenerates into a special form of TransD. At this time, the dimensions of the entity vector space and the relation vector space are the same in the TransD model.

Compared with the PTransE model, the PTransD model believes that the direct relation between entities is also an embodiment of the relation path. When calculating the model loss, the direct relation and the relation path can be treated the same, thereby simplifying the form of the loss function. The

PTransD model retains the calculation ideas for path reliability proposed in the PTransE model and optimizes it. Besides, the PTransE model embeds entities and relations into the same space, while the PTransD model refers to the idea of the TransD model to embed entities and relations into different spaces, and complete the projection of the entity vector into the relation space dynamically. Thereby, the semantic structure of the knowledge graph can be modeled more clearly.

C. Knowledge Graph Completion

Although a common knowledge graph may have millions of entities and hundreds of millions of relations, these graphs may still be relatively sparse. Knowledge graph completion is to discover new information through the existing knowledge graph. According to the different objects in the triad of knowledge graph to be completed, the completion task of the knowledge graph is divided into three sub-tasks of head entity completion, relation completion, and tail entity completion. Head entity completion refers to when giving the relation and tail entity in the triple, we need to give head entities that can form reasonable triples with them. For example, give the relation "state of" and the tail entity "U.S.A", the possible head entity of the condition can be "California" or "Texas". The relation completion and tail entity completion are the same. It can be seen that in the completion task, the entity or relation that can constitute a triple is not unique.

It is not difficult to use the trained PTransD model for knowledge graph completion tasks. Taking the tail entity completion task as an example, for a triplet $T = (h, r, ?)$ whose tail entity is missing, we need to find a suitable tail entity t so that T becomes a valid triplet. Let $e_1, e_2, \cdots, e_n$ represent the n entities of the knowledge graph, and put them one by one into the missing position of T to form candidate triplets, denoted as $T_i = (h, r, e_i)$. Through the scoring function (12), the PTransD model gives the scores of the n candidate triples. After that, the top k candidate triples with the highest scores is taken as the model completion results, where k is a manually set parameter.

IV. EXPERIMENTS AND RESULTS ANALYSIS

In this section, we illustrate our experimental result of knowledge graph completion with the PtransD model.

A. Dataset

For the experimental data set, the public data set FB15k for validation of the translation model effect. The FB15k dataset is the largest commonly used single-language knowledge graph studied in recent years [3] [4] [5] [6] [7]. It contains about 15,000 entities from Freebase [2] and related triples. The detailed statistics of this data set are shown in Tab. I.

TABLE I
FB15K DATA SET STATISTICS

# Relation	# Entity	# Train	# Valid	# Test
1345	14951	483142	50000	59071
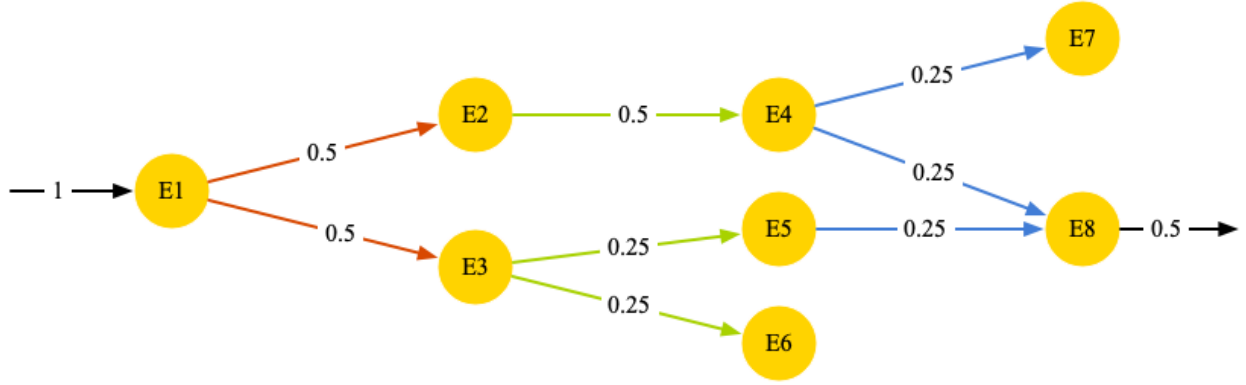
Fig. 2. An example for calculating relation path's reliability. Entities are represented by yellow dots and numbered from 1 to 8. Three kinds of relation r_1, r_2 and r_3 are displayed with red, green and blue arrows, respectively. When calculating the reliability of the relation path $p = (r_1, r_2, r_3)$ between E_1 and E_8, we assign 1 unit of resource to E_1 and make the resource flow along the graph. Finally, we'll obtain 0.5 unit of resource at E_2, which means the reliability of p is 0.5.

B. Experiments

In the experiment, the FB15k dataset is selected to evaluate the PTransD model on the knowledge completion task. The evaluation of the model is completed in three sub-tasks: i) head entity completion, ii) relation completion, and iii) tail entity completion. For each sub-task, the results of three indicators, Mean Rank, MRR [9], and Hits@10, are given. Mean Rank is the mean of correct entities' or relations' rank among the completion result. The MRR indicator calculates the mean value of ranks' reciprocal. Hits@10 indicates the proportion of valid entities or relations ranked in the top-10. In addition, we also calculate the averages of the head entity completion and tail entity completion on these three indicators, as the model's overall completion evaluation criteria. Literature [3] and [4] summarize the results of common existing model evaluations, and this paper refers to some of the results as a comparison of this model.

When training the PTransD model, we used the mini-batch SGD optimizer to optimize the loss function (13). The 2-norm of the entity vector, the relationship vector, and each projection vector is also limited to 1 to prevent the model from obtaining a trivial solution by increasing the norm. All the parameters of the PTransD model are initialized randomly. Corrupt triplet used for training is generated by replacing the head entity or the tail entity of the relation path. We found that there is a large number of relations among entities. In the dataset FB15k, the number of 2-hop relation paths is more than one hundred million while the 3-hop relation path's amount is more than 60 times more. It's very time consuming to consider all the relation paths between certain two entities. To reduce the amount of calculation, we limited the maximum length of the relation path to 2. The confidence of each path was calculated through the train set before training. We also discarded the path whose reliability is less than 0.01 to speed up the training process.

The PTransD model was verified on FB15k using multiple parameters. The batch size was fixed to 4800 and the

maximum training epoch was set to 500. The average loss of model's completion on the validation set of FB15k was used as the basis of the early stop. The best result occurs when setting the margin to 1, the embedding dimension to 100, and the learning rate to 0.01, see Tab. II for details. Tab. III shows the detailed evaluation results of our model together with other models in FB15k.

TABLE II
EVALUATION RESULS OF PTRANSD IN FB15K

Task	Mean Rank	MRR	Hits@10%
head	190	0.266	50.4
relation	3	0.773	96.9
tail	189	0.271	54.6
total	189	0.268	52.5

TABLE III
EVALUATION RESULTS IN FB15K

Model	Mean Rank	Hits@10%
TransE [5]	243	34.9
TransH(bern) [6]	212	45.6
TransR(bern) [7]	198	48.2
TransD(bern) [4]	194	**53.4**
PTransE(ADD,2-STEP) [3]	200	51.8
PTransD(our)	**189**	52.5

Compared with common existing models, the PTransD model in the FB15k data set has a lower Mean Rank than TransD, and similar to the optimal result among these models on Hits@10, which shows that PTransD is effective in considering multi-step relation and embedding schemes for entities and relations. However, because the PTransD model can learn a richer relation between two entities from the relation path, its performance in accurately predicting missing entities or relations is lower than that of the TransD model, so its performance is slightly insufficient on the Hit@10 indicator.

V. CONCLUSION AND FUTURE WORK

This paper explores the application of representation learning in knowledge graphs, focusing on the effects of translation models on knowledge graph completion. We introduce common translation models and compares the advantages and disadvantages of different models. Inspired by the construction ideas of the PTransE model and TransD, we proposed a translation model PTransD based on relational paths and dynamic projection. The PTransD model is a generalization of the TransD model. When the relation path length is limited to 1 and the reliability of the relation path is not taken into consideration,, the PTransD model is converted into a form in which the TransD model has the same dimensions in the entity space and relation space. Compared with the PTransE model, the proposed model embeds entities and relationships into different spaces, and projects entities into the relationship space dynamically according to the relations and entities themselves, thereby making the model more expressive. We verified the completion effect of the PTransD model in the benchmark dataset FB15k. The result indicates the rationality of PTransD for multi-step relations.

There is still much work to do for further research. At present, our work is based on the Trans series of models. These models are based on the structure and learning of the knowledge graph, but the semantic information in the knowledge graph is not only included in the structure, but also the text itself. Our follow-up work will focus on how to use the information in the knowledge graph to complete the knowledge graph completion problem.

REFERENCES

[1] George A. Miller. Wordnet: A lexical database for english. *Commun. ACM*, 38(11):39–41, 1995.

[2] K. Bollacker, C. Evans, P. Paritosh, T. Sturge, and J. Taylor. Freebase: A collaboratively created graph database for structuring human knowledge. In *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*, page 1247–1250, 2008.

[3] Y. Lin, Z. Liu, H. Luan, M. Sun, S. Rao, and S. Liu. Modeling relation paths for representation learning of knowledge bases. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 705–714, 2015.

[4] G. Ji, S. He, L. Xu, K. Liu, and J. Zhao. Knowledge graph embedding via dynamic mapping matrix. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 687–696, 2015.

[5] A. Bordes, N. Usunier, A. Garcia-Durán, J. Weston, and O. Yakhnenko. Translating embeddings for modeling multi-relational data. In *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 2*, page 2787–2795, 2013.

[6] Z. Wang, J. Zhang, J. Feng, and Z. Chen. Knowledge graph embedding by translating on hyperplanes. 2014.

[7] Y. Lin, Z. Liu, M. Sun, Y. Liu, and X. Zhu. Learning entity and relation embeddings for knowledge graph completion. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, page 2181–2187, 2015.

[8] Linyuan Lü and Tao Zhou. Link prediction in complex networks: A survey. *Physica A: statistical mechanics and its applications*, 390:1150–1170, 2011.

[9] N. Craswell. *Mean Reciprocal Rank*, pages 1703–1703. 2009.

Modeling HiBrinto Ontology to Develop Knowledge Management Portal for Highway Bridge Construction

Shanmuganathan Vasanthapriyan
Department of Computing and Information Systems,
Sabaragamuwa University of Sri Lanka,
Belihuloya, Sri Lanka
priyan@appsc.sab.ac.lk

Kuhaneswaran Banujan
Department of Computing and Information Systems,
Sabaragamuwa University of Sri Lanka,
Belihuloya, Sri Lanka
bhakuha@appsc.sab.ac.lk

Abstract—**Highway bridges play an important role in traffic reduction in main cities. Constructing, maintaining and repairing highway bridges is a knowledge-intensive activity that inputs knowledge from different sources to take timely and correct decisions. Since these are knowledge-intensive activities, the proper knowledge management (KM) system can help to make the proper decisions, knowledge sharing, knowledge capturing, etc. easier. The base for the KM system is the ontology which is an explicit specification of conceptualization. In this paper, we introduce a novel HiBrinto, a highway bridge ontology for developing a KM portal for highway bridge construction, maintenance, and repair. The field study of the KM portal clearly emphasizes that the developed KM portal will pave the way for seamless knowledge sharing and quick decision making for highway bridge-related activities.**

Keywords-Highway Bridge, Knowledge, Management, Ontology, HiBrinto, Semantic Web.

I. INTRODUCTION

Bridges are part of the road infrastructure and are an important link in a road network [1]. In this context, highway bridges play an important role in reducing traffic. Highway bridges are made of steel, reinforced concrete, or wood. Girders are most often used in building steel highway bridges. The preferred design of a highway bridge is with driving in the upper part (Deck Bridge); this creates better traffic conditions for automobiles and makes the bridge easier to maintain [2].

The Semantic Web is an extension of the current web to allow computers and persons to share information (not data) based on context (not hypertext). Ontology is one of the core web features [3]. Knowledge Management (KM) is the process of making the effective use of information and knowledge in an organization to achieve the goals [4]. It allows people to access and use the best knowledge when it is necessary and facilitates learning [5, 6].[7].

For the creation of information systems, the development, distribution, and use of common communication principles, vocabulary and ontologies are essential. Therefore, we discuss certain key topics in this paper relevant to our research such as highway bridges and ontologies [7]. The overall objective of our research is to develop a KM portal that can cater to the needs of people who are related to highway bridge construction. The key to achieving this objective is the ontology which we named as HiBrinto (**Hi**ghway + **Br**idge + **Onto**logy)

The remainder of the paper is organized as follows. Section 2 discusses the literature review and related works. Section 3 describes the complete research design for the HiBrinto ontology. Finally, Section 4 concludes this paper with directions for future work.

II. LITERATURE REVIEW AND RELATED WORKS

Construction companies engaged in more routine and repeatable work should also strive to make more effective output of products, uniform procedures, preparation and so on [8]. Information needs to be quickly and easily communicated and beneficial to others. Project teams and people within a company should be empowered and be able to share their experiences with others [3, 9].

Three areas of consideration that must be addressed for the effective construction of highway bridges are; (i) artistic and esthetic, (ii) analytical and (iii) scientific and realistic. Given that most bridge projects currently being undertaken by multidisciplinary teams, it is fairly easy to address the first two concerns. The last one is often the most complicated. [2]. Project periods for these bridges will have to shorten to respond to public demand to minimize road congestion and the flow of traffic during project, often within a few feet of workers and equipment [8].

In, "Building an ontological knowledgebase for bridge maintenance" by Ren, et al.[7], they have developed a system that caters to all the phases in the bridge maintenance life cycle. It covers the maintenance-related knowledge for all types of bridges. In the work of J France-Mensah, et al. [10], they developed Integrated Highway Planning Ontology (IHP-Onto) which is a shared representation of knowledge about pavement assets, M&R planning, and inter-project coordination. This is work is done in high ways.

III. RESEARCH DESIGN

Our research design consists of two major parts; (i) Modeling of HiBrinto ontology, (ii) Development of Highway Bridge KM Portal. The high-level methodological roadmap for the development of the KM portal for highway bridge decision making is shown in Figure 1.
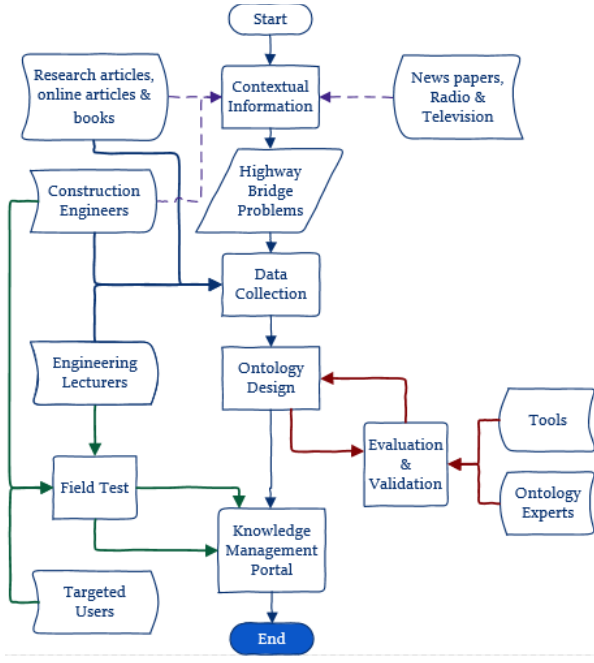
Figure 1. The high-level methodological roadmap for the development of the KM portal for highway bridge decision making

A. Ontology Modeling

In this section, we present how the HiBrinto ontology is developed which is going to act as the base for KM Portal development in the next stage.

1) Data collection

Context refers to a representative item that enables the external environment of a concept to be represented [11]. If any data that gives information context to a person, entity or event is known as contextual information. In our research, the targeted users are; construction engineers, maintenance engineers, quality engineers, RDA management, academic staff, researchers, engineering undergraduates.

The data collection was done by using grounded theory [12]. Grounded theory is a technique that involves building hypotheses through methodical data collection and analysis. In comparison to the hypothetical-deductive model of the scientific method, this approach employs inductive reasoning. Research with the grounded theory is likely to start with a query or even with qualitative data collection [13].

Since the researchers' are not experts in the construction domain, the relevant data were obtained through formal and informal expert collaboration and extensive literature surveys. Five construction engineers who mainly deal with the highway bridge construction over many years under different projects and three academic lecturers from the reputed university of Sri Lanka participated in the process of data collection. They have been interviewed formally and informally several times during the whole research period. Besides, thirty personalities from the construction site with different job roles and thirty academic students also participated in the process of identifying the problems faced by them (i.e., the actual need for the research is realized by them). Further, construction manuals [14-17] and

several works of literature [3, 8, 18-24] were surveyed throughout the research period.

2) Competency Questions

The Competency Questions (CQs) are questions of natural language which define and limit the scope of knowledge that is represented in the ontology. [6]. CQs play a major role in the lifecycle of ontology modeling as they reflect the ontology requirements [25]. CQs work as a requirement's specification of the HiBrinto ontology. Table 1 shows part of the formulated CQs

TABLE I.   COMPETENCY QUESTIONS

Competency questions for the HiBrinto ontology
What are the different types of highway bridges available?
What are the major components of a bridge?
What are the special components of a particular type of highway bridge?
Which are quality checks used to check the quality of a particular type of highway bridge?
What are the activities conducted under each quality check?
In which duration each quality check procedures should be applied?
Which maintenance techniques can be the most suitable for a particular type of?
What are the environmental concerns to be considered in the highway bridge management process?
What are the remedies/actions to be taken if the quality check results fail?

3) Taxonomy Development

An extensive taxonomy had to be developed as a common platform for interacting ontologies. Taxonomic relations ("is-a", "is-part-of", "is-kind-of", "is-a-type-of") allow any sub-concept to inherit the characteristics of its super-concept [3]. Computers may derive new knowledge from existing knowledge by using taxonomies [22].

At the end of the data collection, the taxonomy for HiBrinto was created. Eleven super-classes were identified; Activities, Actors, HighwayBridges, BridgeComponent, Divisions, Equipment, Materials, ProsCons, Maintenance, Construction, and Parameters. The sub-classes for the superclasses are also identified. For example, "EnvironmentalConcerns", "QualityCheck", "Duration", "MaintenanceTechniques", "RepairTechnique", "ManagementPractices" are sub-class of "Maintenance" class. Further, the "QualityCheck" class contains "QCCode" and "QCItem" as its sub-class. The high-level class hierarchy of HiBrinto ontology is shown in figure 2.

Figure 2. The high-level class hierarchy of HiBrinto ontology

A glossary of about 3500 terms was compiled from well-established sources, like building manuals, textbooks, research papers, and informal expert interviews. It is not feasible to list all taxonomy here. Some of the major domains are explained below.

a) Bridge Components

Every bridge will have some basic components known as "basic-components" such as "Superstructure", "Bearings" and "Substructure". Each basic components will have some sub-components. For example, "Superstructure" will have "Bearing", "Parapet wall", "Flooring" etc. Further, some bridges will have specific components. For example, "Arch" which is a sub-component of "Superstructure" is used for arch bridge construction and "cable" which is a sub-component of "Superstructure" is used for suspension, cable-stayed bridge, etc. [16].

b) Actors

There are many actors in the scene of bridge construction. We broadly categorize them into Executive (Engineering and professional), Semi Executive / Officer (Administration, Finance, Developing program/ Duty concerning, Implementation), and so on [26].

c) Divisions

Each actor identified is assigned to one or more "division". Some of the divisions identified are Planning Division, Engineering Service Division, and Maintenance Management & Construction Division, etc. [26].

d) Quality checks

Each bridge has to undergo some quality checks in certain durations. Some of the quality checks identified are; tension check for beam, bearing pad durability, foundation settlement check, deck roughness, etc. If the results quality check fails, a proper repair plan will be executed according to the severity case. Each quality check has a quality check code and quality check item.

4) Ontology Modeling

Modeling the ontology manually is a complex and time-consuming task [27]. According to Vasanthapriyan [28], the principles, methods, and tools for initiating, developing and maintaining ontologies are investigated in the ontology engineering approach. There are many different methodologies proposed to model the ontologies in many works of literature [29-32]. After reviewing all, we selected Grüninger and Fox's methodology [30] for our work as it publishes a formal approach for designing the ontology and also it provides a framework for evaluating the developed ontology [33]. Grüninger and Fox's methodology [30] focuses on building ontology-based on first-order logic by providing strong semantics.

a) Classes

An ontology is a systematic definition of the architecture. The concept is defined by classes and relationships. The classes comprise category; subclass, superclass, intersection class, union class, and complement class [34].

The taxonomies identified were converted into classes. During the modeling of HiBrinto ontology, some special types of axioms such as Instantiation, Assertion, Subsumption, Domain, Range, and Disjointness are included. The classes have been created in Protégé OWL Ontology Editor 5.5. Figure 3 shows the part of high-level classes modeled using Protégé OWL Ontology Editor 5.5.
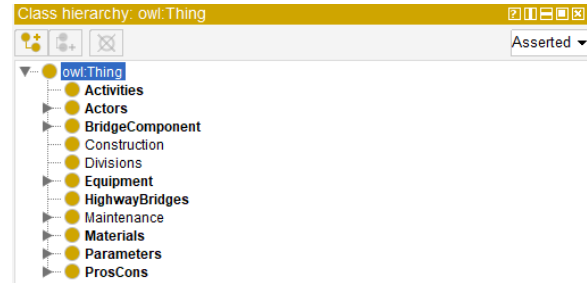
Figure 3. Part of high-level classes modeled using Protégé OWL Ontology Editor 5.5

b) Object properties & Datatype properties

The associative relationships (object properties) are to identify the concepts and relationships with meaningful relations and to define the relationships and their inverse relationships. For example, "BridgeComponent" isComponentOf, "HighwayBridges". The inverse isComponentOf of is hasComponent. Another example is "QualityCheck" hasDuration "Duration".

A datatype property is defined as an instance of the built-in OWL class owl:DatatypeProperty. The needed datatype properties are also defined such as qualityCheckStatus, hasDuration, hasLength, hasMinWorkers, hasMinSiteEngineer, and hasMinSurveyor, etc.

c) Individuals

Various individual instances were added to the class hierarchy. To use the reasoner to test our rules, we instantiated 3,857 individuals. Individuals which we created include bridges, activities, equipment, actors and so on. To create them, we used names such as bridge101, check_sealability, driller, environmental_officer and so forth

d) Axioms

These major concepts (classes and sub-classes) and relationships (properties) are also bound by some axioms. A set of axioms have been also developed. The following sample process illustrates some of the axioms used quality check.

- Each highway bridge has to conform highway bridge quality check

$$\forall HighwayBridges \exists Qualitycheck$$
$$\supset conforms(HighwayBridges, QualityCheck)$$

- Each highway bridge quality check has a quality check safety standard

$$\forall QualityCheck \exists SafetyStandard$$
$$\supset has(QualityCheck, SafetyStandard)$$

- Each highway bridge quality check has a checking method that is either destructive or nondestructive method

$$\forall QualityCheck \; \exists Method$$
$$\in (desctructive, nondestructive)$$
$$\supset has(QualityCheck, Method)$$

- Each highway bridge quality check has quality check code (QCCode) and quality check item (QCItem)

$$\forall QualityCheck \; \exists (QCCode \wedge QCItem)$$
$$\supset has(QualityCheck, QCCode, QCItem)$$

e) DL Queries

Since we were designing with OWL 2 Web Ontology Language [32] for the semantic web, we used Description Logic (DL) which is a decidable fragment of FOL for our scenario. We have evaluated the competency questions to see whether the ontology meets the users' requirements during the internal design process. The DL expressions have been used to query the ontology. For this purpose, we used the DL query facility which is available in Protégé-OWL Ontology Editor 5.5.

B. KM Portal Development

Ontology and semantic web systems have strong logic capabilities [35]. This segment discusses the construction of a knowledge platform to share knowledge regarding highway bridges. It was developed on the distributed system framework of Java J2EE. The five layers of our knowledge framework are; Ontology, Experience Sharing and Knowledge Validation, Storage, Reasoning, and Knowledge Sharing Layer and they are shown in Figure 4.
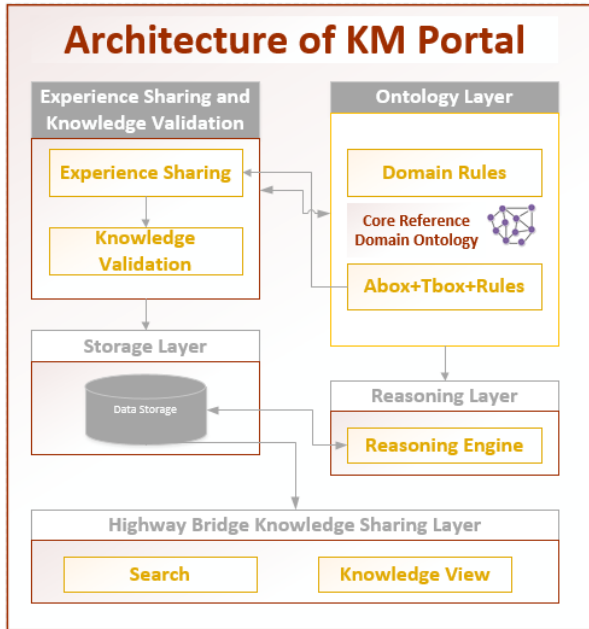
Figure 4. The architecture of KM Portal

1) Ontology Layer

Our developed HiBrinto ontology including its domain rules, axioms, etc. is in the ontology layer. Using the Protégé-OWL

Ontology Editor 5.5, these concepts and their relationships were partly described in section "Ontology Modeling".

2) Experience Sharing and Knowledge Validation Layer

Through the Experience Sharing layer, the construction engineers can annotate their highway bridge knowledge with the support of the construction standard terms. The shared knowledge is then transformed into the semantic data in a machine-understandable format of the triple structure by the semantic data generator.

3) Storage Layer

We used Triple-store, which stores RDF triples. Using SPARQL the queries were made. Since Jena TDB is a component of Jena for RDF storage and query, it was selected in this study. It supports the full range of Jena APIs and can be used as a high performance of the RDF store on a single machine.

4) Reasoning Layer

Highway bridge rules were generated with Protégé-SWRL Editor. It is a plugin in the Protégé-OWL Ontology Editor 5.5 environment. It supports the Jess Rule Engine. The Semantic Web Rule Language (SWRL) is based on a combination of OWL with the Rule Markup Language. It provides inference capabilities from existing OWL ontology.

5) HiBrinto Knowledge Sharing Layer

Knowledge Sharing Layer includes two functionalities that use Semantic Web technologies: (1) basic search, and (2) Advanced Search. SPARQL has been used as the query language to retrieve highway bridge knowledge from the semantic data storage. The basic search provides a simple triple pattern matching service, which is one of the most frequently used functions for searching documents in the Semantic Web. Besides, Advanced Search Option includes, logical operators (AND or NOT or OR), so that users can combine different options to retrieve knowledge.

C. Evaluation of HiBrinto and KM Portal

The quality of the ontology is very much important for its usefulness. To avoid the defects when using the ontology, its quality should be verified and validated. We verified and validated our ontology in different ways; (i) OOPS! - Online ontology evaluator, (ii) Reasoner – Inbuilt tool in Protégé OWL Ontology Editor 5.5, (iii) Ontology experts. We did not incorporate domain experts fully in this phase because the ontology can be understood by who is having computer science knowledge. But whenever issues were identified they were contacted to clarify the issues. The developed KM portal was evaluated using field tests.

1) Tools

a) OOPS!

OOPS! is a web-based method for detecting possible mistakes that could lead to modeling errors, independent of any context for ontology development. This method is intended to support ontology developers in the ontology validation process, which can be separated into diagnostics and repairs. OOPS! helps to identify some of the most common pitfalls in ontological developments OOPS, for example

- Warns of when: the domain or range of a connection is described as a two or more class intersection. In case such classes could not exchange cases, this alert may deter thinking issues.

- No naming convention is used in the ontology element identifiers. In this situation, maintenance, usability, and ontology consistency could be enhanced.

- In ontology, a loop between two classes is included in the hierarchy. The identification of this condition may avoid problems with modeling and reasoning.

Table 2 describes the part of the pitfalls identified for the modeled ontology, description and solution proposed. Three layers existed, including critical, important and minor. The critical degree is very vital and must be fixed to prevent ambiguity in the ontology. Both minor and important instances have been updated to render ontology better.

TABLE II.   PITFALL DESCRIPTION AND SOLUTION PROPOSED

Pitfall	Description	Solution
Missing annotations (4850 cases | Minor)	Creating an ontology element without providing understandable annotations to it.	Included the ontology annotations
Missing domain or range in properties (65 cases | Important)	Object and (or) datatype properties without domain or range	Added the missing domain and range
Inverse relationships not explicitly declared (186 cases | Minor)	Except for the symmetric properties, others do not have an inverse relationship.	Included missing inverse relationships
Defining multiple domains or ranges in properties (6 cases | Critical)	More than one domain or range is defined for a property.	Modified the multiple domains and ranges

b) Reasoner

A semantic reasoner (also known as reasoning engine, rules engine, or simply a reasoner), is a software able to infer logical consequences in the modeled ontology from a set of asserted facts or axioms. We utilized the FaCT++ inbuilt reasoner tool available in the Protégé OWL Ontology Editor 5.5. According to Tsarkov and Horrocks [36], "FaCT++ is a new sound and complete DL reasoner designed as a platform for experimenting with new tableaux algorithms and optimization techniques".

2) Ontology Experts

With the support of two ontology experts we tested the ontology with the deficiencies of the artifacts we used. The expert is not an author and is not associated with our research team. Several approaches for testing ontologies were present in the literature. Our ontology experts considered (a) syntax (b) structure, (c) semantics, (d) terminology, (e) meaning and (f) representation to conduct the assessment. The primary goals of the expert evaluation are: (a) whether the HiBrinto ontology meets the requirements, norms, (b) coverage of the Highway Bridge and (c) internal quality control. The remarks of the ontology experts have also been revised.

Some of the comments given by the experts are; Manchester syntax was followed, all concepts follow is-a relationships, the whole ontology was viewed using OntoGraph, thirty-two concepts and eight object properties do not have understandable names, very few CQs were needed to be modified as highlighted and so on.

3) Field Test

One of the most important tools to determine the validity of the suggested ontology was the actual implementation and testing with the end-users. Different categories of 20 end users were selected for this purpose. First, a training session was carried out with system end-users. The end-users were given a brief introduction to the project and what is expected from them. Then the demonstration for using the system was done. Finally, they were allowed to use the system. Since we hosted the system in the localhost, end-users were allowed to use the system in a restricted environment. Proper facilities were made for their comfort. The end-users were allowed to use the system for 4 hours.

Then, a survey was conducted which consists of a set of questions to check whether developed ontology was able to (i) express highway bridge knowledge (ii) support highway bridge knowledge sharing (iii) support highway bridge knowledge retrieval and (iv) user satisfaction. The survey uses a Likert scale of 1 to 5, with 1 (worst) and 5 (best). Their assessment was largely positive. More than two-thirds of the end-users participated in the survey responded with 4, and 5 ratings. Some of the questions asked and Mode (Likert scale) is shown in the Table 3

TABLE III.   QUESTIONS ASKED AND MODE (LIKERT SCALE)

Question	Mode (Likert scale)
How easy was it to navigate in the system?	5
How representative are the terms used?	4
Can the system be used by the persons who do not know the highway bridge construction field?	5
How did the system responsible for your search?	4
Are you satisfied with the system?	4

IV.   CONCLUSIONS AND FUTURE WORKS

The otology has been utilized in many types of research in different domains such as agriculture, medical, dental, software testing, economics, etc. But a very few researches have been done in the construction domain [3, 22]. But none of the researches has been conducted in the highway bridge domain. In this paper, we presented the HiBrinto ontology to represent highway bridge domain knowledge which includes highway bridge concepts, properties, and their relationships that can be used to help decision making for the whole lifecycle of the highway bridge. The full version of our ontology has 427 entities, 726 properties which include both object properties and datatype properties, and 3,857 individuals.

Since our focus in this research was on highway bridges, the HiBrinto ontology can be expanded further for the other types of bridges as well. Further, reasoning engine with Query-enhanced Web Rule Language (SQWRL) can be incorporated into

HiBrinto knowledge searching to support more accurate and effective knowledge sharing.

ACKNOWLEDGMENT

REFERENCES

[1] Chhim Phalla, You Dara, Sitthy Panhavuth, Nin Menakak, Eam Sovisoth, Long Davuth, et al., "Bridge Inspection Manual," R. I. Department, Ed., ed: Ministry of Public Works and Transport, 2018.

[2] A. Pipinato, Innovative bridge design handbook: Construction, rehabilitation and maintenance: Butterworth-Heinemann, 2015.

[3] T. E. El-Diraby and K. Kashif, "Distributed ontology architecture for knowledge management in highway construction," Journal of Construction Engineering and Management, vol. 131, pp. 591-603, 2005.

[4] R. Farooq, "A conceptual model of knowledge sharing," International Journal of Innovation Science, vol. 10, pp. 238-260, 2018.

[5] H. S. Robinson, P. M. Carrillo, C. J. Anumba, and A. M. Al‐Ghassani, "Knowledge management practices in large construction organisations," Engineering, Construction and Architectural Management, 2005.

[6] D. Wiśniewski, J. Potoniec, A. Ławrynowicz, and C. M. Keet, "Analysis of Ontology Competency Questions and their formalizations in SPARQL-OWL," Journal of Web Semantics, vol. 59, p. 100534, 2019.

[7] G. Ren, R. Ding, and H. Li, "Building an ontological knowledgebase for bridge maintenance," Advances in Engineering Software, vol. 130, pp. 24-40, 2019.

[8] C. Rossimel and J. Wong, "Accelerated bridge construction for level crossing removal in a high-traffic metropolitan environment," in Australian Small Bridges Conference, 9th, 2019, Surfers Paradise, Queensland, Australia, 2019.

[9] J. H. A. Peeters, "Composite bridge deck and bridge construction," ed: Google Patents, 2019.

[10] J. France-Mensah and W. J. O'Brien, "A shared ontology for integrated highway planning," Advanced Engineering Informatics, vol. 41, p. 100929, 2019.

[11] A. Chebba, T. Bouabana-Tebibel, and S. H. Rubin, "Context in ontology for knowledge representation," in Advanced Computational Methods for Knowledge Engineering, ed: Springer, 2015, pp. 311-320.

[12] A. Strauss and J. Corbin, "Grounded theory methodology," Handbook of qualitative research, vol. 17, pp. 273-85, 1994.

[13] J. Mills, A. Bonner, and K. Francis, "The development of constructivist grounded theory," International journal of qualitative methods, vol. 5, pp. 25-35, 2006.

[14] H. E. Wahls, Design and construction of bridge approaches vol. 159: Transportation Research Board, 1990.

[15] M. J. Ryall, N. Hewson, G. Parke, and J. Harding, The manual of bridge engineering: Thomas Telford, 2000.

[16] W.-F. Chen and L. Duan, Bridge engineering handbook: construction and maintenance: CRC press, 2014.

[17] S. Alampalli and W. J. Moreau, Inspection, evaluation and maintenance of suspension bridges: CRC Press, 2015.

[18] C. Middleton and P. Thoft-Christensen, "Assessment of the reliability of concrete bridges," 1997.

[19] A. H. Cooper and J. M. Saunders, "Road and bridge construction across gypsum karst in England," Engineering Geology, vol. 65, pp. 217-223, 2002.

[20] S.-I. Yang, D. M. Frangopol, and L. C. Neves, "Service life prediction of structural systems using lifetime functions with emphasis on bridges," Reliability Engineering & System Safety, vol. 86, pp. 39-51, 2004.

[21] C. Zhou and W. Wang, "Highway bridge construction process simulation base on 4D visualization," in Asphalt Material Characterization, Accelerated Testing, and Highway Management: Selected Papers from the 2009 GeoHunan International Conference, 2009, pp. 138-145.

[22] A. Benevolenskiy, Ontology-based modeling and configuration of construction processes using process patterns: Institut für Bauinformatik, Fakultät Bauingenieurwesen, TU Dresden, 2016.

[23] R. M. Choudhry, "Risk Analysis Related to Cost and Schedule for a Bridge Construction Project," in Perspectives on Risk, Assessment and Management Paradigms, ed: IntechOpen, 2019.

[24] G. Morcous, M. Maguire, and M. K. Tadros, "High Performance Materials for Concrete Bridge Construction," in International Congress and Exhibition" Sustainable Civil Infrastructures", 2019, pp. 48-63.

[25] C. Bezerra, F. Freitas, and F. Santana, "Evaluating ontologies with competency questions," in 2013 IEEE/WIC/ACM International Joint Conferences on Web Intelligence (WI) and Intelligent Agent Technologies (IAT), 2013, pp. 284-285.

[26] J. I. C. A. (JICA), "Data collection survey on primary bridges on national roads and maintenance system of bridges," 2013.

[27] A. I. Walisadeera, A. Ginige, and G. N. Wikramanayake, "Conceptualizing crop life cycle events to create a user centered ontology for farmers," in International Conference on Computational Science and Its Applications, 2014, pp. 791-806.

[28] S. Vasanthapriyan, J. Tian, and J. Xiang, "An Ontology-Based Knowledge Framework for Software Testing," in International Symposium on Knowledge and Systems Sciences, 2017, pp. 212-226.

[29] M. Fernández-López, "Overview of methodologies for building ontologies," 1999.

[30] M. Grüninger and M. S. Fox, "Methodology for the design and evaluation of ontologies," 1995.

[31] N. F. Noy and D. L. McGuinness, "Ontology development 101: A guide to creating your first ontology," ed: Stanford knowledge systems laboratory technical report KSL-01-05 and Stanford medical informatics technical report SMI-2001-0880, Stanford, CA, 2001.

[32] Y. Sure, S. Staab, and R. Studer, "On-to-knowledge methodology (OTKM)," in Handbook on ontologies, ed: Springer, 2004, pp. 117-132.

[33] S. Vasanthapriyan and K. Banujan, "An Ontological Approach for Dental Extraction Decision Making and Knowledge Dissemination–A Pilot Study for Dental Extraction Forceps," 2019.

[34] A. T. Choksi and D. C. Jinwala, "A novel way to relate ontology classes," The Scientific World Journal, vol. 2015, 2015.

[35] S. Vasanthapriyan, J. Tian, D. Zhao, S. Xiong, and J. Xiang, "An ontology-based knowledge management system for software testing," in The Twenty-Ninth International Conference on Software Engineering and Knowledge Engineering (SEKE), 2017, pp. 522-525.

[36] D. Tsarkov and I. Horrocks, "FaCT++ description logic reasoner: System description," in International joint conference on automated reasoning, 2006, pp. 292-297.D. Tsarkov and I. Horrocks, "FaCT++ description logic reasoner: System description," in International joint conference on automated reasoning, 2006, pp. 292-297.

An Extended Knowledge Representation Learning Approach for Context-based Traceability Link Recovery

Guoshuai Zhao, Tong Li*, Zhen Yang
Beijing University of Technology
zhaogs23@foxmail.com, litong@bjut.edu.cn, yangzhen@bjut.edu.cn

Abstract—Software artifact traceability is widely recognized as an essential factor for effectively managing the development and evolution of software systems. However, such traceability links are usually missed in practice due to the time pressure. Although an increasing number of studies have been carried out to recover such links, they all rely on calculating the textual similarity between artifacts without appropriately considering the context of each artifact. In this paper, we propose a novel approach to recover requirements traceability links between use cases and code, which extends Description-Embodied Knowledge Representation Learning (DKRL) model to comprehensively characterize software artifacts by embedding both text information and interrelationships. Such meaningful embeddings are then used to train traceability link classifiers by using machine learning and triple classification techniques. Experimental results show that our approach is superior to existing approaches.

Index Terms—Traceability Link Recovery, Knowledge Graph, Knowledge Representation Learning

I. Introduction

Software artifact traceability is essential for comprehending, maintaining, and evolving software programs [1]. However, creating traceable links is often abandoned due to time pressures in practice. In addition, considering the ever-changing requirements, continuously maintaining the traceability links is even more time-consuming. As a result, there is a strong need to automatically recover such links between existing software artifacts with acceptably high accuracy. Considering that software artifacts typically involve natural languages, many researchers have investigated the automatic recovery of traceability links by leveraging information retrieval and natural language processing techniques [2], [3]. Specifically, such approaches mainly rely on calculating text similarity between software artifacts, but ignore the context information of software artifacts.

Although the text-similarity plays an essential role in correlating software artifacts, we argue that the context of software artifacts also renders important clues for establishing the traceability links among artifacts. In particular, the context of software artifacts can typically be modeled as a graph, which connects an artifact with related ones via certain relationships. For example, a class diagram specifies the interrelationships among classes, which can serve as the context of each individual class. Similarly, use case diagrams represent context of the involved use cases. Intuitively, representing software artifacts (e.g., use cases) by incorporating its context would yield more meaningful results. Description-Embodied Knowledge Representation Learning (DKRL) [4] has been well recognized as an efficient representation learning approach, which captures both the structural information of explicit relationships and the textual descriptions of entities. Considering the context of software artifacts can be represented in terms of entities and relations, DKRL would contribute to comprehensively and meaningfully representing the context software artifacts and eventually help with the identification of traceability links among software artifacts.

In this paper, we propose a novel approach called Traceability Link Recovery-Knowledge Representation Learning(TLR-KRL)[1] to recover requirements traceability links between use cases and code based on Extended DKRL. Specifically, we extends the DKRL model to comprehensively characterize software artifacts by embedding both text information and structural relationships. Such meaningful embeddings are then used to train traceability link classifiers by using supervised machine learning algorithms. All traceability link candidates obtained from the classifier will be further screened to get the final result. Overall, the contributions of this paper can be summarized as below.

- Propose a systematic approach for recovering traceability links between use cases and code based on knowledge representation techniques, which can effectively characterize the context of the software artifacts.
- Extend DKRL model with a systematic process for developing negative samples in order to enhance the embedding of software artifacts.
- Design and conduct a series of experiments to evaluate our approach, the results of which show that our approach is superior to existing approaches.

The remaining part of this paper is organized as follows. We first review and discuss related work in Section II. We detail our approach in Section III. In Section IV, we evaluate our method through four experiments. In Section V, we conclude this paper and discuss future work.

---

[1] https://github.com/Shniya3/TLR-KRL

II. Related Work

1) Information Retrieval Technology: Information retrieval techniques are widely used in traceability link recovery [5]. Scholars have adopted various methods based on information retrieval: vector space model (VSM) [6], latent semantic index (LSI) [7], Latent Dirichlet allocation (LDA) [8] etc. Some researchers focus on other types of information of software artifacts besides textual information. McMillan et al. recover traceability links with textual and structural information according to "related requirement share related source code elements". [2] Wang et al. model the source code as a graph structure and mines the structural information in it through the graph embedding model [3]. However, we believe that there are multiple explicit relations between entities (such as class and method in the source code graph) are more suitable for mining structural information through knowledge representation learning methods. Jin Guo et al. introduce domain knowledge for word embedding, and predict the probability of link existence through RNN, in order to solve the "the term mismatch" problem [9].

2) Knowledge Graph and Knowledge Representation Learning: The main goal of the knowledge graph is to describe the various entities and concepts that exist in the real world and the explicit relationships between them. Relations are used to describe the relationship between two entities. The knowledge graph describes the knowledge in a structured form. People usually organize knowledge in the knowledge graph in the form of a network. Each node in the network represents an entity (person name, place name, etc.), and each edge represents the relationship between entities. Therefore, most of the knowledge can often be triple $(entity1, relation, entity2)$ to represent, corresponding to an edge and two nodes connected in the knowledge graph [10]. Although effective in representing structured data, the underlying symbolic nature of such triples usually makes KG hard to manipulate [11].

Konwledge representation learning has been investigated as an effective means to solve the above problems. The key idea of knowledge representation learning is to embed the entities and relationships in the knowledge graph into a continuous vector space, simplifying the manipulation while preserving the inherent structure of the KG [11]. After embedding, the vector representation of entities and relations is useful for downstream tasks, such as triple classification [12], KG completion [4], and so on. Today, TransE and its extensions are widely recognized in knowledge representation learning research [13] [12] [4].

Xie et al. uses entity descriptions to extend the TransE and proposes dkrl model [4]. DKRL model learn knowledge representations with both triples and descriptions, i.e.,structure-based representations and description-based representations. Structure-based representations do better in capturing information in gold triples of the Knowledge graph, while description-based representations do better in capturing textual information in entity descriptions [4].

III. Traceability Link Recovery-Knowledge Representation Learning(TLR-KRL)

We focus on recovering the traceability links between use cases and code case. Our proposed traceability link recovery approach is shown in Figure 1. Firstly, preprocess the software artifacts. We construct the software artifacts into the structure of KG and get the description of the entities in KG. Secondly, we extend the DKRL model to represent use cases and code comprehensively. We use the extended DKRL model to represent use cases and code cases. Thirdly, all traceability link candidates obtained from machine learning classifier and meaningful representation will be further screened to get the final results.
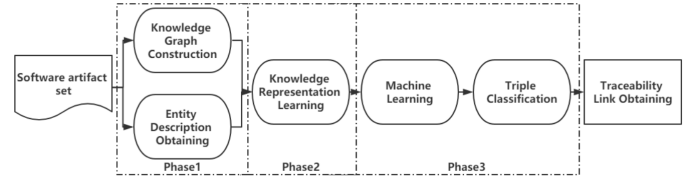
Fig. 1. TLR-KRL Overview

A. Phase 1: Preprocess

1) Software Artifact Knowledge Graph Construction: In order to capture the context information of software artifacts, we construct software artifacts as knowledge graphs. First of all, we define the entity types and relationships of the software artifact knowledge graph based on [3], as shown in figure2. Because our data set is developed in Java, the file name is the same as the public class name in the file. So the code case id is equal to the public class name in the code case. Other entities connected through these relationships become the context of a software artifact. We added member variables and member methods of a class to make vector representation of class more fine-grained. We add relations between use cases to capture context information between use cases. The construction process of the software artifact knowledge graph is shown in figure 3. Next, introduce the flow of figure 3.
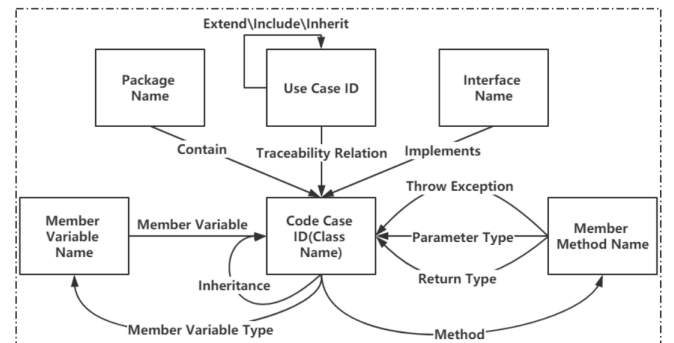
Fig. 2. Entity Type and Relation Definition in KG

- *Extract Relation between Use Cases.* In the use case, the "event flow of system" part is linked to other use cases through the use case name. So by comparing the use case names, we obtain a set of triples with relation "Inherit",
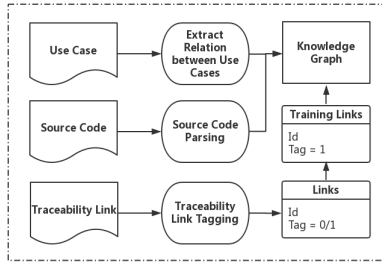
Fig. 3. Software Artifact Knowledge Graph Construction

"Include" and "Extend". Because there are fewer use case pairs with three kinds of relations, we call the relations between use cases unified as "Use Case to Use Case".

- *Source Code Parsing.* We parse the code through the source code parsing tool[2] to obtain classes, member variables, member methods, and their interrelations. Then we build the knowledge graph according to Table 1.

- *Traceability Link Tagging.* We tag the links between use cases and code. Tuples (Use Case, Class) without traceability link are tagged as 0. Tuples (Use Case, Class) with traceability links are tagged as 1. Tuples (Use Case, Class) with traceability links are added to the software artifact knowledge graph as (Use Case, Traceability Relation, Class).

2) Entity Description Obtaining: The process of obtaining entity description is shown in figure 4.
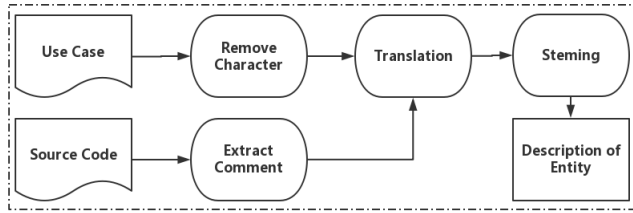
Fig. 4. Entity Description Obtaining

- *Remove Character.* Remove the title, number, punctuation, and special characters in the use case to get use case entity description.

- *Extract Comment.* We use regular expressions to get comments in the code and map them to class entities and method entities. Besides, the names of classes, member variables, and member methods are also natural languages with important meanings [14]. So the names are added to the description of entities.

- *Translation.* ETour is developed in Italian. The software artifacts contain Italian words, so we translate entity description into English.

- *Stemming.* We transform the verbs, nouns, adjectives, and adverbs in the entity description into prototypes. Software developers usually use the abbreviation of words when using common words. So we restore the abbreviations of words into word prototypes. For example, "database"

[2]https://github.com/yeweimian21/AST_JDT.

Table 1 Variable definitions.

Definition	Description
$S = \{(h, r, t)\}$	S represents training set for Extended DKRL model. (h, r, t) is a triple. h, t are entity. r is a relation.
$h, t \in E$	E represents entities set.
$r \in R$	R represents relations set.
$u \in U, c \in C$	U represents use cases set. C represents code set.
r_{Trace}	r_{Trace} represents the relation which we name "Traceability Relation."
$T' = \{(u, r_{Trace}, c)\}$	T' represents the negative samples set. These samples are traceability links between u and c tagged as 0 in preprocess.
u_d, c_d	u_d, c_d represents description-based representation
u_s, c_s	u_s, c_s represents structure-based representation

is generally abbreviated to "DB"; "delete" is generally abbreviated to "del." At this point, we get the knowledge graph of software artifacts and the description of entities in the knowledge graph.

B. Phase 2: Software Artifact Embedding

The process of phase 2 and phase 3 is shown in figure 5. As shown in phase 2 of the figure 5, phase 2 embeds the software artifact KG and its entity descriptions through Extended DKRL model in order to obtain the meaningful representation of software artifacts.

In general, the software requirements described by the use case and the functionality of the software should be equal. However, the application scenario described by a software requirement is often completed by multiple classes. The comments of the class describe the variables and methods of this class, which results in software requirements and textual information of the class being usually different. We argue that the representation of software artifacts with context information are helpful to recover the traceability links. So we use the Extended DKRL model to mine the context information and the text information of entity description in the KG of software artifacts. After extended DKRL training, the obtained software artifact vector contains its information and context information, which is helpful for the application of downstream tasks.

Extended DKRL represents the DKRL model that changed the negative sample construction process. In the training process of the DKRL model, we need to construct negative samples according to triples. Since negative samples are randomly constructed, it is possible to construct false negative samples. Because we tagged some traceability links in the preprocess, we improved the negative sample construction process with T' (we give some definitions to help the following statement, as
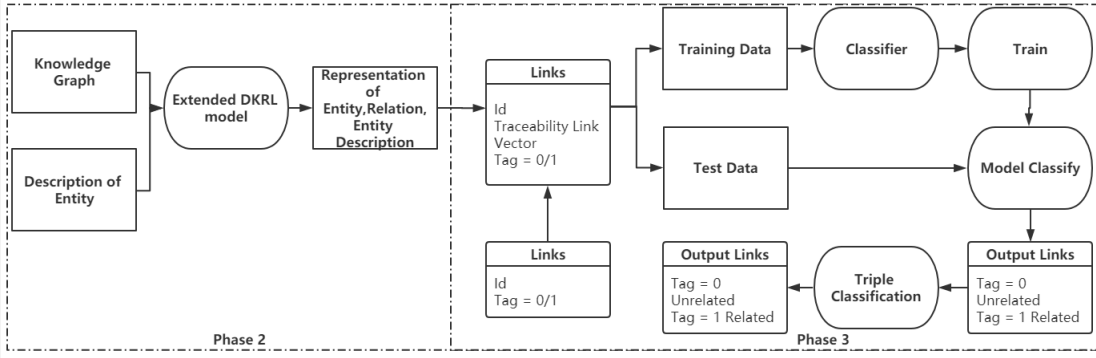
Fig. 5. Phase 2. Knowledge representation learning and Phase 3. Recover Traceability Link

shown in Table 1). Our improved negative sample construction process is as follows:

$$
S'_{(h,r,t)} = \begin{cases} \{(h',r,t) \mid (h',r,t) \in T'\} \cup \\ \{(h,r,t') \mid (h,r,t') \in T'\} \\ \quad if \ r \neq r_{Trace} \\ \{(h',r,t) \mid h' \in E\} \cup \{(h,r,t') \mid t' \in E\} \\ \quad if \ r = r_{Trace}" \end{cases} \quad (1)
$$

The set of corrupted triples, constructed according to Equation (1). When constructing a negative sample of triple whose relation is "Traceability Relation," randomly select use cases or classes to corrupt, we replace the use case with other use cases or class with other classes and make the negative samples belong to T'. When the relation is not "Traceability Relation," training triples with either the head or tail replace by a random entity.

C. Phase 3:Recover Traceability Link

1) Traceability Link Vector Definition: First, we represent traceable links between use cases and classes as Equation (2).

$$
t_{u,c} = (c_s - u_s) \oplus (c_d - u_d) \quad (2)
$$

"$\oplus$" represents the stitching of two vectors. According to the tagging results during Phase 1, if there is a traceability link between u and c, $t_{u,c}$ is tagged as 1; otherwise, $t_{u,c}$ is tagged as 0.

2) Train and Test Model: We train the classifier with all tagged $t_{u,c}$ vectors. There are many common classifiers, such as decision tree, gradient boosting decision tree, Gaussian naive Bayes, and SVM(We use the sklearn[3] library to call the classifier). Identified traceability links are tagged with 1, while others are tagged with 0 and will be further analyzed by using triple classification.

3) Reclassification of Negative Label Samples: Knowledge representation learning usually uses score function to calculate the reliability of triples. Triple Classification is to confirm whether a given triple $(h, Traceability\ relation, t)$ is correct or not according to its score, i.e., binary classification on a triple [12]. The decision rule for classification is simple: for a

triple $(h, Traceability\ relation, t)$, if the score (by the score function f_r) is below a relation-specific threshold σ_r, then predict positive.

Specifically, we first calculate the average score of traceability link triple in Extended DKRL train data. Then, if the triple to be classified as a score below than $\frac{average\ score}{rate}$, the triple is classified as 1 (with traceability link).

IV. EXPERIMENT

A. Dataset

The dataset of our Research is eTour[4]. It is an electronic touristic guide developed by students in Italy. It contains 58 use cases, 116 code cases, and 366 correct traceability links. (Use case, code case) without traceability link in eTour is regarded as wrong link, totaling 6362. In our experiment, the ratio of training set to test set is 4:3.

B. Research Questions and Experiment Design

- Question 1: Can the Extended DKRL model effectively mine software artifacts textual information and the context of software artifacts?
  Experiment 1: We evaluate the embedding results based on the visualization of entity vectors and traceability vectors. Because the entity and the traceability vector are both high-dimensional vectors, the high-dimensional vectors need to be reduced in dimension. After the high-dimensional vector is visualized, observe the distribution of the vector. There should be a clear demarcation between different types of entity vectors. Traceability link vectors should have clear demarcation or clustering. Besides, the quality of the knowledge representation learning method can be demonstrated through the performance of downstream tasks.
- Question 2: Can SVM effectively recover traceability links compared to other classifiers?
  Experiment 2: We use the same train set and test set to compare the classification results of multiple classifiers, such as decision tree, gradient boosting decision tree (GBDT) and Gaussian naive Beyes (Gaussian NB).

---

[3]www.scikit-learn.org

[4]http://www.cs.wm.edu/semeru/tefse2011/.

Experiment 2 compares multiple classifiers without performing triples classification.

- Question 3: Can TLR-KRL effectively recover traceability links?

  Experiment 3: We compare our approach with ML+Logical Reasoning [14], UD-CSTI(VSM) and UD-CSTI(JS) [15]. These models are the best performing models to use the same dataset. In addition, in order to prove the validity of the modified negative sample construction process, experiment 3 also add TLR-KRL(DKRL) to the experimental comparison.

- Question 4: Can triple classification recover traceability links effectively?

  Experiment 4: We first fix the results of the extended DKRL and SVM. Then experiment 4 observes whether the triple classification is useful by changing the threshold of triple classification.

C. Metric

We use well-known IR metrics to evaluate the performance of traceability recovery methods [5].

$$precision = \frac{|cor \cap ret|}{|ret|}\% \tag{3}$$

$$recall = \frac{|cor \cap ret|}{|cor|}\% \tag{4}$$

$$F1\text{-}score = \frac{2 \times precision \times recall}{precision + recall} \tag{5}$$

Specifically, cor represents the sets of correct links and ret is the full set of retrieved links. Based on these two variables, the metrics $precision$, $recall$ and $F1\text{-}score$ can be calculated, which are shown in equation (3)-(5)

D. Analysis of the Results
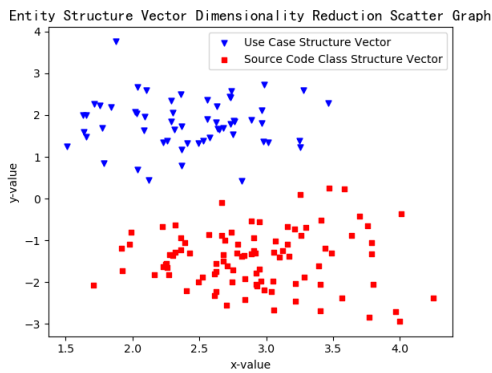
Fig. 6. Entity Structure Vector Dimensionality Reduction Scatter Graph

1) Experiment 1: As we can see in figure 6, use case structure vectors and classes structure vectors have a clear demarcation. As we can see in figure 7, use case description vectors and class description vectors have a clear demarcation.
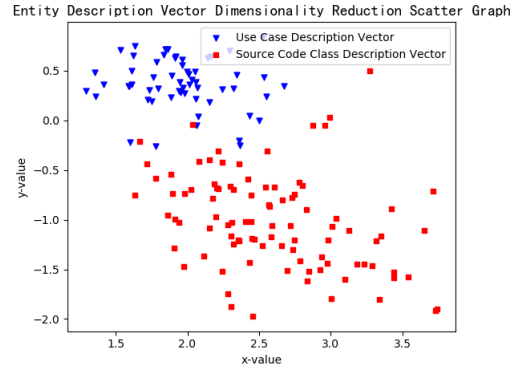
Fig. 7. Entity Description Vector Dimensionality Reduction Scatter Graph
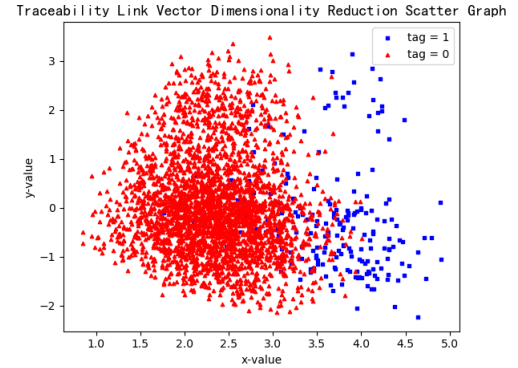
Fig. 8. Traceability Link Vector Dimensionality Reduction Scatter Graph

As shown in figure 8, traceability link vectors with different labels are distributed on both sides of the figure. Traceability link vectors that are tagged as 0 are well clustered on the left side and can be clearly separated from the other type of vectors, showing that our approach can effectively mine textual information and the context of artifacts.
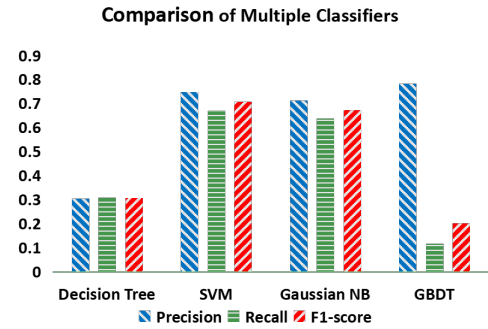
Fig. 9. Comparison of Multiple Classifiers

2) Experiment 2: As we can see in figure 9, the $precision$ of GBDT is slightly higher than SVM, but the $recall$ is significantly lower than SVM. The $F1 - score$ of SVM with the polynomial kernel is the best among the compared classifiers because SVM can better classify linear inseparable problems. Because $F1 - score$ comprehensively evaluates the performance of the classifier, our subsequent experiments use SVM as the classifier.
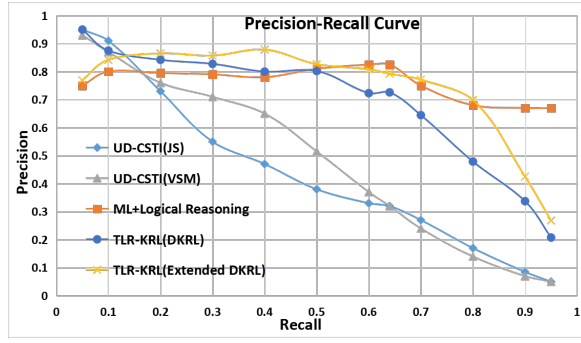
Fig. 10. The Precision-Recall Curves Graph

3) Experiment 3: When evaluating a traceability link recovery work, it is common to compare *precision* at different *recall* level. Figure 10 shows the *precision-recall* curve of ML+Logical Reasoning, UD-CSTI(VSM), UD-CSTI(JS), TLR-KRL(DKRL) and TLR-KRL(Extended DKRL) respectively. It can be seen that our method has a significant improvement in the *recall* rate of 0.2-0.5. The performance of our method is close to that of the ML+Logical approach within the *recall* rate of 0.5-0.8. It shows that TLR-KRL(Extended DKRL)s can recover traceability links effectively. In addition, the improvement of the DKRL negative sample construction process is also effective, because the *precision-recall* of downstream tasks has been improved.
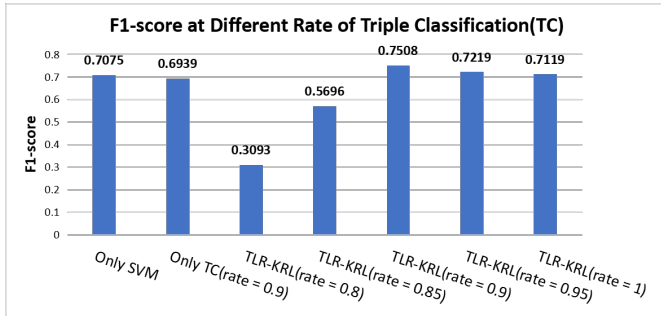
Fig. 11. F1-score at Different Rate of Triple Classification(TC)

4) Experiment 4: We fixed the hyper-parameter of Extended DKRL and SVM to explore whether the triple classification is effective. As shown in figure 11, the performance only using SVM and only using triple classification is similar. When we change the threshold of triple classification to 0.9, the performance is improved. As the rate increases, $F1-score$ gradually decreases, because the threshold is gradually reduced to reduce the *recall*. But TLR-KRL is still better than using the only SVM. To sum up, triple classification can effectively recover traceability links.

V. CONCLUSION AND FUTURE WORK

In this paper, we propose a novel approach to recover requirements traceability links between use cases and code based on DKRL. our approach extends DKRL model in order to embed software artifacts with regard to both of their textual

descriptions and their structural context. A series of experiments have been conducted, the results of which show that our approach has outperformed existing traceability recovery approaches In the future, we first plan to further evaluate our approaches with additional data sets. Moreover, we want to apply our approach to recover traceability links among other software artifacts, investigating whether our approach can be generalized to deal with various software artifacts. Finally, we envision an empirical case study with our industrial partners.

REFERENCES

[1] A. D. Lucia, F. Fasano, R. Oliveto, and G. Tortora, "Enhancing an artefact management system with traceability recovery features," *20th IEEE International Conference on Software Maintenance, 2004.*, 2004.

[2] C. McMillan, D. Poshyvanyk, and M. Revelle, "Combining textual and structural analysis of software artifacts for traceability link recovery," in *2009 ICSE Workshop on Traceability in Emerging Forms of Software Engineering*. IEEE, 2009, pp. 41–48.

[3] S. Wang, T. Li, and Z. Yang, "Using graph embedding to improve requirements traceability recovery," in *International Conference on Applied Informatics*. Springer, 2019, pp. 533–545.

[4] R. Xie, Z. Liu, J. Jia, H. Luan, and M. Sun, "Representation learning of knowledge graphs with entity descriptions," in *Thirtieth AAAI Conference on Artificial Intelligence*, 2016.

[5] M. Borg, P. Runeson, and A. Ardö, "Recovering from a decade: A systematic mapping of information retrieval approaches to software traceability," *Empirical Softw. Engg.*, vol. 19, no. 6, p. 1565–1616, Dec. 2014. [Online]. Available: https://doi.org/10.1007/s10664-013-9255-y

[6] G. Antoniol, G. Canfora, G. Casazza, A. De Lucia, and E. Merlo, "Recovering traceability links between code and documentation," *IEEE transactions on software engineering*, vol. 28, no. 10, pp. 970–983, 2002.

[7] M. Lormans and A. Van Deursen, "Reconstructing requirements coverage views from design and test using traceability recovery via lsi," in *Proceedings of the 3rd international workshop on Traceability in emerging forms of software engineering*, 2005, pp. 37–42.

[8] H. U. Asuncion, A. U. Asuncion, and R. N. Taylor, "Software traceability with topic modeling," in *2010 ACM/IEEE 32nd International Conference on Software Engineering*, vol. 1. IEEE, 2010, pp. 95–104.

[9] J. Guo, J. Cheng, and J. Cleland-Huang, "Semantically enhanced software traceability using deep learning techniques," in *2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE)*. IEEE, 2017, pp. 3–14.

[10] L. Zhiyuan, S. Maosong, L. Yankai, and X. Ruobing, "Knowledge representation learning:a review," *Journal of Computer Research and Development*, vol. 53, no. 2, pp. 247–261, 2016.

[11] Q. Wang, Z. Mao, B. Wang, and L. Guo, "Knowledge graph embedding: A survey of approaches and applications," *IEEE Transactions on Knowledge and Data Engineering*, vol. 29, no. 12, pp. 2724–2743, 2017.

[12] Z. Wang, J. Zhang, J. Feng, and Z. Chen, "Knowledge graph embedding by translating on hyperplanes," in *Twenty-Eighth AAAI conference on artificial intelligence*, 2014.

[13] A. Bordes, N. Usunier, A. Garcia-Duran, J. Weston, and O. Yakhnenko, "Translating embeddings for modeling multi-relational data," in *Advances in neural information processing systems*, 2013, pp. 2787–2795.

[14] S. Wang, T. Li, and Z. Yang, "Exploring semantics of software artifacts to improve requirements traceability recovery: A hybrid approach," in *2019 26th Asia-Pacific Software Engineering Conference (APSEC)*. IEEE, 2019, pp. 39–46.

[15] A. Panichella, C. McMillan, E. Moritz, D. Palmieri, R. Oliveto, D. Poshyvanyk, and A. De Lucia, "When and how using structural information to improve ir-based traceability recovery," in *2013 17th European Conference on Software Maintenance and Reengineering*. IEEE, 2013, pp. 199–208.

Identifying Security Concerns Based on a Use Case Ontology Framework

Imano Williams, Xiaohong Yuna

Computer Science
North Carolina Agricultural and Technical State University
Greensboro, U.S.A
irwilli1@aggies.ncat.edu, xhyuan@ncat.edu

Abstract— **Identifying security concerns in an application can be difficult, especially if the analysts lack security knowledge. We propose a use case ontology that can help to identify security concerns based on the use case specifications. We demonstrate the feasibility of the ontology by systematically applying the ontology to use case specifications expressed in Web Ontology Language (OWL). The proposed approach can help model the interrelationship of concepts in the use case and possibly use queries to group use cases that may have similar security concerns. This approach could allow analysts to identify parts of the use cases with similar security concerns and could potentially reduce reoccurrences of known vulnerabilities in software applications. Lastly, we discuss future work about creating an automated tool for recommending attack patterns for the security requirements process.**

Keywords-Domain Ontology; Security Concerns; Use Cases; Secure Software Engineering

I. INTRODUCTION

Ontological modeling of software artifacts has been used in the requirements and design phases to address security issues [1]. Some ontologies have used software artifacts such as security requirements [2] and use cases [3] to aid in knowledge acquisition and the conceptualization of reusable domain-specific software security information. According to Veres, et al. [4], ontologies can be used to track the dependencies between requirements as the project becomes realistically complex. However, when ontologies are used to elicit security requirements, the security requirements elicited depend on the ontology that was used [5]. Furthermore, requirement analysts may not have adequate security knowledge to choose the most appropriate ontology and then use it correctly [6]. In addition to security knowledge, requirement analysts also need domain knowledge to identify security concerns. Therefore, the modeling of the system based on an ontology can be difficult for security requirement analysts.

In this paper, we introduce a use case ontology framework to identify security concerns based on use case descriptions. The following observations motivated us to build the ontology: (1) Reports from Open Web Application Security Project (OWASP) Top 10 [7] and Common Vulnerability Exposure[1] (CVE) have shown the frequent reoccurrences of known vulnerabilities, such as SQL Injection and Cross-site Scripting. (2) Many of the reported vulnerability exploits started from the web interface of an application. (3) In a software development team, different understandings of what to secure in software under development (SUD) may lead to ambiguous, incomplete, and inconsistent security concerns being identified by the stakeholders.

We created the ontology framework for identifying security concerns by (1) Identifying the Assets and Web Components of a use case that can guide requirements analysts to raise security concerns via use case steps; (2) Creating concepts and attributes for the proposed ontology based on the results of steps (1); and (3) Associate the use case steps (or flows) to specific security concerns using semantic rules; (4) Based on this ontology, semantic queries can be run to find similar use case flows that may have similar concerns. This ontology framework can help reduce reoccurrences of know vulnerabilities by identifying similar security concerns across different functionalities of an application and different applications.

The rest of the paper is organized as follows: Section II presents the proposed ontology. Section III defines rules for identifying security concerns. Section IV discusses how the ontology can be used to identify security concerns. Section V demonstrates how the ontology framework is used with a specific example. Section VI discusses related work. Finally, Section VII concludes the paper and discusses future work.

II. THE PROPOSED USE CASE ONTOLOGY

The proposed ontology was developed using the steps suggested by Noy and McGuinness [8] with an evolutionary approach. These steps include defining the scope of the ontology, reusing existing ontologies, enumerating important terms, defining classes, defining properties, defining cardinality, and creating instances (individuals). We adopted concepts that are related to security concerns in [9], the Restricted Use Case Model (RUCM) [10], and verb categories for web tasks from [11-13] to develop the ontology. The RUCM is a use case template that specifies 26 restriction rules on the natural language, keywords for control structures, and that every flow path in a use case should have a post-condition. In addition to using a more restrictive use case template, we used a dialog descriptive use case format. The dialog use case format includes graphical user interface (UI) components in the use case flows.

The rationale for using dialog descriptive use case includes the following: (1) the web components in the use case could guide requirements analysts to raise security concerns; (2) use cases are rudimentary software development artifacts that can represent the system navigational structure from a graphical user interface aspect; (3) according to Salini and Kanmani [14], the user interface and navigational structure are the main features of applications' web interfaces that must be analyzed and (4) currently, we focus on the constrained system where the design of the interaction is more precise than just providing the intent of the use case.
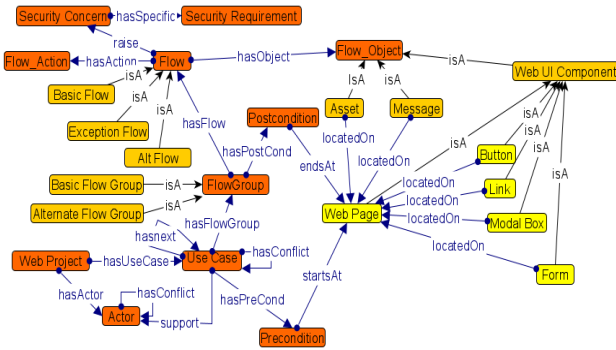
Figure 1 The proposed use case ontology for identifying security concern
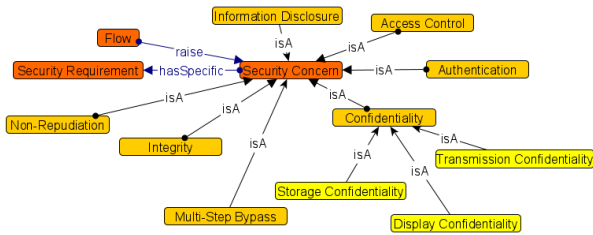
Figure 2 Security concern subclasses

In Fig. 1, we have shown some of the major concepts that were taken from different sources to build the proposed ontology. Fig. 2 shows the concepts related to security concerns. These security concern concepts were adopted from ISO/IEC 27001:2013 [15], ISO/IEC 27000:2018 [2], and [9]. Next, we provide the definitions of some of the core concepts:

- **Use Case**: Represents the intended interactive steps between an external entity and the system.
- **Actor**: Represents a human or an external system that interacts with the system to accomplish the services of the use case.
- **Flow**: It specifies the logical steps that an actor takes to complete the services of the use case.
- **Web UI Component**: An interface component of the application that actors interact with to complete the services of the use case. In our ontology, we created concepts for the button, link, web page, and modal box.

- **Flow Action**: This is an operation that is performed by the Actor or the SUD to complete a Flow. Some examples are "*The user updates the username.*" and "*The system displays the 'login'* web page."
- **Asset/Message**: An intangible valuable resource, such as a password that is worth protecting. Here we focus on data the user provides via some user input.
- **Security Concern**: Matters of interest related to security exploits that may affect use case flows based on the action, web components, and the asset. The subclasses are authentication, authorization, confidentiality, integrity, non-repudiation, identity, and security auditing.
- **Security Requirements**: Conditions that must be satisfied to address a security concern.

An object property is represented as "o (D → R)", which means a class D (domain) is related to another class R (range) by o, the object property. Some of the object properties are:

- **hasFlow** (FlowGroup → Flow)
- **hasFlowBefore** (Flow → Flow), an inverse of **hasFlowAfter** (Flow → Flow)
- **hasActor** (Use Case → Actor)
- **raise** (Flow, Use Case, Post Condition → Security Concern)
- **display** (System → Web Page, Modal Box, Message)
- **validate** (System → Asset)
- **hasFlowObject** (Flow → Web UI Component, Message, Asset)

A data property is represented as, "d (C → r)", which means that class C has data property, d, with range r. Some of the data properties are:

- **hasActionType** (Action → ["passive", "active"])
- **hasLinkParameter** (Link → ["non-sensitive", "sensitive", "both", "none"])
- **hasInformation** (Asset, Web Page → ["non-sensitive", "sensitive", "both", "none"])
- **hasAppLocation** (Use Case → ["authenticated", "unauthenticated", "both"])
- **hasPurpose** (Use Case → ["create", "read", "update", "delete"])
- **hasInteractionFlow** (Use Case → ["multi", "single"])
- **isValidationFlow** (Flow, → ["yes", "no"])

To formally specify the classes (concepts), along with their object properties, data properties, and quantifier restriction, we used the Web Ontology Language (OWL) [16]. We used the second level OWL 2, OWL-DL, which provides maximum expressiveness while retaining the inference capabilities of an ontology [17] and semantic queries over the knowledge.

III. RULES FOR IDENTIFYING SECURITY CONCERNS

We defined rules using Semantic Web Rule Language (SWRL) [18] to identify security concerns based on the assets,

web UI components, and the actions in the use case flow. Currently, we defined eight rules for identifying security concerns that are listed below. These rules are not exhaustive.

Rules for the use case flow concepts:

1) *Asset <sensitive> && Save → Storage Confidentiality*, which means that an Asset instance with a "sensitive" data property value that should be saved has storage confidentiality.

2) *Asset <sensitive> && Display → Display Confidentiality*, which means that an Asset instance with a "sensitive" data property value should have display confidentiality.

3) *Asset && Validate → Multi-Step Bypass*, which means that an Asset instance that is being validated by the system multi-step bypass. This security concern occurs when a user can bypass some validation logic to get to another flow in the use case.

4) *Button <active> && Click → Non-Repudiation && Transmission Confidentiality*, which means that a Button instance with an active data property (i.e., it makes changes to the system file system) associated with a click action have non-repudiation and transmission confidentiality.

5) *Link <sensitive> && Click → Transmission Confidentiality,* which means that a click action on a Link instance with "sensitive" parameter data property value has transmission confidentiality.

6) *Button, Link && Click → Information Disclosure*, which means when the system generates messages (warning, error, or, confirm) because of an action on a link or button has Information Disclosure.

Rules for the use case concept:

7) hasAppLocation *<authenticated or both> →* Authentication, which means a Use Case instance with authenticated or both boundary type has of authentication.

Rules for the actor concept:

8) *Conflicting Use Cases && Actor → Separation of Duties*, which means a use case that is followed by a conflicting use case, should not have the same Actor. Conflicting meaning that the same Actor cannot use two or more use cases. This rule is an extension of Rule 2.

Apart from having these eight rules, we can create informal rules (not in SWRL) to be used as SPARQL Protocol, and RDF Query Language (SPARQL) query, such as:

1) *Non-Permitted Actor && Web Pages → Inaccessible Web Pages* means that a user role is not permitted to view restricted web pages. .

IV. THE PROPOSED ONTOLOGY FRAMEWORK APPROACH

In this section, we describe how the ontology could be in a framework to identify security concerns. Fig. 3 shows the three major phases, along with the respective sub-phases.

In the first phase, *Identifying Instances*, the instances based on the use case concepts are identified along with their object and data properties. We start by identifying the name of the use case along with its data properties, and then we find the instances that are related to the use case via its object properties.
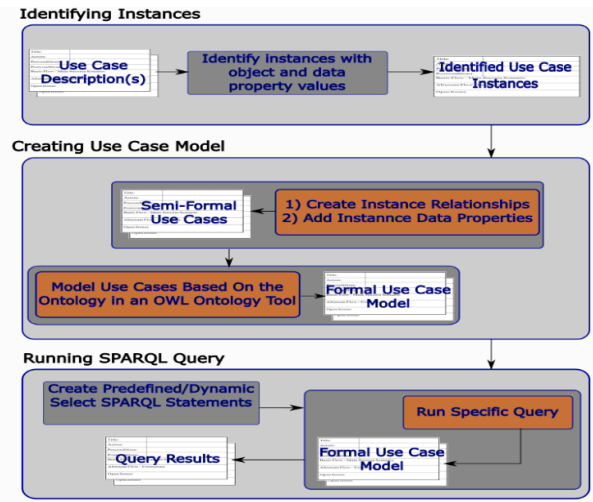
Figure 3 The Proposed Ontology Framework

For example, we identify the Actor instance and then find its data and object properties. Overall, we perform a depth-first identification of the instances (via concepts) with the data and object properties, then recursively perform a depth-first identification on the next instance that is related to the current instance via the object property. As a result, we identify the Asset, Action, and Web UI Component from the flows of the use cases along with their inter-relationships. For example, we can specify the *hasFlowBefore* and *hasFlowBefore* object properties for the current flow or the web pages that an asset is located.

In the second phase, Creating Use Case Model, the output of the first phase is used to create the semi-formal RDF triples (*subject, predicate, object*) of use case descriptions. Next, Protégé [19], the OWL 2 editor, is used to create the instances and their relationships with their object and data properties that were identified in the first phase in the ontology. During this phase, the Pellet[3] reasoner is run regularly to continually check the consistency of the asserted facts being added in the ABox (asserted facts about the use cases) of the ontology.

In the third phase, Running SPARQL Query, SPAQRL is used to query the ABox to find security concerns based on the inferred facts using the SWRL rules in Section III. For example, if several Flow instances can be affected by Multi-Step Bypass, the query will return the Flows. We use the approach proposed by Uschold and Gruninger [20] to evaluate the ontology based on motivating scenarios, informal competency queries, and formal competence queries to help identify security concerns for the modeled application. To run SPARQL queries, we use Snap-SPARQL [21] plugin[4] in Protégé that supports reasoner inferences using the Pellet plugin. Pellet, an OWL reasoner plugin in Protégé, is used to assist in answering the queries about the security concerns.

V. DEMONSTRATING THE ONTOLOGY FRAMEWORK

A. A Case Study

We used a mock online[5] shopping web application to demonstrate the ontology framework. The total number of instances (individuals) modeled were 11 use cases, four actors, 22 web pages, 24 assets, 17 links, eight buttons, 90 flows, and 14 actions in the proposed ontology. We had to edit the use cases to conform to the standards of the Restricted Use Case Model [10]. The edited use cases and ontology file are located at Use_Case_Ontology_for_Security_Concern[6]. Fig. 4 shows a description of the "*Create New Account*" use case. In the use case description, we used a double underline to mark the action and single underline to mark the object in the sentence to be modeled in the ontology as triples.

```
USE CASE UC3: Create New Account
Actors: Primary – Unregistered Customer
Preconditions: The system is displaying a 'Home' Webpage to the user.
Basic flow:
1. The user clicks on 'Create New Account' link.
2. The system displays the 'New Account Information' screen.
3.The user enters the FirstName, Last Name, Street Address, City, State, Country,
Postal Code, Card Number, Card Type, and Card Expiry Date.
4. The user clicks on the 'Create' button.
5. The system validates that the FirstName, Last Name, Street Address, City, State,
Country, Postal Code, Card Number, Card Type, and Card Expiry Date are correct.
6. The system displays the 'Signup Information' webpage.
7. The user enters the username and password.
8. The user clicks on the "Sign Up" Button.
9. The system validates that the username does not exist.
10. The system displays the 'Account Confirmation' webpage along with the
FirstName, Last Name, Street Address, City, State or Province, Country, Postal
Code, Telephone Number, Card Number, Card Type, and Card Expiry Date.
11. The user clicks on the "Verification" button.
12. The system displays the 'Account Information' Page.
Post Condition: The system saves the FirstName, Last Name, Street Address, City,
State or Province, Country, Postal Code, Telephone Number, Card Number, Card
Type, and Card Expiry Date. The system is displaying the 'Account Information'
Page to the user.
Bounded Alternate Flow: N/A
Global Flow: N/A
Specific Flow: N/A
```

Figure 4 The Create New Account Use Case

In the *Identifying Concepts* phase, we identify the concepts in the sequence of Use Cases → Actors → Preconditions → Flow Groups → Flow (subject, predicate, object) → Postconditions along with their data and object properties. The *has Boundary* data property of the "*Create New Account*" use case is on the "*unauthenticated*" side of the application. There is only one Actor, Unregistered Customer, for this use case. To initiate the use cases, the Actor would start from the "Home" page. Next, we move onto the basic flow to identify the predicate and object. The user is sending account information that the system must validate in basic flow four before the user can supply the signup username and password to complete the use case (basic flow 5). Therefore, the interaction is a "*multiple-step*" (temporal expression). We determine the Asset instance based on the data the actor supplied to the system or vice versa. The use case's purpose is "*active (inserting)*" since the use case is making changes to the file system to create the new account. Fig. 5 shows the representation of basic flow six, along with its object and data properties in the Protégé editor. The object property *hasFlowBefore* has value basic flow five. Additionally,

the inferred flows before and after are shown through the *dependsOnFlowBefore* and *subsequentFlow* object properties.
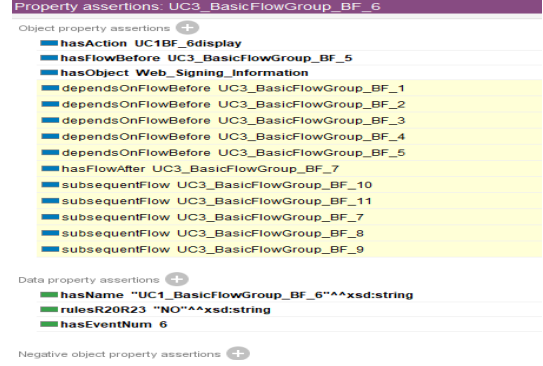
Figure 5 Basic Flow in Protégé
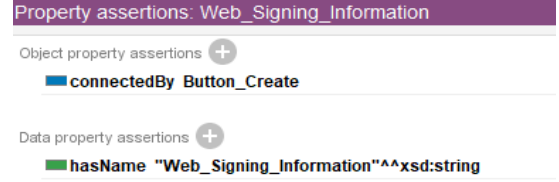
Figure 6 Signup Information Web Page in Protégé

Fig. 6 shows the Signup Information Web Page from basic flow six. The object property *connectedBy* shows the navigational path to get to the web pages. In this case, it is the "Create" button located in basic flow four. It could also be a Link instance. Furthermore, the "Sign Up" button in flow eight in Fig. 4 would have a *buttonLocatedOn* object property of Signup Information Web Page in the ontology.

Once we have completed the Creating Use Case Model phase for all the use cases, the next phase is Running SPARQL Queries based on the defined SWRL rules. We used SNAPSPAQRL to run the queries since it supports inferring once a reasoner is running. The SWRL rule for rule 3, Multi-Step Bypass is *Flow (?f), Asset(?a), Action (Validate), has Asset(?f, ?a), has Action(?f,Validate), Mult_Step_ByPass(?m) → raise (?f, ?m)*. We can then query ontology to find the consequent of the SWRL rule once the antecedent is true. So, we can have informal and formal queries to search:

1. **Informal Query**: Which flows are affected by the *Multi-Step Bypass security concern?*
2. **Formal Query** SELECT **?useCase ?flow** WHERE {
   ?useCase uc:hasFlowGroup ?flowGroup.
   ?flow uc:isPartOfGroup ?flowGroup.
   ?flow uc:raise uc:MultiStepBypass. }

Table I shows the partial results of running the above formal query.

Table I Multi-Step Bypass Query Results

Use Case	Flow Affected
UC11_Login	UC11_MF5
UC3_Create_New_Account	UC3MF5
UC3_Create_New_Account	UC3MF9

The rules in Section III are not the complete SWRL ruleset. Apart from running the queries based on the defined SWRL rules, an analyst can also create and run new queries based on reported CVEs. For example, CVE-2018-14398[7], a*n issue was discovered in Creme CRM 1.6.12. The value of the cancel button uses the content of the HTTP Referrer header and could be used to trick a user into visiting a fake login page to steal credentials*. This vulnerability is related to a user clicking a button component on the web page. Therefore, we can create a SPARQL query to find the pages that involve users clicking a button. The query would be the following:

1. **Informal Query:** Which flows display a button, and which pages are these buttons located?

2. **SPARQL Query:** SELECT **?flow ?button ?webpage** WHERE {**?useCase** uc:hasFlowGroup ?flowGroup. **?flow** uc:isPartOfGroup ?flowGroup. **?flow** uc:hasFlowObject **?button** . **?button** a uc:Button ; uc:buttonLocatedOn **?webpage** . }

From the query results in Table II, UC2_Search Catalog – basic flow three, UC9_Make Online Payment – basic flow (success path) six, UC3_Create New Account – basic flow four, UC4_Update Account Information – basic flow five, and UC8_Apply for Financing – basic flow six are affected by Rule 6 (information disclosure) when a user clicks a button. We can run similar queries to find other parts of the system that could be exposed to other CVEs.

Table II Button in Use Case Model Similar To CVE-2018-14398

Flow	Button	Web Page
UC9_BF6	Button_Submit	Web_Make__Payments_Page
UC2_BF3	Button_Search	Web_Main_Page
UC3_BF4	Button_Update	Web_New_Acct_Information_Page
UC4_BF5	Button_Finish	Web_Update_Acct_Information_Page
UC8_BF6	Button_Submit	Web_Make__Payments_Page

B. Discussion

In section V.A, we demonstrate how the ontology framework can be applied to a specific use case. The object and data properties show that specific rules could be used to find similar parts of different use cases that may share the same security concern. So, we focus more on providing a modeling process to find common security concerns through specific scenarios inspired by reports from OWASP and CVE, where the interface of the application is concerned.

In terms of performance, the instance Identification and Use Case Model Creation phases of the framework are task intensive. We had to take precautionary steps so that we did not miss sub-tasks, such as extracting the information from the use case description that is related to the concepts in the ontology. Manually populating the ontology by copying the semi-model from the document to the ontology via Protégé is tedious and time-consuming. Also, it is easy for the ontology to become inconsistent when using the wrong individual for the range of an object property. Furthermore, an analyst may forget to add information from the semi-model. It took more than 2 hours to populate the ABox of the ontology for the 11 use cases. As a result, an automated process to identify what to populate the

ABox of the ontology is needed to make the process take less effort by a user.

VI. RELATED WORK

Gärtner, et al. [3] developed an integrative security knowledge model that identifies vulnerabilities from software requirements (use cases) based on reported security incidents. They conducted a case study that showed how different use cases were related to a similar misuse case, but their proposed knowledge structure was not able to identify interrelationship between use cases that may have the same misuse case. Rago, et al. [22] used text mining to identify quality attributes such as modifiability, performance, availability, security from use case description. Their work aimed to help requirements engineers skim through requirements documentation efficiently, in order to identify potential quality attributes such as performance, security, mobility, and testability. However, in terms of security quality attribute, they did not delve into security concerns. Wouters, et al. [23] proposed a semi-formal ontology for the reuse of similar use cases by defining labels, concepts and relations to create rules and queries in an inference machine to find similar use cases. Our work is similar to theirs in the conceptual model of user interaction with UI in use case. However, our ontology includes more detailed UI components concepts such as button, web page, URL. Couto, et al. [24] automated the extraction of requirements patterns based on stakeholders formalizing use case specification by using OWL inference capabilities to address typical implementation solutions. Dermeval, et al. [25] suggested that ontologies could be used for representing requirements and architectural knowledge and support reasoning through traceable links between them. This paper does not focus on bridging the gap between requirement and architectural design phases, but the concepts such as web page, button, other web UI components can be linked to artifacts in architectural design and subsequent phases, which help with traceability. Decker, et al. [26] represent use cases in a requirements document ontology to semi formalize the representation of actors interacting with the system through user story descriptions. Kang and Liang [9] developed a security ontology for software development, a model-driven approach, where security concerns play a role in the analysis, design, implementation, testing, and maintenance stages of the SDLC. Our approach is different from Kang and Liang [9] since we focus on applying security concerns to use case instead of representing use cases as ontologies for development.

Our work is different from the related literature in that it mapped specific flows in use cases to security concerns based on data and object properties.

VII. CONCLUSION AND FUTURE WORK

This paper introduced a preliminary work on using an ontology framework during the early software development phases to identify security concerns based on use cases. We have manually and effectively created relationships between different use case concepts. These relationships have the potential to relate use case concepts to security concerns. Even though we can use the ontology to identify security concerns, manually

---

[7] https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2018-14398

representing the user case in the ontology can be time inefficient for many use cases.

Also, this ontology currently works with a predefined set of rules for identifying security concerns. CVE provides information on many security attacks that are based on different CVE scenarios. For example, different parts of the use case can be exploited with XSS. New rules and queries can be developed to find where in the use case that could be affected XSS.

In future work, we intend to develop a web-based tool to automatically extract and populate the relevant information from use cases into the ontology. As a result, the tool will semi-automatically query ontology the parts of the use case that matches the security concerns rule. The work presented in this paper is a part of a larger project to help recommend relevant attack patterns as part of the security requirements process. We will evaluate the usability of the ontology framework in a user study with the participants in software engineering courses and the security requirements community.

REFERENCES

[1]. A. Souag, R. Mazo, C. Salinesi, and I. Comyn-Wattiau, "Reusable knowledge in security requirements engineering: a systematic mapping study," *Requirements Engineering,* vol. 21, pp. 251-283, 2016.

[2]. C. Schmitt and P. Liggesmeyer, "Getting grip on security requirements elicitation by structuring and reusing security requirements sources," *Complex Systems Informatics and Modeling Quarterly,* pp. 15-34, 2015.

[3]. S. Gärtner, T. Ruhroth, J. Bürger, K. Schneider, and J. Jürjens, "Maintaining requirements for long-living software systems by incorporating security knowledge," in *Requirements Engineering Conference (RE), 2014 IEEE 22nd International*, 2014, pp. 103-112.

[4]. C. Veres, J. Sampson, S. J. Bleistein, K. Cox, and J. Verner, "Using semantic technologies to enhance a requirements engineering approach for alignment of IT with business strategy," in *Complex, Intelligent and Software Intensive Systems, 2009. CISIS'09. International Conference on*, 2009, pp. 469-474.

[5]. A. Souag, C. Salinesi, R. Mazo, and I. Comyn-Wattiau, "A Security Ontology for Security Requirements Elicitation," in *ESSoS*, 2015, pp. 157-177.

[6]. H. Guan, H. Yang, and J. Wang, "An ontology-based approach to security pattern selection," *International Journal of Automation and Computing,* vol. 13, pp. 168-182, 2016.

[7]. T. OWASP, "10 2017," OWASP Top 10 Application Security Risks–2017, 2018.

[8]. N. F. Noy and D. L. McGuinness, "Ontology development 101: A guide to creating your first ontology," ed: Stanford knowledge systems laboratory technical report KSL-01-05 and Stanford medical informatics technical report SMI-2001-0880, Stanford, CA, 2001.

[9]. W. Kang and Y. Liang, "A security ontology with MDA for software development," in *Cyber-Enabled Distributed Computing and Knowledge Discovery (CyberC), 2013 International Conference on*, 2013, pp. 67-74.

[10]. T. Yue, L. C. Briand, and Y. Labiche, "Facilitating the transition from use case models to analysis models: Approach and experiments," *ACM Transactions on Software Engineering and Methodology (TOSEM),* vol. 22, p. 5, 2013.

[11]. D. Ko, S. Kim, and S. Park, "Automatic recommendation to omitted steps in use case specification," *Requirements Engineering,* pp. 1-28, 2018.

[12]. J. Jurkiewicz and J. Nawrocki, "Automated events identification in use cases," *Information and Software Technology,* vol. 58, pp. 110-122, 2015.

[13]. S. Tena, D. Díez, P. Díaz, and I. Aedo, "Standardizing the narrative of use cases: A controlled vocabulary of web user tasks," *Information and Software Technology,* vol. 55, pp. 1580-1589, 2013.

[14]. P. Salini and S. Kanmani, "Security requirements engineering process for web applications," *Procedia engineering,* vol. 38, pp. 2799-2807, 2012.

[15]. I. O. f. Standardization, ISO/IEC 27001: 2013: Information Technology--Security Techniques--Information Security Management Systems--Requirements: International Organization for Standardization, 2013.

[16]. I. Horrocks, P. F. Patel-Schneider, and F. Van Harmelen, "From SHIQ and RDF to OWL: The making of a web ontology language," *Journal of web semantics,* vol. 1, pp. 7-26, 2003.

[17]. C. Welty, D. L. McGuinness, and M. K. Smith, "Owl web ontology language guide," W3C recommendation, W3C (February 2004) *http://www*.w3. org/TR/2004/REC-owl-guide-20040210, 2004.

[18]. I. Horrocks, P. F. Patel-Schneider, H. Boley, S. Tabet, B. Grosof, and M. Dean, "SWRL: A semantic web rule language combining OWL and RuleML," *W3C Member submission,* vol. 21, p. 79, 2004.

[19]. M. A. Musen and T. the Protégé, "The Protégé Project: A Look Back and a Look Forward," *AI matters,* vol. 1, pp. 4-12, 2015.

[20]. M. Uschold and M. Gruninger, "Ontologies: Principles, methods and applications," *The knowledge engineering review,* vol. 11, pp. 93-136, 1996.

[21]. M. Horridge and M. Musen, "Snap-SPARQL: a java framework for working with SPARQL and OWL," in *International Experiences and Directions Workshop on OWL*, 2015, pp. 154-165.

[22]. A. Rago, C. Marcos, and J. A. Diaz-Pace, "Uncovering quality-attribute concerns in use case specifications via early aspect mining," *Requirements Engineering,* vol. 18, pp. 67-84, 2013.

[23]. B. Wouters, D. Deridder, and E. Van Paesschen, "The use of ontologies as a backbone for use case management," in *European Conference on Object-Oriented Programming (ECOOP 2000), Workshop: Objects and Classifications, a natural convergence*, 2000.

[24]. R. Couto, A. N. Ribeiro, and J. C. Campos, "Application of ontologies in identifying requirements patterns in use cases," *arXiv preprint arXiv:1404.0850,* 2014.

[25]. D. Dermeval, J. Vilela, I. I. Bittencourt, J. Castro, S. Isotani, P. Brito, *et al.*, "Applications of ontologies in requirements engineering: a systematic review of the literature," *Requirements Engineering,* vol. 21, pp. 405-437, 2016.

[26]. B. Decker, E. Ras, J. Rech, B. Klein, and C. Hoecht, "Self-organized reuse of software engineering knowledge supported by semantic wikis," in *Proceedings of the Workshop on Semantic Web Enabled Software Engineering (SWESE)*, 2005, p. 76.

Towards High Quality Recommendations: A Goal-Oriented and Ontology-Based Interactive Approach

Ronaldo Gonçalves Junior, Robert Ahn, Tom Hill, Lawrence Chung

Department of Computer Science
The University of Texas at Dallas
Richardson, TX, USA
{ronaldo.goncalves, robert.sungsoo.ahn, chung}@utdallas.edu,tom.hill.fellow@gmail.com

Abstract—**Recommender systems are aimed to offer good recommendations when the search space is too big or even uncertain. For example, when deciding on a movie to watch or a restaurant to go. However, when users are not satisfied with such results, they might spend a considerable amount of time interacting with the system while being unsatisfied. Most commercial recommender systems seem to lack methods to understand user needs while guiding the user with an intelligent human-like interactive process towards a high-quality recommendation. In this paper, we present a goal-oriented and ontology-based interactive framework for high quality recommendations. Through a goal-oriented approach, the recommender system address and define user needs by using quality attributes, the so-called Non-Functional Requirements (NFRs), such as responsiveness, efficiency, accuracy of the recommendation, and so on. A list of potentially competing recommendations, pertaining to a user interest, are evaluated using machine learning (ML) algorithms, and, by following an interactive approach, system and user exchange information derived from the ontology in order to avoid problems related to uncertain, vague, or even incorrect, user inputs. Results from the assessment of a real online database (TMDb) combined with an experimental study shows that the proposed approach is able to guide the user towards high quality recommendations.**

Keywords- Recommender Systems, Interactive Systems, Non-Functional Requirements, Ontology, Goal-Orientated, Machine Learning

I. INTRODUCTION

Recommender systems are aimed to offer good recommendations to users (e.g., a good restaurant to go) and can be especially useful in scenarios where it is difficult for the user to evaluate options individually [1]. For instance, more than 500 hours of video are uploaded to YouTube every minute[1]. In the absence of a systematic approach, a vast amount of content that is potentially interesting to the user is likely to remain unexplored. Many commercial systems, including streaming platforms such as Amazon[2] and Netflix[3], have recommender systems in place to help users navigate through a seemingly boundless set of contents. Even though recommender systems are more common and popular, many challenges still require attention [2]. Some techniques have been proposed to better understand users' needs by making use of the communication

between a user and the system [3][4], but most recommender systems seem to lack methods to understand user needs while guiding the user with an intelligent, human-like, interactive process towards high quality recommendations.

In this paper, we present a goal-oriented and ontology-based interactive framework for high quality recommendations. System and user interact in order to increase the prospect of satisfactory recommendations. Through a goal-oriented approach, we consider important quality attributes, or the so-called Non-Functional Requirements (NFRs), such as responsiveness, efficiency, accuracy of the recommendation, and so on. In this approach, a list of potentially competing recommendations pertaining to a user interest are evaluated by using machine learning (ML) algorithms, with the necessary features coming from the ontology. Results from the assessment of a real online database (TMDb) combined with an experimental study shows that the proposed approach is able to guide the user towards high quality recommendations.

II. BACKGROUND

Recommendation algorithms are typically divided into three different approaches [5][6]: Collaborative Filtering (CF), Content-Based (CB), and Hybrid systems. CF is one of the most common types of recommender systems [7], where the similarity between users is utilized to build recommendations [8]. Traditional CB recommender systems utilizes the similarity between items to build recommendations and are less wide-spread [9], but have been proven to contribute to the research field [10]. Finally, Hybrid systems can provide other benefits that the previous types do not offer [11][12]. For instance, Hybrid recommender systems have been used to incorporate justification in recommendations to achieve customer acceptance and trust [13].

Some of the related work closest to our approach perform item selection as a step in a conversational process and the system inquiries about item attributions while waiting for user response [14]. Other interactive approaches have similar concepts for recommendations with different contributions [4][15], such as knowledge based systems that use facts and rules to improve recommendations [16][17]. However, to the best of our knowledge, there is no consideration of user needs in

---

a Goal-Oriented approach, where satisfactory recommendations are modeled using NFRs and further analyzed using measurable observations. Another important aspect is that there is limited consideration of domain-dependent ontologies within the interactive steps of the recommendation process.

III. The Proposed Approach

The process of the proposed approach consists of four steps: 1) *Model NFR Softgoals*, 2) *Build recommendations*, 3) *Explore ontology-based concepts*, and 4) *Interact with the user*. The following sections describe each of these steps in further detail.

A. Step 1: Model NFR Softgoals

The proposed approach uses quality attributes, or the so-called Non-Functional Requirements (NFRs), to define high quality recommendations. In the context of recommendations, high quality can be further decomposed into different NFRs in order to better identify user needs. More specifically, a high quality recommendation may be AND-decomposed into three different NFR Softgoals: an efficient recommendation, a responsive recommendation and an accurate recommendation. Each of these Softgoals can be further decomposed and connected to other NFRs. For instance, an efficient recommendation may help a responsive recommendation, but an accurate recommendation has some negative contribution to responsiveness. An example of a complete model for high quality recommendations is shown in Fig. 1. It is worth mentioning that it is possible to make refinements to the graph to include other quality attributes such as security, user-friendliness, cost, and so on.

Note that it is important to define how these requirements will be satisficed. Operationalization Softgoals are related to external entities, such as ML algorithm results and user interaction. Each Softgoal may be marked by the labels satisfied, weakly satisfied or denied based on these external observations. These labels will, subsequently, be propagated to other goals bottom-up [18] until high quality recommendation can be validated or invalidated.

B. Step 2: Build recommendations

Using a Machine Learning-Based approach, it is possible to create prediction models that will help identify which items the user might be interested in. Consequently, instead of showing a list with all available items, we can show a reduced list with items that exclusively pertain the interest of the user. The first step in building recommendations is the acquisition of user data. This may be done by using private databases, publicly available datasets, etc. If user data is not available, it is possible to perform a customer survey, purchase data from providers, and so on. After acquiring user data, it is important to verify that undesired information is removed, such as null values, unavailable entries, and so on. Last but not least, the right format must be ensured. The main input feature for the CF recommender system algorithm utilized in this paper needs to be in the form of {*<userID, itemID, rating>*}. This implementation of the CF algorithm includes the computation of the distance between different users, which can be done by using a Pearson Correlation score, Cosine function, among others [5]. In other

words, these scores measure how similar two users are. The scores for all users are stored in order to avoid repeated computation. The execution of the recommendation algorithm may take long periods of time, especially when dealing with considerably large datasets. However, by storing these values there is no need to re-execute the algorithm for the same data.
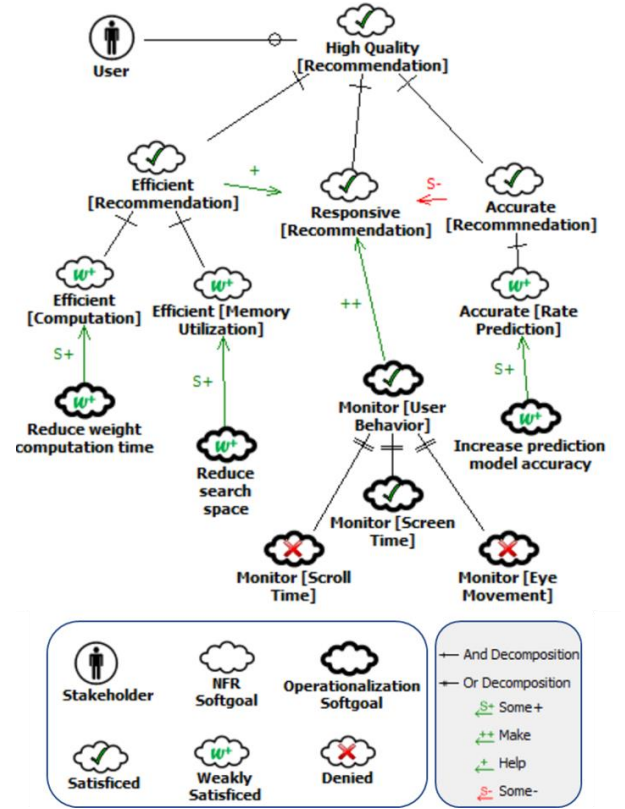
Fig. 1. A model for high quality recommendations.

C. Step 3: Explore ontology-based concepts

It is important to explicitly represented concepts such as User, Softgoals, and Recommendation Model in a domain-independent approach to avoid omissions while mapping Goal-Orientation (Step 1) and machine learning (Step 2). In addition, some concepts may be derived from a domain-dependent ontology. A complete set of concepts and their relationships can be found in Fig. 2. This ontology is domain-independent and can be used for various domains. In this step, we want to identify domain level concepts to guide the user for better recommendations. The movie domain is the most common domain for research in recommender systems [2]. For this reason, Fig. 3 shows portion of a domain-dependent diagram for the movie domain. The proposed approach uses each ontology concept as options to interact with the user. Each ontology concept is directly or indirectly connected to *Movie*, e.g., *Genre* and *Entertainment* respectively. Let *CN(a,b)* be the proposition that *a* is a conceptual neighbor of *b*, such that:

$$CN(a, b) \leftrightarrow \exists c[(a \in c) \land (b \in c) \land (c \neq M)] \quad (1)$$

The propositional statement (1) states that *a* is a conceptual neighbor of *b* *iff* there exists an arbitrary element *c* in the movie domain, such that both *a* and *b* are members of *c*, and *c* is not

Movie (*M*). For example, both Information and Entertainment and members of Genre, and Genre is not Movie (M). Thus, *CN(Information, Entertainment)* evaluates to *True*. The next section shows the usage of this definition during the interaction with the user.
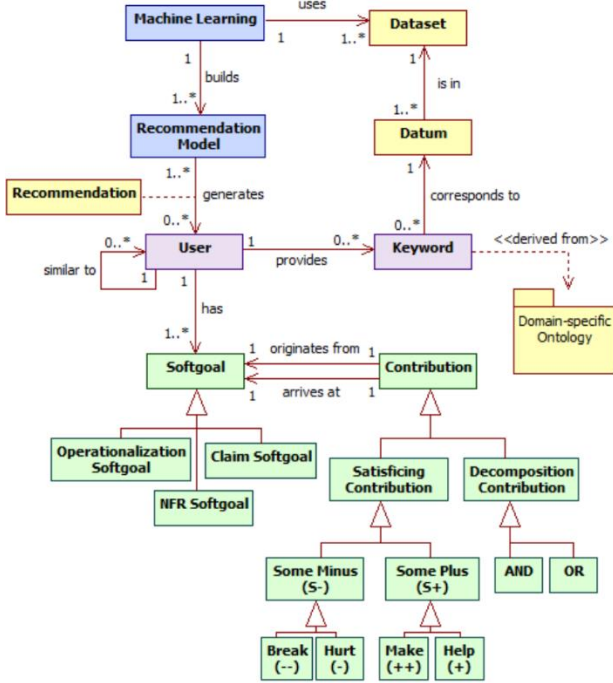
Fig. 2.   Domain-independent ontology diagram for the proposed approach.
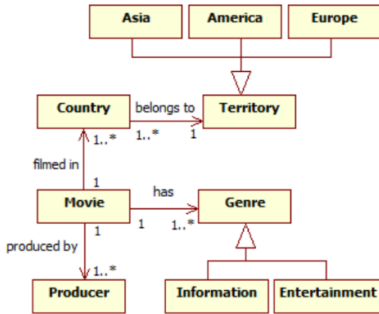
Fig. 3.   An example of ontology concepts for the movie domain.

D. Step 4: Interact with the user

By interacting with a user, a recommender system may be able to better understand the user needs. This step of the proposed approach generates options derived from the ontology and present them to the user towards more satisfactory recommendations. To illustrate this step for the movie domain, let us consider that a user is looking for movie recommendations which are similar to the movie *The Mission (1986)*. First, it is necessary to monitor the user behavior to identify when help is needed. Consider that the user defines a time threshold of 60 seconds for screen time, i.e., the time spent on a screen without performing any actions, such as clicking, scrolling, etc. When this threshold is reached, the system will attempt to gather additional information from the user and provides options for selection. For the movie domain, we present the following set of options: {*Actor, Award, Country, Genre, ..., Producer*}. Note that these are ontology-based options for the movie domain. Different options would be available for a different domain. To illustrate this step, let us assume that the user does not remember the title of the movie, but selects *Brazil* as *Country*. Note, however, that *The Mission (1986)* takes place in Argentina.

The system will filter recommendations from other countries, except Brazil, and goes back to monitoring. The user may provide information multiple times. However, if the user is not able to provide additional information and help is still needed, the system will suggest alternatives. At this point, the system will evaluate each given option and modify the search space to comprehend conceptual neighbors of each option, one at a time. By using the conceptual relationships computed in Step 3, the system adjusts the user input from *Brazil* to the following set: {*Argentina, Bolivia, Brazil, Chile, ..., Venezuela*}, which is the set of conceptual neighbors of *Brazil* with respect to *South America*. Now, instead of movies that took place in Brazil, we also include movie recommendations that took place in many other countries, including the correct country, Argentina. This step can be executed multiple times, and through each iteration, we go back to Step 1 of the proposed approach to rebuild the NFR model and check if we achieved the goal of high-quality recommendations.

IV.   AN EXPERIMENT FOR THE MOVIE DOMAIN

In this experiment, we monitor the user screen time and, by following the Step 1 of the proposed approach, we mark the corresponding Softgoal in Fig. 1 as satisficed and mark the others as denied. By following the label propagation process [18], since at least one of the sub goals for *Monitor user behavior* is satisfied, *Responsive recommendation* is marked as satisficed. It is important to mention that, at this point, we cannot verify if we achieved *Efficient recommendation* or *Accurate Recommendation*. In this case, we leave these goals unlabeled and proceed to the next step of the proposed approach.

The data used in Step 2 for this experiment is from a popular dataset (MovieLens [19]) that was collected from The Movie Database [4] (TMDb). This dataset contains more than 45 thousands of movies, 26 million ratings and 270,000 users. Note that, due to space limitations, we show here the results for the first iteration of the proposed approach by using the CF algorithm, implemented in Java. With the similarity between users stored, it is now possible to recommend movies to users. By following the Step 3 of the proposed approach, all conceptual neighbors for the movie domain are built and stored as interaction options. Finally, by following the Step 4 of the proposed approach, it is possible to see a significant reduction in the search space. For instance, if a user selects *Italian* as *Language*, the search space is reduced from 45,466 movies to 1,529. Multiple aspects are worth mentioning here. First, different users will have different results based on the interaction. Second, the recommender system does not recommend the entire dataset, only movies predicted to be satisfactory. However, the original input for the ML algorithm will be reduced, which narrows the recommendations

---

significantly and lowers the amount of training examples being used, decreasing the weight computation time.
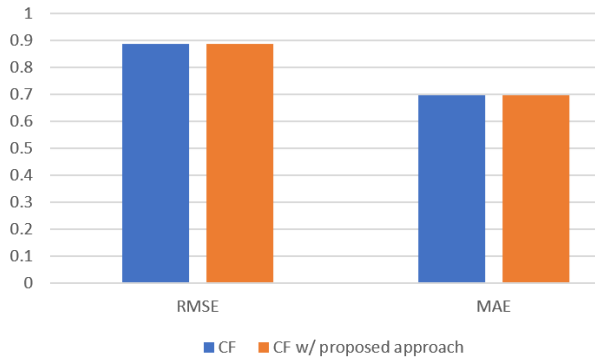
Fig. 4. CF algorithm results by accuracy (RMSE and MAE).

Once step 4 is completed, we go back to Step 1 to check if we achieved high quality recommendations in the NFR model. Since the search space and weight computation time were both reduced by a certain degree, we mark the corresponding Softgoals as weakly satisfied. Finally, for an arbitrary selection of users that rated movies, the first step of the proposed approach increased the accuracy of the results. Note that both Root Mean Squared Error (RMSE) and Mean Average Error (MAE) metrics were reduced by a considerably small margin, as shown in Fig. 4. Hence, *Increase prediction model accuracy* is marked as weakly satisfied. A complete iteration over the process of the proposed approach will result in a model fully labeled as shown in Fig 1. For this experiment, note that *High Quality Recommendation* is marked as satisfied after the label propagation process [18]. In other words, we can proceed to Step 2 once again to build more satisfactory recommendations. Note that this process can be repeated indefinitely, by check if *High Quality Recommendation* is satisfied or denied.

V. DISCUSSION AND FUTURE WORK

This work presented an approach that considers Non-Functional Requirements (NFRs) as key drivers for helping the user find good recommendations through an interactive system. Results from the assessment of an online database (TMDb) combined with an experimental study show that users are guided towards high quality recommendations by using the proposed approach. The work presented in this paper, to the best of our knowledge, is one of the first to propose the usage of a Goal-Oriented approach where satisfactory recommendations are modeled using NFRs and further analyzed by measurable observations. The outcomes of the proposed approach include: 1) a typical set of high quality goals and relationships between them, as well as ways to achieve them; 2) a domain-independent ontology for recommendations; and 3) an approach for the interaction between user and system to guiding the user towards more satisfactory recommendations.

Regarding our future work, we plan to to investigate ways to improve accuracy in a more impactful manner, as well as experimenting for different domains other than movies. Moreover, we intend to implement a tool to support the approach during all steps of the process for recommendations.

REFERENCES

[1] P. Resnick and H. R. Varian. Recommender systems. Commun. ACM, 40(3):56–58, March 1997.

[2] D. Jannach, M. Zanker, M. Ge, and M. Gröning. Recommender systems in computer science and information systems – a landscape of research. In Christian Huemer and Pasquale Lops, editors, E-Commerce and Web Technologies, pages 76–87, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[3] T. Mahmood and F. Ricci. Improving recommender systems with adaptive conversational strategies. In Proceedings of the 20th ACM Conference on Hypertext and Hypermedia, HT '09, pages 73–82, New York, NY, USA, 2009. ACM.

[4] L. McGinty and B. Smyth. On the role of diversity in conversational recommender systems. In Proceedings of the 5th International Conference on Case-based Reasoning: Research and Development, ICCBR'03, pages 276–290, Berlin, Heidelberg, 2003. Springer-Verlag.

[5] J. S. Breese, D. Heckerman, and C. Kadie. Empirical analysis of predictive algorithms for collaborative filtering. In Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence, UAI'98, pages 43–52, San Francisco, CA, USA, 1998. Morgan Kaufmann Publishers Inc.

[6] C. He, D. Parra, and K. Verbert. Interactive recommender systems: A survey of the state of the art and future research challenges and opportunities. Expert Systems with Applications, 56:9 – 27, 2016.

[7] M. Zanker, M. Jessenitschnig, D. Jannach, and S. Gordea. Comparing recommendation strategies in a commercial context. IEEE Intelligent Systems, 22(3):69–73, May 2007.

[8] P. Resnick, N. Iacovou, M. Suchak, P. Bergstrom, and J. R. Grouplens: An open architecture for collaborative filtering of netnews. In Proceedings of the 1994 ACM Conference on Computer Supported Cooperative Work, CSCW '94, pages 175–186, New York, NY, USA, 1994. ACM.

[9] S. B. Ticha, A. Roussanaly, A. Boyer, and K. Bsaïes. User semantic preferences for collaborative recommendations. In EC-Web, 2012.

[10] M. Gemmis, P. Lops, G. Semeraro, and P. Basile. Integrating tags in a semantic content-based recommender. In Proceedings of the 2008 ACM Conference on Recommender Systems, RecSys '08, pages 163–170, New York, NY, USA, 2008. ACM.

[11] R. Burke. Hybrid recommender systems: Survey and experiments. User Modeling and User-Adapted Interaction, 12(4):331–370, Nov 2002.

[12] I. Cantador, A. Bellogín, and P. Castells. A multilayer ontology-based hybrid recommendation model. AI Commun., 21(2-3):203–210, April 2008.

[13] P. Symeonidis, A. Nanopoulos, and Y. Manolopoulos. Providing justifications in recommender systems. IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans, 38(6):1262–1272, Nov 2008.

[14] C. A. Thompson, M. H. Göker, and P. Langley. A personalized system for conversational recommendations. J. Artif. Int. Res., 21(1):393–428, March 2004.

[15] L. Ardissono, G. Petrone, and M. Segnan. A conversational approach to the interaction with web services. Computational Intelligence, 20(4):693–709, 2004.

[16] A. Felfernig, G. Friedrich, D. Jannach, and M. Zanker. An integrated environment for the development of knowledge-based recommender applications. Int. J. Electron. Commerce, 11(2):11–34, December 2006.

[17] F. Ricci, L. Rokach, and B. Shapira. Introduction to recommender systems handbook. In Recommender systems handbook, pages 1–35. Springer, 2011.

[18] L. Chung, B. A Nixon, E. Yu, and J. Mylopoulos. Non-functional requirements in software engineering, volume 5. Springer Science & Business Media, New York, NY, USA, 2012.

[19] F. M. Harper and J. A. Konstan. The movielens datasets: History and context. ACM Trans. Interact. Intell. Syst., 5(4):19:1–19:19, December 2015.

SHAMROQ: Towards semantic models of regulations

Patrick D. Cook, Susan A. Mengal, Siva Parameswaran

Department of Mechanical Engineering, Department of Computer Science, Department of Mechanical Engineering
Texas Tech University, Lubbock, TX 79409−3104
patrick.d.cook@ttu.edu, susan.mengel@ttu.edu, siva.parameswaran@ttu.edu

Abstract— **Regulatory documents contain a rich set of provisions that requirement engineers must observe in software requirements. If a requirement engineer fails to accurately interpret or include the provisions in the software requirements, then a right, privilege, or obligation could be omitted or incorrectly applied – resulting in a violation. When a violation occurs, complaints are filed, penalties are imposed, and in some instances, the responsible party goes to prison; thus, this paper introduces SHAMROQ, a methodology to systematically acquire software requirements from regulations, and demonstrates the methodology using a section of the Health Insurance Portability and Accountability Act (HIPAA). SHAMROQ is applied to a case study to show that it is possible to use the basic activity pattern with modality, description logic, and Hohfeldian legal concepts to analyze, classify, and model the legal relationships to ascertain meaning, context, and structure.**

Keywords- Knowledge representation, Semantic Web, OWL, Resource Description Framework, SHAMROQ

I. INTRODUCTION

Regulations contain a rich set of provisions that requirement engineers must observe in software requirements [1]. However, if requirement engineers fail to accurately interpret or include provisions in software specifications, then a right, privilege, or obligation could be omitted or incorrectly applied – resulting in a violation [2]. When a violation occurs, a complaint is filed, a penalty is imposed, and in some instances, the responsible party goes to prison.

In fact, between April of 2003 and the end of January 2020, The U.S. Department of Health and Human Services (HHS) Office for Civil Rights (OCR) received more than 227,866 Health Insurance Portability and Accountability Act (HIPAA complaints about violations of the Privacy Rule. As a result, civil penalties of $116,203,582 were settled or imposed.[1]

Apart from over $116 million in penalties, building regulatory compliant software systems presents several challenges [3]. First, regulations may complement, overlap, or contradict at the federal, state, and local levels. Secondly, regulations are continually changing, plagued with ambiguity, and often accompanied by previous administrative rulings, reference handbooks, and other guidelines published to facilitate interpretation [4]. Third, the influence of case law (the interpretation of the law by the judicial process) over statutory law (the written law passed by the legislature) poses other challenges because the courts could add new interpretations to the statues from court rulings [5]. Additionally, regulations are notable for frequent references to other sections, also known as cross-references [6], and regulations contain domain-specific language or jargon – sometimes called "legalese" [1].

Despite these challenges, researchers offer several approaches to aid requirement engineers in building regulatory compliant software systems. Approaches included logic models [7, 8], extracting formal specifications from regulations [9], goal-oriented approaches [10], production rules [11], machine learning [12] and access control [13] . More recently, researchers are using semantic web technologies to aid requirement engineers in building regulatory compliant software systems [14].

Semantic web technologies are promising because they provide a common framework that facilitates interoperability across applications, organizations, and jurisdictional boundaries. Moreover, the semantic web offers a family of technologies that enable requirement engineers to create data stores, construct vocabularies, and write rules for dealing with data[2] . In this research, we leverage semantic web technologies, in particular, the Web Ontology Language (OWL) to aid requirement engineers in systematically acquiring software requirements from regulations.

The purpose of this descriptive, embedded, single-case study is to develop and validate the SHAMROQ methodology. At this stage in the research, SHAMROQ is generally defined as the systematic process to examine all words and phrases written in a regulatory document, classify patterns that correspond to Hohfeldian legal concepts, and model the regulations using the basic activity pattern with modality.

The remainder of this paper is organized as follows: Section 2 reviews the background and related work; Section 3 outlines the methodology; Section 4 describes the case study; Section 5 presents the findings; Section 6 examines the threats to validity; and Section 7 discusses the conclusion and future work.
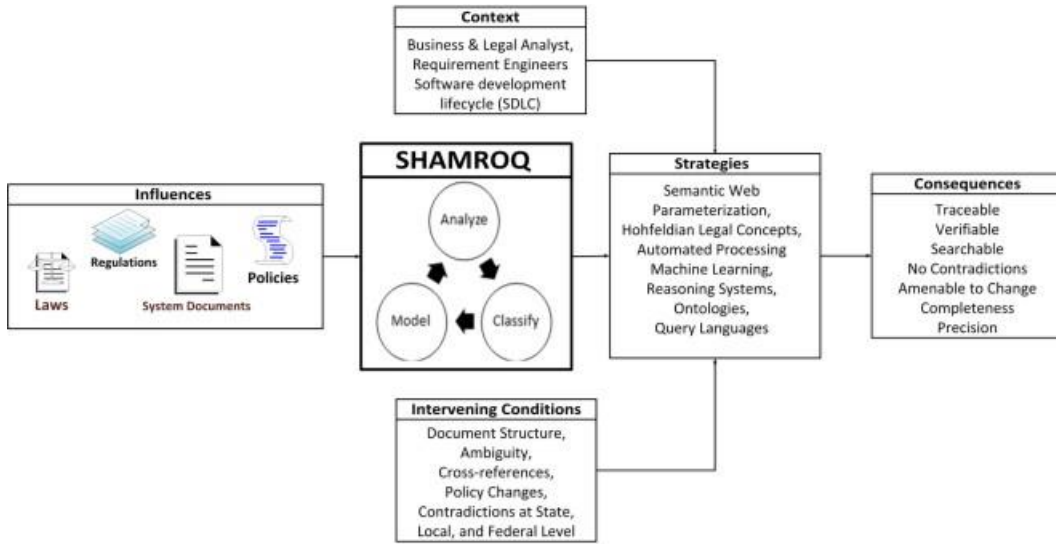
Figure 1. SHAMROQ - Theoretical Framework

II. BACKGROUND AND RELATED WORK

In this section, the background and related work in requirements engineering to extract software requirements from regulations is considered.

A. Background

Our working definition of requirements engineering, borrowed from Pamela Zave and generalized by Phillip A. Laplante, is the following: "Requirements engineering is the branch of engineering concerned with the real-world goals for, functions of, and constraints on systems. It is also concerned with the relationships of these factors to precise specifications of system behavior and to their evolution over time and across families of related systems" [15].

Laplante argues that software systems are bifurcated along functional (i.e., what the system does) and nonfunctional (how well the system does it under observable quality attributes) requirements. SHAMROQ provides a means to address both, however, this paper concentrates on nonfunctional.

Nonfunctional requirements are further broken down into design/implementation constraints, economic constraints, operating constraints, and political/cultural constraints. In this work, we will focus on the political/cultural constraint; i.e., the laws and regulations category, of nonfunctional requirements.

B. Related Work

Researchers use a variety of approaches to extract requirements from regulations and model them for system development. A comprehensive survey of the approaches is outlined by Otto [3]. Here, we concentrate on the related work that directly influences the ideas in our research: Semantic Parameterization [1, 2, 16-18], Frame-based [19, 20], and Production Rules [11, 21, 22].

Semantic Parameterization [1, 2, 16-18] is a process to represent a domain of interest in a structural way using Description Logic [23]. This process happens over three phases.

In phase 1, phrase heuristics are applied to natural language features, so that noun phrases, pronouns, intentional and extensional synonyms, and polysemes are differentiated. In phase 2, a dictionary is used to assign meaning to the words so that the domain is grounded. In phase 3, the dictionary is used to identify the tacit relationships to build a meta-model. As a result, Restricted Natural Language Statements (RNLS) are modeled using the basic activity pattern with modality. RNLS are derived from the original text and are restricted to one discrete activity. The RNLS is then represented by the basic activity pattern using one unary relation and two asymmetric, binary relations. The *unary relation* defines the root concept σ, while parameters use *associative relations* α, and values use *declarative relations* δ.

To further illustrate the point, Breaux uses the following RNLS as an example: "***The provider may share information***." Figure 2 depicts the *unary relation* σ (activity$_1$) in the shaded region. The *associated relations* α (activity$_1$, actor$_1$), α (activity$_1$, action$_1$), α (activity$_1$, object$_1$), is captured by the shaded region and oval. The *declarative relations* δ (actor$_1$, provider), δ (action$_1$, share), δ (object$_1$, information) is captured by connecting the directed arrow between the oval in the shaded region with the ovals outside of it.
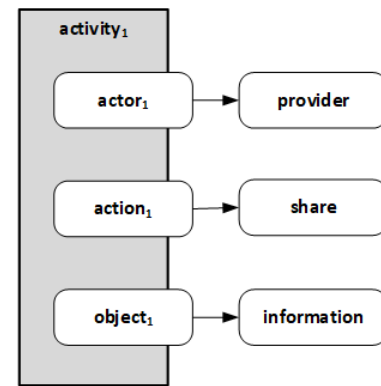
Figure 2. Basic Activity Model

Breaux contends that these three relations represent a complete parameterization process when all words and phrases written in a regulatory document are assigned or subsumed by a parameter or value. In instances where parameter values require concepts with additional parameters, then a second parameterization takes place with an additional associative and declarative relations.

In figure 3, we see the additional *associative relations α* (activity_1, purpose_1), and the additional declarative relations δ (purpose_1, activity_2) that represents the RNLS: "***The provider may share information to market services.***" Note, the preposition "to" is indicative of an additional *associative relation*.

This research builds on Semantic Parameterization and extends this work in the following ways. First, we codify the unary relation, as a root node, with a Hohfeldian legal concept [24, 25]. For example, the statement, "***The provider may share information to market services,***" uses the modal verb "may," to establish the root node "Privilege Activity."
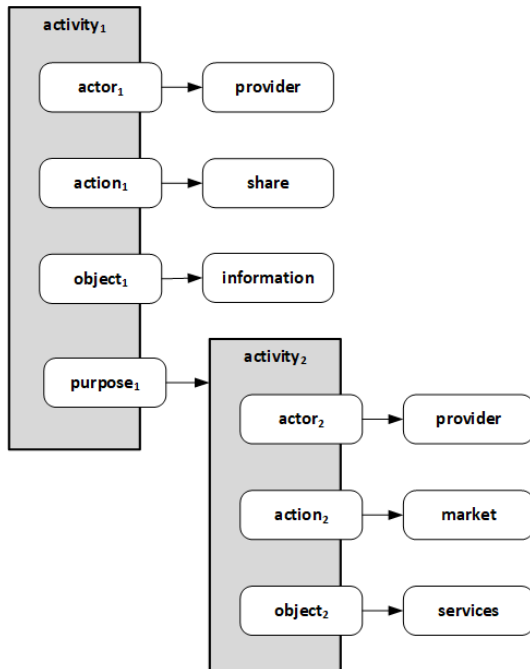
Figure 3. Basic Activity Model with Purpose

Secondly, we reduce the steps presented by Breaux [16] from UNLS, RNLS, Activity Model (3 steps) to UNLS, Activity model (2 steps). Third, we combine the associate and declarative relations and explicitly represent the activity as Resource Description Framework (RDF) *triples*. Next, we add a meta-data-model to the basic activity pattern that consists of the following attributes: a unique identifier, category, title, priority, and degree of necessity. We capture the triples in Figure 4.

Figure 4 shows the root node, PrivilegeActivity, and the associate relations as predicates (i.e., hasActor, hasAction, hasObject, and hasPurpose). Also depicted are the declarative relations as objects (i.e., provider, share, information, and PurposeActivity), and the meta-data-model as predicates.

The Frame-Based Requirements Analysis Method (FBRAM) [19, 20], is another means of extracting requirements from regulations. Breaux uses FBRAM to annotate the regulatory document manually in order for a tool to parse the annotations to extract the requirements. From this extraction, three artifacts are produced: an upper ontology, a context-free markup, and a document model.

The upper ontology is used to classify regulatory statements and consists of three concepts: a statement-level used to categorize individual regulatory statements, a phrase-level used to categorize individual regulatory phrases, and an abstract placeholder. The context-free markup describes the structure using concepts and logical connectives. The analyst uses the context-free markup to make some interpretation about the text and aligns the upper ontology in a manner that removes ambiguity.

The document model describes how the document is organized using a hierarchical representation. Moreover, the document model enables traceability between the requirements and the section, subsections, and paragraphs of the original regulations. The requirements are represented as HTML, in a table format, and contain the frame type; i.e., the type of requirement, the pattern, and the traceability information

Similarly, to FBRAM, we examine the natural language features of regulatory documents and map concepts to an ontology. However, our approach differs from FBRAM in that we extract software requirements directly from the regulations with natural language processing techniques and use Web Ontology Language Description Logics (OWL-DL) to express requirements as opposed to a document model to formalize the legal syntax. Moreover, our work focuses on all eight Hohfeldian legal concepts – not just rights and obligations.

The Production Rule Methodology [11, 21, 22] codifies four sections of the HIPAA Privacy Rule (§164.520, §164.522, §164.524 and §164.526) SWI-Prolog software application [21]. A production rule is a knowledge representation technique that is stated using horn clauses connected by logical operators [22]. Each rule consists of a two-part structure: an antecedent and a consequent. If the antecedent set of conditions resolves to true, then the consequent set of actions takes place. A collection of rules creates a knowledge base. The interaction with this knowledge base requires the top-level query using an inference engine; for example, backward chaining, as a reasoning strategy to execute on the rules base [22].

Prior to getting started, the production rule methodology requires an ontology and some legal text as input. Then, a preparatory step (Create Rule Patterns of Ontological Concepts) followed by two activities (Specify Production rules and Refactoring, respectively) takes place. In the preparatory step, production rule patterns are created from the ontology. The first activity, specify production rules, requires five steps.

In step 1, normative phrase analysis is used to classify rules based on the words and phrases used in the legislation. In step 2, identify rule parameters, the objective is to identify the subject of the statement, the relation the actor can change, the action the actor has the right or obligation to perform, and the source of the rule. In step 3, identify preconditions, the legal preconditions
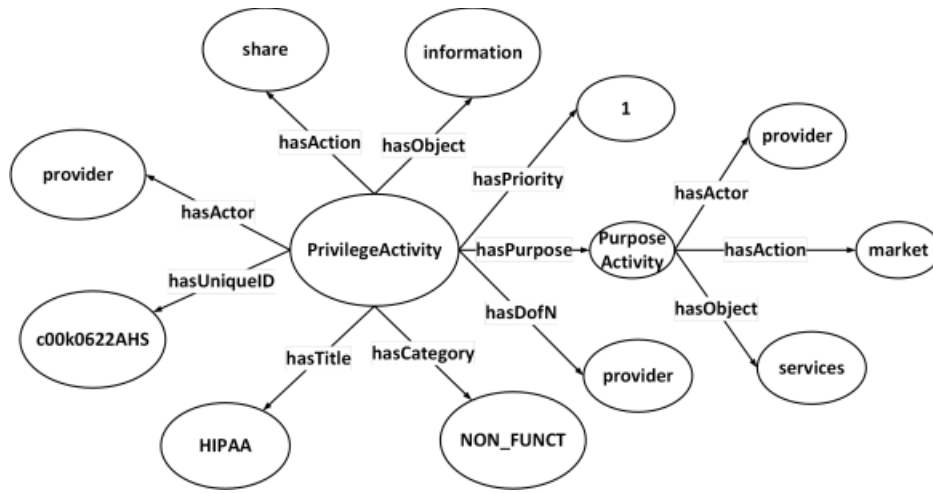
Figure 4. Semantic Web Parameterization

that enable the rule to be true are captured. In step 4, remove the rule, disjunctions, the statements in the legislation that are separated by an "or," are split into separate statements. Finally, in step 5, identify rules implied by the ontology, the software engineer may deduce other facts. After the completion of the first activity, a complete production rule model exists. However, the second activity, which refactors the rules base to remove duplicates, provides an opportunity to improve the design.

Like the production rule methodology, we use a multi-step process to extract requirements from regulations. We evaluate the natural language phrases and classify patterns that correspond to Hohfeldian legal concepts. Unlike the production rule methodology, we read the regulations directly from a file, segment the sentences from the regulations, tokenize the strings, tag the words with parts of speech, and chunk the sentences in a manner that can be modeled with OWL-DL.

III. METHODOLOGY

The research methodology is borne out of a constructivist worldview [26]. The philosophical idea around constructivism is to seek understanding of the world in its real-world context and is typically associated with qualitative research. Moreover, constructivists interpret meanings others have about the world or generate theory inductively as opposed to starting with theory. This induction is directly in contrast to the postpositivist worldview. The philosophical ideas of the postpositivist start with a theory, collect data to support or refute the theory, then revise, and are typically associated with quantitative research.

The constructivist worldview is necessary in this research because we seek to understand a phenomenon in its real-world context. Therefore, we adhere to the qualitative research design and the case study strategy of inquiry. Yin [27] describes a case study as an empirical method that takes an in-depth analysis of a contemporary phenomenon within a real-world context. Case study design includes four types: single-case embedded, single-case holistic, multi-case embedded, and multi-case holistic. The choice to use a single vs. multi-case study design is based on the number of cases in a study.

A case is a centralized phenomenon that exists within a real-life context. Within the context of software engineering, a case may range from a software development project to a process, product, team, technology, specific role, or policy [28]. Consequently, if only one case exists, then it is best to select a single case study. However, if two or more cases exist, then it is best to select the multi-case study design.

The choice to use an embedded vs. holistic is based on whether the case study has multiple units of analysis; i.e., subunits, or the case study examines the global nature of a phenomenon. Yin defines a unit of analysis as the actual source of information (e.g., a person, organizational document, or an artifact.)

Runeson [28] elaborates on the unit of analysis for software engineering as a project, group, or a decision. In short, a unit of analysis is the phenomenon within a case that is examined. On the other hand, to examine the global nature of a phenomenon means a holistic view of the case is assessed, and there are no subunits. Therefore, if the case study has multiple units of analysis, then one selects an embedded case study. If the case study looks at the nature of the whole phenomenon, then one selects a holistic approach.

This research employs an embedded, single-case study research design as defined by Yin and as recommended for software engineering by Runeson. This design is intentionally chosen with a long-range strategy in mind. We intend to leverage the results of this descriptive study to support future studies that will be prescriptive in nature. In the next section, we outline the case, units of analysis, research questions, theoretical framework, and strategy for mitigating threats.

IV. CASE STUDY

1) Case Selection

This study purposely selects HIPAA regulation §164.510 (a)(1), as illustrated in figure 2, because this specific provision of the regulation provides normative phrases, continuations, exceptions, and parameter values that are concepts with other parameters.

2) Units of Analysis

Yin defines the unit of analysis as the element within the case study for which the data is collected [27]. For software engineering research, Runeson stated that the unit of analysis might be some element of the project, the methodology, or some aspect of the ongoing development or maintenance [28]. Here, the unit of analysis consists of the natural language features (i.e., the keywords, sentences, phrases, and clauses) that form the parameters applied to the basic activity model.

3) Research Questions

Creswell declares that qualitative research questions are central with associated sub-questions [26]. A central research question takes a broad view and explores a central phenomenon. In this study, the following main central question outlines a broader view of the purpose statement to describe and explain the SHAMROQ methodology. To what extent can SHAMROQ be used to build a knowledge base? What is the SHAMROQ framework? How does the SHAMROQ methodology work in practice to extract requirements from regulatory documents?

4) Quality Assurance – Mitigating Threats to Validity

The quality of a case study is evaluated based on its ability to identify and mitigate threats to validity. Yin outlines four tests to assess the threats to validity. The four tests are construct, internal, external, and reliability. This study addresses three out of the four threats to validity. Internal validity applies to explanatory or causal studies and does not apply to descriptive or exploratory studies [27].

5) Theoretical Framework

SHAMROQ represents a contribution to the body of knowledge formalized by a systematic literature review (SLR) as outlined by Barbara Kitchenham [29] and meta-ethnography synthesis as outlined by Noblit and Hare [30]. A meta-ethnography synthesis (MES) uniquely and systematically defines a qualitative process for generating theory, which involves induction and interpretation. Meta-ethnography places emphasis on maintaining alignment with the original research articles and encourages researchers to extend beyond the original ideas of the research [30].

A clear finding of the SLR and MES was that several strategies are required to analyze, classify, and model regulations. SHAMROQ is a manifestation of those strategies. In the next section, we provide an overview of the SHAMROQ and will answer the central research question, to what extent can SHAMROQ be used to extract requirements from regulations.

V. FINDINGS AND DISCUSSION

This section presents the evolution of SHAMROQ and answers the research questions.

A. What is the SHAMROQ Framework?

SHAMROQ is an acronym that embodies the strategies used to build a knowledge base: semantic web parameterization, Hohfeldian legal concepts, Artificial Intelligence, Metadata Enrichment, Reasoning System, Ontologies, and Query language. Collectively, these seven core strategies provide requirement engineers a means to analyze, classify, and model functional and nonfunctional requirements using the semantic

web. As illustrated in figure 1, there are four main artifacts that influence SHAMROQ.

First, depicted in figure 1, are the laws that are established by Congress. Secondly, are the regulations (i.e., rules) that implement a statue or act as a guide. Third are the system documents that represent the stakeholder's needs, goals, deliverables, constraints, limitations, security, and performance criteria. Some examples of system documents are a Statement of Work (SOW), Software Requirement Specification (SRS), and Concept of Operations (CONOPS). Finally, are the policies that are an assortment of legal artifacts to include executive orders and presidential actions.

The context depicted in figure 1 shows the people involved and contends that they must be a part of the software development lifecycle. The intervening conditions include the characteristics that laws, regulations, and policy artifacts exhibit that make them both beneficial and problematic; in particular, the legal document structure, ambiguity, cross-references, and frequent changes.

The next construct of the framework, strategies, are reflective of the techniques to carry out the analysis, classification, and modeling of the artifacts that influence the framework – given the context and intervening conditions. As a result, the strategies yield a set of consequences that make legislative documents traceable, verifiable, searchable, absent of contradictions, complete, precise, and amenable to change.

Given the number of strategies to unpack, we narrow the scope of this paper to semantic web parameterization, Hohfeldian legal concepts, and ontologies. In the next section, we describe how the SHAMROQ methodology works in practice to extract requirements from regulations.

§ 164.510 Uses and disclosures requiring an opportunity for the individual to agree or to object.
(a) *Standard: Use and disclosure for facility directories—*
    (1) *Permitted uses and disclosure.* Except when an objection is expressed in accordance with paragraphs (a)(2) or (3) of this section, a covered health care provider may:
      (i) Use the following protected health information to maintain a directory of individuals in its facility:
        (A) The individual's name;
        (B) The individual's location in the covered health care provider's facility;
        (C) The individual's condition described in general terms that does not communicate specific medical information about the individual; and
        (D) The individual's religious affiliation; and
      (ii) Use or disclose for directory purposes such information:
        (A) To members of the clergy; or
        (B) Except for religious affiliation, to other persons who ask for the individual by name.

Figure 5. §164.510 (a)(1) of HIPAA

B. How does the SHAMROQ methodology work in practice to build a knowledge base of regulatory documents.

In this section, we describe the methodology that supports the framework. The methodology aids practitioners in providing a formal means to represent legal provisions, minimize

ambiguity, trace a requirement from its basic activity model back to its origin, and enrich the data model with metadata. What follows is a description of the automated process to analyze and classify and the manual process to model. To illustrate, we use the example of §164.510 (a)(1)(i)(A) of the HIPAA Privacy Rule, to describe each phase.

1) Analysis

In the analysis phase, we download the XML version of the regulations from the govinfo.gov website [3] and perform the following automated processing on the text using the Python programming language and Natural Language Toolkit (NLTK) [31]. First, the XML file is scanned to obtain the root node of the regulation. Secondly, preprocessing is performed on the document by traversing the root node and extracting the information associated with the node.tag, the node.attrib, and node.text. The analysis phase concludes when the results are stored in a python dictionary that contains the metadata, header, and body structure.

2) Classify

In the classification phase, the python dictionary is taken from the analysis phase and NLTK aids in performing sentence segmentation on the body of the python dictionary which contains the regulatory text. Next, NLTK helps to segment the regulatory text into sentences, to tokenize each sentence into words, and to tag each word with a part of speech. Finally, a grammar, illustrated in figure 6, helps to chunk the tagged words into eight categories: section, topic, noun phrase, exception, modality, conjunction, continuance, and action.

```
SECTION: {<\(><DT|CD><\)>?}
Topic: {<JJ>+<NNS><\.>}
EXCEPTION: {<IN><WRB><DT><NN.*>+}
NOUN_PHRASE: {<IN>?<DT>?<JJ.*>*<NN.*>+}
MODALITY: {<MD>}
CONJ: {<CC>}
CONTINUANCE: {<\:>}
ACTION: {<RB.*>*<VBZ>?<VB|VBN|VBZ>*<RB|RBR>?}
```

Figure 6. Grammar

Chunks, with a focus on noun phrases and verbs, are inspected manually in the output to identify an actor, action, object, target, exception, or constraint. A search for the modal verb is performed to assign to the root activity. Table 1 outlines the normative phrases [1] that align with Hohfeldian legal concepts[24, 25].

Table 2 captures the predicates that are aligned with the subject, verb, and object along with other attributes to support building a model that represent the regulation. The classification phase ends when all words and phrases written in a regulatory document are mapped to a category.

TABLE I. HOHFELDIAN CLASSIFICATION

Serial No.	Modality and Normative Phrase Correlation	
	Normative Phrase	*Concept*
1	has a/the right to, retains the right to	Right

TABLE I. (continued)

Serial No.	Modality and Normative Phrase Correlation	
	Normative Phrase	*Concept*
2	must, is required to, shall, may not, is prohibited, is subject to	Obligation
3	may, may elect not to, is not required to, requirement does not apply, is permitted to, at the election of, is not subject to	Privilege
4	does not have a right to	No-Right
5	authorize termination of, must obtain an authorization, may revoke, may terminate	Power
6	provide that <actor> will/must, obtain assurance	Liability
7	None	Immunity
8	may not authorize	Disability

The sentences are further examined manually to ascertain continuances [32]. Continuances are clauses that break into multiple constituent parts. The constituent parts are appended to the base clause and must be classified and modeled separately in the following manner [21].

TABLE II. ACTIVITY CLASSIFICATIONS

Serial No.	Classification Scheme	
	Predicate	*Description*
1	hasActor	The subject of the clause and answer the ICM question who
2	hasAction	The verb of the clause and answers the question what
3	hasModality	The auxiliary verb that corresponds to a Hohfeldian legal concept
4	hasObject	The verb of the clause and answers the question what
5	hasTarget	The person, place, or thing receiving an action
6	hasPurpose	The goal or objective of the clause
7	hasException	Contains keywords that express an exception
8	hasConstraint	Contains keywords that express a constraint
9	hasSource	Contains the legislation source section

a) § 164.510 (a)(1)(i)(B)

Except when an objection is expressed in accordance with paragraphs (a) (2) or (3) of this section, a covered health care provider may: (i) Use the following protected health information to maintain a directory of individuals in its facility: *The individual's location in the covered health care provider's facility;*

a) 164.510 (a)(1)(i)(C)

Except when an objection is expressed in accordance with paragraphs (a) (2) or (3) of this section, a covered health care provider may: (i) Use the following protected health information to maintain a directory of individuals in its facility: *The individual's condition described in general terms that does not communicate specific medical information about the individual; and*

---

[3] https://www.govinfo.gov/content/pkg/CFR-2019-title45-vol2/xml/CFR-2019-title45-vol2-sec164-510.xml
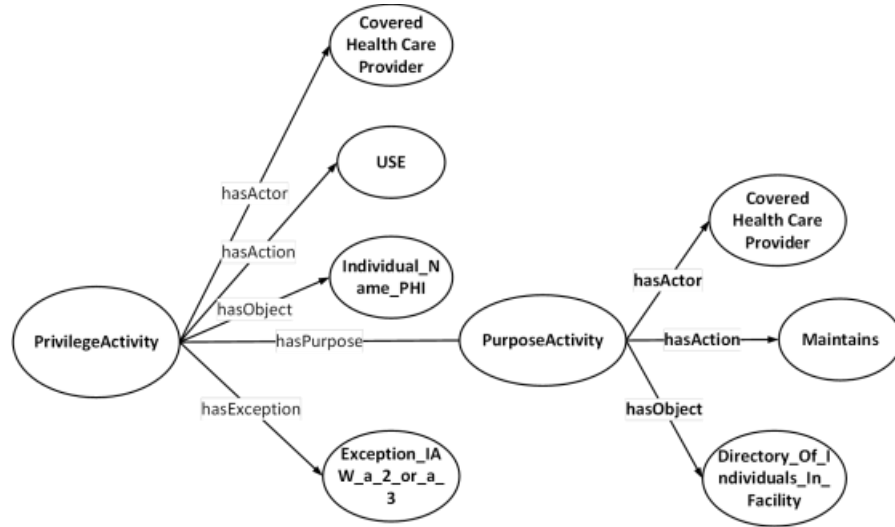
Figure 7. Semantic Web Parameterization

b) § 164.510 (a)(1)(i)(D)

Except when an objection is expressed in accordance with paragraphs (a) (2) or (3) of this section, a covered health care provider may: (i) Use the following protected health information to maintain a directory of individuals in its facility: *The individual's religious affiliation; and*

3) Model

In this phase, the output of the classification phase is inspected manually and protégé [33] is used to represent an organized, logical representation of concepts and categories using semantic web parameterization. The root activity contains the assignment based on the modal verb in the normative phrase. To provide more clarity, we continue with the example of § 164.510 (a)(1)(i)(A).

In figure 7, the illustration shows the results of the Semantic Web Parameterization process, in particular, the assignment of the root activity $\sigma'(\text{PrivelegeActivity}_0)$. The root activity is a privilege according to the phrase, "a covered health care provider may:" A look in Table 1, serial 3 shows the modal verb "may" maps to a privilege – the root activity.

The combined associate and declarative relations become the following triples: $\alpha'(\text{privelegeActivity}_0, \text{hasActor}_0, \text{some Health care Provider})$, $\alpha'(\text{privelegeActivity}_0, \text{hasAction}_0, \text{some Use})$, $\alpha'(\text{privelegeActivity}_0, \text{hasObject}_0, \text{some Name Individual Activity})$, $\alpha'(\text{privelegeActivity}_0, \text{hasPurpose}_0, \text{some Purpose Activity})$, $\alpha'(\text{privelegeActivity}_0, \text{hasException}_0, \text{some CFR 164 (a) (2)\_Activity})$, and $\alpha'(\text{privelegeActivity}_0, \text{hasException}_0, \text{some Exception\_IAW\_a\_2\_or\_a\_4})$

VI. THREATS TO VALIDITY

In this paper, we used section §164.510 of HIPAA as a descriptive, embedded, single-case study to develop and validate the SHAMROQ methodology. To assess the quality of a case study, Yin describes four criteria: construct validity, internal validity, external validity, and reliability. Internal validity is used for explanatory or causal case studies and not for descriptive or exploratory studies [27]. Therefore, internal validity is not tested here. In this section, we discuss construct validity, external validity, and reliability.

Construct validity assesses the correctness of operational measures by evaluating the means in which the researcher collects data, builds, or validates theory, and reports results [27]. Yin outlines three case study tactics to mitigate the threats to construct validity: use multiple sources of evidence, establish a chain of evidence, and use key informants to review the draft case study [27].

Although the case study in this paper uses one section of HIPAA, the basis of the SHAMROQ methodology is grounded in the literature and based on prior theories of semantic parameterization [16], description logic [23], and Hohfeldian legal concepts [24, 25]. We establish a chain of evidence by following our methodology and retaining copies of all artifacts. Lastly, the authors listed here reviewed the draft case study report.

External validity assesses whether the results are specific to the phenomenon under investigation or are applicable more generally [27]. We acknowledge several threats to external validity in our case study. First, we only examine one legal text within one regulatory domain - HIPAA. However, we purposely selected §164.510 because this provision provides normative phrases, continuations, exceptions, and parameter values that are indicative of legal text. Further studies modeling more legal texts across multiple domains will serve to validate and refine the methodology.

Reliability assesses whether the research can be independently verified, using the same methodology, to yield the same results [27]. Researchers independently verifying our case study are likely to use different grammar rules, identify a different combination of noun phrases, and identify a different ontology. Therefore, a small probability exists that the exact results of our case study could be replicated.

However, reliability is improved by evaluating reliability against our documented process and case-study database.

VII. CONCLUSION AND FUTURE WORK

In this paper, we introduced SHAMROQ, a methodology to examine all the natural language features in a regulatory document, group the features according to their shared characteristic, and model the features using the semantic web parameterization. We analyzed HIPAA regulation §164.510, which contained provisions with normative phrases, continuations, exceptions, and cross-references that are indicative of legal text. Our findings show that it is possible to use the basic activity pattern with modality, description logic, and Hohfeldian legal concepts to analyze, classify, and model the legal relationships to ascertain meaning, context, and structure.

Future work will include automating the manual steps to generate a semantic model from the noun phrases produced by the chunked grammar. Moreover, we will refine and validate SHAMROQ with a more extensive legal corpus across multiple regulatory domains and evaluate to what extent a multi-class classification machine learning algorithm can classify Hohfeldian legal concepts for semantic modeling.

REFERENCES

[1]   T. D. Breaux, M. W. Vail, and A. I. Anton, "Towards regulatory compliance: Extracting rights and obligations to align requirements with regulations," in *Requirements Engineering, 14th IEEE International Conference*, 2006: IEEE, pp. 49-58.

[2]   T. D. Breaux and A. I. Antón, "Analyzing regulatory rules for privacy and security requirements," *Software Engineering, IEEE Transactions on,* vol. 34, no. 1, pp. 5-20, 2008.

[3]   P. N. Otto, A. Antón, and others, "Addressing legal requirements in requirements engineering," 2007 2007: IEEE, pp. 5-14. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4384161[Online]. Available: files/272/abs_all.html

[4]   E. Kamsties, "Understanding ambiguity in requirements engineering," in *Engineering and Managing Software Requirements*: Springer, 2005, pp. 245-266.

[5]   M. J. Sergot, F. Sadri, R. A. Kowalski, F. Kriwaczek, P. Hammond, and H. T. Cory, "The British Nationality Act as a logic program," *Communications of the ACM,* vol. 29, no. 5, pp. 370-386, 1986.

[6]   J. C. Maxwell, A. I. Anton, and J. B. Earp, "An empirical investigation of software engineers' ability to classify legal cross-references," in *Requirements Engineering Conference (RE), 2013 21st IEEE International*, 2013: IEEE, pp. 24-31.

[7]   L. E. Allen, "Symbolic logic: A razor-edged tool for drafting and interpreting legal documents," *Yale LJ,* vol. 66, p. 833, 1956.

[8]   T. J. Bench-Capon, G. O. Robinson, T. W. Routen, and M. J. Sergot, "Logic programming for large scale applications in law: A formalisation of supplementary benefit legislation," in *Proceedings of the 1st international conference on Artificial intelligence and law*, 1987: ACM, pp. 190-198.

[9]   T. D. Breaux, A. I. Ant, and #243, "Mining rule semantics to understand legislative compliance," presented at the Proceedings of the 2005 ACM workshop on Privacy in the electronic society, Alexandria, VA, USA, 2005.

[10]   A. Antón, J. B. Earp, A. Reese, and others, "Analyzing website privacy requirements using a privacy goal taxonomy," 2002 2002: IEEE, pp. 23-31. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1048502[Online]. Available: files/279/abs_all.html

[11]   J. C. Maxwell, A. I. Ant, and #243, "The production rule framework: developing a canonical set of software requirements for compliance with law," presented at the Proceedings of the 1st ACM International Health Informatics Symposium, Arlington, Virginia, USA, 2010.

[12]   J. Cleland-Huang, A. Czauderna, M. Gibiec, and J. Emenecker, "A machine learning approach for tracing regulatory codes to product specific requirements," presented at the Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 1, Cape Town, South Africa, 2010.

[13]   M. J. May, C. A. Gunter, and I. Lee, "Privacy APIs: Access control techniques to analyze and verify legal privacy policies," 2006: IEEE, pp. 13-pp.

[14]   P. Slootweg, L. Rutledge, L. Wedemeijer, and S. Joosten, "The Implementation of Hohfeldian Legal Concepts with Semantic Web Technologies," *AI4J–Artificial Intelligence for Justice,* p. 65, 2016.

[15]   P. A. Laplante, *Requirements engineering for software and systems*. CRC Press, 2017.

[16]   T. D. Breaux and A. I. Antón, "Analyzing goal semantics for rights, permissions, and obligations," 2005: IEEE, pp. 177-186.

[17]   T. D. Breaux, A. I. Antón, and J. Doyle, "Semantic parameterization: A process for modeling domain descriptions," *ACM Transactions on Software Engineering and Methodology (TOSEM),* vol. 18, no. 2, p. 5, 2008.

[18]   T. D. Breaux and A. Anton, "Deriving semantic models from privacy policies," in *Policies for Distributed Systems and Networks, 2005. Sixth IEEE International Workshop on*, 2005: IEEE, pp. 67-76.

[19]   T. D. Breaux and A. I. Antón, "A systematic method for acquiring regulatory requirements: A frame-based approach," *RHAS-6), Delhi, India,* 2007 2007.

[20]   T. D. Breaux, *Legal requirements acquisition for the specification of legally compliant information systems*. ProQuest, 2009.

[21]   J. C. Maxwell and A. I. Anton, "A refined production rule model for aiding in regulatory compliance," North Carolina State University. Dept. of Computer Science, 2010.

[22]   J. C. Maxwell and A. I. Anton, "Developing production rule models to aid in acquiring requirements from legal texts," in *Requirements Engineering Conference, 2009. RE'09. 17th IEEE International*, 2009: IEEE, pp. 101-110.

[23]   F. Baader, I. Horrocks, C. Lutz, and U. Sattler, *Introduction to Description Logic*. Cambridge University Press, 2017.

[24]   W. N. Hohfeld, "Fundamental legal conceptions as applied in judicial reasoning," *The Yale Law Journal,* vol. 26, no. 8, pp. 710-770, 1917.

[25]   W. N. Hohfeld, "Some fundamental legal conceptions as applied in judicial reasoning," *Yale Law Journal,* pp. 16-59, 1913.

[26]   J. W. Creswell, *Research design: Qualitative, quantitative, and mixed methods approaches*. Sage, 2013.

[27]   R. K. Yin, *Case study research and applications: Design and methods*. Sage publications, 2017.

[28]   P. Runeson, M. Host, A. Rainer, and B. Regnell, *Case study research in software engineering: Guidelines and examples*. John Wiley & Sons, 2012.

[29]   B. Kitchenham and P. Brereton, "A systematic review of systematic review process research in software engineering," *Information and Software Technology,* vol. 55, no. 12, pp. 2049-2075, 12// 2013, doi: http://dx.doi.org/10.1016/j.infsof.2013.07.010.

[30]   G. W. Noblit and R. D. Hare, *Meta-ethnography: Synthesizing qualitative studies*. Sage, 1988.

[31]   E. Loper and S. Bird, "NLTK: the natural language toolkit," *arXiv preprint cs/0205028,* 2002.

[32]   W. M. Wilson, L. H. Rosenberg, and L. E. Hyatt, "Automated analysis of requirement specifications," 1997, pp. 161-171.

[33]   M. A. Musen, "The protégé project: a look back and a look forward," *AI matters,* vol. 1, no. 4, pp. 4-12, 2015.

On the Reuse of Knowledge to Develop Intelligent Software Engineering Solutions

José Ferdinandy Silva Chagas
Federal Rural University of Semi-Arid
Intelligent Software Engineering Group
ferdinandy@ufersa.edu.br

Luiz Antonio Pereira Silva
Federal University of Campina Grande
Intelligent Software Engineering Group
luizantonio@copin.ufcg.edu.br

Mirko Perkusich
Intelligent Software Engineering Group
mirko@virtus.ufcg.edu.br

Ademar França de Sousa Neto
Federal University of Campina Grande
Intelligent Software Engineering Group
ademar.sousa@virtus.ufcg.edu.br

Danyllo Albuquerque
Intelligent Software Engineering Group
danyllo.albuquerque@virtus.ufcg.edu.br

Dalton Cézane Gomes Valadares
Federal Institute of Pernambuco
Intelligent Software Engineering Group
dalton.cezane@caruaru.ifpe.edu.br

Hyggo Almeida
Federal University of Campina Grande
Intelligent Software Engineering Group
hyggo@dsc.ufcg.edu.br

Angelo Perkusich
Federal University of Campina Grande
Intelligent Software Engineering Group
perkusic@dee.ufcg.edu.br

Abstract—**Intelligent Software Engineering (ISE) is currently a hot topic in research. Besides being a promising field, it brings many challenges. Therefore, there is a need for guidelines to help researchers to build an ISE solution. The goal of this study is to identify patterns in developing ISE solutions. For this purpose, we analyzed 42 studies, using a thematic analysis approach, to understand how they reused knowledge and applied it to solve a SE task. As a result, we developed a thematic network composed of the main concepts related to knowledge reuse for ISE. Further, we identified that researchers use external and internal knowledge sources, and mostly rely on structured data to develop ISE solutions. Despite this, there are alternatives such as eliciting data from humans and literature to identify metrics or build knowledge-based systems. Overall, we concluded that there many research opportunities to guide the construction of ISE solutions.**

I. Introduction

The processing power of modern computers increased considerably, enabling Artificial Intelligence (AI) to reach streets, houses, cities, and people daily [22], [18]. Hence, AI advances bring new challenges and opportunities, such as automating or supporting the execution of Software Engineering (SE) tasks [27], [10]. On the other hand, bringing AI systems to the market also brings challenges that can be addressed by applying SE. As a consequence, recently, the field denominated as Intelligent Software Engineering (ISE) has emerged. ISE is an ambidextrous field focusing on (i) applying intelligent techniques to solve SE problems and (ii) using SE to improve AI systems [53], [44]. In this paper, we focus only on (i), which, itself, is not a very recent phenomenon dating back to the 1980s [37]. As the definition for what is an *intelligent technique*, we follow Perkusich et al. [44], in which this term was defined as: "the exploration of data (from digital artifacts or domain experts) for knowledge discovery, reasoning,

learning, planning, natural language processing, perception or supporting decision-making".

In industry, companies such as Facebook and Amazon have been applying intelligent techniques (i.e., search-based algorithms) to solve SE problems [41]. In academia, it is a hot topic [44]. For instance, researchers have proposed the application of NLP to manage requirements [28] and the use of natural-language-based chatbots to improve the productivity of developers [25]. Moreover, through ML, the researchers can use accessible software repositories, with a lot of available data, to continuously learn and improve the software reuse [50]. Perkusich et al. [44] performed a systematic literature review on ISE in the context of agile software development and highlighted the following research themes: Search-Based Software Engineering (SBSE) [19], machine learning for SE [55], recommender systems for SE [15], Bayesian networks for SE [35], software analytics [34], Big Code [2] and decision analysis for SE [56]. Among the SE problems that researchers are addressing with intelligent techniques, we can list: Effort Estimation ([48], [51], [13], [23]), Risk Management ([9]), Software Testing ([29], [24], [42], [31]), Team Formation ([30], [7]), and Requirements Engineering ([47], [26], [38], [46]).

Developing an ISE solution is a complex task because it demands knowledge regarding the SE task at hand and intelligent techniques, and, to the best of our knowledge, there are few proposals of conceptual models or general guidelines to develop ISE solutions. The literature presents guidelines for ISE subfields such as data mining for software engineering [54], [17], machine learning for software engineering [33], Search-Based Software Engineering [20], and data-driven solutions for agile projects [10]. It also presents guidelines for applying intelligent techniques for general purposes, such as building Bayesian networks [39], [32]. Despite having their value,

the existing studies focus on a specific intelligent technique. The problem is that defining the intelligent technique should not be the starting point of defining an ISE solution. The solution designer only selects the intelligent technique to be applied after evaluating the existing available knowledge (e.g., data stored in CASE tools or repositories) and the software engineering problem to be tackled (e.g., estimate effort for a given task). Therefore, to help in the early stages of building an ISE solution, we argue that there is a need for general guidelines.

To address this need, we analyzed 42 studies, identified by Perkusich et al. [44], that applied intelligent techniques to several SE tasks to identify patterns and provide a holistic view on how to develop ISE solutions from the perspectives of Knowledge Management (KM) and reuse-driven software engineering.

This paper synthesizes our findings by presenting a thematic network and the identified patterns on how the applied intelligent techniques relates to the reused knowledge. Further, it discusses the implications for research and practice. The rest of the paper is structured as follows. Section II presents the applied methodology to perform the thematic analysis. Section III describes the conceptual model. Section IV discusses the model development challenges, application, and impact. Section V lists main threats to validity. Finally, Section VI presents our final remarks, emphasizing the research contribution and limitations, and suggesting future works.

II. RESEARCH METHODOLOGY

The goal of this study is to identify patterns in developing ISE solutions. For this purpose, we model the problem of developing ISE solutions from the perspective of knowledge-reuse, in which we assume that an intelligent technique reuses data, information, or knowledge, which might be available through digital artifacts or domain experts, to solve SE problems. Given this, we defined the following research questions:

- RQ1 - How is knowledge reused in the context of ISE?
- RQ2 - What is the relationship between the type of reused knowledge and the applied intelligent technique?

RQ1 focuses on classifying existing ISE solutions in terms of the type of knowledge sources used and, if the case, what are the knowledge transformation techniques employed by researchers to feed intelligent technique algorithms.

RQ2 focuses on identifying patterns between the type of reused knowledge and the applied intelligent techniques. The answer to this research question might indicate trends, which might serve as guide researchers and practitioners interested in developing ISE solutions.

To answer the research questions, we employed a thematic analysis approach following the guideline proposed by Cruzes and Dyba [8]. The guideline proposes five research steps: (i) data extraction, (ii) code data, (iii) translate codes into themes, (iv) create a model of higher-order themes, and (v) assess the trustworthiness of the synthesis.

As the data source for the first step (i), we used a subset of the studies reported by Perkusich et al. [44], which identified intelligent techniques applied to agile software development.

Despite restricting the scope to agile, Perkusich et al. [44] report ISE solutions for diverse SE tasks such as effort estimation, requirements prioritization, and risk management; and using digital artifacts and humans expertise as knowledge sources. Therefore, we judged that analyzing the ISE solutions reported by Perkusich et al. [44] as being sufficient, given the scope of this study.

To assure that our results are based only on high-quality studies, guaranteeing the trustworthiness of the synthesis (step v), we filtered the 104 papers following the quality scoring performed by Perkusich et al. [44]. Perkusich et al. [44] used the instrument proposed by Dyba and Dingoyr [11] to assess the quality of the studies. We present the quality criteria in what follows.

1) Is the paper based on research (or is it merely a "lessons learned" report based on expert opinion)?
2) Is there a clear statement of the aims of the research?
3) Is there an adequate description of the research context?
4) Was the research design appropriate to address the aims of the research?
5) Was the recruitment strategy appropriate to the aims of the research?
6) Was there a control group with which to compare treatments?
7) Was the data collected in a way that addressed the research issue?
8) Was the data analysis sufficiently rigorous?
9) Has the relationship between the researcher and participants been considered to an adequate degree?
10) Is there a clear statement of findings?
11) Is the study of value for research or practice?

For each quality criteria, Perkusich et al. [44] rated the studies using a boolean scale in which "1" means "yes" and "0" means "no". Therefore, the quality score ranges in the interval [0, 11], which is composed only of Integer numbers. We only evaluated studies with a quality score equal to seven or higher, resulting in 42 studies. The complete list of evaluated study is made available here[1].

After having identified the 42 studies to be used as our data source, one researcher analyzed them and extracted publication details (e.g., title and year), the applied intelligent technique, SE task, level of automation [12] and segments of text relevant given our research questions. Afterward (step ii), each segment of text was analyzed and labeled by a researcher, generating a set of codes. We used an integrated approach, in which we defined a "start list" of codes based on our expertise in the field, but we remained open for new concepts that could become apparent. Another researcher checked the coded segments to avoid researcher bias.

Examples of codes defined a priori are the type of knowledge source (i.e., tacit, explicit, or both), and type of data (i.e., structured, non-structured, both). Conversely, as an example of a code that became apparent during data analysis was techniques to "transform" unstructured data into structured data such as text mining, ontology, and qualitative analysis. We discuss these concepts in Section III.

---

[1]https://bit.ly/2Q596MK
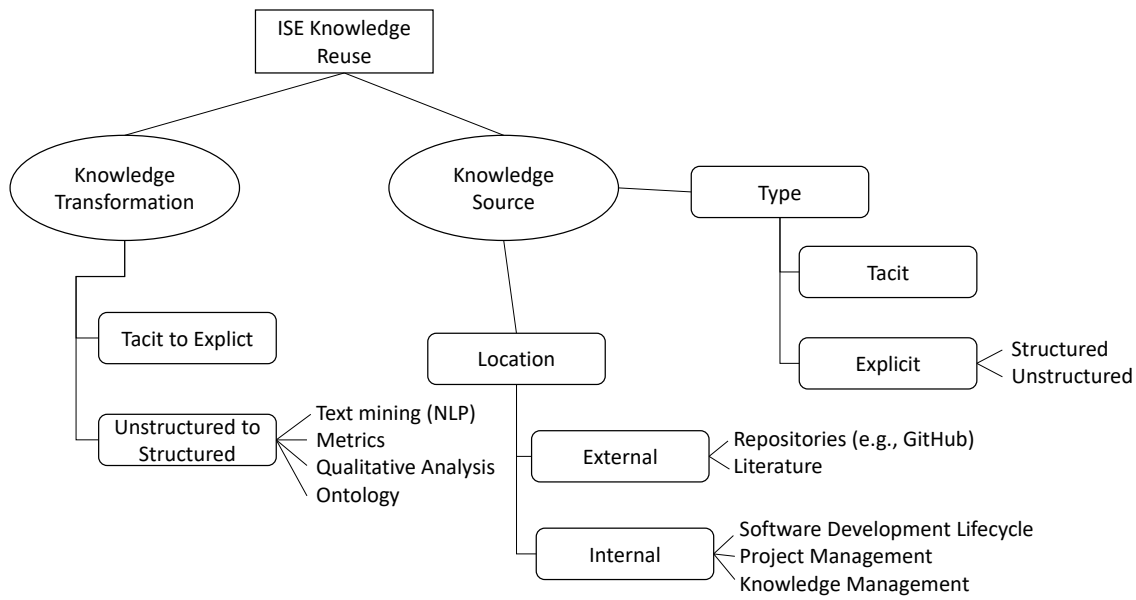
Fig. 1. ISE Knowledge Reuse thematic network

To define and structure the themes (steps iii and iv), the researchers analyzed the codes during workshops. At the end of this process, we developed a thematic network [3], organizing the concepts related to knowledge reuse for ISE. Afterward, we used the identified themes to classify the studies and analyze our research questions.

III. Thematic Network

This section discusses the thematic network, shown in Figure 1, that resulted from analyzing the 42 papers from Perkusich et al. [44], as discussed in Section II. The thematic network focuses on structuring the concepts related to knowledge reuse in the context of ISE. Therefore, the Global Theme (represented as a rectangle in Figure 1) encompasses the ISE solutions reuses knowledge.

During our analysis, two middle-order themes (represented by ellipsis in Figure 1) emerged: *Knowledge Source* and *Knowledge Transformation*. Every ISE solution, in some way or another, uses knowledge for a SE task. The *Knowledge Source* theme represents the possible types of sources in which the solution designer or algorithm might collect the necessary knowledge. We further refined this theme by identifying the themes *Type* and *Location*.

The theme *Type* refers to the two possible types of knowledge described in classical KM literature: *Tacit* or *Explicit*. Tacit knowledge refers to knowledge that is only stored in the minds of stakeholders (e.g., programmers, software engineers, and project managers). Explicit knowledge refers to knowledge that is codified and stored in digital (or physical) artifacts. There are two types of explicit knowledge, *Structured* and *Unstructured*. We defined that structured data refers to data high-organized and easily processed by a machine (e.g., relational database search). Conversely, unstructured data cannot be processed using conventional tools. In our context, mostly,

unstructured data refers to text (e.g., requirements, system logs, and source code) but could include audio and video, for instance.

The *Location* theme characterizes where the necessary knowledge might be found. For this theme, we identified two options: *External* and *Internal*. An external source refers to sources that are external to a given organization, such as a repository and the literature. Many researchers in data-driven ISE use repositories such as GitHub [16] for data mining, but the literature is an important source of knowledge for ISE. For instance, Hearty et al. [21], Perkusich et al. [45] and Freire et al. [14] identified features for their proposed models, partially, based on information collected from the scientific and grey literature.

Internal sources refer to knowledge that is available within an organization. It is the case for data produced during the Software Development Lifecycle, Project Management, and Knowledge Management activities. Given this, it is vital to notice that there are cases in which the necessary data is not readily available to solve de SE task at hand. In these cases, the ISE designer must develop tools to collect such data and integrate it them existing processes followed by the organization or evaluate the possibility of transforming existing knowledge into usable data for intelligent techniques, which is discussed in what follows.

The *Knowledge Transformation* theme refers to transforming knowledge that is available, but not ready to be used for ISE. It is the case when we have unstructured or tacit knowledge that we wish to use. In the case of unstructured data, it is necessary to transform it into structured. We identified such as text mining (e.g., Natural Language Processing) [28], the use of software metrics [36], qualitative analysis (e.g., coding) [49] and ontology [6] for this purpose. In the case of having tacit knowledge, it must be transformed into explicit. This process

Theme	%	Freq.	Distribution
Intelligent Technique			
Search and optimization	21%	9	
Reasoning under uncertainty	19%	8	
Mathematical model	14%	6	
Multiple Criteria Decision Analysis	7%	3	
Machine learning	21%	9	
Rules	5%	2	
Others	12%	5	
Level of Automation			
3	45%	19	
4	55%	23	
Non-Structured to Structured			
None	10%	4	
Metrics	72%	28	
Text mining (NLP)	5%	2	
Qualitative analysis (	8%	3	
Others	13%	5	
Uses tacit knowledge?			
No	62%	26	
Yes	38%	16	
Number of experts			
0-10	89%	40	
11-50	4%	2	
> 50	7%	3	
External Type			
Repository	45%	19	
Scientific Literature	38%	16	
None	14%	6	
Internal type			
Software Development Process	31%	13	
Project Management Process	62%	26	
Knowledge Management Process	2%	1	
None	10%	4	
Data type			
Only structured	83%	35	
Both	17%	7	
Only unstructured	0 %	0	

Fig. 2. Frequencies of Themes.

might transform tacit knowledge structured or unstructured. In the latter case, it is necessary to transform it into structured. For instance, Perkusich et al. [43] elicited knowledge from 46 Scrum experts through the Delphi method and an online survey to construct a Bayesian network for assessing Scrum projects.

IV. DISCUSSION

This section discusses the research questions (see Sections IV-A and IV-B) presented in Section II and the implications for research and practice (see Section IV-C).

A. ISE Knowledge Reuse

We used the thematic network presented in Figure 1 to classify the studies and identify the trends on knowledge reuse for ISE. Figure 2 presents the frequencies for each theme, as they were extracted from the studies. In what follows, we discuss each of the results for each of the basic themes (represented as rectangles with rounded borders in Figure 1 presented in the thematic network.

For the theme *Location*, 36 (86%) studies used some form of external knowledge source, while 42 (95%) used internal sources. Regarding the studies that relied on external sources, 19 studies used data from a repository and 16, from the scientific literature. Regarding the studies that relied on internal sources, 13 studies collected data from artifacts produced during the Software Development Lifecycle, 26 from artifacts produced by the Project Management Process, and only 1

from Knowledge Management Processes. The collected data indicates that researchers when developing ISE solutions, look for wherever places necessary to find data. Despite this, care should be taken when deploying ISE solutions in practice, because having different data sources raises the complexity in operating and maintaining them.

For the theme *Type*, 16 studies used tacit knowledge in the development cycle of the ISE solution, mostly (89%) eliciting it from ten or fewer experts. For the development cycle, we included a potential evaluation of the developed ISE solution by humans. Therefore, we considered that studies that developed expert systems such as Perkusich et al. [45], Odzaly et al. [40] and data-driven studies that evaluated their solution with humans (e.g., Chaves-González et al. [4]) equally. The reasoning applied is that either way, the tacit knowledge of humans was used to develop the ISE solution. It is essential to notice that we did consider here human knowledge regarding the intelligent technique itself (e.g., knowledge to define the fitness function of a genetic algorithm), but only related to the SE task at hand.

Further, 35 (83%) used only structured data, while 7 (17%) used both types of data (i.e., structured and unstructured). For instance, Hearty et al. [21] used only structured data (i.e., metrics and rules in the form of probability functions) to build a Bayesian network to predict the velocity of an XP team. Conversely, Lucassen et al. [28] presents a tool that uses metrics (i.e., structured data) as indicators of a well-written user story and processes user stories (i.e., unstructured data), calculating a quality score for them.

Regarding the theme *Knowledge Transformation*, we observed that the most popular form to transform tacit knowledge to explicit is in the form of questionnaires such as done by Perkusich et al. [43]. Further, regarding the transformation of unstructured to structured data, we observed that the most popular procedure is to use software metrics (28 studies). For instance, Abouelela and Benedicenti [1] define a set of metrics and use them to build a Bayesian network to predict the velocity and delivered quality of an XP team. A few studies used Qualitative Analysis (3) and Text mining (2). For instance, Lucassen et al. [28] process text from user stories using Natural Language Processing algorithms. Therefore, we identify a pattern, in the evaluated studies, to use software metrics as the preferred means for knowledge transformation.

B. Relationship between type of reused knowledge and applied intelligent technique

We triangulated the results of our classification (see Section IV-B with the data collected by Perkusich et al. [44] regarding the types of intelligent techniques applied by the studies. As presented in Figure 2, 18 studies focus on, necessarily, data-driven solutions, this is the case for *Search and Optimization* and *Machine learning*. The remaining intelligent techniques might be applied as a result of eliciting expert knowledge or exploring digital artifacts. For instance, it is the case for Bayesian networks, which can be constructed solely based on expert knowledge, available data, or both. Therefore, as expected, if there is enough structured data, any of the intelligent techniques presented by Perkusich et

al. [44] might be used. Unfortunately, in practice, most of the artifacts produced by the software development process are unstructured; the case for requirements, test cases, source code, and system logs. In these cases, researchers have used tools to process unstructured data extracting metrics (e.g., Chidamber and Kemerer metrics [5]) from it automatically or processing them using text mining algorithms.

In the cases of not having enough data, there are two alternatives: (i) elicit data from humans or literature to identify metrics or (ii) build knowledge-based systems. For the first case, after identifying the metrics that are crucial to solve the SE task at hand, it is necessary to develop tools to, ideally, collect them automatically during software development or project management activities. Given that the necessary data is available, a data-driven intelligent technology such as Machine Learning can be used to infer new knowledge or make predictions, for instance. The second option is to extract knowledge from experts and develop, for instance, an expert system using a Bayesian network or a Rules-based system (e.g., production rules). In this case, ideally, it is necessary to develop a tool that collects the input automatically or from humans to infer whatever is necessary (e.g., estimate effort for a given task).

C. Implications for research and practice

This study has several implications for research and practice. For research, we have mapped how knowledge is used for ISE solutions and identified patterns on how the type of reused knowledge relates to the applied intelligent technique. The reported information might guide researchers to develop ISE solutions having a more holistic view of their development process. Despite this, the analyzed studies focus on supporting decision-making, not having a high level of automation, using the classification described in Feldt et al. [12]. Therefore, we believe that there is a need to further refine the presented thematic network by analyzing studies with higher levels of automation. Further, there is a need for more studies to define guidelines for researchers in building ISE solutions, through the form of checklists, catalogs, taxonomies or reference models; especially, focusing on the early stages of developing an ISE solution, which, usually, relies on evaluating the available knowledge to solve the SE task at hand.

Also, this study showed that researchers, when building ISE solutions, rely on publicly available repositories. The implications for this point are twofold: first, it demonstrates the importance of having data available to build ISE solutions, and second, to be able to validate and compare ISE solutions focusing on the same SE task.

For practitioners, this study shows how knowledge is reused by ISE solutions and can be an inspiration for them to use tools and participate in research that helps to make explicit knowledge available. As a consequence, better ISE tools can be developed that can, potentially, make them more efficient.

V. Threats to validity

This section discusses this study's threats to validity following the classification proposed by Wohlin et al. [52]: construct, internal, conclusion, and external validity.

- *Construct validity*: we analyzed the studies following a thematic analysis approach, in which multiple researchers participated to avoid bias. Despite this, it is possible that the resulting thematic network (Figure 1) and extracted data (Figure 2) are not representative of the are due to subjective bias.
- *Internal validity*: to assure credibility in our findings, multiple researchers checked the extracted coding, themes, and the data presented in Figure 2.
- *Conclusion validity*: since we classified the study to identify patterns using the developed thematic network, there is the risk that, since there is a threat to the construct validity of the thematic network, it influenced the extracted data and, consequently, our conclusions regarding the relationship between concepts.
- *External validity*: Moreover, since the analyzed studies focus on supporting decision-making, they do not represent all types of ISE solutions. Therefore, the constructed thematic network might not be representative of ISE solutions with higher levels of automation. Despite this, we believe that including ISE solutions with higher levels of automation might identify more basic terms for high granular, but would not have impact middle-order themes, since they follow from classic KM concepts.

VI. Conclusion

In this study, we explored patterns in developing ISE solutions, focusing on knowledge reuse by analyzing 42 papers. As a result, we developed a thematic network that relates the main concepts in this topic. Further, we identified that researchers use external and internal knowledge sources, and mostly rely on structured data to develop ISE solutions. Despite this, we showed alternatives, such as eliciting data from humans and literature to identify metrics and build knowledge-based systems (e.g., expert systems) when structured data is not readily available to be used for solving a SE task.

The main limitation of the study is only having evaluated ISE solutions that focus on supporting decision-making. Further, the study also identifies several opportunities for future work, including refining the thematic network by analyzing ISE solutions with higher levels of automation and defining guidelines for researchers to build ISE solutions, especially, giving instructions on the early-stages process of an ISE solution conceptualization.

References

[1] M. Abouelela and L. Benedicenti. Bayesian network based xp process modelling. *International Journal of Software Engineering and Applications*, 1(3):1–15, 2010.

[2] M. Allamanis, E. T. Barr, P. Devanbu, and C. Sutton. A survey of machine learning for big code and naturalness. *ACM Computing Surveys (CSUR)*, 51(4):81, 2018.

[3] J. Attride-Stirling. Thematic networks: an analytic tool for qualitative research. *Qualitative research*, 1(3):385–405, 2001.

[4] J. M. Chaves-González, M. A. Pérez-Toledano, and A. Navasa. Software requirement optimization using a multiobjective swarm intelligence evolutionary algorithm. *Knowledge-Based Systems*, 83(Supplement C):105 – 115, 2015.

[5] S. R. Chidamber and C. F. Kemerer. A metrics suite for object oriented design. *Software Engineering, IEEE Transactions on*, 20(6):476–493, 1994.

[6] R. Colomo-Palacios, I. González-Carrasco, J. L. López-Cuadrado, and Á. García-Crespo. Resyster: A hybrid recommender system for scrum team roles based on fuzzy and rough sets. *International Journal of Applied Mathematics and Computer Science*, 22(4):801–816, 2012.

[7] A. A. M. Costa, F. B. A. Ramos, M. Perkusich, A. S. Freire, H. O. Almeida, and A. Perkusich. A search-based software engineering approach to support multiple team formation for scrum projects. In *SEKE*, pages 474–473, 2018.

[8] D. S. Cruzes and T. Dyba. Recommended steps for thematic synthesis in software engineering. In *2011 international symposium on empirical software engineering and measurement*, pages 275–284. IEEE, 2011.

[9] H. K. Dam. Artificial intelligence for software engineering. *XRDS: Crossroads, The ACM Magazine for Students*, 25(3):34–37, 2019.

[10] H. K. Dam, T. Tran, J. Grundy, A. Ghose, and Y. Kamei. Towards effective ai-powered agile project management. In *2019 IEEE/ACM 41st International Conference on Software Engineering: New Ideas and Emerging Results (ICSE-NIER)*, pages 41–44. IEEE, 2019.

[11] T. Dybå and T. Dingsøyr. Empirical studies of agile software development: A systematic review. *Information and software technology*, 50(9-10):833–859, 2008.

[12] R. Feldt, F. G. de Oliveira Neto, and R. Torkar. Ways of applying artificial intelligence in software engineering. In *2018 IEEE/ACM 6th International Workshop on Realizing Artificial Intelligence Synergies in Software Engineering (RAISE)*, pages 35–41. IEEE, 2018.

[13] G. R. Finnie and G. E. Wittig. Ai tools for software development effort estimation. In *Proceedings 1996 International Conference Software Engineering: Education and Practice*, pages 346–353. IEEE, 1996.

[14] A. Freire, M. Perkusich, R. Saraiva, H. Almeida, and A. Perkusich. A bayesian networks-based approach to assess and improve the teamwork quality of agile teams. *Information and Software Technology*, 100:119–132, 2018.

[15] M. Gasparic and A. Janes. What recommendation systems for software engineering recommend: A systematic literature review. *Journal of Systems and Software*, 113(Supplement C):101 – 113, 2016.

[16] G. Gousios and D. Spinellis. Mining software engineering data from github. In *2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C)*, pages 501–502. IEEE, 2017.

[17] M. Halkidi, D. Spinellis, G. Tsatsaronis, and M. Vazirgiannis. Data mining in software engineering. *Intelligent Data Analysis*, 15(3):413–441, 2011.

[18] M. Harman. The role of artificial intelligence in software engineering. In *2012 First International Workshop on RAISE*, pages 1–6. IEEE, 2012.

[19] M. Harman, S. A. Mansouri, and Y. Zhang. Search-based software engineering: Trends, techniques and applications. *ACM Comput. Surv.*, 45(1):11:1–11:61, Dec. 2012.

[20] M. Harman, S. A. Mansouri, and Y. Zhang. Search-based software engineering: Trends, techniques and applications. *ACM Computing Surveys (CSUR)*, 45(1):11, 2012.

[21] P. Hearty, N. Fenton, D. Marquez, and M. Neil. Predicting project velocity in xp using a learning dynamic bayesian network model. *IEEE Transactions on Software Engineering*, 35(1):124–137, Jan 2009.

[22] H. Hourani, A. Hammad, and M. Lafi. The impact of artificial intelligence on software testing. In *2019 IEEE JEEIT*. IEEE, 2019.

[23] S.-J. Huang, N.-H. Chiu, and L.-W. Chen. Integration of the grey relational analysis with genetic algorithm for software effort estimation. *European Journal of Operational Research*, 188(3):898–909, 2008.

[24] H. Jin, Y. Wang, N.-W. Chen, Z.-J. Gou, and S. Wang. Artificial neural network for automatic test oracles generation. In *International Conf. on Computer Science and Software Engineering*. IEEE, 2008.

[25] C. Lebeuf, A. Zagalsky, M. Foucault, and M.-A. Storey. Defining and classifying software bots: a faceted taxonomy. In *Proceedings of the 1st Intern. Workshop on Bots in Software Engineering*. IEEE Press, 2019.

[26] J. Lin, M. S. Fox, and T. Bilgic. A requirement ontology for engineering design. *Concurrent Engineering*, 4(3):279–291, 1996.

[27] M. Lowry and M. R. Lowry. Knowledge-based software engineering. In *Handbook of Artificial Intelligence, volume IV*, pages 241–322. Addison-Wesely, 1989.

[28] G. Lucassen, F. Dalpiaz, J. M. E. van der Werf, and S. Brinkkemper. Improving agile requirements: the quality user story framework and tool. *Requirements Engineering*, 21(3):383–403, 2016.

[29] D. J. Mala, V. Mohan, and M. Kamalapriya. Automated software test optimisation framework–an artificial bee colony optimisation-based approach. *IET software*, 4(5):334–348, 2010.

[30] L. S. Marcolino, A. X. Jiang, and M. Tambe. Multi-agent team formation: diversity beats strength? In *Twenty-Third International Joint Conference on Artificial Intelligence*, 2013.

[31] A. M. Memon, M. E. Pollack, and M. L. Soffa. Hierarchical gui test case generation using automated planning. *IEEE TSE*, 2001.

[32] E. Mendes. *Practitioner's knowledge representation: a pathway to improve software effort estimation*. Springer Science & Business, 2014.

[33] T. Menzies. Practical machine learning for software engineering and knowledge engineering. In *Handbook of Software Engineering and Knowledge Engineering: Volume I: Fundamentals*, pages 837–862. World Scientific, 2001.

[34] T. Menzies and T. Zimmermann. Software analytics: so what? *IEEE Software*, 30(4):31–37, 2013.

[35] A. T. Misirli and A. B. Bener. Bayesian networks for evidence-based decision-making in software engineering. *IEEE Transactions on Software Engineering*, 40(6):533–554, June 2014.

[36] R. Moser, W. Pedrycz, and G. Succi. Incremental effort prediction models in agile development using radial basis functions. In *SEKE*, pages 519–522. Citeseer, 2007.

[37] J. Mostow. Foreword what is ai? and what does it have to do with software engineering? *IEEE Transactions on Software Engineering*, (11):1253–1256, 1985.

[38] J. Mylopoulos, L. Chung, and E. Yu. From object-oriented to goal-oriented requirements analysis. *Communications of the ACM*, 1999.

[39] M. Neil, N. Fenton, and L. Nielson. Building large-scale bayesian networks. *The Knowledge Engineering Review*, 15(3):257–284, 2000.

[40] E. E. Odzaly[1] and D. S. Des Greer[1]. Lightweight risk management in agile projects. 2014.

[41] I. Ozkaya. The golden age of software engineering [from the editor]. *IEEE Software*, (1):4–10, 2019.

[42] S. Parnami, K. Sharma, and S. V. Chande. A survey on generation of test cases and test data using artificial intelligence techniques. *International Journal of Advances in Computer Networks and its Security*, 2(1), 2012.

[43] M. Perkusich, H. O. de Almeida, and A. Perkusich. A model to detect problems on scrum-based software development projects. In *Proceedings of the 28th Annual ACM Symposium on Applied Computing*, pages 1037–1042, 2013.

[44] M. Perkusich, L. C. e Silva, A. Costa, F. Ramos, R. Saraiva, A. Freire, E. Dilorenzo, E. Dantas, D. Santos, K. Gorgônio, et al. Intelligent software engineering in the context of agile software development: A systematic literature review. *Information and Software Technology*, 119:106241, 2020.

[45] M. Perkusich, G. Soares, H. Almeida, and A. Perkusich. A procedure to detect problems of processes in software development projects using bayesian networks. *Expert Systems with Applications*, 42(1):437 – 450, 2015.

[46] F. B. A. Ramos, A. A. M. Costa, M. Perkusich, H. O. Almeida, and A. Perkusich. A non-functional requirements recommendation system for scrum-based projects. In *SEKE*, pages 149–148, 2018.

[47] H. B. Reubenstein and R. C. Waters. The requirements apprentice: Automated assistance for requirements acquisition. *IEEE TSE*, 1991.

[48] K. Srinivasan and D. Fisher. Machine learning approaches to estimating software development effort. *IEEE TSE*, 21(2):126–137, 1995.

[49] A. Turani. Applying case based reasoning in agile software development. *Journal of Theoretical and Applied Information Technology*, 78(1):120, 2015.

[50] D. P. Wangoo. Artificial intelligence techniques in software engineering for automated software reuse and design. In *4th ICCCA*. IEEE, 2018.

[51] J. Wen, S. Li, Z. Lin, Y. Hu, and C. Huang. Systematic literature review of machine learning based software development effort estimation models. *Information and Software Technology*, 54(1):41–59, 2012.

[52] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén. *Experimentation in software engineering*. Springer Science & Business Media, 2012.

[53] T. Xie. Intelligent software engineering: Synergy between ai and software engineering. In *Intern. Symposium on DSE*. Springer, 2018.

[54] T. Xie, S. Thummalapenta, D. Lo, and C. Liu. Data mining for software engineering. *Computer*, 42(8):55–62, 2009.

[55] D. Zhang and J. J. Tsai. Machine learning and software engineering. *Software Quality Journal*, 11(2):87–119, Jun 2003.

[56] L. Zhu, A. Aurum, I. Gorton, and R. Jeffery. Tradeoff and sensitivity analysis in software architecture evaluation using analytic hierarchy process. *Software Quality Journal*, 13(4):357–375, 2005.

An Information Fusion based Evolution Requirements Acquisition Method for Mobile Applications

Yuanbang Li[1], Rong Peng[1]*, Bangchao Wang[1], Dong Sun[2]

School of Computer Science, Wuhan University, Wuhan, China

IT Management Department, Haitong Securities Co.Ltd, Shanghai, China

(lybang,rongpeng,wangbc)@whu.edu.cn, 13362999@qq.com

Abstract—**User feedbacks and market changes are both important sources of requirements evolution. Accurately capturing the evolutionary demands from user feedbacks and market changes are extremely important for providers of mobile apps to adjust evolutionary strategies of products. However, many challenges, such as divergent demands and conflicting demands, hinder the process of evolutionary requirements acquisition from multiple sources. Thus, eliciting and merging information from multiple sources is vital to make intelligent evolution decisions. In this paper, an evolution requirements acquisition method based on information fusion is proposed, which comprehensively utilizes its functionality statements, its online comments and its similar apps' online comments to refine evolutionary requirements. By evolution point ranking and selection, it provides a feasible and reasonable way to recommend the evolutionary requirements for the next release of the mobile application.**

Keywords information fusion; evolutionary requirements; kernel concerns; mobile app; user comments.

I. INTRODUCTION

Since the launch of Apple App Store and Google Play in 2008, mobile applications have penetrated into multiple aspects of people's daily life such as communication, games, reading, shopping, social networking, scheduling, working and so on [1].

Users are increasingly interested in downloading and installing mobile applications (apps) to obtain convenient services as more and more apps are published in app stores. Meanwhile mobile apps are facing fierce market competition. On the one hand, the proliferation of homogenous apps brings great challenges to the sustainable development of mobile apps [2]; on the other hand, user expectations are becoming more and higher with the improvement of mobile apps [3]. If mobile apps fail to respond demand changes timely, they will be replaced by other apps soon. Therefore, continuously paying attention to user feedbacks, market changes and main competitors are vital for mobile apps to stand out in fierce competition.

However, tracing, integrating and prioritizing the demands elicited from user feedbacks, market changes and main competitors are always huge challenges for providers due to the cost and time constraints [4]. Hence, in this paper, an information fusion based evolution requirement acquisition method for mobile apps is proposed, which not only provides a merge algorithm to synthesize the information of its own feedback and similar apps' feedback, but also proposes a ranking method to evaluate the importance of the evolutionary points, which provides a

feasible way to determine the priorities of the evolution requirements of certain mobile application.

In this paper, Section 2 introduces the relevant work. Section 3 to 5 introduces the whole method. Section 6 demonstrates the effectiveness of the method through a case study. Section 7 summarize the paper.

II. RELATED WORKS

There are two main ways for mobile applications to acquire evolutionary requirements: monitoring based evolutionary requirements acquisition and application market analyses based evolutionary requirements acquisition.

A. Monitoring Based Evolutionary Requirements Acquisition

Monitor the changes of user behaviors and system performance indicators, find problems or bottlenecks in time are important for mobile apps to analyze evolutionary requirements. Therefore, deploying various performance monitoring tools to detect performance indicators such as response time, resource utilization and data transmission rate in real-time has become important means to guide system improvement [6]. For example, Instagram deploys Munin to fulfil network resource monitoring, Dogslow to fulfil process monitoring, and Redis to fulfil database query traffic monitoring [7]. However, monitoring-based evolutionary requirements acquisition is prone to discovering system anomalies and performance bottlenecks, but not suitable for capturing evolutionary requirements arising from user experiences or expectation changes.

B. Application Market Analyses Based Evolutionary Requirements Acquisition

Basic information, technical information and market information of a mobile app are needed when it is submitted to app market to facilitate user retrieval. The basic information of the application mainly includes the developers, size, function description and characteristics description. The technical information includes function interfaces, class libraries and resource manifest files, which can be obtained by reverse analysis. The market information consists of the price, category, download records and reviews of the app.

The methods in this category can be divided into three kinds: feature analysis, version engineering and commentary analysis.

Feature analysis methods mainly focus on extracting applied features from all the available information sources include app descriptions and resource listing files, internal functions, permission and comments by NLP, topic modeling , clustering and other technologies[8-10]. The

methods have been widely used in app recommendation and version evolution.

Version engineering methods focus on version information and release strategies. The recommendation of app and the formulation of version strategy are realized based on the analysis of the relationship between version external function interfaces, download volumes, comments and sales volumes [11-13].

Comment analysis methods focuses on extracting useful information from online comments of apps. These methods categorize and summarize the comments with other information such as version and download to understand the concerns and complaints of users by using the technology of classification, topic extraction, affective analysis, association mining and regression analysis [14-16].

III. ACQUISITION PROCESS OF EVOLUTIONAL REQUIREMENTS

Evolutional requirements can be extracted from user comments. However, effective comments from which evolutionary requirements can be extracted are few because they were written spontaneously by ordinary users with the main purpose of describing their own feelings. Therefore, it is not enough. The acquisition should be broaden to gather the useful information from the comments of its similar apps, namely the apps with similar functions and the apps from competitors or potential competitors in the market.

As shown in Fig.1, a mobile application evolutionary requirements acquisition process is designed to integrate user requirements from multiple sources, which include the application information, its user comments and the user comments of its similar apps.

Firstly, Kernel Concerns (KCs) are automated extracted from its own comments and its similar apps' comments; and then, they are used to generate a specific Scenario Model Instance (SMI) for each comment; after that, Aggregated Scenario Models (ASMs) are generated and merged according to the similarity of the kernel concerns of SIMs and ASMs; then, an association establishment algorithm is utilized to establish the associations between the ASMs and the Functional Structure Tree (FST) created according to the application information; finally, a ranking strategy is employed to prioritize the potential evolutionary points.
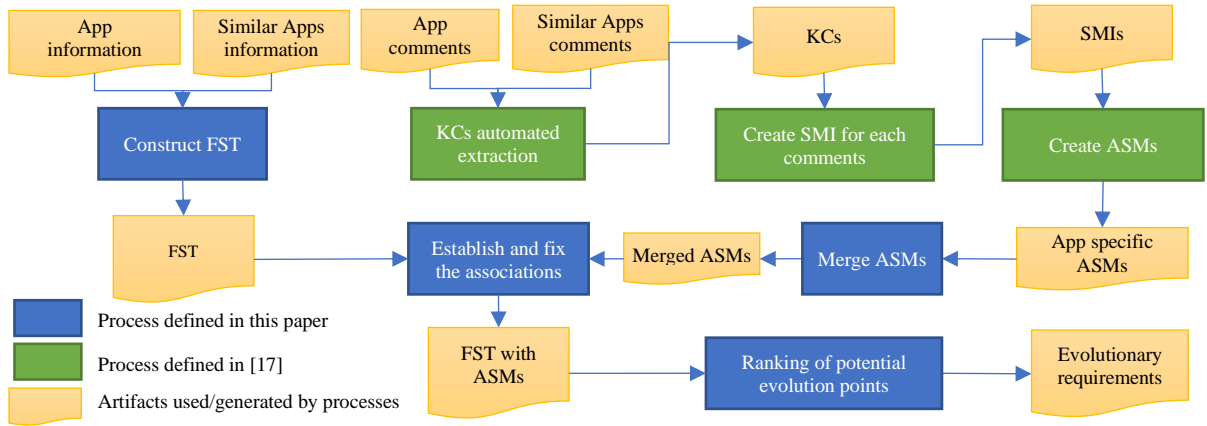
Fig. 1. Acquisition process of evolutional requirements

IV. MODEL DEFINITION

A. Definition of Functional Structure Tree

The app information registered in App Store is represented in Functional Structure Tree (FST), which is defined as follows:

Definition 1 Functional Structure Tree is modeled as T= (N, R), where N = {N_0, N_1... N_n} denotes the set of function nodes in T (n >=0) and N_0 is the root node and represent the whole system; each functional node N_i contains two attributes: function name and function description, denoted as Ni:=<Name, Description>; R={<N_i, N_j>, i ≠ j} represents the set of relationships among function nodes in T, and <N_i, N_j> indicates that N_j is a sub-function of N_i.

FST construction process is as following: firstly, construct the app's FST according to the basic and technical information provided to App Stores; after that, supplement the FST with the functions of its similar apps abiding by the following rules:

Traverse the function description of each similar app: for each function f_i in the description:

- If a matching node can be found in FST, record it as an alias if the function name is inconsistency with the node's name;
- Otherwise, if f_i is a sub-function of the function f_j which has a matching node N_j is FST, add a new node N_i for f_i and establish a relation < N_j, N_i >; otherwise, add a new node N_i for f_i and establish a relation < N_0, N_i >.

An example is shown as Fig. 4.

B. Definition of Scenario Model Instance

Definition 2 Scenario Model Instance (SMI) describes in which scenario the demand is needed or the defect happens. Its core element is KernelConcern, which has the attributes of HasTriggers, HasApperences, HasTerminal, HasOS and HasAppV [17].

For space limit, we only shown an example of SMI for the comment C_1 "Quit without prompt after open the positioning function" in Fig.2. The technical detail of how to construct SMI can be found in [17].
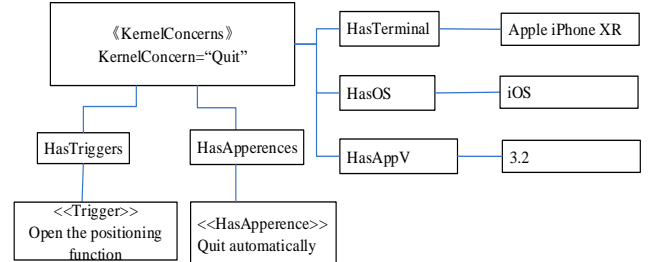
Fig. 2. SMI for the Comment C_1

C. Definition of Aggregated Scenario Model

Definition 3 Aggregated Scenario Model (ASM) is an aggregated model of multiple scenario model instances with the same kernel concerns.

The algorithm of ASM construction are also elaborated in [17]. Therefore, we only show an example of ASM with the kernel concern "Quit" in Fig.3. The model is integrated by 15 SMIs whose kernel concerns are all "Quit". And the number in each rectangle represents the frequency of the attribute appears. It is worth mentioning that the number of trigger is not 15 because some of the comments do not specify their trigger events.
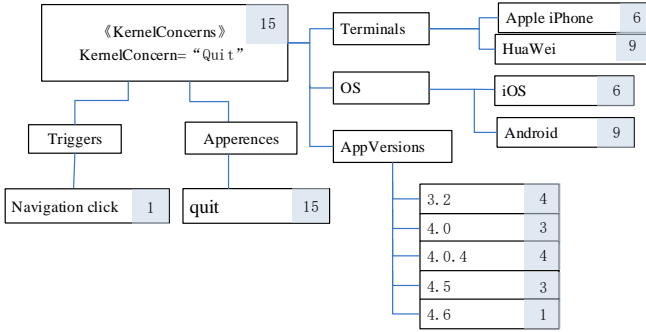
Fig. 3. Aggregated scenario model

V. RANKING AND SELECTION OF POTENTIAL EVOLUTIONARY POINTS

Ranking and selection of potential evolution points should be related not only to the importance of a function or the severity of a defect but also to the degree of the user attention to them. Therefore, the following evolution points

The step of ranking for defect feedback evolution point are as follows:

Step1: Construct FST for the app according to the process described in section 4.1.

Step2: Construct SMIs and ASMs for the app and its similar apps according to [17].

Step3: Merge the ASMs of all apps.

In this step, the ASMs of different apps are merged according to the similarity of their kernel concerns, which is judged by the requirements analyst. Once the analyst decides which two ASMs can be merged, the merge algorithm can be carried out automatically.

Algorithm 1: Merge algorithm of ASMs
Input: ASM M_1, M_2; M_1, M_2 are the ASMs to be merged
Output: ASM M
1: Initialize ASM M = M_1
2: Initialize the counter of the kernel concern of M:
       M.times += M_2.times
3: for each a∈AttributesSet do
4:     for each v∈M_2.a.V do
5:         if (v∈M.a.V) then
6:             M.a.v.times += M_2.s.v.times
7:         else
8:             M.a.V.add(v)
9:             M.a.v.times= M_2.s.v.times
10:        end if
11:    end for
12: end for
13: return M

The input of the algorithm is two ASMs, M_1 and M_2, and the output is the merged M. M is initialize to M_1 (Line 1), and the counter of the KernelConcern of M is set to the sum of the counters of the KernelConcern of M_1 and M_2 (Line 2). For each attribute a in AttributesSet (Line 3), traverse each value v in M_2.s.V(Line 4): if v already exists in M.s.V, sum M.s.v.times and M_2.s.v.times (Line 5-6); otherwise, add v to M.s.V, and assign M.s.v.times to M_2.s.v.times (Line 7-9).

For example, as shown in Fig. 3,{Triggers, Appearances, Terminals, OS, AppVersions} are the AttributesSet of the ASMs; and the value set of the attribute "Terminals" is {"Apple iPhone", "HuaWei"}.

Step4: Establish the association between the ASMs and the FST according to the following algorithm:

Algorithm 2: Association establishment algorithm between ASMs and FST
Input: ASMSet MS, FST T;
Output: RelationSet RS
1: RS =NULL // Initialize the relation set RS to NULL
2: for each asm∈MS do
3:      for each $lnode$∈T.leafNodeSet do
4:          if (asm.KernelConcern.IsAssociatedWith($lnode$))
5:          then RS.add(asm.KernelConcern, $lnode$)
6:          end if
7:      end for
8: end for
9: return RS

Algorithm 2 aims to establish the association between ASMs and FST. For each asm in the ASM set MS, traverse each leaf node $lnode$ of FST T (Line 2-3): if the kernel concern of asm is associated with $lnode$ which is determined by analyst, add a relation between them to the relation set RS (Line 4-5). Finally, the relation set RS is returned (Line 9).

Step5: Ranking of the ASMs

The importance of an ASM_i $G(ASM_i)$ is measured by the product of the importance of the model's kernel concern $GA(ASM_i)$, the importance of the function associated with the model $Gf(ASM_i)$ and the user attention to the model $Gu(ASM_i)$, as shown in formula 1.

$$G(ASM_i) = GA(ASM_i) \cdot GF(ASM_i) \cdot GU(ASM_i) \quad (1)$$

$GA(ASM_i)$ is determined by requirement engineers and can be divided into three levels{0.5,1,2}, which represent{not serious, normal, serious}. $GF(ASM_i)$ is also divided into three levels{0.5,1,2}, which represents {unimportant, normal, important}. The value of $GF(ASM_i)$ that is not associated with any functional of the ASM is set to 1 by default. Of course, requirements engineers can also set other levels in the specific implementation process.

$GU(ASM_i)$ can also be divided into three levels{0.5,1,2}. It is calculated by the number of times the user pays attention to the concern, as shown in formulas 2- 4.

$$GU(ASM_i)= \begin{cases} 0.5 \ if(\ T_i < minp) \\ 1 \ if(\ T_i \geq minp \&\& T_i \leq maxp) \\ 2 \ if(\ T_i > maxp) \end{cases} \quad (2)$$

$$minp = min + (max - min)/3 \quad (3)$$

$$maxp = min + 2 \times (max - min)/3 \quad (4)$$

Where T_i indicates the number of times ASM_i is concerned; max and min indicates the maximum and minimum number of times a ASM is concerned, respectively; minp and maxp divide the range between min and max into three ranges on average.

Based on the score calculated of $G(ASM_i)$, the ASMs with TOP n scores are recommended as the evolution requirement points.

VI. CASE STUDY

Take Baidu Map with the version of 4.6 as the sample app, and take Amap with the version of 5.0 as its similar app. 840 user comments of Baidu Map and 960 user comments of Amap are crawled as feedbacks.

The evolution points are selected according to the process described in section V, which is specified as follows.

Step 1: The FST of Baidu Map was built as Fig.4.
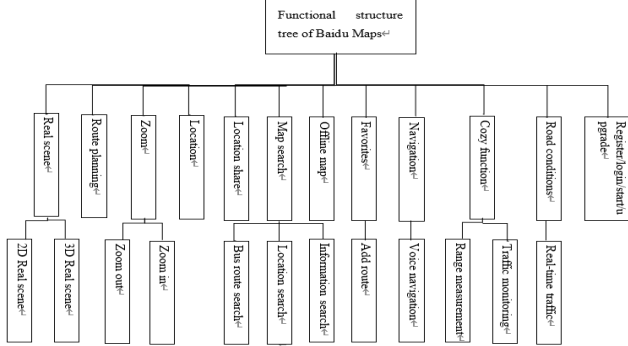
Fig. 4. FST of Baidu Maps

Step 2: Seven feedback ASMs are constructed through the analysis of the comments, namely "deviation", "inaccurate", "slow", "auto-exit, "flash-screen", "down-time" and "waste-of-slow".

Step 3: "Inaccurate" and "deviation" are merged as they indicate the same meaning. Finally, six ASMs are retained.

Step 4: The associations between these ASMs and the FST are established using algorithm 2. The associations between ASMs and function nodes of FST are built as follow:<deviation, location>, <slow, location>, <auto-exit, navigation>, <flash-screen. start>. No functions are associated to the ASM with concern "down-time" and "waste-of-flow".

Step 5: Set the GA, GF value of each ASM under the guidance of the analyst and calculate GU according to the formula 2-4 based on the times of the feedbacks. Finally, calculate the importance of each ASM by formula 1 and sort them in descending order. The result is shown in Table 1.

Table 1. The result of the ranking of evolution points

N	F	R	T	GU	GA	GF	G(ASM$_i$)
1	Deviation	Location	720	2	2	2	8
2	Slow	Location	490	1	1	2	2
3	Auto-exit	Navigation	150	0.5	2	2	2
4	Flash-screen	Start	190	0.5	2	2	2
5	Down-time	/	90	0.5	2	1	1
6	Network flow waste	/	160	0.5	1	1	0.5

Note: F indicates the kernel concern of the ASM; R indicates the function associate with the ASM; T indicates the feedback times of the kernel concern of the ASM; GU, GA and GF indicate GU(ASM$_i$), GA(ASM$_i$) and GF(ASM$_i$), respectively.

As shown in the table, the most important evolutionary requirement is to settle the problem of "deviation" in the "location" function; and the least important one is to settle the problem of "Network flow waste".

VII. SUMMARY

In this paper, a multi-source information fusion based evolution requirements acquisition method for mobile apps is proposed. It synthesizes the information provided to App Stores, the feedbacks of the app and its similar apps to acquire evolutionary requirements. By ranking the ASMs according to their importance and user attention, most urgent evolutionary requirements can be found. The case study on Baidu Map show that the ranking is similar to its version history, which verifies the effectiveness of the method.

REFERENCES

[1] William Martin, Federica Sarro, Yue Jia, et al. 2016. A Survey of App Store Analysis for Software Engineering[J]. Research Note of UCL Department of Computer:1-56.

[2] Cuiyun Gao, Hui Xu, Junjie Hu, et al. 2015. AR-Tracker: Track the Dynamics of Mobile Apps via User Review Mining[C]//Proceedings of the Service-Oriented System Engineering (SOSE), 2015 IEEE Symposium on. IEEE, 284-290.

[3] Claudia Iacob, Rachel Harrison. 2013. Retrieving and analyzing mobile apps feature requests from online reviews[C]//Proceedings of the 10th Working Conference on Mining Software Repositories (MSR).41-44.

[4] Lawrence Bernstein, C. M. Yuhas. 2014. Software Requirements[M]. Apress, 73-106.

[5] Xie zhongwen, li tong, dai fei, etc. 2011. A paraconsistent meta-model of requirements for software evolution[J]. journal of Jiangsu university(natural science edition), 32(5):562-568. Doi:10.3969/j.issn.1671-7775.2011.05.013.

[6] Liyin Tang, Vinod Venkataraman, Charles Thayer. 2012. Facebook's Large Scale Monitoring System Built on HBase[C]//Proceedings of the Strata Conference.

[7] Dong Sun, Rong Peng, Wei-Tek Tsai. 2014. Understanding Requirements Driven Architecture Evolution in Social Networking SaaS: An Industrial Case Study[C]//Proceedings of the Service Oriented System Engineering (SOSE), 2014 IEEE 8th International Symposium on. IEEE, 230-236.

[8] Borja Sanz, Igor Santos, Carlos Laorden, et al. 2012. On the Automatic Categorisation of Android Applications[C]//Proceedings of the 9th IEEE Consumer Communications and Networking Conference.149-153.

[9] Jieun Kim, Yongtae Park, Chulhyun Kim, et al. 2014. Mobile application service networks: Apple's App Store[J]. Service Business, 8(1):1-27.

[10] Lavid Ben Lulu David, Tsvi Kuflik. 2013.Functionality-based clustering using short textual description: helping users to find apps installed on their mobile device[C]//Proceedings of the International Conference on Intelligent User Interfaces.N/A.

[11] Diya Datta, Sangaralingam Kajanan. 2013. Do App Launch Times Impact their Subsequent Commercial Success? An Analytical Approach[C]//Proceedings of the International Conference on Cloud Computing and Big Data.205-210.

[12] Israel J. Mojica Ruiz, Meiyappan Nagappan, Bram Adams, et al. 2016. Analyzing Ad Library Updates in Android Apps[J]. IEEE Software, 33(2):74-80.

[13] Gunwoong Lee, T. S. Raghu. 2014. Determinants of Mobile Apps' Success: Evidence from the App Store Market[J]. Journal of Management Information Systems, 31(2):133-170.

[14] Ning Chen, Jialiu Lin, Steven C. H. Hoi, et al. 2014. AR-miner: mining informative reviews for developers from mobile app marketplace[C]//Proceedings of the 36th International Conference on Software Engineering.767-778.

[15] W. Maalej, H. Nabil. 2015. Bug report, feature request, or simply praise? On automatically classifying app reviews[C]//Proceedings of the Requirements Engineering Conference.116-125.

[16] Guzman E, Aly O,Bruegge B. Retrieving diverse opinions from app reviews//Empirical Software Engineering and Measurement (ESEM), 2015 ACM/IEEE International Symposium on. IEEE, 2015: 1-10.

[17] Sun D, Peng R. A Scenario Model Aggregation Approach for Mobile App Requirements Evolution Based on User Comments[M]//Requirements Engineering in the Big Data Era. Springer Berlin Heidelberg, 2015: 75-91.

Detecting and Modeling Method-level Hotspots in Architecture Design Flaws

Ran Mo, Shaozhi Wei, Ting Hu, Zengyang Li

School of Computer Science & Hubei Provincial Key Laboratory of Artificial Intelligence and Smart Learning

Central China Normal University

moran@mail.ccnu.edu.cn, wsz@mails.ccnu.edu.cn, 826473959@qq.com, zengyangli@mail.ccnu.edu.cn

Abstract—In large-scale software systems, the majority of change-prone files are usually architecturally connected, and their architectural connections often exhibit design flaws, which propagate change-proneness among files and increase maintenance costs. Complementary to the identification and definition of the files involved architecture design flaws, the automatic detection at the method level is important for developers to understand fine-grained contributors to the architecture flaws that have incurred high maintenance costs. In this paper, we propose an approach to identify method-level hotspots. Each hotspot contains a group of evolutionarily connected methods or attributes that participate in architecture design flaws and continuously accumulate maintenance difficulties. Our investigations on six large-scale projects show that the attributes or methods captured in method-level hotspots are more change-prone than the others. The results also show that the growth trend of the maintenance effort spent on each method-level hotspot could be modeled and monitored by our approach, which sheds light in the decision of design refactoring.

Index Terms—Software Architecture, Software Maintenance, Fine-grained Code Change

I. INTRODUCTION

Software architecture has significant impacts on the maintenance of a software project. Numerous studies have been proposed to investigate the software quality by examining software architecture. For example, prior studies of Xiao et al. [18] and Kazman et al. [13] have shown that the majority of change-prone files are architecturally connected, and design flaws in these connections could propagate change-proneness among files. Consequently, it is difficult to eliminate files' change-proneness without resolving the architecture design flaws among them. The automatic detection of architecture design problems has also been studied. Xiao et al. [19] presented four types of architecture debts which have significant and long-term impact maintenance costs over time. Mo et al. [14, 15] formally defined and automatically detected a suit of architecture design flaws, which have a significant impact on files' change-proneness.

However, all of these studies are conducted at the file level, none of them investigated method-level participants of architecture design flaws. After identifying the files involved in architecture design flaws, it is still worth for developers to explore which attributes and methods are responsible for the change-proneness of these involved files, especially, for the files with hundreds of attributes or methods. This could raise questions for developers or architects: Which attributes or methods of the flawed files need to be fixed for maintenance and refactoring? Can these 'problematic attributes or methods be detected automatically? How do the involved attributes or methods evolve over time?

In this paper, we formally define the concept of *Method-level Hotspot*: a group of evolutionarily coupled attributes or methods that participate in architecture design flaws and continuously accumulate maintenance costs. To detect a *method-level hotspot*, 1) our approach first extracts the attributes or methods from the files involved in an instance of architecture flaws that identified by the techniques in [14]; 2) based on these extracted attributes or methods, and the fine-grained history output by *ChangeDistiller* in [3], our approach will automatically identify a *method-level hotspot* in a project. Given a *method-level hotspot*, our approach will automatically model its evolution trend in terms of maintenance effort spent on its involved attributes and methods. We use three typical regression models, linear, exponential and logarithmic models, to model each hotspot' evolution to monitor whether the hostspot has been accumulating maintenance costs steadily, dramatically or slowly.

We have validated the effectiveness of our approach by using six large-scale projects. According to the evaluation results, we have found that: 1) attributes or methods involved in method-level hotspots are more change-prone than the other attributes or methods; 2) most of the detected method-level hotspots could be modeled by one of the three typical regression models, the modeling results are useful for developers to understand the evolution trend for each hotspot.

The rest of this paper is organized as follows: Section II presents the background concepts. Section III describes the definition of the method-level hotspots. Section IV presents our evaluation methods and results. Section V discusses. Section VI shows related work and Section VII concludes.

II. BACKGROUND

We now introduce the basic concepts and techniques behind our work.

A. Architecture Design Flaws

In [14], the authors defined and validated a suite of hotspot patterns, the recurring architectural design flaws in software systems. They presented that files involved in the detected design flaws are really change-prone. Using the techniques in [14], we identify four file-level architecture design flaws: 1) *Unstable Interface* – a highly influential file have a large number of dependents and changes frequently with many of its dependents in the revision history; 2) *Modularity Violation* – the structurally independent files frequently change together as recorded in the project's revision history; 3) *Unhealthy Inheritance* – a super class depends on its sub classes or a client class depends on both a super-class and its sub classes; 4) *Clique* – a group of files forming a strongly connected component are tightly coupled with cyclic dependency. In general, each instance of the architecture design flaws capture a group of files, and these detected files have caused high maintenance costs. From each group of flawed files, we could extract the attributes and methods that participate in each architecture design flaw.

B. Method-level Changes

Revision history records maintenance activities of a project. By examining the change history of attribute and methods, we could get the data of evolutionary relations among attributes or methods and investigate how they evolved. Following the techniques in [3], we extract fine-grained changes from a project's revision. Furthermore, we categorize these fine-grained changes into eight types of *method-level change operations* on attributes or methods:

- *ATTRIBUTE_ADRT_CHANGE*: Adding, Deleting, Renaming an attribute or changing the type of an attribute.
- *ATTRIBUTE_MODIFIER_CHANGE*: Changing the modifier of an attribute, such as changing the accessibility of an attribute or finalizing an attribute, etc.
- *METHOD_ADR_CHANGE*: Adding, Deleting or Renaming a method.
- *METHOD_MODIFIER_CHANGE*: Change the modifier of a method, such as changing the accessibility of a method or finalizing a method, etc.
- *METHOD_PARAMETER_CHANGE*: Adding, Deleting, Rename parameters, or changing the type or ordering of parameters.
- *METHOD_BODY_CHANGE*: Change the body of a method.
- *METHOD_RETURN_CHANGE*: Adding, Deleting or Changing the type of a method return.
- *DOCUMENT_CHANGE*: Adding, Deleting or Updating the documentation of an attribute or a method.

Using the suite of *method-level change operations*, we capture how an attribute or method was changed, and by how many times.

III. IDENTIFICATION AND MODELING

To investigate method-level hotspots, our approach proceeds as follows: 1) identifying *method-level hotspots*; 2) modeling the evolution of identified method-level hotspots; 3) visualizing the identified method-level hotspots.

A. Method-level Hotspot Definition

Using the concepts and technique in [14], a set of file groups flawed by architecture flaws will be detected from a project. Although the attributes or methods in each file group participate in an architecture flaw, not all of them contribute to incurring maintenance costs into the group. For each file group (an instance of architecture flaw [14]), we aim to identifying their attributes or methods that propagate changes among the group and cause maintenance costs. Therefore, we define a *Method-level Hotspot* to be a group of *evolutionarily coupled* attributes or methods that participate in architecture design flaws and continuously accumulate maintenance difficulties. The rationale is, first, if an attribute or method could always be changed independently, then we consider this attribute or method is not coupled with others in the group, changes to it won't influence the others; second, a hotspot should continuously cause maintenance costs in the system. We don't need to worry about the methods or attributes that are inactive with respect to changes, because they won't cause maintenance costs in the future.

Based on the definition, we formally calculate a method-level hotspot as a sequence of tuples :

$$M - hotspot = (\langle mSet_1, mSetCost_1 \rangle, \langle mSet_2, mSetCost_2 \rangle, ..., \langle mSet_m, mSetCost_m \rangle)$$

(1)

where m is the number of history periods the hotspot has been evolved through.

B. Method-level Hotspot Identification

Given an instance of architecture flaw, let F to be the set of involved files, and M to be the universal set of all attributes and method within these files. To identify the group of evolutionarily coupled attribute or methods in M, we checked the co-changes between the attributes or methods in M by mining a particular period of revision history. If an attribute or method hasn't changed together with any other attributes or methods in M during the period of history, we consider it is evolutionarily independent to the others in M. Assume $mSet_k$ is the maximal group of evolutionarily coupled attributes or methods in M, then $mSet_k$ contains a subset of attributes and methods in M, and these attributes or methods should satisfy:

$$mSet_k : \forall m_i \in mSet_k, \exists m_j \in mSet_k | cochange(m_i, m_j)$$

(2)

where $i \neq j$, $i, j = [1, 2, 3, ..., n]$, n is the number of attributes or methods in the $mSet_k$. k means the kth period of history. $cochange(m_i, m_j)$ means m_i and m_j have been changed together in the same commits. The co-changes between m_i and m_j are calculated from the given $period_k$ of revision history.

Based on each M of an architecture flaw instance, we detected the *method-level hotspot* consisting of a sequence of $mSet$ and $mSetCost$ by using different history periods. In this paper, we back-forwardly decreased the history period by

a 6-month history interval. For example, if we have detected an architecture flaw instance from a version of project A, which was released in 2016-07 and its history began in 2015-09. Let the universal set of attributes and methods involved in this instance be M_a, to construct the hostpot, we calculated its $mSet$ sequence by using four history periods: 2015-03 - 2016-07, 2015-03 - 2016-01, and 2015-03 - 2015-07. The number of $mSet$ in the hotspot may be less than 3, because attributes and methods in M may not change together at the early history. Besides, we also calculated the maintenance costs spent on the attributes or methods in each $mSet$ to be $mSetCost$ by using the same history period.

We consider a *method-level hotspot* that has been accumulating maintenance costs, if two conditions are satisfied: 1) the hotspot has been involved for an enough long time. In this paper, since all of the projects have a long history, we required that a hotspot should have evolved for 3 years, and we sampled six months as the history interval; 2) attributes or methods involved in the hotspot should continuously incur maintenance costs. Let $\langle mSet_1, mSetCost_1 \rangle$, and $\langle mSet_n, mSetCost_n \rangle$ be the first and last elements in a method hostopt, $mSetCost_n$ should be larger than $mSetCost_1$, where $mSetCost_i$ means the total number of changes made on the attributes or methods in $mSet_i$ by using corresponding history period.

C. Method-level Hotspots Modeling

Given a *method-level hotspot*, our approach automatically models its evolution in terms of maintenance costs. In this way, we could present the variation trend of these method-level hotspots in terms of maintenance costs. Our approach uses the sequence of mSetCost as an input, and searches the best regression model for it. Since the selected history periods are cumulative, we applies three typical regression models in our approach, each type of regression models presents a different maintenance evolution of method-level hotspots:

- *Linear model*, which describes the method-level hotspot accumulates maintenance costs steadily over time.
- *Exponential model*, which describes the method-level hotspot accumulates maintenance costs dramatically over time.
- *Logarithmic model*, which describes the method-level hotspot accumulates maintenance costs slowly over time.

D. Method-level Hotspots Visualization

For each *method-level hotspot*, we proposed a *Method-level DSM* (M-DSM) to model each of its $mSet$. A M-DSM is extended from the DSM, which is proposed by [1]. A DSM is a square matrix whose rows and columns are labels with the same elements in the same order. A cell in a DSM present the structure or co-change relations between an element in row and an element in column. Elements in the original DSM could be files, classes or packages. We used the M-DSM to present the attributes or methods and their relations. Instead of presenting the co-change numbers in a cell, the cells in a M-DSM show *method-level co-change pairs*. We define a *Method-level Co-change Pair* to be a pair of change operations introduced in

Section II. Each *co-change pair* shows how two attributes or methods changed together and the number of its occurrence. Considering a M-DSM contains 2 method, $method1$ and $method2$, 1) these two methods were both involved in a commit, where $method1$'s parameter and $method2$'s body were changed; 2) these two methods were involved in another commit, where $method1$'s parameter and $method2$'s return type were changed. Then, the method-level DSM will be shown as in Figure 1.

		1	2
1	method1	{1}	[(MPC->MBC):1;(MPC->MRC):1;]
2	method2	[(MRC->MPC):1;(MRC->MPC):;]	{2}

Fig. 1: Example of M-DSM showing co-change pairs
MRC: METHOD_RETURN_CHANGE;
MPC: METHOD_PARAMETER_CHANGE;
MBC: METHOD_BODY_CHANGE

IV. EVALUATION

In this section, we report our evaluation subjects, methods and results.

A. Research Question

To evaluate the effectiveness of our approach, we investigate the following research questions:

RQ1: *Are the attributes or methods involved in method-level hotspot really change-prone?*
A positive answer to this question would demonstrate that the detected method-level hotspot really capture the change-prone attributes and methods in the project, which deserve more actions for refactoring.

RQ2: *Could we model the evolution of detected method-level hotspots?* To answer this question, we are attempting to investigate whether the evolution of method-level hotpots could be effectively monitored. A positive answer to this will enable us to understand how the detected method-level hotspts evolve over time.

B. Subjects

Six Apache open-source projects have been chosen as the subjects, which differ in size, domain and other project characteristics: Camel is an integration framework; Cassandra is a distributed NoSQL database management system; CXF is a services framework; Hadoop is a tool for distributed Big Data processor; OpenJPA is a Java persistence project; PDFBox is a library for manipulating PDF documents.

We list the basic facts for each studied project in Table I. The column "#Members" presents the total number of attributes and methods in a project. The column "#Commits" presents the number of revisions over the time period from the begin to the selected release date for each project. All projects' revision histories are extracted from GitHub[1]. The column "#History Length" shows the number of months from the begin to the selected release date.

---

[1]https://github.com/

For each project, we first obtained its source code and chose its stable version as our research subject. Then we used Understand[2] to generate a file dependency report. Given the revision history and the file dependency file, we used the toolset in Mo et al.'s work [14] to detected architecture flaws. We used *ChangeDistiller* [3] to extract the method-level revision history. Given all the flaws and method-level history as inputs, our tool automatically detects all *method-level hotspots* and fits them into regression models.

TABLE I: Researched Projects

	Release	#Members	#Commits	#History Length
Camel	2.15.5	84,734	24,933	113 months
Cassandra	2.1.13	41,767	19,333	89 months
CXF	3.0.9	54,210	11,160	96 months
Hadoop	2.6.3	70,054	12,506	86 months
OpenJPA	2.4.1	25,896	4,729	123 months
PDFBox	1.8.10	19,236	4,337	102 months

C. Evaluation Results

To quantify the maintenance effort, we use a typical history measures: *Change Frequency (CF)*, the number of times an attribute or method has been changed in commits with a given period of revision history.

RQ1: *are the attributes or methods methods captured in method-level hotspots notably change-prone?*

For each project, we calculated the average change frequency (avg_CF) values for all the attributes or methods involved in *method-level hotspots* and the attributes or methods not involved in any *method-level hotspot*. Table II reports the comparison results between two sets of average values. "avg_CF" means the average CF for attributes or methods involved in method-level hotspots, "avg_nCF" means the average CF for the other attributes or methods. "Inc." means by how much the average value has increased by comparing the attributes or methods in hotspot to the attributes or methods not in hotspots. We calculated the Inc. of change frequency as:

$$Inc. = (avg\_CF - avg\_nCF)/avg\_nCF \times 100\% \quad (3)$$

From Table II, we can observe that all the avg_CF values are larger than the avg_nCF. The greatest increase happens in the *CXF* project. 147% means that, in this project, the attributes or methods involved in method-level hotspots were changed twice more often than the attributes or methods which are not involved in any hotspot. The smallest increase is still as high as 74% in the *Cassandra* project.

To rigorously validate this observation, we employ the *Wilcoxon signed-rank test*, a non-parametric statistical hypothesis test for comparing two related samples, to test whether the population of avg_CF is significantly larger than the

TABLE II: Comparison between avg_CF and avg_nCF

	avg_CF	avg_nCF	Inc.
Camel	1.94	0.85	129%
Cassandra	2.92	1.68	74%
CXF	2.18	0.88	147%
Hadoop	1.98	1.05	89%
OpenJPA	1.85	0.78	138%
PDFBox	2.29	1.17	95%

population of avg_nCF over the six projects. We defined the hypotheses as follows:

Null Hypothesis: H0, the population of avg_CF is not significantly larger than the population of avg_nCF.

Alternative Hypothesis: H1, population of avg_CF is significantly larger than the population of avg_nCF.

The p-value of this test is less than 0.05, thus H1 is accepted. The results indicate that there exists statistically significant differences between avg_CF and avg_nCF over all 6 projects. Therefore, we can claim that that attributes or methods captured in method-level hotspots will be more change-prone, and hence cause higher maintenance costs in a project.

RQ2: *Could we model the evolution of detected method-level hotspots?*

A *method-level hotspot* captures a group of evolutionarily coupled attributes or methods that accumulates maintenance costs. We investigate this problem to answer whether we can monitor the growth trend of the maintenance effort spent on each *method-level hotspot* over time.

In this paper, we fit a hotspot's growth trend of maintenance costs to one of the three models: linear, exponential and logarithmic regression models, which indicating the attributes or methods in a method-level hotspot accumulates maintenance costs steadily, extremely fast or slowly respectively. For each method-level hotspot, the regression model with highest R^2 will be selected to be the best fit for it. Besides, the P-value of each fitting model should be less than 0.05, which guarantees that the derived model is significant.

Following the guidelines in the work of [10, 12], where the authors described R^2 = 0.75, 0.5 and 0.25 as substantial, moderate and weak models, respectively, we summarized the fitting results in Table III. Column "#Hotspot" means the total number of *method-level hotspots* detected from a project. Columns "Lin", "Exp" and "Log" present the number of hotspots that fit into linear, exponential or logarithmic models respectively. The following "Pt." columns show the ratios to the total number of detected hotspots. In the column of "$0.5 \leq R^2 < 0.75$", "Num" means the number percentage of hotspots fitted into a regression model with a R^2 from 0.5 to 0.75. The last two columns shows the number and percentage of hotspots which couldn't or weakly fit into one of the three regression models.

Using "Camel" as an example, we can see that there are 907 method-level hotspots were detected from this project,

and 69% of these hotspots can be substantially modeled by the regression models ($R^2 \geq 0.75$). 48% of all detected hotspots could fit into a linear model, means these hotspots accumulate maintenance costs steadily. 8% and 13% of all detected hotspots could fit into the exponential and logarithmic models respectively. Only 69 hotspots, 8% of all detected hotspots, couldn't fit into a substantial or moderate regression model.

The last row of Table III presents that, considering all the detected method-level hotspots over all projects together, 67% of them could be fitted into a substantial regression model. 52% of all hotspots could be modeled by linear models. For both exponential and logarithmic models, there are 7% of all hotspots follow a substantial fitting. Only 8% of all the detected hotspots can not or weakly fit into a model. In summary, our approach could model the growth trend of maintenance costs for each hotspot..

Figure 2 shows the example of a linear fitting. We can observe that the selected release date of this project is "2015-11", and attributes or methods in this hotspot started to be co-changed before 2010-07, but after 2010-01, since the history interval is 6 months. The maintenance costs trend of this hostspot is fitted into a linear model, which has a $R^2 = 0.98$, with a formula as: $y = 64.2x + 88.6$.
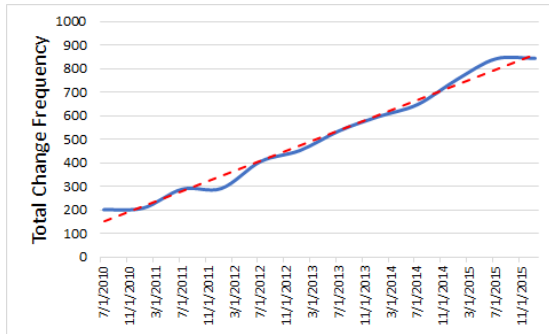
Fig. 2: Hotspot fitted into a linear model
$R^2 = 0.98$; Formula: $y = 64.2x + 88.6$

D. Results Summary

Based on the evaluation results, we can positively answer our research questions as follows:

RQ1: If an attribute or method is involved in a *method-level hotspot*, it is more likely to have higher change-proneness.

RQ2: The majority of detected method-level hotspots could be modeled by one of the three regression models, the modeling results will help developers to understand the evolution trend for each hotspot.

V. DISCUSSION

A. Threats to validity

First, we can not guarantee that change frequency is the best proxies for maintenance effort. In our future work, we intend to use more proxy measures for our analysis, for example, the bug frequency, the time frame for each issue, the budget spent,

etc. Thus we could further demonstrate the effectiveness of our approach and tool.

Second, we only applied our research on six Apache open source projects, hence we can not claim that our results are generalizable across all software projects. However, we chose projects with different sizes and domains to partially address this issue. A larger study employing more projects and more metric types would improve the validity of our conclusions.

Third, the detection of our *method-level hotspot* needs to mine the project's revision history. We need to examine the co-changes between attributes or methods. The availability and accuracy of the method-level history information heavily depends on the project's protocols.

Finally, since the history periods are accumulative, we only used three typical regression models to model the growth trend of maintenance costs. We can not grantee the completeness of our applied models. But our tool and our approach are scalable, which enables easy additions for new regression models.

B. Future Work

We are planning to apply our approach on more projects to further demonstrate the effectiveness of our approach and detection tool. We also plan to investigate whether the *method-level DSM* could help to examine underlying problems in a project and help to explore the refactoring opportunities for each *method-level hotspot*, such as splitting files, combining methods, etc.

VI. RELATED WORK

In this section we compare our approach with the following research areas.

Defect Prediction and Localization: There are numerous studies [2, 5, 8, 9, 11] aimed at predicting and locating error-prone/change-prone files by using file metrics, file change history, or both. For example, Jones et al. [11] obtained the ranking information of each statement and used the information to assist fault location. Nagappan et al. [16] studied different complexity metrics and demonstrated that a combination of these metrics are useful predictors for defects and successful for defect prediction. Cataldo et al. [2] investigated the density of change coupling and showed that it correlated with failure proneness. Ostrand et al. [17] demonstrated that a combination of files metrics and file change history can be used to effectively predict defects.

However, all these studies treat the error-/change-prone files individually but don't consider the architectural connections among these files. Consequently, even if the predicted files were modified, the root causes of high-maintenance costs would still exist there, because architecture design flaws haven't been eliminated. Our study focuses on the attributes or methods participating in architecture design flaws.

Code and Architecture Quality: Gamma et al. [6] introduced commonly occurring software design problems. They presented design patterns as proven solutions to these recurring problems. Fowler [4] introduced the concept of a "bad smell" to identify code problems and provide refactoring references.

TABLE III: Distribution of Architecture flaws' Regression Models

| Project | #Hotspot | $R^2 \geq 0.75$ | | | | | | | | $0.5 \leq R^2 < 0.75$ | | Other | |
		Lin	Pt.	Exp	Pt.	Log	Pt.	Total	Pt.	Num	Pt.	Num	Pt.
Camel	907	437	48%	77	8%	116	13%	630	69%	208	23%	69	8%
Cassandra	737	477	65%	15	2%	16	2%	508	69%	209	28%	20	3%
CXF	712	344	48%	69	10%	43	6%	456	64%	167	23%	89	13%
Hadoop	924	668	72%	63	7%	40	4%	771	83%	121	13%	32	3%
OpenJPA	544	129	24%	15	3%	12	2%	156	29%	289	53%	99	18%
PDFBox	246	76	31%	62	25%	53	22%	191	78%	54	22%	1	0%
Total	4,070	2,131	52%	301	7%	280	7%	2712	67%	1048	26%	310	8%

Garcia [7] investigated and presented some bad smells from architectural perspectives. These methods presented the concepts and principles for the design solutions or problems, but still leave much of the effort to developers, and depend on the skill of the architecture analysts.

Automatic detection of architecture problems has been widely studied. Mo et al.'s [14, 15] work formally defined a suite of architecture hotspot patterns—recurring architecture smells—in a project. Xiao et al.'s [18] work helps to detect architecture roots, file groups where the constituent files are architecturally connected and cause high maintenance costs. However, all of this work focuses on file-level problems. Even if we found the flawed connections among files, it still worth for developers or architects to examine the details in the flawed files. In particular, for large-scale and long-term projects, file sizes increase as software evolves. What is worse, the larger and complicated a file, and the more attributes or methods it will have. In our approach, we automatically detect and model the group of attributes or methods or fields which participate in architecture flaws and accumulate maintenance costs.

VII. CONCLUSION

In this paper, we have formally defined the concept of *Method-level Hotspot*, a group of evolutionarily coupled attributes or methods that participate in architecture design flaws and continuously incur maintenance difficulties. Our approach could automatically detect method-level hotspots from a project and model the evolution trend of the detected method-level hotspots in terms of maintenance costs. Our approach also uses *method-level co-change pairs* to visualize the co-change details between attributes or methods in a *method-level hotspot*.

From our analysis on six large-scale open source projects, we have demonstrated that the attributes or methods involved in *method-level hotspots* have significantly higher change-proneness compared to the attributes or methods not involved in any *method-level hotspot*. We have also presented that our approach could model most of the identified method-level hotspots into one of the three typical regression models. The modeling results could help developers understand the evolution trend of each hotspot and provide guidance for the refactoring decisions.

ACKNOWLEDGMENTS

REFERENCES

[1] C. Y. Baldwin and K. B. Clark. *Design Rules, Vol. 1: The Power of Modularity*. MIT Press, 2000.

[2] M. Cataldo, A. Mockus, J. A. Roberts, and J. D. Herbsleb. Software dependencies, work dependencies, and their impact on failures. *IEEE Transactions on Software Engineering*, 35(6):864–878, July 2009.

[3] B. Fluri, M. Wuersch, M. PInzger, and H. Gall. Change distilling: Tree differencing for fine-grained source code change extraction. *IEEE Transactions on Software Engineering*, 33(11):725–743, 2007.

[4] M. Fowler. *Refactoring: Improving the Design of Existing Code*. Addison-Wesley, July 1999.

[5] H. Gall, K. Hajek, and M. Jazayeri. Detection of logical coupling based on product release history. In *Proc. 14th IEEE International Conference on Software Maintenance*, pages 190–197, Nov. 1998.

[6] E. Gamma, R. Helm, R. Johnson, and J. M. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1994.

[7] J. Garcia, D. Popescu, G. Edwards, and N. Medvidovic. Identifying architectural bad smells. In *Proc. 13th European Conference on Software Maintenance and Reengineering*, pages 255–258, Mar. 2009.

[8] T. L. Graves, A. F. Karr, J. S. Marron, and H. P. Siy. Predicting fault incidence using software change history. *IEEE Transactions on Software Engineering*, 26(7):653–661, 2000.

[9] N. Gupta, H. He, X. Zhang, and R. Gupta. Locating faulty code using failure-inducing chops. In *Proc. 20th IEEE/ACM International Conference on Automated Software Engineering*, pages 263–272, 2005.

[10] J. F. Hair, C. M. Ringle, and M. Sarstedt. Pls-sem: indeed a silver bullet. *Journal of Marketing Theory and Practice*, 19(2):139–151, 2011.

[11] J. A. Jones, M. J. Harrold, and J. Stasko. Visualization of test information to assist fault localization. In *Proc. 24thInternational Conference on Software Engineering*, 2002.

[12] J. Joseph F. Hair, G. T. M. Hult, C. Ringle, and M. Sarstedt. *A Primer on Partial Least Squares Structural Equation Modeling (PLS-SEM)*. Sage, Thousand Oak, 2013.

[13] R. Kazman, Y. Cai, R. Mo, Q. Feng, L. Xiao, S. Haziyev, V. Fedak, and A. Shapochka. A case study in locating the architectural roots of technical debt. In *Proc. 37th International Conference on Software Engineering*, May 2015.

[14] R. Mo, Y. Cai, R. Kazman, and L. Xiao. Hotspot patterns: The formal definition and automatic detection of architecture smells. In *Proc. 12thWorking IEEE/IFIP International Conference on Software Architecture*, May 2015.

[15] R. Mo, Y. Cai, R. Kazman, L. Xiao, and Q. Feng. Architecture anti-patterns: Automatically detectable violations of design principles. *IEEE Transactions on Software Engineering*, 2019.

[16] N. Nagappan, T. Ball, and A. Zeller. Mining metrics to predict component failures. In *Proc. 28th International Conference on Software Engineering*, pages 452–461, 2006.

[17] T. J. Ostrand, E. J. Weyuker, and R. M. Bell. Predicting the location and number of faults in large software systems. *IEEE Transactions on Software Engineering*, 31(4):340–355, 2005.

[18] L. Xiao, Y. Cai, and R. Kazman. Design rule spaces: A new form of architecture insight. In *Proc. 36rd International Conference on Software Engineering*, 2014.

[19] L. Xiao, Y. Cai, R. Kazman, R. Mo, and Q. Feng. Identifying and quantifying architectural debt. In *Proc. 38thInternational Conference on Software Engineering*, pages 488–498, 2016.

Threat and Security Modeling for Secure Software Requirements and Architecture

Michael Shin, Don Pathirage, Dongsoo Jang
Department of Computer Science, Texas Tech University
Lubbock, Texas, USA
michael.shin@ttu.edu, don.pathirage@ttu.edu, dongsoo.jang@ttu.edu

Abstract – Most of the threat modeling approaches do not stipulate when and what types of threats should be identified and modeled in each software development phase. This paper addresses a threat and security modeling approach in software requirements and architecture. The threats to software systems are classified and modeled as input and output, class and message threats in software requirements, and message communication threats in software architecture so that the security countermeasures are modeled and designed against the threats. The modeling of threats and security countermeasures is described by means of the underlying meta-models of software requirements and software architecture models. An online shopping system is used to demonstrate the approach.

Keywords – Threat; Security Modeling; Meta-Model; Software Requirements; Software Architecture.

I. INTRODUCTION

The identifying and modeling of threats to software systems are the starting point of developing secure systems that protect sensitive assets from the threats. A secure system is specified and designed together with security countermeasures to neutralize the threats. A secure system enables to defeat only the attacks caused by the identified and modeled threats. A security-sensitive system can be vulnerable to attacks if any critical threats are not identified and modeled in software development.

Several threat modeling approaches [Schneier99, Abi-Antoun06, Torr05, Sindre05, McDermott99, Srivatanakul05, Lund11, Shevchenko18] have been suggested to model the threats to software systems. However, most of the threat modeling approaches do not stipulate when and what types of threats should be identified and modeled in software development. The approaches might overlook the threats unnoticed because they may require all threats to be identified in only one development phase (e.g., requirements specification or design).

This paper aims at developing a threat and security modeling approach that describes when and what types of threats are identified and modeled in software development as well as the modeling of security countermeasures against the threats. The approach to modeling threats and security is described by means of the meta-model, which is a model of software system models.

II. RELATED WORK

Threat Modeling. Threats in a system have been modeled by several approaches, which include attack trees [Schneier99], data flow diagrams [Abi-Antoun06, Torr05], and UML-based modeling [Sindre05, McDermott99, Srivatanakul05]. Attack trees [Schneier99] provide an approach to modeling and analyzing the threats of systems, and the threats are analyzed in terms of attacker's capabilities. The design models in the research [Abi-Antoun06, Torr05] are specified with data flow diagram, and the threats to the models are identified and analyzed using scenarios of each function in a system. Several threat modeling approaches, such as misuse cases [Sindre05], abuse cases [McDermott99], and HAZOP (Hazard and Operability Analysis) [Srivatanakul05], have been developed for object-oriented software systems. The approaches model threats using the use case model in UML and capture security requirements against the threats. The CORAS in [Lund11] is a model-driven risk analysis that assesses the risks of assets in the system. The authors in [Shevchenko18] describes threat modeling methods that target different parts of the development process.

Secure Software Development. The studies in [Lodderstedt02] proposed a new modeling language based on UML for the model-driven development of secure distributed systems. The research in [Turpe17] describes the interplay of three dimensions: threats, security goals, and system design. Security patterns in [Fernandez13, Schumacher06] address the broad range of security issues that should be considered in the stages of software development lifecycle.

In earlier work by a coauthor in [Gomaa04], an approach has been described to model complex applications by modeling security requirements and designs separately from application requirements and designs using the UML notation. In later work by the coauthor in [Shin07], an approach has been described for modeling the evolution of a non-secure application to a secure application in terms of a requirements model and a software architecture. The recent work of the coauthors proposes the design of reusable secure connectors [Shin17a] and describes security failure-tolerant requirements specification and analysis [Shin17b, Shin18].

III. THREATS IN SOFTWARE REQUIREMENTS AND ARCHITECTURE MODELS

The threats to systems are determined by considering the sensitive assets that are described in software requirements and

architecture. Fig. 1 depicts the threats to sensitive assets in software requirements and software architecture. The threats to input and output (I/O) are induced by an actor's interactions with a system, which are described in software requirements specification that is modeled in this paper by means of the use case model in the unified model language (UML) [Booch05, Rumbaugh04, Gomaa11]. An actor's sensitive input to and output from a system can be compromised, whereas an actor's untrusted input can compromise the system.

The threats to classes occur when the objects of classes in the static model for requirements analysis are engaged in processing or storing sensitive data. The static model is used in UML to depict the static structural aspects of a system by defining the classes in the system, their attributes, and relationships between classes.

A sensitive message passed between objects in the communication model in requirements analysis can be threatened. The dynamic aspect of a system is captured in the communication model of UML through objects and the messages passed between objects. The sensitive messages passed between objects can be threatened.

The messages communicated between components in software architectures can be compromised when the message communication between components are insecure. A message in software architectures is sent by a component to another via a connector that encapsulates the detail of message communication. Insecure connectors can breach the sensitive messages communicated between components.
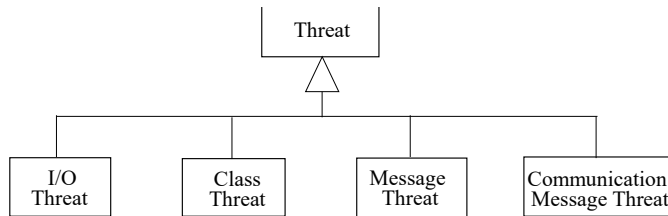
Fig. 1 Threats in software requirements and architecture

IV. Threat and Security Modeling in Requirements Specification

The threats and security in requirements specification are modeled by extending the underlying meta-model [Gomaa08] of the use case model, which is depicted in Fig. 2 in which the threat and security each are modeled as meta-classes. A meta-model is described with the meta-classes and relationships between the meta-classes. The meta-classes associated with threat and security are highlighted in gray in Fig. 2.

The threats of requirements specification concentrate on the threats to I/O that can be caused by the interaction between actors and the system. An actor's input to the system can contain sensitive data, such as a user's credit card number, which needs to be protected. On the other hand, an attacker can exploit a system with a suspicious input to the system, so the input must be verified. Some actor's access to a system must be authenticated and/or authorized in order not to disclose sensitive data in the system.

The threats to I/O are identified by means of analyzing the use case description that describes an actor's input to the system and the system's responses to the actor. The threat is described

[Shin17b, Shin18b] in terms of threat name, threat type, threat point, security asset, description, and security need.
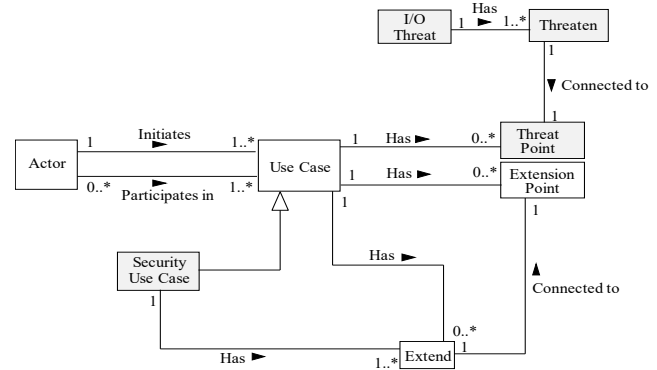
Fig. 2 Meta-Model of Use Case Model with threat and security

A threat to I/O is modeled together with a use case (Fig. 2) in which the threat is represented using the use case notation. Fig. 3 depicts an *unauthenticated ID* input threat, which threatens the *make order request* use case at the order request threat point. The *make order request* use case is an application use case in an online shopping system [Gomaa11], verifying a customer credit card payment and processes a customer's order request.
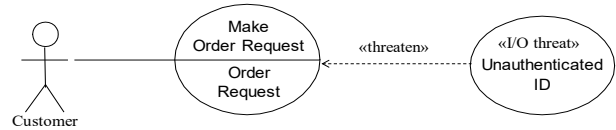
Fig. 3 Threat to Make Order Request use case

A security countermeasure against an I/O threat is specified using a security use case [Gomaa04], separately from application use cases (Fig. 2). When a system requires a security countermeasure, the security use case extends the application use case at an extension point. The *check account password* security use case (Fig. 4) is taken for the *make order request* use case against the *unauthenticated ID* input threat (Fig. 3).
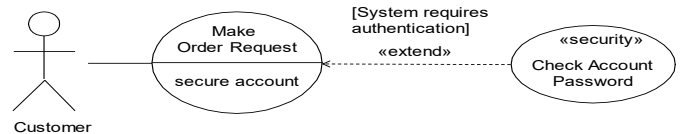
Fig. 4 Security use case for Make Order Request use case

V. Threat and Security Modeling in Requirements Analysis

A. Class Model

The threats and security in the static model for software requirements analysis are described in the underlying meta-model [Gomaa08] of the class model (Fig. 5b). The threats to classes are described as the class threat meta-class that threatens the class meta-class through the threaten meta-class. The security countermeasures are described as the security class meta-class that is specialized from the class meta-class. The

class meta-class supports the use case meta-class in the use case meta-model (Fig. 5a), which needs classes to implement the functionality of the use case.
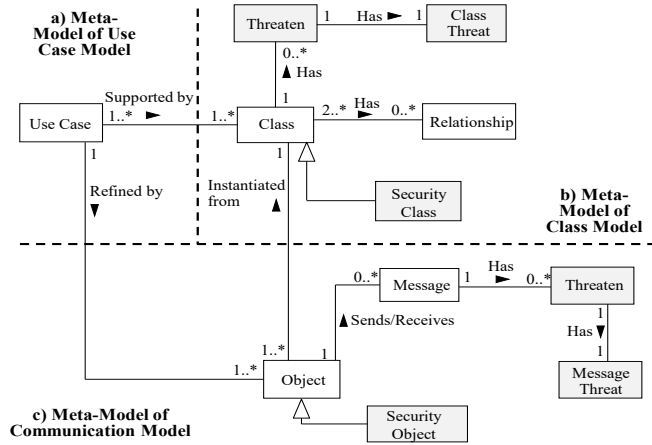
Fig. 5 Meta-Model of Requirements Analysis

The class threats are identified by considering the role of each class, such as interface, entity, control, or application logic [Gomaa11]. An interface class that interfaces to and interacts with an actor must be secure to receive sensitive input from or send sensitive output to the actor. An entity class that stores data might have sensitive data (e.g., patient medical record) that requires security. A control class that contains the system's sensitive state information or coordination logic might be compromised. An application logic class that contains the details specific to applications might be tampered with.

Part of the static model for *make order request* use case (Fig. 4) is depicted in Fig. 6a using the class diagram, which includes the *Customer Account* class that stores customer's account data in the *make order request* use case (Fig. 4), and *Account Password Checker* security class that verifies a customer's account password in the *check account password* security use case (Fig. 4). A credit card disclosure class threat (Fig. 6b) is identified and modeled for the *make order request* use case. Also, the security countermeasure against the credit card class disclosure threat is modeled with *Encryption* and *Decryption* security classes (Fig. 6a). Fig. 6b depicts the class threat description for credit card disclosure class threat.

a) Class Threat and Security Classes for Customer Account class **b) Class Threat Description for Credit Card Disclosure**
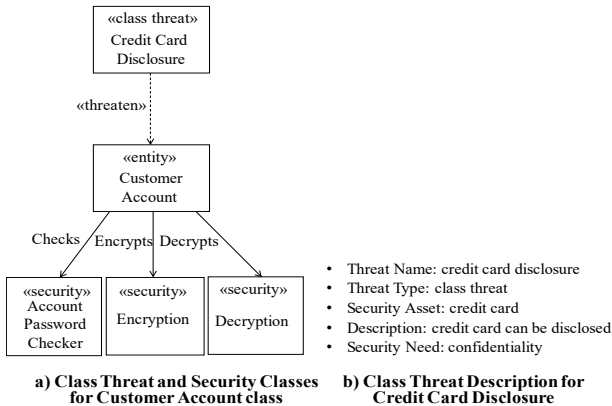
Fig. 6 Credit Card Disclosure class threat and its description

B. Communication Model

The meta-model [Gomaa08] of the communication diagram (Fig. 5c) describes the threats to messages passed between objects and the security against the threats. The security object meta-class is specialized from the object meta-class (Fig. 5c), which is utilized to refine the use case meta-class in the use case model (Fig. 5a).

The message threats in the communication model can be the threats to confidentiality, integrity, or non-repudiation security of messages from the application perspective. When a sensitive message is sent by an object to another, the message must be confidential or must not be tampered with. Also, non-repudiation may be required to prove the message presence between objects later. The message threats are determined by examining the messages passed between objects on the communication diagrams. The security objects against the message threats are determined and incorporated into the communication diagrams.

The *Order Request Repudiation* message threat is modeled in Fig. 7a in which it threatens the *order request* message in the *make order request* use case when the *Customer Interface* object sends the *order request* message to the *Delivery Order* entity object. Fig. 7b depicts the message threat description for the *Order Request Repudiation* message threat. As the *Order Request Repudiation* message threat is identified, a digital signature is taken as a security countermeasure, which is realized by means of the *Digital Signature Generator* security class (Fig. 7a), and the *Digital Signature Verifier* security class (Fig. 7a).

a) Order Request Repudiation threat **b) Message Threat Description for Order Request Repudiation threat**
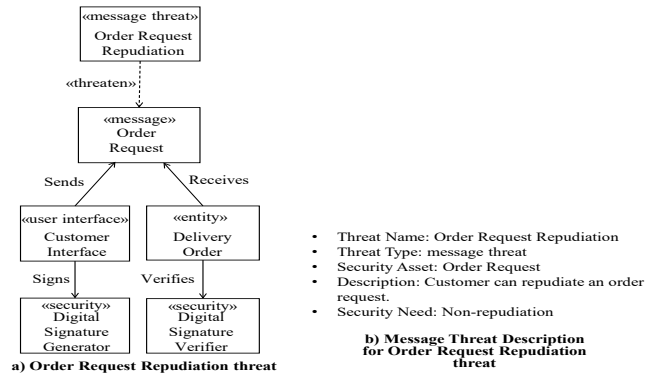
Fig. 7 Order Request Repudiation message threat and its Description

VI.   THREAT AND SECURITY MODELING IN SOFTWARE ARCHITECTURE

The threats and security in software architecture are described in the underlying meta-model of software architecture model (Fig. 8b) in which a component sends a message to or receives it from another through a connector. The message communication threat meta-class threatens the message meta-class between components. The secure connector meta-class is specialized from the connector meta-class. The component meta-class consists of object meta-class in the meta-model of communication model (Figs. 8a and 5c).

The message communication threats are identified in terms of integrity, confidentiality, non-repudiation, authentication and authorization from the software architecture perspective.

The sensitive messages communicated between components can be tampered with or can be disclosed. Fig. 9a depicts a message communication threat to the *order request* message sent by a Customer component to a Delivery Order component in which the order request can maliciously be changed. Fig. 9b depicts the message communication threat description for the *order request* message.

a) Meta-Model for Communication Model
b) Meta-Model for Software Architecture

Fig. 8 Meta-Model of Software Architecture

a) Changed Order Request Message Communication threat
b) Changed Order Request Message Communication Threat Description
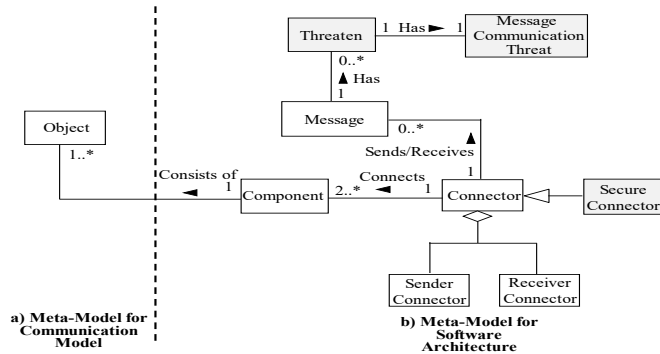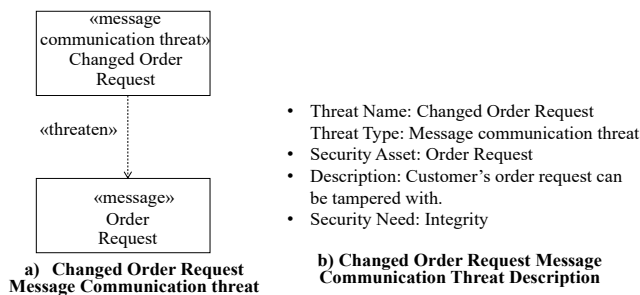
Fig. 9 Changed Order Request communication message threat

For countering the message communication threats, a secure connector [Shin17a, Shin18a] is designed by separately considering the message communication pattern and the security patterns required by application components. A secure connector (Fig. 8) is a distributed connector, which consists of a secure sender connector for a sender application component, and a secure receiver connector for a receiver application component.

VII. CONCLUSIONS AND FUTURE WORK

This paper describes a threat and security modeling approach for software requirements and architecture, which is described by means of the meta-model of the use case model, static model, communication model, and software architectural model. The threats are classified and modeled as I/O threat, class threat, message threat and message communication threat. The security countermeasures are modeled as a security use case for an application use case, a security class for an application class, a security object for an application object, and a secure connector for a simple connector.

This paragraph describes future research for the threat and security modeling of software requirements and architectures. A way of future research could be designing security fault-tolerant secure connectors, which tolerate the breaches of software architectures.

REFERENCES

[Abi-Antoun06] M. Abi-Antoun, D. Wang and P. Torr, "Checking Threat Modeling Data Flow Diagrams for Implementation Conformance and Security", International Conference on Automated Software Engineering, pp. 392-396, Atlanta, Georgia, USA, November 6-9, 2007.

[Booch05] G. Booch, J. Rumbaugh, and I. Jacobson, "The Unified Modeling Language User Guide", 2nd Edition, Addison Wesley, Reading MA, 2005.

[Fernandez13] Fernandez, E. B., 2013. Security Patterns in Practice, Wiley.

[Gomaa11] H. Gomaa, "Software Modeling and Design: UML, Use Cases, Patterns, and Software Architectures", Cambridge University Press, 2011.

[Gomaa04] H. Gomaa and M. E. Shin, "Modeling Complex Systems by Separating Application and Security Concerns" 9th IEEE International Conference on Engineering of Complex Computer Systems (ICECCS 2004), Italy, April 2004.

[Gomaa08] H. Gomaa and M. E. Shin, "Multiple-View Modeling and Meta-Modeling of Software Product Lines," IET Software, Volume 2, Issue 2, April 2008, pp. 94 – 122.

[Gomaa11] H. Gomaa, "Software Modeling and Design: UML, Use Cases, Patterns, and Software Architectures", Cambridge University Press, 2011.

[Lodderstedt02] T. Lodderstedt, D. Basin, J. Doser, "SecureUML: A UML-Based Modeling Language for Model-Driven Security", Fifth International Conference on the Unified Modeling Language, pp. 426-441, Dresden, Germany, September 30 – October 4, 2002.

[Lund11] M. S. Lund, B. Solhaug, and K. Stølen, "Model-Driven Risk Analysis", Springer Berlin Heidelberg, 2011.

[McDermott99] J. McDermott and C. Fox, "Using Abuse Case Models for Security Requirements Analysis," In Proceedings of 15th Annual Computer Security Applications Conference (ACSAC`99), pp. 55-64, Phoenix, Arizona, December 1999.

[Rumbaugh04] J. Rumbaugh, G. Booch, and I. Jacobson, "The Unified Modeling Language Reference Manual", 2nd Edition Addison Wesley, Reading MA, 2004.

[Schneier99] B. Schneier, "Attack trees: Modeling security threats," Dr. Dobbs Journal, pages 21–29, December 1999.

[Schumacher06] Schumacher, M., Fernandez, E. B., Hybertson, D., Buschmann, F., Sommerlad, P., 2006. Security Patterns, Wiley.

[Shevchenko18] N. Shevchenko, T. A. Chick, P. O'Riordan, T. P. Scanlon, and C. Woody, "Threat Modeling: A Summary of Available Methods", SEI/Carnegie Mellon University, July 2018.

[Shin07] M. E. Shin and H. Gomaa, "Software Modeling of Evolution to a Secure Application: From Requirements Model to Software Architecture," Science of Computer Programming, Volume 66, Issue 1, pp. 60-70, April 2007.

[Shin17a] M. E. Shin, H. Gomaa, and D. Pathirage, "Model-based Design of Reusable Secure Connectors," 4th International Workshop on Interplay of Model-Driven and Component-Based Software Engineering (ModComp2017), September 17, Austin/Texas, USA, 2017.

[Shin17b] M. E. Shin and D. Pathirage, "Security Requirements for Tolerating Security Failures," 29th International Conference on Software Engineering and Knowledge Engineering, Pittsburgh, USA, July 5-7, 2017.

[Shin18] M. Shin, D. Pathirage, and D. Jang, "Analysis of Security Failure-Tolerant Requirements," The 30th International Conference on Software Engineering and Knowledge Engineering (SEKE2018), San Francisco Bay, California, USA, July 1-3, 2018.

[Sindre05] G. Sindre and L. Opdahl, "Eliciting Security Requirements with Misuse Cases," Requirements Engineering, vol. 10, Issue 1, pp. 34-44, January 2005.

[Srivatanakul05] T. Srivatanakul, "Security Analysis with Deviational Techniques," PhD thesis, Department of Computer Science, University of York, UK, 2005.

[Torr05] P. Torr, "Demystifying the Threat-Modeling Process," IEEE Security and Privacy, vol. 03, no. 5, pp. 66-70, September/October 2005.

[Turpe17] S. Turpe, "The trouble with security requirements", 25th International Requirements Engineering Conference (RE), pp. 122-133, Lisbon, Portugal, September 4-8, 2017.

Knowledge-based Interface transition diagram for SRS(Software Requirements Specification) in mobile application*

Taeghyun Kang
Department of Computer Science
University of Central Missouri
Warrensburg, MO, USA
tkang@ucmo.edu

Hyungbae Park
Department of Computer Science
University of Central Missouri
Warrensburg, MO, USA
park@ucmo.edu

Venkata Inukollu
Department of Computer Science
Purdue University Fort WAYNE
Fort Wayne, IN, USA
inukollv@pfw.edu

Abstract—**This paper presents a phased development of user requirements for mobile applications. The usage of mobile applications has become increasingly prevalent and indicates the most rapid and exponential growth. Nonetheless, the number of unsatisfactory apps has been growing owing to the miscommunication between stakeholders and developers. Low-fidelity and prototype tools help to define user requirements by visualizing the needs of the users. However, a user who doesn't have any foundation in using prototype tools, will have difficulties in representing the requirements sufficiently. In this paper, the authors have proposed solutions to efficiently develop user requirements based on their understanding of the application and prototype tool. The authors of this paper suggest incremental developments of requirements in accordance with the user's knowledge in expressing the requirements for a mobile application.**

Index Terms—**Interface transition diagram, UML, Requirement Engineering, mobile application.**

I. INTRODUCTION

With the exponential rise in the number of mobile device users, the demand for superior quality apps is also increasing. The number of smartphone users is expected to reach up to 3.5 billion [1] and mobile apps are estimated to generate more than 935 US billion dollars in revenues [2]. To develop high quality mobile applications desired by the end users, the customer requirements should be clearly defined and each element of the application needs to be efficiently presented on a page. Therefore, many software projects use a prototype to record customer's requirements and also ensure not to misrepresent the requirements to the development team. Various prototype software tools are used in the IT industry, however, the tools need to be used differently based on the classification of the software product. There are 2 fundamental types of Software product:generic product and customized product.

Generic software products, such as gaming apps or apps that provide a service to the users are marketed and sold to anyone who need them. Software development team analyzes and gathers the needs of the customers and processes the project according to the requirements. The requirements and layout of a generic product are usually defined by the development team. Developers and Designers are required to invest time in learning to use the prototype tool. Though, each prototype software tool provides different interface and functionalities, usage of the tool is not a big constraint given the diverse experiences of the developers and designers.

In the case of the customized product, a user who does not have the necessary programming skills, places a request with the development company for the desired product. Requirements in a customized product are defined outside of the development team. Also, there are significant limitations for the end user or customer to learn the prototype tool and deliver the desired functions with the screen layout.

UML(Unified Modeling Language) has become the international standard to analyze user's requirements and to define static and dynamic model of an application [3]. Use case diagrams and use case descriptions are used to define customer's requirements. Class diagrams represent static structure of an application and sequence diagrams depict dynamic interaction between objects in an application. These diagrams are used as a guide for software development. However, the survey of UML practitioners shows considerable variation using the above diagrams [4]. Various devices such as mobile devices, self-driving cars, and digital machines, have been developed and many types of software applications that come with it also have varied, but there is no standard guide on which UML diagrams should be used for each type of application to improve the quality of an application. Also, while UML diagram specification is released with time, there is no newly introduced UML diagram which can support new devices and paradigms like agile. In addition, the assumption that UML design is not required in Agile, undermines the advantages of using UML.

This paper describes an approach to define user requirements for mobile applications by considering the characteristics of a user. Characteristics to develop good and efficient requirements for mobile applications are defined, and, class of users are identified based on the user's knowledge in expressing the requirements for mobile application. Different levels of ITD (Interface transition diagram) are designed to enhance communication between customers and developers.

II. Related Work

The interviews and prototype are the most widely used techniques to elicit requirements in mobile application [5]. the literature review also shows that user participation is the most common issue in the requirement gathering process.

Use Case diagram is introduced by Ivar Jacobson, it is widely used to represent functional requirements [6]. The functionality of an application is described using actors, use cases, an association between actor and use case, and system boundary. Use Case diagram provides an overview of the system and defines key requirements [7]. However, there are limitations in expressing interface layout and finger gestures, which are key elements of mobile application.

The behavior of an application on mobile devices relies on finger gestures, sensors, and location data unlike software running on a desktop. Furthermore, due to the limited screen size, screen layout and design have a significant impact on the success of a mobile application [8], [9]. Therefore when customers define their requirements, the requirements engineering tool should encourage customers to consider the characteristics of a mobile applications.

The software requirements specification describes the capabilities of an application [10]. The SRS includes functional, non-functional requirements and external interface requirements. Also, a separate interface specification document can be developed for a critical aspect of an embedded system. It describes the layout of an application, connections between an application and other software components, an interface between the software components and hardware, and communication functions such as email and network protocols.

R. Hennicker et al. [11] proposed UWE(UML-based Web Engineering Approach) to design user interfaces. UWE consists of three design phases: conceptual, navigation and presentation design. The conceptual design shows internal structure of an application and navigation design identify the instances in an application and shows when they are used. The instances which is identified in navigation space model, are used in the presentation design which shows an abstract interface. The author also proposed the UML profile to build an abstract user interface and storyboarding scenarios from the navigation space model [12].

P. Abrahamsson et al. [13] proposed The Mobile-D approach to overcome technical constraints of mobile environment. The approach is consist of well-known agile practices such as continuous integration, pair programming, and user-centered focus. However it does not provide detail guides how user express their requirements.

III. Characteristics of Good Requirements for Mobile Application

A. Agility

The Agile Manifesto was published for lightweight software development and to focus on customer's satisfaction [14]. The result of the study indicates that agile methods are well suited for the development of mobile applications due to the dynamic and incomplete requirements of the mobile application [15]. Thus, quick response to customer request is a key factor for a successful project. Similarly, customers must communicate their requirements to the development team as clearly and quickly as possible. Due to the limitation of the screen size, the interface layout in an application and events based on finger gesture need to be reflected in the requirement. Many prototype applications have been developed and used to express customer's requirements, but customers are often unfamiliar with the use of these prototype tools. Customers therefore need a way of expressing their requirements which is not constrained by a prototype tool.

B. Finger gestures in mobile application

Interacting with mobile devices is greatly different from interacting with desktops or laptops. The desktop applications mainly use keyboard and mouse. However, mobile applications can be controlled by various finger gestures [16]. Therefore, these finger movements should be represented at design phase. The basic finger movements that control mobile applications are explained in table I. Mobile platforms such as android, IOS, and windows phone support more core finger gestures. For example, Android uses two-finger swipe down to immediately display switches for wi-fi, Bluetooth, mobile data, and the like. Thus, clear and diverse finger gesture information should be shared between customers and developers for representing customer's requirements accurately. Furthermore, more finger gestures can be defined according to the advancement of the mobile device's hardware. If the prototype tool does not immediately reflect these new changes, the customer cannot accurately reflect the desired requirements.

Gesture	Description
Tap	Press and release a portion of the screen
Double Tap	Press and release same part of the screen twice within a certain time(1second)
Long Tap	Press and hold the same part of the screen for a certain time (1second)
Drag	Press a portion of the screen, then move holding the finger on the screen and release
Flick	Press a portion of the screen, Quickly move and release.
Pinch in	Pinch inward
Pinch out	Pinch outward
Rotate	Press a portion of the screen with two finger, and rotate left and right
pan	Hold a side of palm on the screen, then Quickly move and release.
Shake	Move a smartphone up and down or side to side with rapid movements
Scroll	Move one finger across the screen without lifting. Drag a list up or down

TABLE I
Basic finger gestures to control mobile application

C. Interface design

In mobile applications development, the UI design is considered as one of important phases in the development process and the user requirements are highly dependent on the interface design [17]. Thus, the layout of an application, the events

and detailed design should be considered together in tandem to express user requirements. Text-based requirement specifications are not intuitive and often lead to misinterpretations.

IV. CLASSIFICATION OF USER GROUP

User requirements of a customized product are defined outside of the development team. Unambiguous, clear, understandable and complete requirements should be delivered to the development team. However, most end users are unfamiliar with how to develop a good and requirements document. Thus, a developer needs to know the characteristics of a user to be able to distinctly understand the action items and what information should be gathered from the customer. The characteristics of a user in defining requirements is summarized in Table II.

Type of software product	Knowledge of using a prototype tool (Y/N)	Determined layout (Y/N)	Detailed Design (Y/N)
Customized product (C1)	N	N	N
Customized product (C2)	N	Y	N
Customized product (C3)	N	Y	Y
Customized product (C4)	Y	N or Y	N
Customized product (C5)	Y	Y	Y
Generic product (P1)	N/A	N/A	N/A

TABLE II
CHARACTERISTICS OF CUSTOMER

The class of an user can be identified by five groups for customized product and one group for generic product, based on the knowledge of using a prototype tool and the ability to design user interfaces. The first group in the category of customized product, does not know how to use the prototype tool. In addition, a layout and detailed design is not determined. In the scenario of first group, the Customer can only define functionalities of an application. The second User Group also has no experience in using prototype tool. Nevertheless, they have more specific requirements regarding functionalities and layout of the application's. Therefore, detailed design choices are to be suggested by development team. The user in the third group has detailed requirements for design and interface layout. So they need a way of succinctly communicating requirements without using a prototype tool. Lastly, customers can express their requirements using a prototype tool. Customer's requirements can be changed in the process of listening to developers and other design experts. However, base requirements specifications and interface design are determined by the customers and these tasks can be omitted in the development process done by developers. The detailed design cannot be developed without considering the application's layout. thus, such a user group is not classified in the table.

The authority of requirements and interface design for generic products are owned by the development team. They develop an application to compete with other products in the market. In order to understand the needs of customers in the market, the development team can survey or interview a selected group of users to define the required mobile application characteristics. During the market research, the developer requires a communication tool such as use case diagrams to understand the needs of the users. However, use case diagrams are not suitable for identifying the requirements related to mobile application's layout and detailed interface design.

V. INTERFACE TRANSITION DIAGRAM FOR REQUIREMENT SPECIFICATIONS OF MOBILE APPLICATION

Various approaches are used to determine a customer's requirements. Nonetheless, there are still many constraints in delivering accurate requirements. Use case diagrams assist in visualizing user requirements and do not require long training session to learn, but, use case diagrams cannot describe detailed functionalities and user interface. Interview or survey techniques are document based; thus the initial requirements can be misleading over time. Requirements for a customized product should be defined at the user's level. Requirements of mobile applications can be expressed using low, medium, and high fidelity founded on the user's knowledge of prototype and design tools.

A. Low fidelity diagram

Low fidelity (LF) diagrams can be used for the C1 group in the Table II. LF diagrams can express basic finger gestures and interface transition according to them. Fig 1 depicts LF diagram that is divided into three sections like class diagrams. The first section identifies user interface using an interface name. Second section is used to briefly describe the main purpose of the page and last section contains element name, type, and finger gesture. Finger gesture makes UI transition. UI transition is represented using solid line with closed and filled arrowhead.
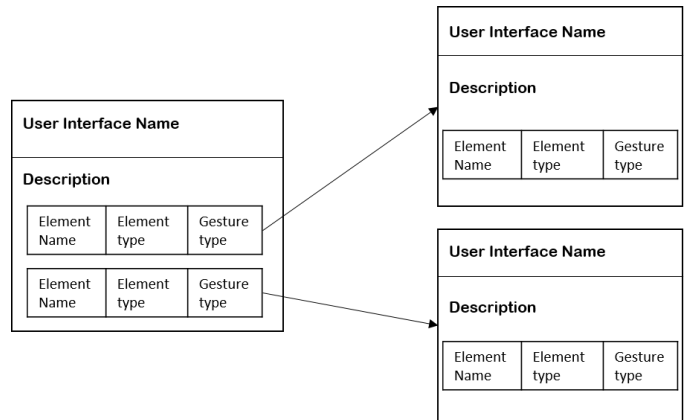
Fig. 1. Low fidelity interface transition diagram for UI transition

The popup menu is indicated by (p) in the page name. If the gesture on the element in the popup page triggers page transition, it is represented by using solid line with closed and filled arrow head as seen in Fig 2. On the other hand, if the gesture closes the popup menu and gets the user to remain on the page that opens the popup menu, the gesture

is represented using dashed line with closed and filled arrow head. Relationship between elements is represented by a thick solid line surrounding the elements group.
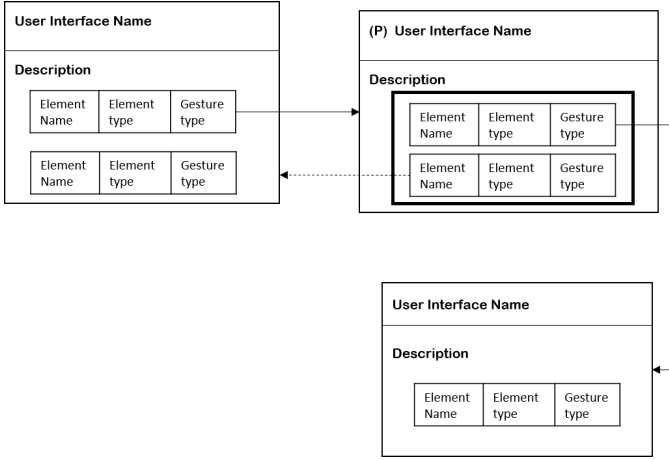
Fig. 2. Low fidelity interface transition diagram for grouping of element and popup menu

If the finger gesture activates an action only when it satisfies a specific condition, it is described by using diamond like flowchart. The condition in the diamond symbol requires a "Yes" or "NO" response and branch to the different user interface accordingly (Fig 3).
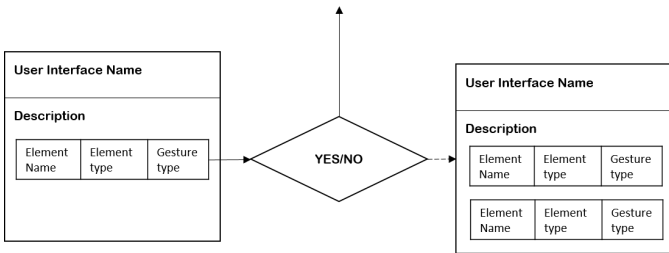
Fig. 3. Conditional Transition of an interface

One finger gesture in a mobile application can trigger two or more events and these actions can be represented with open arrow and a solid line (Fig 4). The purpose of LF diagram is to define elements, action, and finger gestures that trigger the action without considering a layout and detailed design.

B. Medium fidelity diagram

Medium fidelity(MF) transition diagram is used to determine the layout of an interface with information in LF transition diagram. The C2 user group in table II, can use MF diagram to define requirement specifications. Customer places an element without considering the details and style of the design. Events that trigger an interface transition are described on the transition line (Fig 5). Type of transition line and conditional events are expressed in the same way as LF.

If LF is already implemented and additionally customer defines MF, customer defines a layout for each interface and
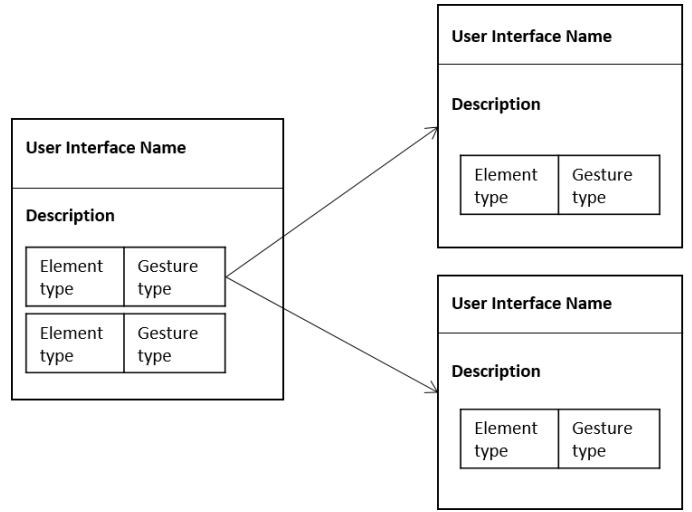
Fig. 4. A finger gesture can activate two different events
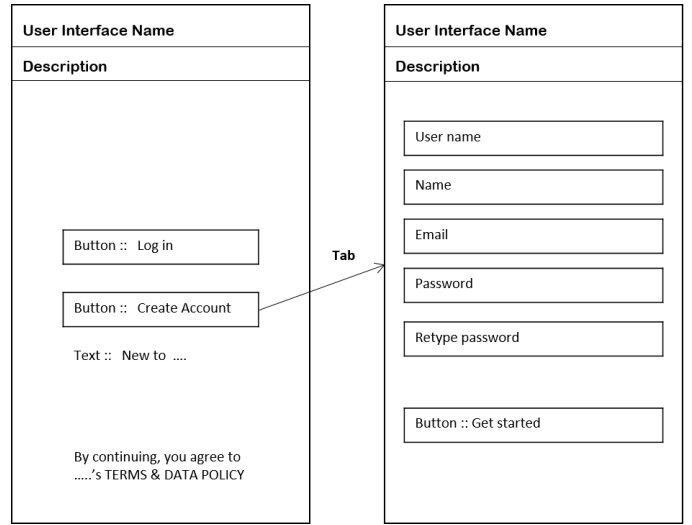
Fig. 5. Medium fidelity interface transition diagram

associates the page layout with LF using the page name and the element name (Fig 6).

C. High fidelity diagram

User group C3 in the table II can use High Fidelity (HF) diagram to define their requirements. C3 user group can define and provide a basic design information such as size of an element and color (Fig 7). However, most end users do not have professional designing skills/knowledge. Thus, an interface designer of the development team needs to provide a detailed guide on the MF diagram.

VI. INTERFACE TRANSITION DIAGRAM FOR REQUIREMENT SPECIFICATIONS OF MOBILE APPLICATION

Requirements specification is one of the primary and crucial steps in the software development life cycle. Incorrectly defined or misrepresented requirements are one of the key factors that lead to the failure of the whole project. The following

case study illustrates the process of defining user requirements for a "Log In" page of a mobile application using LF, MF and HF diagram. Customers with inadequate knowledge of mobile application development, can define only functional requirements using LF diagram.
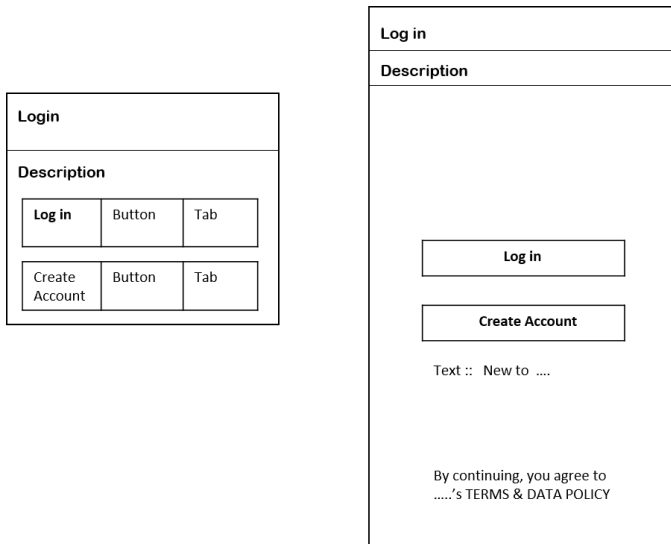
Fig. 6. mapping between LF and MF
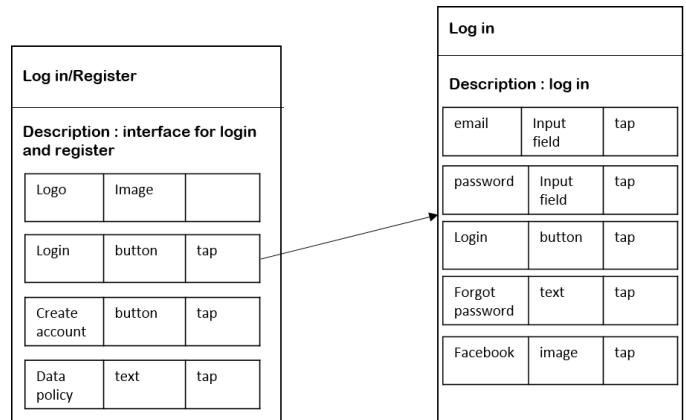
Fig. 8. Low fidelity interface transition diagram for Login page

The functional requirements of the login page, can be represented by defining element's name, type and event (Fig 8). Development team needs to evaluate and clarify the customer's requirement and suggest a layout of the application by using the MF diagram (Fig 9). If a customer has a desired layout for an application, the customer can define functional requirements and layout with the help of the MF diagram. In this scenario, the development team can save time in defining the interface layout. Nevertheless, in order to proceed with the actual mobile application development, the requirements of detailed design need to be specified and defined distinctly.
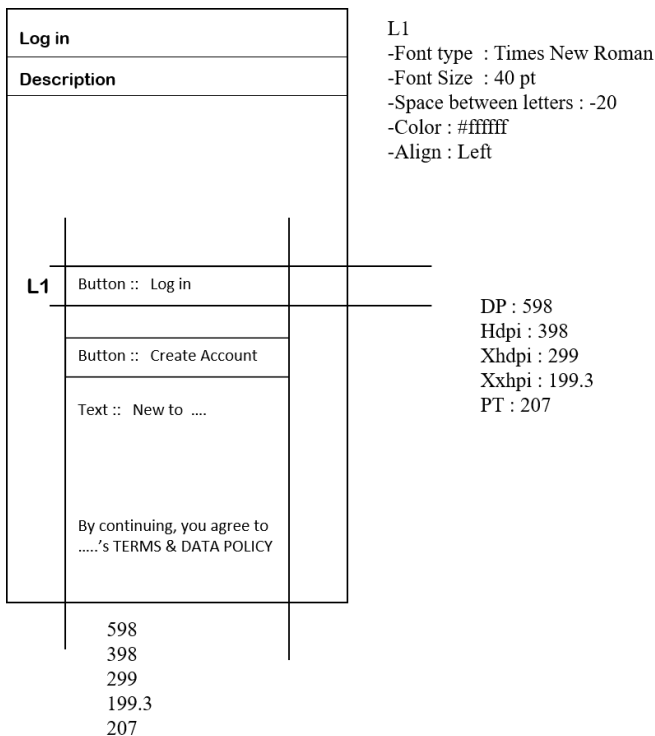
Fig. 7. A finger gesture can activate two different events
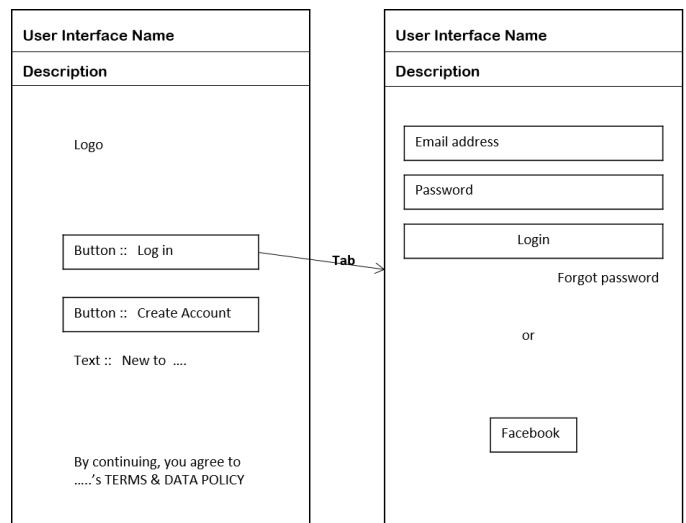
Fig. 9. Medium fidelity interface transition diagram for Login page

If a customer doesn't have specific requirements on the design, the development team needs to elicit design requirements with a prototype or HF diagram. HF diagrams are limited in

expressing detailed design, but, they can present basic designs and facilitate collaboration with the design team.

VII. CONCLUSION

This paper has described an approach to define user requirements for mobile application based on the user's knowledge of using prototype tool. Most users are unfamiliar with the prototype tool and often do not know how to express their requirements.

Users can use Low fidelity, Medium fidelity or High fidelity diagrams. LF diagrams can identify element and event on the page. Users who have requirements of interface layout, can use MF diagrams. Basic design elements such as color, font, alignment, and size of element can be expressed using HF diagram. Furthermore, LF, MF, and diagrams enable incremental definition of requirements. Since the inputs and outputs of each diagram are clearly defined, the developers and users can easily identify their tasks to be done next.

This paragraph describes the future research for the interface transition diagrams. The diagrams need to be validated by further applying to different applications. Also, systematic management of changed requirements need to be developed.

REFERENCES

[1] https://www.statista.com/statistics/330695/number-of-smartphone-users-worldwide/

[2] https://www.statista.com/statistics/269025/worldwide-mobile-app-revenue-forecast/

[3] Kobryn, C. (1999). UML 2001: A standardization odyssey. Communications of the ACM, 42(10), 29-37.

[4] Dimensions of UML Diagram Use: A Survey of Practitioners

[5] H. Dar, M. I. Lali, H. Ashraf, M. Ramzan, T. Amjad, B. Shahzad, "A systematic study on software requirements elicitation techniques and its challenges in mobile application development", IEEE Access, vol. 6, pp. 63859-63867, 2018.

[6] I. Jacobson, M. Christerson, P. Jonsson. and G. Overgaad, "Object-Oriented Software Engineering: A Use Case Driven Approach," Addison-Wesley, Wokingham, 1992.

[7] F. Alhumaidan, "A Critical Analysis and Treatment of Important UML Diagrams Enhancing Modeling Power," Intelligent Information Management, Vol. 4, No. 5, pp. 231-237, 2012.

[8] A.Wasserman, Software Engineering Issues for Mobile Application Development, Proc. of the FSE/SDP workshop on Future of software engineering research, FOSER 2010, IEEE Comp.b16 Soc. Press, pp. 397-400

[9] J. Dehlinger and J. Dixon, "Mobile application software engineering: Challenges and research directions," in Proceedings of the Workshop on Mobile Software Engineering. Springer, 2011, pp. 29-32.

[10] Wiegers KE (1999) Software requirements. Microsoft Press, Redmond, WA

[11] Hennicker R., Koch N.: A UML-based Methodology for Hypermedia Design. In Proceedings of UML 2000, Evans, A., Kent, S. (Eds), LNCS, Vol. 1939. SpringerVerlag (2000) 410-424.

[12] HENNICKER R. and KOCH N. 2001. Modeling the User Interface of Web Apllications with UML. In Practical UML-Based Rigorous Development Methods, Workshop of the pUML-Group at the UML´01, Gesselschaft für Informatik, Köllen Druck-Verlag

[13] P. Abrahamsson, A. Hanhineva, H. Hulkko, T. Ihme, J. Jäälinoja, M. Korkala, J. Koskela, P. Kyllönen, and O. Salo, "Mobile-D: An Agile Approach for Mobile Application Development," presented at OOPSLA 2004, Vancouver, Canada. 2004.

[14] Kent Beck, "Manifesto for Agile Software Development". Agile Alliance, http://agilemanifesto.org, 2001.

[15] Flora, H. K., Chande, S. V., Wang, X. (2014). Adopting an agile approach for the development of mobile applications. International Journal of Computer Applications, 94(17)

[16] C. Villamore, D. Willis and L. Wroblewski, Touch Gesture Reference Guide, April 15, (2010)

[17] Kumar, N. A., Krishna, K. H., Manjula, R. (2016). Challenges and best practices in mobile application development. Imperial Journal of Interdisciplinary Research, 2(12).

Correct Software by Design for Software-Defined Networking: A Preliminary Study

Liang Hao, Xin Sun, Lan Lin, Zedong Peng
Department of Computer Science, Ball State University, Muncie, IN 47306, USA
{lhao, xsun6, llin4, zzpeng}@bsu.edu

Abstract

We report our experience of applying rigorous software specification and design methodologies to the development of applications for the emerging software-defined networking (SDN) paradigm. While much of the prior work in the SDN space focused on creating novel algorithms and protocols, in this paper we take the position that the implementation of those algorithms and protocols on the SDN platform is a hard problem on its own that deserves a systematic treatment from the software engineering perspective. Through a concrete case study of implementing an essential switching algorithm as an SDN app, we expose the challenges stemmed from the unique three-tier architecture of SDN, and propose a rigorous approach that flows from functional requirements through stepwise refinement to design and implementation. Our case study shows promises of the proposed approach in supporting correctness arguments for the software developed for the SDN platform.

1 Introduction

Software-defined networking (SDN) is an emerging technology that has completely transformed modern networking, with widening adoptions in industries such as IT, telecommunications, retails, and healthcare, to name a few. At the core of the technology is the ability to apply software solutions to hard, long-standing networking problems while cutting the operation costs via automation.

Until very recently the focus of the SDN community was on creating new and SDN-specific algorithms and protocols that can take advantage of the unique capabilities of SDN to solve sophisticated networking problems, such as highly-dynamic, fine-grained traffic engineering and adaptive intrusion detection. However, little attention was given to the software engineering aspects of app development, as it was assumed that the implementation of those algorithms and protocols was straightforward. More recently there has

been some work, including the authors' own, on the testing and orchestration of the SDN apps that leveraged techniques from the software engineering discipline. But to the best of our knowledge, there has not been a systematic investigation on the implementation of the SDN apps guided by software engineering principles and methodologies.

In this paper we take the position that implementing SDN apps is a hard problem on its own that deserves a systematic treatment from the software engineering perspective. We present a case study of implementing a basic yet essential switching algorithm on the SDN platform. Our solution takes two iterations of rigorous software specification and design. It flows from functional requirements through stepwise refinement to design and implementation. We report our experience that shows promises of the proposed approach, which also supports correctness arguments for the software developed for the SDN platform.

2 The MAC Learning Algorithm

We introduce in this section a preliminary case study, i.e., the MAC learning algorithm. We first describe the algorithm and its implementation on a traditional switch. We then describe how the SDN architecture differs fundamentally from the traditional network architecture, and expose the challenges of migrating the same algorithm to SDN that stem from the architectural difference.

2.1 On a Traditional Switch

The MAC learning algorithm is implemented on every traditional switch, typically in the firmware. It is the core switching algorithm that enables a switch to forward packets toward their destinations. The algorithm builds the switch table and at the same time utilizes the table to determine the switch port to which a packet should be directed. An entry in the switch table contains (1) the hardware identification number, termed *Media Access Control (MAC) address*, of some host or router in the network; this address is
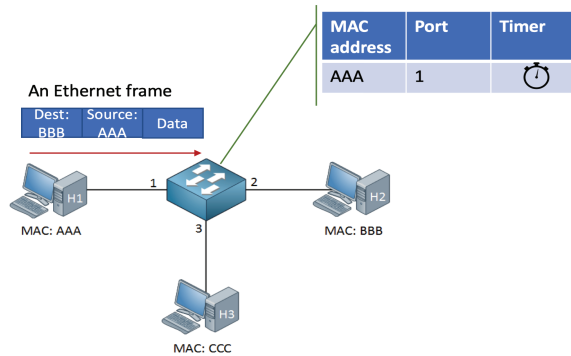
Figure 1. The learning algorithm working on a traditional network

used as the key for indexing the table, (2) the switch port leading toward that MAC address, and (3) a timer to delete the entry in the future in case it becomes stale. The table is stored in the memory and is initially empty. More specifically, the algorithm has the following components (also illustrated in Figure 1):

• For each incoming packet received on a port, the switch creates an entry in the switch table if such an entry does not exist. The entry contains (1) the MAC address in the packet's *source* address field, (2) the port from which the packet arrived, and (3) a timer set to expire after some period of time. If such an entry already exists, the timer will be refreshed. If there exists an entry with the same key (i.e., the MAC address) but a different port, the port will be updated based on the new information, and the timer be reset.

• For each incoming packet, the switch uses the MAC address in the packet's *destination* address field to look up the table. If the MAC address is listed, the switch will send the packet out the associated port; otherwise, the switch will *flood* the packet out all active ports except the incoming port (the port the packet was received on).

• The switch deletes an entry in the table when the associated timer expires. This is to handle potential topology change, e.g., hosts being removed or relocated.

2.2 The SDN Architecture

The unique three-tier architecture of SDN (illustrated in Figure 2) has important implications on the development of software for SDN, so we briefly describe it here. At the bottom tier are the hardware boxes, commonly called *SDN switches*. Compared to hardware boxes (such as switches and routers) in a traditional network, SDN switches are much dumber (and also cheaper). A traditional switch or router has the intelligence, provided by the device firmware, to decide for itself how to handle incoming packets, as the firmware implements various networking algorithms and protocols. In contrast an SDN switch does not have such
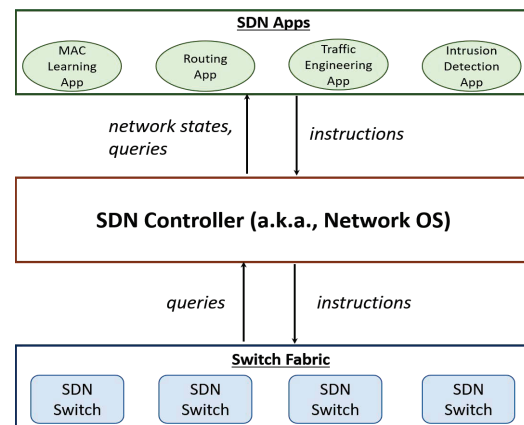
Figure 2. The three-tier SDN architecture

intelligence; it relies on the controller (the middle tier in the SDN architecture) to provide "instructions" (technically called *FlowMod messages* or simply *FlowMods*) on how to handle packets and then act accordingly. The controller is a software platform that runs on any commodity PC server, and can be viewed as the operating system for the network. On the one hand the controller interacts with the hardware boxes and provides instructions (i.e., FlowMods) to them upon request. On the other hand it provides a set of APIs that supports individual SDN apps running on top of it. The SDN apps (the top tier) collectively implement the intelligence of the network. Each app typically manages/optimizes one aspect of the network, such as switching, routing, traffic engineering, intrusion detection, etc. They obtain an abstract representation of the network state (e.g., topology, traffic load) from the controller and output to the controller instructions on how the network state should be modified and how incoming packets should be handled. The controller then compiles the instructions received from all apps to generate FlowMod messages and sends FlowMod messages to the SDN switches.

2.3 The Challenges of Migrating the Algorithm to SDN

As explained above, in the SDN paradigm the MAC learning algorithm is to be implemented as an app running on top of the controller. This app will be responsible for building the tables, one for each switch. The switches are only capable of querying the controller (which in turn consults the app) to obtain necessary instructions in the form of FlowMod messages to forward incoming packets. Such a query from a switch contains the source and destination MAC addresses of the packet and the switch port on which the packet was received. This allows the app to create or update the entry corresponding to the source MAC address in the querying switch's table. A FlowMod message from the

controller back to the switch is in the format of "send any packet with source MAC address s and destination MAC address d to the port p" or "flood any packet with source MAC address s and destination MAC address d".

It is important to note that, the SDN architecture requires the querying switch to *cache* any FlowMod message received from the controller in a local table structure called *FlowMod table*. The switch will then use the cached Flow-Mod messages to process subsequent packets with the *same source and destination MAC addresses*, without querying the controller again. A switch may cache as many instructions as its memory space permits. As a critical measure to save memory space, any cached instruction will be deleted after it has not been utilized for a period of time. This caching technique is critical to optimizing the packet processing speed on the switches, as querying the controller introduces significant delays. As an example, imagine that a large file is being transmitted over a local-area network from the host s to the host d. Tens of thousands of packets will be transmitted, all with the same source and destination MAC addresses. A switch w queries the controller when it receives the *first* packet and caches the FlowMod message from the controller. The switch will then use the cached FlowMod to forward all subsequent packets of the same file without querying the controller again. Hence the controller will only see the first packet of that file.

The fact that the SDN controller, and consequently all apps running on top it, only see a *small fraction* of all the packets in the network has an important implication on the implementation of the MAC learning algorithm, or more specifically, on refreshing the timers associated with table entries. Recall that an entry will be refreshed every time a packet from the same source MAC address is received. Because the controller does not see most of the packets, it will not be able to effectively refresh the timers. As a result, many of the timers will unnecessarily expire, causing the app to instruct the switches to flood much more frequently than necessary. Continuing from the file transmission example and imagining the switch w receives a new packet with s as the *destination* MAC address shortly after the file transmission. It queries the controller which in turn consults the app. But the app has timed out the table entry corresponding to the MAC address s because it did not see any of the subsequent packets of that file. Thus the app, through the controller, will instruct the switch to flood the packet. Clearly the flooding is unnecessary in this case because the information contained in the expired table entry is still valid. As flooding causes significant bandwidth overhead that degrades the network performance, it is highly undesirable. (On the other hand, the switch does see all the packets of the file. We will explore in Section 5 how this knowledge of the switch may be leveraged to prevent the MAC learning app from unnecessarily timing out table entries.)

3   Leveraging Rigorous Software Specification and Design Methodologies

Developing a reliable SDN app, just as developing a reliable piece of *any* software, relies on rigorous methods for code development and testing, and a development process that is based on more than heuristics. In our opinion, each SDN app should first be treated as a black box, and flow naturally through a sequence of requirements specification, design, implementation, and testing steps. In migrating the MAC learning algorithm from traditional network to SDN, we applied two rigorous methods for software specification and design, i.e., Prowell and Poore's *sequence-based specification and stepwise refinement* [10, 12, 13] and Exman's *linear software models and the modularity matrix* [8, 7].

Sequence-based specification was developed in the 90's by the University of Tennessee Software Quality Research Laboratory. It converts ordinary, functional requirements to a precise specification that defines software's response to any possible input sequence, through a systematic *sequence enumeration* process. In this process, sequences of system inputs are enumerated in length-lexicographic order and mapped to software's outputs, and grouped in equivalence classes based on behavior described in (software) requirements. The completed enumeration encodes a formal model in the form of a finite state machine (a Mealy machine) that is refined into design and implementation [10, 12, 13].

Linear software models and the modularity matrix were recently developed by Iaakov Exman in the study of real software system composition, as a formal theory of modularity. He proposed that the composition of a software system can be represented by a modularity matrix, whose rows and columns represent *functionals* (a generalization of methods) and *structors* (a generalization of classes), respectively, and 1/0-valued matrix elements indicate asserted links (associations) (or lack of) between rows (functionals) and columns (structors). He proved that a standard modularity matrix, in which one has only linearly independent structors and functionals, must be both square and block-diagonal, with disjoint diagonal blocks representing independent system modules. He showed that canonical systems strictly obey linear software models, and larger systems tend to agree with bordered linear software models with a few outliers near the diagonal block borders. The outliers point to areas of coupling that need to be resolved in system design [8, 7].

We first applied sequence-based specification to derive a rigorous specification from functional requirements for our chosen case study, and refined it into a state-based specification and design. Then we applied the modularity matrix to validate the modular design refined from the formal specification. We derived the specification in two iterations, with new findings at the end of the first iteration, which we in-

corporated into the second iteration's work product.

4 Our Solution: The First Iteration

We started with a natural language description of the behavior of the MAC learning algorithm, i.e., the software requirements, as shown in Table 1. In developing the requirements we also identified a system boundary that cuts the interfaces between the software and its external entities in the software's environment, i.e., the switches (communication with the switches is through the SDN controller), the memory (that stores the lookup tables for the switches), and the timers. Figure 3 depicts our identified system boundary for the first increment.

Table 1. MAC learning algorithm requirements: The first increment

Tag	Requirement
1	On receiving a packet with source MAC address sa, destination MAC address da from switch s and in-port p, if the lookup table for s does not contain entries for either sa or da, the learning switch should add a new entry (sa, p, t) to the same lookup table, start timer t, and flood the same packet to all the ports of s except the in-port p.
2	The output of the learning switch is solely determined by the incoming packet information, the current lookup table status, and the timer events, as encapsulated in the most recent input.
3	On receiving a packet with source MAC address sa, destination MAC address da from switch s and in-port p, if the lookup table for s does not contain an entry for sa but contains an entry for da, the learning switch should add a new entry (sa, p, t) to the same lookup table, start timer t, and forward the same packet to the port of da as specified in the lookup table (for s).
4	On receiving a packet with source MAC address sa, destination MAC address da from switch s and in-port p, if the lookup table for s contains an entry for sa but does not contain an entry for da, the learning switch should overwrite the entry (sa, p, t) to the same lookup table, restart timer t, and flood the same packet to all the ports of s except the in-port p.
5	On receiving a packet with source MAC address sa, destination MAC address da from switch s and in-port p, if the lookup table for s contains entries for both sa and da, the learning switch should overwrite the entry (sa, p, t) to the same lookup table, restart timer t, and forward the same packet to the port of da as specified in the lookup table (for s).
6	On receiving a timer going off event for the MAC address sa in the lookup table for switch s, the learning switch should delete the entry (sa, p, t) from the same lookup table, and stop timer t.

From the identified system boundary we collected software's inputs (stimuli) and outputs (responses), as shown in Table 2 and Table 3.

The sequence enumeration proceeds as follows. One explicitly enumerates all possible stimulus sequences first according to the length, and within the same length lexicographically. For each enumerated sequence, one maps it to

Table 2. MAC learning algorithm stimuli: The first increment

Stimulus (Parameterized)	Shorthand
Packet(source MAC address, destination MAC address, switch, in-port)	$pa(sa, da, s, p)$
Look up table(switch)	$lt(s)$
Timer going off(switch, source MAC address)	$t(s, sa)$

Table 3. MAC learning algorithm responses: The first increment

Response	Shorthand
Forward	forward
Flood	flood
Add a new entry to the lookup table	add
Overwrite an existing entry in the lookup table	overwrite
Delete an existing (expired) entry from the lookup table	delete
Start timer	start
Restart timer	restart
Stop timer	stop

a software's response based on the requirements, and declares it equivalent to a prior sequence if both sequences take the software to the same situation (i.e., internal state). If a stimulus sequence is operationally not realizable (for instance, when a button-pressing event happens before the power-on event), the sequence is mapped to a special *illegal* response, otherwise, it is *legal*. If a sequence is declared equivalent to a prior sequence, it is *reduced*, otherwise, it is *unreduced*. One proceeds from Length n to Length $n + 1$ only extending both legal and unreduced sequences (by every stimulus), until there are no more sequences to extend. At that point the enumeration is complete.
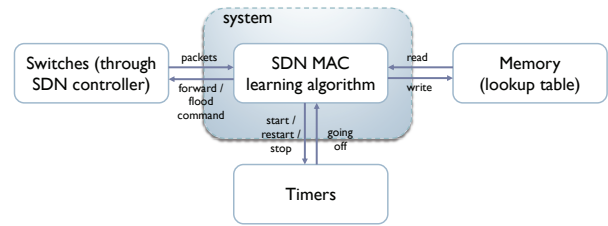
Figure 3. The system boundary for the first increment

An enumeration of the learning switch algorithm is shown in Table 4. The columns are stimulus sequences, their mapped responses, and requirements traces, respectively. We omit the column that shows reductions to prior sequences (as explained later, all the enumerated sequences

are reduced to the first sequence in the table). We started with the empty sequence λ. All the others are Length 1 sequences with either an incoming packet ($pa(sa, da, s, p)$) as the current input, or a timer going off event ($t(s, sa)$). When a packet comes in, we used predicates (in square brackets) to refine the condition based on the lookup table status, i.e., whether the source address sa and the destination address da are in the lookup table for switch s, respectively, in order to define the software's unique response deterministically. It turned out that all the Length 1 sequences are reduced to λ, suggesting a stateless software control that maps current input (with predicate refinement) to current output. We held the following assumptions in sequence enumeration: (1) The $pa(sa, da, s, p)$ and $t(s, sa)$ events are queued by the SDN controller for processing, hence cannot happen simultaneously; and (2) All the timers are set to go off after the same time interval once started/restarted.

Table 4. A MAC learning algorithm enumeration: The first increment

Sequence	Response	Trace
λ	0	Method
$pa(sa, da, s, p)[sa \notin lt(s), da \notin lt(s)]$	add (sa, p, t) to $lt(s)$, start t, flood $pa(sa, da, s, p)$ to all the ports of s except p	$1, 2$
$pa(sa, da, s, p)[sa \notin lt(s), da \in lt(s)]$	add (sa, p, t) to $lt(s)$, start t, forward $pa(sa, da, s, p)$ to the port of da in $lt(s)$	$2, 3$
$pa(sa, da, s, p)[sa \in lt(s), da \notin lt(s)]$	overwrite (sa, p, t) in $lt(s)$, restart t, flood $pa(sa, da, s, p)$ to all the ports of s except p	$2, 4$
$pa(sa, da, s, p)[sa \in lt(s), da \in lt(s)]$	overwrite (sa, p, t) in $lt(s)$, restart t, forward $pa(sa, da, s, p)$ to the port of da in $lt(s)$	$2, 5$
$t(s, sa)$	delete (sa, p, t) in $lt(s)$, stop t	$2, 6$

As one could observe, our first iteration of the specification was a direct, intuitive migration of the traditional MAC learning algorithm, replacing distributed intelligence with centralized intelligence, solely based on the algorithm's behavior. What were overlooked are the constraints enforced by the unique SDN architecture, i.e., the different roles taken by the controller and the switches, as well as the caching of instructions (in the form of FlowMod messages) on the switches. Without considering these requirements the controller would see and handle every packet that goes through every switch, making itself excessively "fat" and causing unnecessary, significant delays to degrade network performance. This observation led to our second iteration of the specification to be discussed next.

5 Our Solution: The Second Iteration

In the second iteration we took into consideration important architectural differences introduced by SDN, i.e.,

Table 5. MAC learning algorithm requirements: The second increment

Tag	Requirement
1	The SDN controller maintains a lookup table for each switch mapping MAC addresses to ports on that switch. Each switch maintains a FlowMod table that maps (source MAC address, destination MAC address, in-port) to (out-port, timer value). Recall from Section 2.3 that the FlowMod table contains the cached FlowMod messages which are instructions from the controller to the switch.
2	On receiving a packet with source MAC address sa, destination MAC address da, in-port p of switch s, if the FlowMod table of switch s has an entry for (sa, da, p), denoted by (op, t), the switch will forward the packet to port op, and restart timer with value t; otherwise, it will send the packet on to the controller for processing.
3	When the controller receives a packet with source MAC address sa, destination MAC address da from switch s and in-port p, if the lookup table for s does not contain an entry for da, the learning switch should add/overwrite an entry (sa, p) to the lookup table, and flood the same packet to all the ports of s except the in-port p.
4	The output of the learning switch is solely determined by the incoming packet information, the current lookup table status, or the FlowRemoved message, as encapsulated in the most recent input.
5	When the controller receives a packet with source MAC address sa, destination MAC address da from switch s and in-port p, if the lookup table for s contains an entry for da (with out-port op), the learning switch should add/overwrite an entry (sa, p) to the lookup table, forward the same packet on to port op, and write a FlowMod message $(s, sa, da, p, op, t, add)$ and its reversed FlowMod message $(s, da, sa, op, p, t, add)$, where t is the timer value, to switch s.
6	When a FlowMod table entry expires, the switch automatically removes it from the FlowMod table, and sends a FlowRemoved message to the controller. When the controller receives a FlowRemoved message from switch s, with source MAC address sa, destination MAC address da, and in-port p, it deletes the lookup table entry (sa, p) for s, and sends a FlowMod message $(s, da, sa, p, delete)$ to remove the reversed FlowMod table entry maintained by s.
7	Any FlowMod table entry maintained by the switch that goes stale (it only goes stale when the out-port becomes incorrect and cannot reach the destination) must eventually time out (expire) to prevent packet loss.

the two-level architecture in which the controller and the switches take on different roles. We notice the following for the SDN environment:

• Lookup tables are on the controller, rather than on the switches, and are outside of the system boundary (of the specified learning algorithm).

• The controller only sees the packets that the switches do not know how to handle (forward).

• The controller sends the switches FlowMod messages that are maintained by the switches. The FlowMod messages have different information packed than the information packed in a lookup table entry.

• No timer is associated with lookup table entries. Timer effect is simulated on the switches.

The new knowledge we learned contributed to our derived requirements for the second increment shown in Table 5. A new system boundary was identified as depicted in Figure 4, from which we defined stimuli and responses in Table 6 and Table 7.
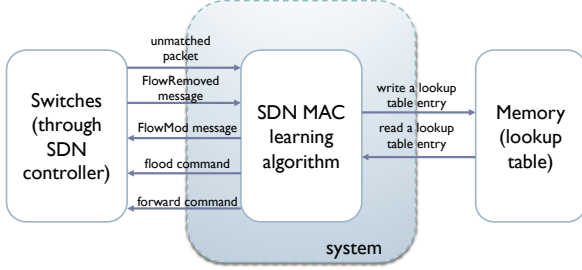
Figure 4. The system boundary for the second increment

Table 6. MAC learning algorithm stimuli: The second increment

Stimulus (Parameterized)	Shorthand
Unmatched packet(switch, source MAC address, destination MAC address, in-port)	$pa(s, sa, da, p)$
Look up table(switch)	$lt(s)$
FlowRemoved message(switch, source MAC address, destination MAC address, in-port)	$frm(s, sa, da, p)$

Table 7. MAC learning algorithm responses: The second increment

Response	Shorthand
Forward	forward
Send a FlowMod message to the switch with the following information: switch, source MAC address, destination MAC address, in-port, out-port, timer value, type (add or delete)	$flowmod(s, sa, da, ip, op, t, add)$ or $flowmod(s, sa, da, ip, delete)$
Flood	flood
Add/Overwrite a new entry to the lookup table with the following information: switch, source MAC address, in-port	$add\text{-}lt(s, sa, p)$
Delete an existing (expired) entry from the lookup table with the following information: switch, source MAC address, in-port	$delete\text{-}lt(s, sa, p)$

We completed a sequence enumeration for the second increment in Table 8. Similarly all Length 1 sequences are reduced to the empty sequence indicating a stateless software control for this simple SDN app. This is because we used predicates to refine the lookup table status, at a certain

level of abstraction, at the receipt of an unmatched packet or a FlowMod message from the switch, to deterministically identify software's behavior while keeping the enumeration productive.

Table 8. A MAC learning algorithm enumeration: The second increment

Sequence	Response	Trace
λ	0	Method
$pa(s, sa, da, p)$ $[da \notin lt(s)]$	$add\text{-}lt(s, sa, p)$, flood $pa(s, sa, da, p)$ to all the ports of s except p	$1, 2, 3, 4$
$pa(s, sa, da, p)$ $[da \in lt(s)]$	$add\text{-}lt(s, sa, p)$, forward $pa(s, sa, da, p)$ to the port of da in $lt(s)$ (denoted by op), $flowmod(s, sa, da, p, op, t, add)$, $flowmod(s, da, sa, op, p, t, add)$	$1, 2, 4, 5, 7$
$frm(s, sa, da, p)$	$delete\text{-}lt(s, sa, p)$, $flowmod(s, da, sa, p, delete)$	$1, 2, 4, 6, 7$

Refinement of a sequence-based specification into design and implementation proceeds with selecting a software architecture and capturing how to gather each stimulus, generate each response, and maintain each system state.

Towards the implementation of an SDN MAC learning algorithm, we selected Mininet [1] as the network emulator, and Floodlight [4] as the open SDN controller. There is a learning switch already provided by Floodlight, which we disabled and replaced with a simple learning switch implemented from scratch based on our formal specification. This consists of two Java files that define a class and an interface: `SimpleLearningSwitch.java` and `ISimpleLearningSwitchService.java`.

In Table 9 and Table 10 we show how one could write Java code to gather each stimulus and generate each response. No state data is needed for this simple stateless control as identified by the specification.

We defined functionals from the requirements in Table 11, and structors from our design (classes and methods) in Table 12. Figure 5 shows a standard modularity matrix for the MAC learning algorithm based on these definitions that obeys linear software models. Due to the simplicity of this app (a single class implementing the algorithm), structors correspond to a group of class methods rather than a group of classes.

Testing of the SDN learning switch algorithm turned out to be a trivial testing problem given the stateless (rather than a stateful) software control. Using predicate refinement, we were able to enforce a trajectory onto a specific lookup table state, enabling a direct mapping from the current input (e.g., an incoming packet) to the current output, and exhaustive testing of all scenarios of uses.

Table 9. MAC learning algorithm stimuli gathering

Stimulus	Design / Implementation
$pa(s, sa, da, p)$	SimpleLearningSwitch.receive(IOFSwitch sw, OFMessage msg, FloodlightContext cntx), msg.getType() == PACKET_IN, sw $\Rightarrow$ s, cntx $\Rightarrow (sa, da)$, msg $\Rightarrow p$
$lt(s)$	Map$\langle$IOFSwitch, Map$\langle$MacAddress, OFPort$\rangle\rangle$, SimpleLearningSwitch.macToSwitchPortMap, macToSwitchPortMap.get(s)
$frm(s, sa, da, p)$	SimpleLearningSwitch.receive(IOFSwitch sw, OFMessage msg, FloodlightContext cntx), msg.getType() == FLOW_REMOVED, sw $\Rightarrow$ s, msg $\Rightarrow (sa, da, p)$

Table 10. MAC learning algorithm response generation

Response	Design / Implementation
forward	SimpleLearningSwitch.pushPacket(IOFSwitch sw, Match m, OFPacketIn msg, OFPort outport)
$flowmod(s, sa, da, ip, op, t, add)$ or $flowmod(s, sa, da, ip, delete)$	SimpleLearningSwitch.writeFlowMod(IOFSwitch sw, OFFlowModCommand command, OF-BufferId buffered, Match m, OFPort output), sw $\Rightarrow$ s, command $\Rightarrow$ OFFlowModCommand.ADD or OFFlowModCommand.DELETE, m $\Rightarrow (sa, da, ip)$, outport $\Rightarrow op$, SimpleLearningSwitch.FLOWMOD_DEFAULT_IDLE_TIMEOUT $\Rightarrow t$
flood	SimpleLearningSwitch.writePacketOutForPacketIn (IOFSwitch sw, OFPacketIn msg, OFPort porttype) Pass in OFPort.FLOOD as porttype
$add\text{-}lt(s, sa, p)$	SimpleLearningSwitch.addToPortMap(IOFSwitch sw, MacAddress sourceMac, OFPort inport)
$delete\text{-}lt(s, sa, p)$	SimpleLearningSwitch.removeFromPortMap(IOFSwitch sw, MacAddress sourceMac)

Table 11. MAC learning algorithm functionals

Functional	Requirement
Processing in-packets: collecting information	3, 4, 5, 6
Processing in-packets: sending on to switches	3, 4, 5
Writing FlowMod messages on switches	2, 4, 5, 6, 7
Maintaining lookup table entries	1, 3, 4, 5, 6

Table 12. MAC learning algorithm structors

Structor	Class Method
Retrieving relevant information from incoming packets	SimpleLearningSwitch.receive, createMatchFromPacket
Sending received packets to switches	SimpleLearningSwitch.pushPacket, SimpleLearningSwitch.writePacketOutForPacketIn
Writing FlowMod messages to switches	SimpleLearningSwitch.writeFlowMod
Processing lookup table entries	SimpleLearningSwitch.addToPortMap, SimpleLearningSwitch.removeFromPortMap, SimpleLearningSwitch.inLookupTable

Figure 5. A standard modularity matrix for the SDN MAC learning algorithm

6 Related Work

Software-defined networking is a new paradigm in computer networking that is gaining significant momentum. The key concept of softwarization was first introduced in the seminal work [11]. For the next several years, the research primarily focused on the technologies that enabled the core platform. This includes the development of the controller software(e.g., Floodlight [4] and OpenDaylight [2]), the communication protocol (called OpenFlow) between the hardware boxes and the software controller [3],

the pipeline packet processing on the hardware boxes [5], and so on.

More recently and as the core SDN platform has been established, the networking community has begun to shift its attention to the development of SDN apps. The development has largely focused on the creation of advanced networking algorithms and techniques that utilize the unique capabilities provided by the SDN platform, such as direct programmability and centralized control, to solve hard and long-standing problems in networking, such as dynamic traffic engineering [9], efficient and intelligent anomaly detection [15], optimized energy efficiency [14], to name a few. Unfortunately the problem of implementing those algorithms and techniques into software applications is largely overlooked, as it has been assumed that the implementation is straightforward. This paper takes the position that the implementation is in fact a non-trivial problem, and our major contributions are to shed light on the implementation complexity, and to propose and sketch an initial solution. As such this work complements the prior research well. To the best of our knowledge, no prior work has systematically addressed the software engineering problems in SDN application development.

Finally, we wish to note that there has been effort (in-

cluding the authors' own work) in applying software engineering techniques in the SDN context, but mainly to the testing [6, 16] and orchestration [17] of SDN apps. In contrast, this paper focuses on the implementation of those applications.

7 Conclusion

The emerging software-defined networking paradigm revolutionizes computer networking and presents new challenges to the software engineering community. The architecture enabled by SDN enforces re-examination of the many assumptions (that have been taken for granted in developing traditional networking software), when one develops SDN control software. We propose a systematic and methodical approach to SDN app development through rigorous software specification and design methodologies, that can be applied to developing new SDN apps, or migrating existing algorithms and protocols to the SDN environment. We illustrate a preliminary case study of the MAC learning algorithm that shows promises of achieving correct software by design for software-defined networking, through a sequence of refinement steps. Our future work includes further validation of our approach via more case studies of more sophisticated SDN apps to address scalability, and utilizing/augmenting existing tool support.

Acknowledgments

References

[1] Mininet: An Instant Virtual Network on your Laptop (or other PC). http://mininet.org.

[2] The OpenDaylight Project. http://opendaylight.org.

[3] OpenFlow Specifications. https://www.opennetworking.org/software-defined-standards/specifications/.

[4] Project Floodlight: Open Source Software for Building Software-Defined Networks. http://www.projectfloodlight.org/floodlight/.

[5] P. Bosshart, G. Gibb, H.-S. Kim, G. Varghese, N. McKeown, M. Izzard, F. Mujica, and M. Horowitz. Forwarding

[6] M. Canini, D. Venzano, P. Perešini, D. Kostič, and J. Rexford. A NICE way to test Openflow applications. In *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation*, pages 127–140, San Jose, CA, 2012.

[7] I. Exman. Linear software models: Standard modularity highlights residual coupling. *International Journal of Software Engineering and Knowledge Engineering*, 24(2):183–210, 2014.

[8] I. Exman. Conceptual integrity of software systems: Architecture, abstraction and algebra. In *Proceedings of the 29th International Conference on Software Engineering and Knowledge Engineering*, pages 416–421, Pittsburgh, PA, 2017.

[9] C.-Y. Hong, S. Mandal, M. Al-Fares, M. Zhu, R. Alimi, K. N. B., C. Bhagat, S. Jain, J. Kaimal, S. Liang, and et al. Before and after: Managing hierarchy, partitioning, and asymmetry for availability and scale in Google's software-defined WAN. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, pages 74–87, Budapest, Hungary, 2018.

[10] L. Lin, S. J. Prowell, and J. H. Poore. An axiom system for sequence-based specification. *Theoretical Computer Science*, 411(2):360–376, 2010.

[11] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner. OpenFlow: Enabling innovation in campus networks. *Computer Communication Review*, 38:69–74, 2008.

[12] S. J. Prowell and J. H. Poore. Foundations of sequence-based software specification. *IEEE Transactions on Software Engineering*, 29(5):417–429, 2003.

[13] S. J. Prowell, C. J. Trammell, R. C. Linger, and J. H. Poore. *Cleanroom Software Engineering: Technology and Process*. Addison-Wesley, Reading, MA, 1999.

[14] M. Rahnamay-Naeini, S. S. Baidya, E. Siavashi, and N. Ghani. A traffic and resource-aware energy-saving mechanism in software defined networks. In *International Conference on Computing, Networking and Communications*, pages 1–5, Kauai, HI, 2016.

[15] A. Santos da Silva, J. A. Wickboldt, L. Z. Granville, and A. Schaeffer-Filho. ATLANTIC: A framework for anomaly traffic detection, classification, and mitigation in SDN. In *IEEE/IFIP Network Operations and Management Symposium*, pages 27–35, Istanbul, Turkey, 2016.

[16] C. Scott, A. Wundsam, B. Raghavan, A. Panda, A. Or, J. Lai, E. Huang, Z. Liu, A. El-Hassany, S. Whitlock, and et al. Troubleshooting blackbox SDN control software with minimal causal sequences. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, pages 395–406, Chicago, IL, 2014.

[17] X. Sun and L. Lin. Leveraging rigorous software specification towards systematic detection of SDN control conflicts. In *Proceedings of the 31st International Conference on Software Engineering and Knowledge Engineering*, pages 193–258, Lisbon, Portugal, 2019.

Dynamic Architecture-Implementation Mapping for Architecture-Based Runtime Software Adaptation

Cuong Cu[1], Rachel Culver[2], and Yongjie Zheng[2]
[1]CyberSource Corporation, Austin, Texas, USA
[2]Department of Computer Science and Information Systems, California State University San Marcos, USA
csc823@gmail.com, culve005@cougars.csusm.edu, yzheng@csusm.edu

Abstract—**Architecture-based software adaptation is a promising method that adapts a software system by evolving its architectural model, which is generally easier to understand and manipulate than source code. The wide-scale practice of the method requires an approach to automatically mapping adaptive changes that are planned and deployed in the architecture to modifications of running code. This involves two main challenges: maintaining architecture-implementation conformance and dynamic software updating. Existing approaches fail to address them simultaneously to enable architecture-based adaptation. This paper presents a novel approach combining an architectural variability implementation mechanism with an architecture framework. The approach automatically updates both source code and running code during architectural evolution. As an initial assessment, we applied the approach to the adaptation of a chat application.**

Keywords—software architecture, architecture-implementation conformance, software evolution

I. INTRODUCTION

A self-adaptive software [15] modifies its own behavior in response to changes in its operating environment, such as end-user input, external hardware devices and sensors, or program instrumentation. *Architecture-based adaptation* [5, 12, 17] is an important result from software architecture research. A software system's architecture is the set of principal design decisions made about the system [19]. It is commonly modeled as a configuration of components connected via interfaces, using an architecture description language (ADL). Architectural models do not contain implementation details and generally are easier to understand and manipulate than source code.

Figure 1 shows an existing infrastructure of architecture-based software adaptation that this research aims to support. It separates adaptation activities into two simultaneous processes: *adaptation management* (the upper half) and *evolution management* (the lower half). Adaptation management monitors and evaluates the application and its operating environment, plans adaptation, and deploys change descriptions in architectural terms (e.g., replacing a component) to the running application. Evolution management is responsible for evolving the application by mapping the deployed architectural changes to modifications of the application's running implementation, while ensuring runtime conformance between the architecture and implementation. This is our focus in this project, and it primarily involves the following two challenges.
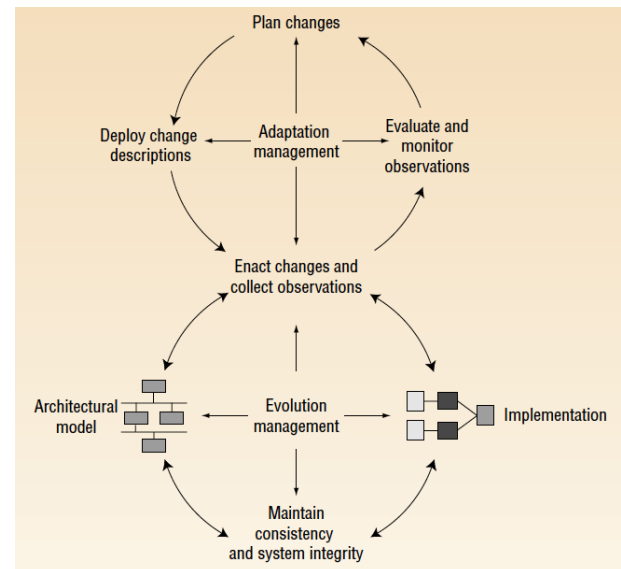
Fig. 1. An infrastructure of architecture-based runtime adaptation [17].

- *Mapping changes in the architecture to automatic updating of source code*. Software architecture may be frequently changed during architecture-based adaptation: a component may be replaced by another component, and a new interface may be added to a component. These changes affect the code in various ways and at different degrees of granularity [4]. Existing architecture-centric approaches often solely rely on code generation [6] to automatically update the code. This is not sufficient because the manually developed code (i.e., user-defined code) also exists and is often mixed with generated code. It is difficult under this circumstance either to protect user-defined code from being overwritten or to update user-defined code accordingly.

- *Dynamically modifying running code after source code is updated*. This is essentially a problem of dynamic software updating [13] that involves several issues, such as swapping code and transferring application state (e.g., the current values of program variables) to new code. None of the existing architecture-implementation mapping approaches [11, 14, 16, 22] addresses these issues. Existing dynamic software updating techniques [1, 10, 13] are mainly for

language-level program adaptation. They typically require extension of existing programming languages or the use of a third-party middleware. These requirements may add to the complexity of implementing architectures and thus are not appropriate for architecture-based runtime adaptation.

In this paper, we present an architecture-implementation mapping approach to supporting architecture-based runtime adaptation shown in Figure 1. An important insight that we have in this research is that the architectural elements that are planned/anticipated to be changed (i.e., *variable*) should be implemented differently from the stable or core elements. Their implementations should be either loosely bound to the rest of the system (e.g., in a separate module) or can be easily identified and updated (e.g., via code annotations). This opens up the opportunity of automatic updating of source code (including user-defined code) when the variable architectural elements are changed as planned during architectural adaptation.

The first contribution of our approach is a novel source code model that implements different kinds of variable architectural elements in specific ways. It extends an existing implementation model that we developed for product line architecture [21, 22], which also involves architectural variability. The original model decouples the generated code and user-defined code of each architecture component into independent code modules (e.g., classes). Our approach further divides the user-defined code into modules that implement the component's main logic and variable interfaces respectively. Moreover, our approach uses an annotative technique in the user-defined code to indicate code fragments (e.g., a single line of code) corresponding to variable architectural elements. When the architecture is changed, our approach automatically updates both generated code (via code regeneration) and user-defined code (via annotation processing) to maintain architecture-implementation conformance.

Additionally, our approach includes a novel software framework named *DynaMyx* that automatically updates running code of a component when its source code is changed (e.g., as a result of mapping architectural changes to source code described above). DynaMyx extends an existing architecture framework, *Myx* [9], which provides built-in implementations (e.g., APIs, abstract classes) for implementing architectures. On top of that, DynaMyx includes modules that encapsulate the logic (e.g., transfer state, swap code) of dynamic software updating from the overlying application. This allows the developer to focus on application-specific logic, while the DynaMyx framework automatically monitors source code, detects its changes, reloads the changed code, and migrates the runtime state to the new code without stopping the running code.

We implemented a prototype of the approach in ArchStudio [2], an Eclipse-based architecture development platform. As an initial assessment, we applied the approach to the adaptation of a chat application that has an explicit architectural model. We changed its architecture while the application was running, and observed that the running code was dynamically updated with our approach. After that, we inspected both the source code and the application's behavior (e.g., functions, runtime data) to assess whether the application functions appropriately and whether its architecture and running code are consistent. We created a video demo [7] to illustrate this process.

II. APPROACH

Figure 2 provides an overview of our approach. The rectangle at the top represents different kinds of variable architectural elements that the approach supports, including replacement of a component, replacement of an interface, and addition of an interface. We use an existing architectural modeling approach and tool called *ArchFeature* [3] that we developed in a prior project. ArchFeature supports modeling and evolution of architectural variations using an existing XML-based ADL, xADL [8]. Our focus in this project is on mapping of architectural changes to both source code and running code. The gray boxes in Figure 2 represent two main contributions of the approach: (1) a source code model (supported by a code generator and an annotation processor) that regulates the implementation of an architecture component to enable automatic modifications of source code; (2) the DynaMyx framework that extends the Myx framework as mentioned in Section I with the capability of automatic updating of running code. Each is introduced in the following subsections.

A. Architectural Variability Implementation

Our approach includes a novel source code model combining code generation, code separation, and an annotative technique in the implementation of an architecture component. As shown in Figure 2, the model divides a component's source code into the following three independent modules and uses a program composition mechanism (e.g., method delegation) to integrate the separated code. This enables a separation of decision space within the implementation of each component and offers a novel way to implement different variations in the architecture.

Generated Code: a module that is generated from the component's architectural specification. It contains routine implementation of the externally visible information (e.g.,
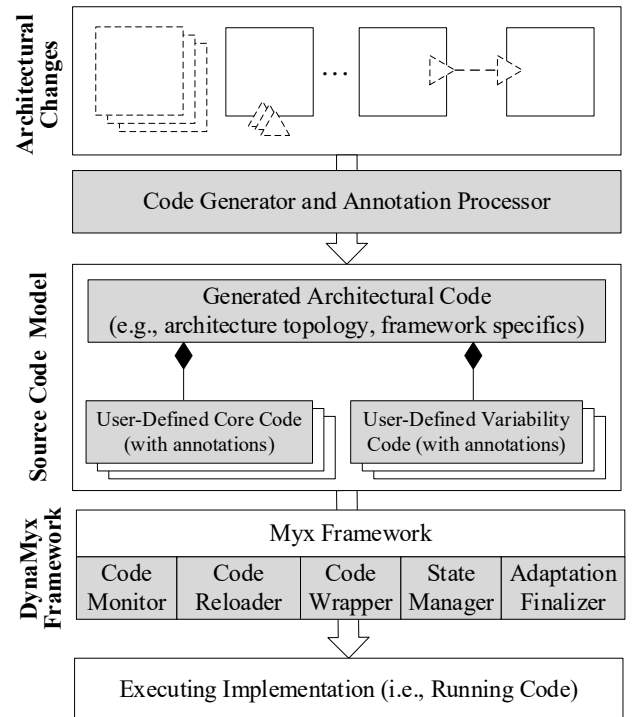
Fig. 2. Approach overview.

interfaces) of the component. The generated code encapsulates knowledge about architecture topology and related variations (e.g., implementation of an optional interface). It does not need or allow manual modification, and implements the application-specific methods (e.g., methods defined in the component's interfaces) by redirecting request to a separate module (i.e., user-defined code modules explained below), where the methods are manually implemented.

User-Defined Core Code: a module that contains manually developed implementation details of the component's main logic. It implements a program interface including the methods that generated code needs the programmer to develop. The user-defined core code represents the internal implementation of the component and encapsulates implementation-specific concerns (e.g., use of code libraries and algorithms). It addresses related architectural variations (e.g., replacement of a component for a different implementation) by switching between alternative user-defined modules, which are represented by the overlapping boxes in Figure 2.

User-Defined Variability Code: a manual module that is separated from the user-defined core code above. It contains implementation details of a construct (e.g., an optional interface) that can vary independently of the component. This reduces the impact of the variation (e.g., inclusion/exclusion of the construct) on the rest of the component's user-defined code. Similar to the core module, the variability module implements a program interface, which only includes methods specific to the construct. A library of variability modules containing different implementation mechanisms may also exist.

Our approach also includes an *architecture-based code annotation* technique that is used in the user-defined code modules described above to indicate optional *fine-grained* code fragments (e.g., a method, a line of code), which may be added or removed corresponding to the adaptive changes made to the architecture. An annotation is defined as a Java annotation (i.e., @Optional) wrapped by a block comment (i.e., /*…*/) as shown below. It contains the name(s) of the feature(s) that the annotated code is related to. Each feature is represented as a predefined value of a Java enum named *Feature* (i.e., *Feature.{feature-name}*). In particular, the *Feature* enum is generated from the architecture (hence architecture-based) and includes the names of all the features that are related to the corresponding component in the architecture. Only the included names can appear in an annotation used in the component's code. In this way, the programmer does not need to manually type in a feature name. When the architecture is changed, the related code fragments and annotations are automatically updated by the annotation processor shown in Figure 2.

/*@Optional(Feature.{feature-name}, …)*/

Figure 3a shows an architecture example of a text-based chat application. The architecture has four components (i.e., rectangles) that are connected via interfaces (i.e., triangles). All the elements drawn using dashed lines are variable for three features: sending system messages (e.g., a smiley face), saving chat history using different mechanisms (e.g., file system, database), and sharing files. Figure 3a also shows an example of variability specification in the xADL language. It defines an optional interface of Component *Server* for *FileSharing*.

```
01 class ServerArch extends MyxComponent{
02   IServer _imp = getCoreImp();
03   IFileSharing fileImp = getFileImp();
04   IFwdMsg out1;
05   IChatHisotry out2;
06   public void init(){
07     out1= MyxUtils.getService("FwdMsg");
08     out2= MyxUtils.getService("ChatHistory");
09   }
10   public void destroy(){}
11   … //other lifecycle methods
12   public void sendMsg(String msg){
13     _imp.sendMsg(msg);
14   }
15   public void uploadFile(File f) {
16     fileImp.uploadFile(f);
17   }
18 }
```



```
01 <interface id="i03">
02   <optional>
03     <feature>
04       FileSharing
05     </feature>
06   </optional>
07 </interface>
```
(a)

```
01 interface IServer {
02   public void sendMsg(String msg);
03 }
```

```
01 interface IFileSharing {
02   public void uploadFile(File f);
03 }
```

```
01 class ServerImp implements IServer{
02   ServerArch _arch;
03   public void sendMsg(String msg){
04     _arch.out1.fwdMsg(msg);
05     /*@Optional(Feature.ChatHistory)*/
06     _arch.out2.saveMsg(msg);
07   }
08 }
```

```
01 /*@Optional(Feature.FileSharing)*/
02 class FSImp implements IFileSharing{
03   ServerArch _arch;
04   public void uploadFile(File f){...}
05 }
```
(b)

Fig. 3. (a) Architecture example of a chat application; (b) Code example of Component *Server*.

Figure 3b shows the code example of Component *Server* implemented using our approach. The generated code (Class *ServerArch*) includes references to user-defined modules (Lines 2-3), references to connected components (Lines 4-5), lifecycle methods and APIs required by the DynaMyx framework (introduced in the following subsection), and application-specific methods (Lines 12-17) that are implemented by calling the corresponding user-defined module. The user-defined core code (Class *ServerImp*) and variability code (Class *FSImp*) each implements a program interface (*IServer* and *IFileSharing*) that is also generated and includes the methods to be manually developed. The user-defined code modules contain architecture-based annotations that are attached to optional code fragments (Line 6 of Class *ServerImp*). Note that the user-defined code may call the methods of other components via its generated code (e.g., Lines 4 and 6 of Class *ServerImp*).

B. DynaMyx Framework

Myx is an existing architecture framework written in the Java programming language. It includes a set of modules as built-in implementations of key architectural elements, which are used to develop an architecture-based application. Myx also encapsulates the logic for bootstrapping the application and regulating interactions (e.g., method calls) between components. This allows application developers to focus on developing application-specific logic. DynaMyx inherits these capabilities (i.e., supporting architecture implementation) from Myx. In particular, DynaMyx maintains Myx's interface (e.g., APIs and lifecycle methods that are underlined in Figure 3b) to the overlying application. All the applications originally built on Myx can still be correctly executed with DynaMyx. This is reflected in Figure 2 as all the DynaMyx modules are underneath Myx and are invisible to the application code above.

DynaMyx extends Myx with the capability of dynamic software updating and hides the related complexity from implementing architectures. This represents a novel and promising approach to supporting architecture-based runtime adaptation. DynaMyx includes five new modules: *Code Monitor*, *Code Reloader*, *Code Wrapper*, *State Manager*, and *Adaptation Finalizer* as shown in Figure 2. It automatically detects source file changes, complies the program from the changed source files, starts it up alongside the old program, transmits its state to the new program, and finally swaps the initialized new program with the old program. The entire process consists of the following five steps.

Step 1 – *Detect code changes*. DynaMyx works with an program development tool (e.g., Eclipse) that automatically complies source code when it is changed. The Code Monitor module of DynaMyx monitors the modified dates of every component's compiled code files. If a change is detected in a component, Code Monitor automatically triggers the steps below to update the component's running code. Other components are not affected during this process.

Step 2 – *Reload changed code into the running system*. The Code Reloader module includes a dedicated code loader, which enforces reloading of a modified code file into the system (e.g., Java Virtual Machine). The code reloading will occur if the component is inactive and is not communicating with other components. This is determined based on Myx's capability of managing component communications as mentioned earlier.

Step 3 – *Create new instances from reloaded code*. The new code instances are not bound to the system at this point.

Step 4 – *Transfer application state to new code instances and initialize them afterwards*. State Manager processes user-defined code to transfer state for the updated components. During this process, it bypasses security scope, especially the private and protected scopes, to access and copy state from old stances to new instances. In particular, State Manager is able to transfer state in a class hierarchy and support properties that are inherited and defined in a parent class. After that, the Manager will initialize new instances with the transferred state by calling the component's Myx lifecycle methods (e.g., *init* underlined in Figure 3b).

Step 5 – *Swap (i.e., bind) new instances into the running system and discard old instances*. A challenge involved at this point is updating the references of other components to old code instances, which will be swapped out of the memory. It is difficult to detect all the related references as this is essentially a problem of dynamic code analysis. DynaMyx addresses the challenge by wrapping the implementation of each component with Code Wrapper (implemented based on Java Proxy). The Wrapper, instead of the component's code, is referenced by the code of other connected components. The Wrapper serves as a delegate that intercepts and redirects the function calls from connected components. At the end of the adaptation, the Adaptation Finalizer module updates the Wrapper to refer to new instances of the component's code and bind new instances into the system.

Table 1. Mapping Runtime Architectural Changes to both Source Code and Running Code.

Runtime architectural changes	Mapping to source code	Mapping to running code
Component addition/removal	Regenerate code to include/exclude the component's architectural code.	Load/unload code and create/destroy instance.
Component replacement	Regenerate code to switch to a different user-defined core module.	Reload new code; create new instance; transfer state; swap code.
Provided interface addition/removal	Regenerate code to include/exclude the interface's architectural code.	Load/unload code and create/destroy instance.
Provided interface replacement	Regenerate code to switch to a different user-defined variability module.	Reload new code; create new instance; transfer state; swap code.
Required interface addition/removal	Regenerate code; process annotations and code fragments in user-defined modules.	Reload new code; create new instance; transfer state; swap code.
Connection addition/removal	Regenerate the bootstrapper program.	Reload the bootstrapper program.

C. Mapping Architectural Changes to Code

Table 1 summarizes our approach's capabilities of mapping typical kinds of runtime architectural changes to both source code and running code based on the implementation model and DynaMyx framework presented in this section. It distinguishes a *provided* (input) component interface from a *required* (output) interface. A provided interface contains the methods implemented within the component, while a required interface contains the methods that are implemented by another component and used by the current component. The connection changes are handled by Myx as mentioned in Section II.B.

III. PRELIMINARY EXPERIENCE

We developed a prototype of the approach in the ArchStudio open-source system as mentioned in Section I. The prototype includes a code generator, an annotation processor, and the DynaMyx framework. The code generator is built using the Eclipse JET code generation engine [6] that follows a template-based code generation paradigm. The code generation templates capture routine implementations of software architecture. The annotation processor is built using the ANTLR parser generator [20]. It automatically identifies and updates code annotations and code fragments corresponding to an architectural change. We integrated these tools with the ArchFeature architectural modeling tool mentioned in Section II. This provides us with a platform where we can assess our approach.

We will consider the architecture-implementation mapping approach presented in this paper successful if (1) it maintains conformance between a software's architecture, its source code, and its running code when the planned architecture changes are deployed; (2) the dynamically-updated running software behaves appropriately (with the updated behavior) and continuously (with the transferred state); (3) the overhead in terms of executing time and memory requirement during adaptation is acceptable. To validate our approach along all these three dimensions, we applied the approach and tools to a chat application.

The chat application was implemented by two Masters students based on the approach presented in this paper. It has a list of features (e.g., *File Sharing*, *Game*, and *Template*) and an explicit architectural model developed using the ArchFeature tool. It has around 15K SLOC, including generated code, user-defined code, and code annotations. The architecture and code are consistent with each other. This was validated using a consistency checking tool of ArchStudio. We assessed the approach by executing the chat application and exercising some of its functions (e.g., chat) to generate runtime state (e.g., chat messages). We then used the ArchFeature tool to change its architecture while the application was running. We exercised different kinds of architectural changes as discussed in the paper, such as component removal and interface addition, which were eventually reflected into the running code by our approach. In the end, we checked architecture-implementation conformance using the tool mentioned above. We also inspected the behavior and application data of the updated chat application to validate whether it still functions correctly.

Figure 4 shows screenshots of the chat application's architectural model (opened in our ArchFeature modeling tool) and user interface (i.e., a chat client window). In one of the experiments, we removed the *Template* feature (selected in the feature list) and its related architectural elements while the application was running. Our approach was then triggered to automatically update the application's source code (via code regeneration and annotation processing) and running code (with DynaMyx) without terminating its execution. When the adaptation was completed, we noticed that a related user interface element (e.g., the button circled in the figure) disappeared since the corresponding code was dynamically removed by our approach. Meanwhile, the application state (e.g., chat messages) was successfully preserved. We created a video demo [7] to illustrate the process described above.

Overall, our approach was able to automatically update both the source code and running code of the chat application when its architecture was changed at runtime. We validated
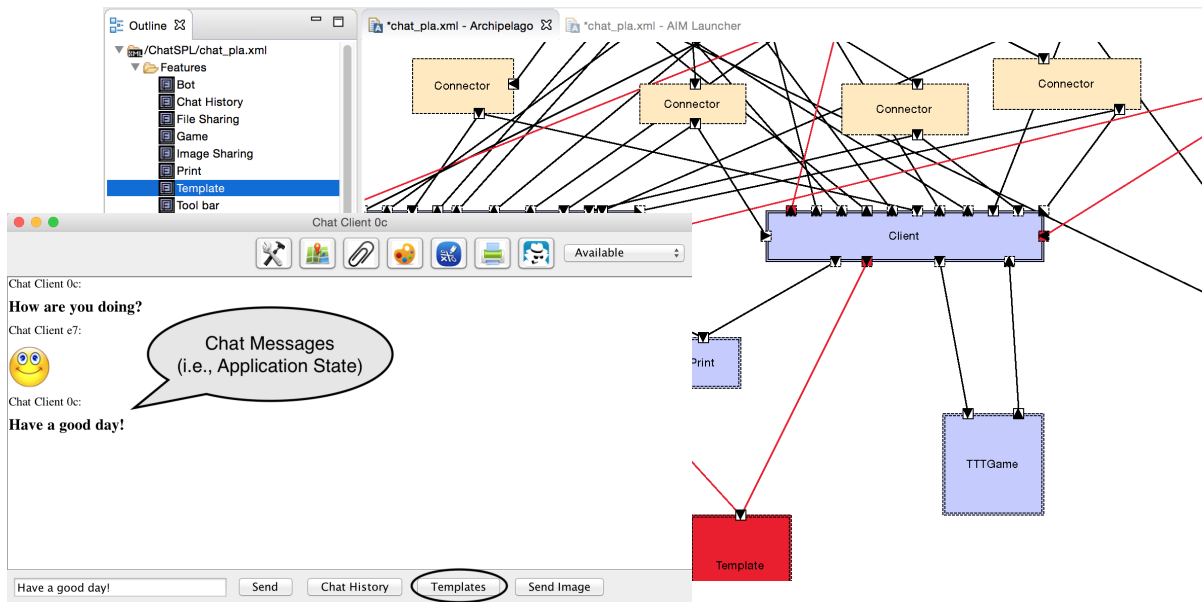
Fig. 4. Architecture-based runtime evolution of a chat application.

conformance between the updated architecture and implementation after each evolutionary operation. The system's new behavior also matched the corresponding architectural changes, and we did not notice any performance degradation during the adaptations. A limitation of DynaMyx that we found is that the new state must be determined and transferred from the existing state. Our approach does not address inferring new state information, which usually requires manual intervention (due to lack of information). For example, it cannot automatically transfer state to a new field added in the new code since this information does not exist in the old code.

IV. RELATED WORK

Several architecture-implementation mapping approaches exist, including programming language design [11], code generation [6], and architecture frameworks [9, 16]. These approaches successfully address the challenge of bridging the abstraction gap between architecture constructs and program elements during the initial development of a software system. They can maintain conformance between the architecture and source code along certain criteria, such as *style conformance* [16], *communication integrity* [11], or *quality concerns* [14]. However, none of them addresses *runtime conformance* between the architecture and running code in architecture-based self-adaptation. Existing architecture frameworks, such as C2 [16] and Myx [9], provide fairly well understood source code that assists developers in implementing systems conforming to an architecture style. They do not support the mapping of architecture changes to code and require an additional mapping approach (e.g., the presented work) to maintain architecture-implementation conformance.

Existing architecture-based runtime adaptation approaches address some important issues in this area, such as adaptation infrastructure [5, 12, 17] and architecture styles [18]. These approaches reveal the benefits of a self-managed software architecture. In terms of mapping architectural changes to running code, they mainly rely on existing architecture-implementation mapping approaches, such as architecture frameworks and code generation, which are not sufficient as described above. For example, existing approaches in this area cannot support architectural changes (e.g., replace an interface) involving the challenges of automatically updating user-defined code and dynamic software updating.

V. CONCLUSION

This paper presents an approach that maintains runtime conformance between the architecture and running system. This is essential to architecture-based runtime adaptation, but fails to be addressed by the existing approaches of dynamic software updating and architecture-implementation mapping. The approach has two main contributions: (1) a variability-specific architecture implementation approach that enables automatic modifications of source code (e.g., user-defined code) during architectural adaptation, and (2) an architecture framework that encapsulates dynamic software updating mechanisms and enables automatic modifications of running code. The initial assessment reveals that our approach is capable of supporting architecture-based runtime software adaptation. We intend to further evaluate the approach through a long-term study with a large software system in the future.

REFERENCES

[1] A. Orso, A. Rao and M. J. Harrold, "A technique for dynamic updating of Java software," International Conference on Software Maintenance, 2002. Proceedings., Montreal, Quebec, Canada, 2002, pp. 649-658.

[2] Archstudio. An Architecture-based Development Environment. http://www.isr.uci.edu/projects/archstudio/, Institute for Software Research, University of California, Irvine.

[3] C. Cu, X. Ye, and Y. Zheng. "XLineMapper: a product line feature-architecture-implementation mapping toolset". In Proceedings of the 41st International Conference on Software Engineering: Companion Proceedings (ICSE 2019). IEEE Press, 87–90. 2019.

[4] D. Garlan, R. Allen and J. Ockerbloom, "Architectural mismatch: why reuse is so hard," in IEEE Software, vol. 12, no. 6, pp. 17-26, Nov. 1995.

[5] D. Garlan, S. Cheng, A. Huang, B. Schmerl and P. Steenkiste, "Rainbow: architecture-based self-adaptation with reusable infrastructure," in Computer, vol. 37, no. 10, pp. 46-54, Oct. 2004.

[6] D. Steinberg, F. Budinsky, M. Paternostro, and E. Merks, EMF: Eclipse Modeling Framework (2nd Edition). Addison-Wesley Professional, 2008.

[7] DynaMyx. https://youtu.be/2zCHz6jovX4

[8] E.M. Dashofy, A. van der Hoek, and R.N. Taylor, "A Comprehensive Approach for the Development of Modular Software Architecture Description Languages," ACM Transactions on Software Engineering and Methodology (TOSEM). 14(2), p. 199-245, April, 2005.

[9] E.M. Dashofy, Myx and myx.fw. http://www.isr.uci.edu/projects/archstudio/myx.html.

[10] H. Seifzadeh, H. Abolhassani, and M.S. Moshkenani, "A survey of dynamic software updating," Journal of Software: Evolution and Process, 25(5), 535-568, 2013.

[11] J. Aldrich, C. Chambers and D. Notkin, "ArchJava: connecting software architecture to implementation," Proceedings of the 24th International Conference on Software Engineering. ICSE 2002, Orlando, FL, USA, 2002, pp. 187-197.

[12] J. Kramer and J. Magee, "Self-Managed Systems: an Architectural Challenge," Future of Software Engineering (FOSE '07), Minneapolis, MN, 2007, pp. 259-268.

[13] M. Hicks, and S. Nettles, "Dynamic Software Updating," ACM Transactions on Programming Languages and Systems (TOPLAS) 27(6), p. 1049-1096, 2005.

[14] M. Mirakhorli and J. Cleland-Huang, "Detecting, Tracing, and Monitoring Architectural Tactics in Code," in IEEE Transactions on Software Engineering, vol. 42, no. 3, pp. 205-220, 1 March 2016.

[15] M. Salehie and L. Tahvildari, "Self-adaptive software: Landscape and research challenges," ACM Trans. Auton. Adapt. Syst. 4, 2, Article 14, 42 pages, May 2009.

[16] N. Medvidovic, N.R. Mehta, and M. Mikic-Rakic, "A Family of Software Architecture Implementation Frameworks," In Proceedings of the 3rd IFIP Working International Conference on Software Architectures. Montreal, Canada, August, 2002.

[17] P. Oreizy et al., "An architecture-based approach to self-adaptive software," in IEEE Intelligent Systems and their Applications, vol. 14, no. 3, pp. 54-62, May-June 1999.

[18] R.N. Taylor, N. Medvidovic and P. Oreizy, "Architectural styles for runtime software adaptation," 2009 Joint Working IEEE/IFIP Conference on Software Architecture & European Conference on Software Architecture, Cambridge, 2009, pp. 171-180.

[19] R.N. Taylor, N. Medvidovic, and E.M. Dashofy, Software Architecture: Foundations, Theory, and Practice. 736 pgs., John Wiley & Sons, 2010.

[20] T. Parr, The ANTLR Parser Generator. http://www.antlr.org/.

[21] Y. Zheng, C. Cu and H. U. Asuncion, "Mapping Features to Source Code through Product Line Architecture: Traceability and Conformance," 2017 IEEE International Conference on Software Architecture (ICSA), Gothenburg, 2017, pp. 225-234.

[22] Y. Zheng, C. Cu, and R. N. Taylor, "Maintaining Architecture-Implementation Conformance to Support Architecture Centrality: From Single System to Product Line Development," ACM Transactions on Software Engineering and Methodology. 27, 2, Article 8, 52 pages, June 2018.

An Automated Goal Labeling Method
Based on User Reviews

Shuaicai Ren* Hiroyuki Nakagawa* Tatsuhiro Tsuchiya*
*Graduate School of Information Science and Technology
Osaka University, Suita, Japan
Email: {s-ren, nakagawa, t-tutiya}@ist.osaka-u.ac.jp

Abstract—Requirements analysis is an important step in a software development process. Summarizing user reviews is an efficient way of requirement elicitation. Nevertheless, it is difficult to manually collect user reviews from app stores. In order to solve this problem, we proposed a method that automatically elicits requirements and establishes a goal model for visualization. To improve the previous method, this paper proposes a method of defining labels of goals. Experimental results demonstrate that those labels can help developers understand requirements more easily and precisely.

Index Terms—Requirements elicitation, goal modeling, user reviews, labeling

I. INTRODUCTION

With the expansion of the mobile phone market, various smartphone applications have been developed, such as SNS, payment applications and social games. To ensure their attraction, developers have to collect feedbacks from application users. In application stores, users can publish reviews to rate applications, ask for a new function or report bug information. Developers can investigate these reviews and extract important requirements to improve applications from them. However, the number of reviews is too large to extract requirements from reviews.

In order to help developers understand user requirements, we have reported a preliminary result of review clustering and a goal model construction from user reviews [19]. The goal model is one of requirements models, which describes requirements as goals to be satisfied. In our previous paper, the construction method structurally visualizes requirements extracted from user reviews in a goal model. Nevertheless, the tool has a problem: it is too difficult to understand what these goals represent. To deal with this problem, we propose a new labeling method in this paper.

The contribution of this paper is **goal labels automatically from user reviews**. We use two methods to give weights to user reviews and select useful sentences from reviews as labels. Two experiments demonstrate the difference between new labels and old ones. According to the results of the two experiments, we report the evaluation of our method.

The structure of this paper is as follows. Section II explains the background of our research; Section III describes the previous construction method in detail; Section IV presents the new labeling method; Section V evaluates the labeling method

and reports the challenges to improve the method; Section VI covers the conclusion and future work.

II. RELATED WORK

Requirements analysis has always been an important step in the software development life cycle. This step analyzes what kinds of requirements need to be satisfied by the system. After analyzing the requirements, developers can make documents which describe the system capability. For better requirements analysis, many methods have been proposed. Palmieri et al. [17] proposed a tool-supported method that integrates a goal-oriented requirement language and feature modeling to handle regulatory goal model families. Bettenburg et al. [2] reported a study about what kinds of information from users are required by developers. That method helps users to improve the quality of bug reports. Higashi et al. [7] provided a method of improving the accuracy of LDA review classification. Chen et al. [4] developed a tool called AR-miner, which can help developers to filter some useless reviews and classify useful ones. As a result, this tool shows the groups of the most "informative" reviews via an intuitive visualization method. Maalej et al. [11] collected massive reviews from Google Store and App Store, and tried to classify them. They compared several types of technologies for language processing and some machine learning methods. In those machine learning methods, Naive Bayes classifier had the best result. Unlike these papers we focus on how to represent user requirements via a goal model. Our work includes eliciting useful sentences through user reviews and labeling goals from the goal model.

Goal models include KAOS [6], i* [23], NFR [14], AGORA [9] and Tropos [21]. In goal models, requirements are described as goals that demand to be achieved. Goals are linked based on the relationship between them. Abstract goals become the parent goals and detailed goals will become the subgoals. For example, the goal "file operations provided" could be a parent goal of the goal "files edit function provided."

To high quality goal models, the requirements from users are necessary.

Users submit requirements or bugs by writing reviews in app stores. Some of these reviews are valuable for developers[16]. Nevertheless, there are also meaningless and low-quality reviews. Due to large numbers of reviews, it is hard to find useful reviews manually[8]. We have reported a preliminary

result of finding useful reviews and building goal models. This preliminary method still has disadvantages. Normally goals have manual labels that describe requirements. The preliminary method labels each goal with a word that is automatically extracted from reviews. The word that seems to be the most relevant to the goal is chosen as a label. Unfortunately those labels may not clearly describe requirements. To make labels more easy to understand, we propose a new method for labeling goals in this paper.

III. GOAL MODEL CONSTRUCTION METHOD

In this section, we will briefly introduce the goal model construction method that we have previously proposed in [19]. This modeling method is mainly composed of two steps: clustering and goal labeling.

Step 1: Clustering. Algorithm 1 shows the process of clustering. First, user reviews are broken down into text. Second, we delete stopwords which appear frequently in reviews but do not have meaning, such as "is, are, a, an, the". Deleting those stopwords is a common approach to process natural language. This approach can help us to find words that show users' requirements more easily. In this method, we added some other words like "ur" (your) and "dis" (this) into the stopwords from NLTK[3]. These words come from users' oral habit and have no relationship with users' requirements.

Next, filtered words are added to a dictionary, and then lists representing bag-of-words (BoW) [12] are generated from the dictionary. BoW represents words in a document and the number of their occurrences. Then, the generated lists of BoW are stored in a matrix as vectors. For example, suppose a document contains following two sentences:

- I cannot open any of my company documents now from the app.
- It won't let me open any preexisting documents.

Lists of BoW after lemmatization and filtering stopwords are ["open": 1, "company": 1, "document" : 1, "app" : 1] and ["open": 1, "preexist": 1, "document": 1]. Finally, these lists are stored in a matrix. After data preparation, Ward's method [22], a hierarchical clustering method, is applied to the matrix and the result of review clustering is obtained. In Ward's method, the minimum variance criterion is used to couple clusters:

$$d_{ij} = d(\{X_i\}, \{X_j\}) = \|X_i - X_j\|^2 \qquad (1)$$

Since two BoWs have the same words "open" and "document" in the above example, the distance between two sentences becomes close by clustering.

Step 2: Labeling. We have reported a preliminary method to label goals. Algorithm 2 shows the process of goal labeling. In this step, we used document frequency to define goal labels. For leaf goals, the goals that do not have subgoals, their labels are clustered words' weight. The weight comes from the document frequency, which means that if a word frequently appears in this cluster, it probably appears in the goal label. For parent goals, their labels come from the words that appear frequently in every subgoal. Through this way we can ensure

---

Algorithm 1 Clustering

1: **Input:** titles and texts of user reviews
2: $array$ = array of $number\ of\ reviews \times size\ of\ vocabulary$
3: **for** $review$ **in** $reviews$ **do**
4:    $wordlist \leftarrow$ lemmatized words not in stopwords
5:    $bows \leftarrow$ bag-of-words (BoW) of $wordlist$ /* generate BoW */
6:    **for** {$word\_id$, $frequency$} **in** $bows$ **do**
7:       $array[\# review][word\_id] \leftarrow frequency$
8: apply Ward method to $array$
9: **Output:** a clustering result

Algorithm 2 Goal labeling

1: **Input:** a coupled cluster
2: $cluster$ = cluster of reviews
3: $wordlist$ = map of {$review[word]$, DF} /* DF: document frequency */
4: **for** $review$ **in** $cluster$ **do**
5:    **for** $word$ **in** $review$ **do**
6:       /* $word$ does not occur in $review$ yet */
7:       **if** $word$ is not in stopword and $word$ is not counted **then**
8:          $wordlist[word]$ += 1
9: exclude words that occur in two sibling goals
10: sort $wordlist$ by DF of $word$
11: select top words from sorted $wordlist$
12: **Output:** a goal description

---

that parent goals are labeled with words which are from each subgoal.

The previous method applies a clustering method for grouping reviews and constructing a hierarchical structure. However, the goal labels obtained by the previous labeling method are still hard to understand. For example, it is hard to know the meaning of the generated label "never, device, book, version, day". In order to solve this problem, we propose a new labeling method in this paper.

IV. NEW LABELING METHOD

The new labeling method utilizes sentences as goal labels. There are two approaches for automatically generating sentences. One approach generates sentences directly from documents. However, this approach has too many limitations, which makes this approach difficult to use in practice. So this paper adopts the second approach, which selects the sentences contained in the documents as a representative. To prevent from containing too many requirements in one goal, we select only one sentence for one goal as the label.

The overview of our new construction method is illustrated in Figure 1. Since the clustering method is the same as the previous method (the green part), this section mainly introduces the labeling method (the blue part). The overview of our new labeling method is illustrated in Figure 1. The labeling method is mainly composed of two methods: *selecting by TFIDF* and *selecting by cosine similarity*. Those two have
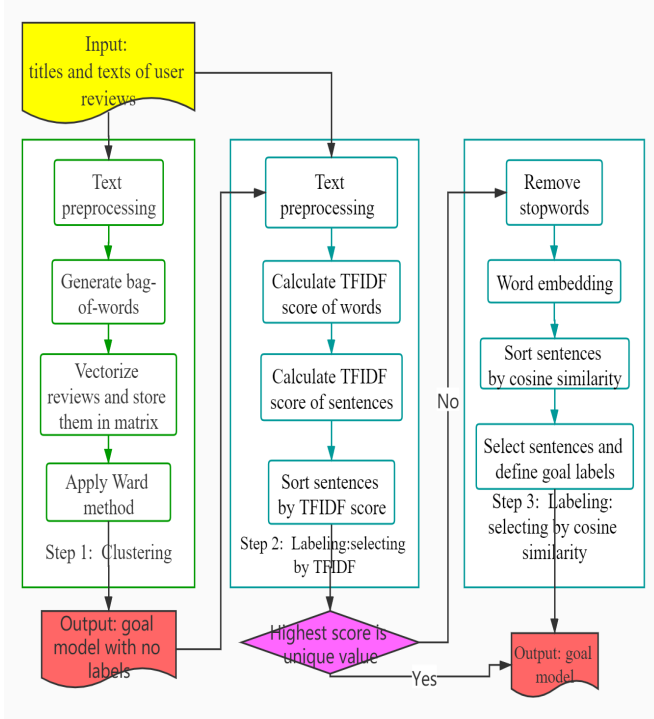
Fig. 1. Overview of the construction method. The green part shows the clustering method and the blue part illustrates the new labeling method.

Algorithm 3 Selecting by TFIDF

1: **Input:** titles and texts of user reviews
2: **for** $review$ **in** $reviews$ **do**
3:   $sentencelist \leftarrow$ divided sentences in reviews /* segment reviews with periods, exclamation marks, and question marks*/
4:   **for** $sentence$ **in** $sentencelist$ **do**
5:     **for** $words$ **in** $sentence$ **do**
6:       $TFIDFwordslist \leftarrow$ calculate TFIDF of words
7:     $TFIDFlist \leftarrow$ calculate TFIDF of sentence
8:   sort $TFIDFlist$ by $TFIDF$ of $sentencelist$
9: **Output:** sentences list

---

different pretreatment process and effects. In pursuit of better results, we decide to combine these two methods.

Step A: Selecting by TFIDF. TFIDF stands for term frequency–inverse document frequency, which is the most frequently applied weighting scheme [1] in text mining. This technique can reflect the importance of a word to a document in a corpus [20]. The importance of a word increases proportionally with the times that this word appears in the document, but it decreases inversely with the frequency that this word appears in the corpus. TFIDF is appropriate for our purpose to select keywords, as it gives a high weight to a word that only appears in one cluster.

Fig. 2 illustrates the overview of step A. Algorithm 3 shows the algorithm for selection by TFIDF. First of all, with preprocessing, these reviews are divided into sentences. The general sentences end with periods, exclamation marks and question marks. In order to prevent sentences from being too
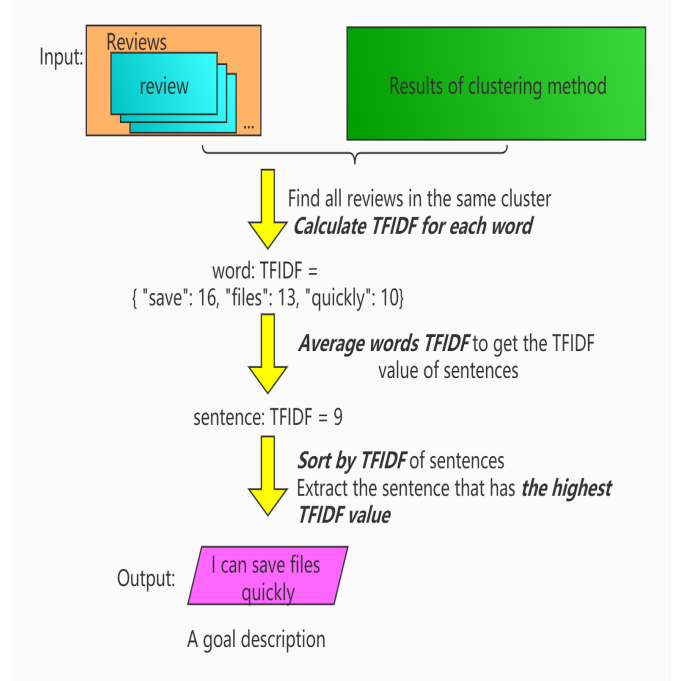
Fig. 2. Overview of labeling by TFIDF.

long and containing more than one requirement in one goal label, we segment reviews also with commas that separate sentences in a compound sentence. Next, the TFIDF score for each word is calculated, and then we average them to get the score of the sentence:

$$TFIDFs = \frac{\sum_{i=1}^{n_s} TFIDF_i}{n_s} \qquad (2)$$

n_s means the number of words of one sentence. $TFIDF_i$ means TFIDF score of the ith word order. Finally, we sort these sentences and select the sentence with the highest TFIDF score.

Step B: Selecting by cosine similarity. In the previous step, TFIDF score used to sort sentences. However, in some cases, the first two sentences have the same score. We require to find the one that is more similar to the cluster. For this we use word embedding techniques. Word embedding is a general name for a set of technologies to process natural language. One method of word embedding is dimensionality reduction in the word co-occurrence matrix [10]. According to this method, a space with many dimensions per word is embedded into a continuous vector space with a lower dimension, and each word or phrase is mapped to a vector on the real number field. Now there are some tools to achieve this method, for example, the word2vector [13] and doc2vector. After getting the word vector, we apply the cosine similarity, which is a measure of similarity between two non-zero vectors of an inner product space, to calculate the similarity between vectors, words or sentences.

First, reviews were filtered by stopwords. Then, we apply doc2vec to achieve sentence embedding. According to doc2vec, vectors of all sentences synthesize the vector of the

cluster so that the cosine similarity between sentences and the cluster can be calculated. The vector with the highest cosine similarity to the cluster will be selected.

V. Experiment and Evaluation

A. Purpose of Experiment

We focus on answering the following two research questions:

- *RQ1:* Do the labels correctly reflect the intent of the goals?
- *RQ2:* Can the labels be properly understood by the developers?

To answer the research questions, we made the goal models first. Reviews for making models were taken from Google Docs with the App Store. Due to the numerous reviews, goal models contain a large number of goals. In order to facilitate the experiment, we extracted a part of the goal model. In this experiment, we used the part corresponding to reviews about the cross-platform function of Google Docs. Three goal models' labels came from manual, the previous labeling method and the new labeling method. As we mentioned in Section IV, we segmented reviews with commas that separate sentences in a compound sentence. In this experiment, sentences that have less than five words were not treated as independent sentences in a compound sentence. The result of modeling and manual labeling is illustrated in Figure 3. Those labels are correct labels. To answer our research questions, two experiments were conducted. We explained the detail of each experiment below.

B. Experiment 1

Design of experiment 1. In the experiment 1, we enumerated the ideal labels made by hand, the labels of the new method and the labels of the previous method. We comprehend the gap between the two kinds of automatically generated labels and the ideal labels and whether automatically generated labels can be understood. This experiment used the correctness rate with ideal labels and understanding rate to evaluate both methods. The evaluation of the correctness rate and understanding rate in the table was completed by the authors.

Results of experiment 1. Table I demonstrates the difference between the previous labels and the current labels. Experiment 1 illustrates that the correctness rate of the previous method is 54%, while the understanding rate is 30%. As for the new method, the correctness rate is 77% and the understanding rate is 100%.

C. Experiment 2

Design of experiment 2. Experiment 2 aims at people other than authors. To comprehend the effect of labels on other people, we produced questionnaires for the examinees. In each questionnaire, examinees were asked about what kind of requirements were reflected. These examinees included four professionals and four non-professionals. The answering order of the questionnaires was random; some examinees were asked

to complete the previous labels questionnaires first, while others were the opposite. As for the correctness rate of labels, we also use correct labels to compare the labels obtained by two methods.

Results of experiment 2. The results of experiment 2 are illustrated in Table II. This figure indicates that new labels have a higher correctness rate in total.

D. Discussion

First, we answer *RQ1*: "Do the labels correctly reflect the intent of the goals? ". The experiment and questionnaire results demonstrate that labels generated by using this method have a higher accuracy rate. But there is one thing to be noted. For abstract goals at the top layers of the goal models, the correctness rate of new labels is lower. Normally, the close to the root goal the label is, the more unreliable the label is. We believed the reason is that we directly select the sentences in the reviews as labels. Users prefer to give feedbacks in detail, such as BUG reports and requirements, so our method could accurately capture the sentences that express these requirements as labels. In terms of sentences that can be used as abstract goals' labels, it is difficult to find them in user reviews. To build more precise labels, we could use other requirements mining methods. Conneau et al. [5] proposed several probing tasks designed to capture simple linguistic features of sentences. We can utilize different methods to deal with top goals and bottom goals. When it comes to bottom goals, we still extract sentences from user reviews as labels. In terms of top goals, we could extract keywords and logical relationships from sentences, and use word embedding and vector synthesis to construct labels.

Here, we answer *RQ2*: "Can the labels be properly understood by the developers?". Compared to previously generated labels, Table II demonstrates that examinees can better understand the labels, but it is still difficult for non-experts. The following improvements should be considered:

- **Construction method needs improvement**:
  To classify reviews in more detail, we require to improve our construction method. Our method uses the cluster distance for determining the goal refinement level. When the developers set the threshold to a small value, the size of each coupled cluster becomes small, and then the total number of goals increases in a model generated. In other words, the proportion of the number of detail goals becomes larger.
- **Logical relationships need to be refined**:
  In this method, the relationship between the two sibling goals is limited to *AND-refinement*. In other words, all sibling goals demand to be achieved. But actually, there are many logical relationships and they may be mentioned in the reviews. Extracting these words from reviews will help us improve the goal model construction method.

VI. Conclusion

In this paper, we reported the experience of an automated labeling method on the basis of user reviews. Our method
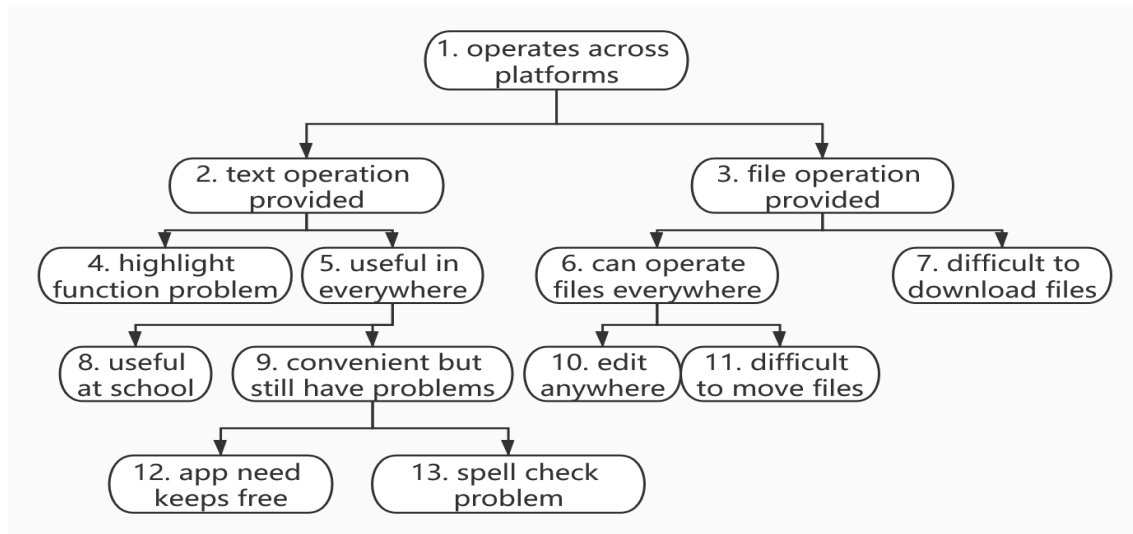
Fig. 3. Labels generated by manual.

TABLE I
LABELS EVALUATION

No.	Correct label	Label type	Label content	Evaluation
1	operates across platforms	Previous	never, device, book, version, day	-/-
		New	Also I never lose anything cause I can access it on any device	C/U
2	text operation provided	Previous	apps, note, time, user, text	-/-
		New	I just hope Google keeps these apps free	-/U
3	file operation provided	Previous	download, platform, live, fact, weird	-/-
		New	Download I hate the fact that you have to buy this	-/U
4	highlight function problem	Previous	text, highlight, browser, copy/cut, superior	-/-
		New	you can't highlight your text	C/U
5	useful in everywhere	Previous	work, school, docs, keep, save	-/-
		New	School I use this for school and it works good	-/U
6	can operate files everywhere	Previous	anywhere, allow, open, document, share	C/U
		New	Easy to access your files anywhere and share with others	C/U
7	difficult to download files	Previous	download, right, template, weird, number	C/-
		New	Templates are annoying I have to download a template	C/U
8	useful at school	Previous	school, ms, finish, right, box	C/-
		New	Use it for school I love it honestly	C/U
9	convenient but still have problems	Previous	note, time, life, feature, helpful	C/U
		New	convenient and a great time saver	C/U
10	edit anywhere	Previous	allow, open, document, share, platform	C/-
		New	edit my documents everywhere that I go	C/U
11	difficult to move files	Previous	live, updates, yesterday, move, adobe	-/-
		New	can't move over files adobe	C/U
12	app need keeps free	Previous	keep, free, save, note, time	C/U
		New	I just hope Google keeps these apps free	C/U
13	spell check problem	Previous	spell, check, belittle, -i, become	C/U
		New	Spell check I can't right-click to fix a word underlined red (spell check) using my chrome book	C/U
C: correct U: understandable -: incorrect or incomprehensible				

TABLE II
LABELS CORRECTNESS RATE

Method & examinee type	Labels correctness rate
Pervious method & non-professionals	26%
Pervious method & professionals	40%
New method & non-professionals	66%
New method & professionals	80%

weights each sentence and selects one of the sentences as a label for a goal. In the evaluation part, we conducted on experiment to evaluate the correctness of the previous method and this method. Then, we used questionnaires to investigate the developer's understanding of the goal model. As for the result of labeling, it is easier and more precise to understand than the previous method generated labels. The correctness rate of labels decreased from the bottom to the top, but the general labels can correctly reflect the intent of the goal model.

For future work, we identify the following improvement points:

- **Abstract label readability**:
  To improve the goal description readability, first, we should refine stopwords. If the number of stopwords is too large or too small, we might miss some requirements. Next, labels require to contain more useful information.

The experiment shows that this method works well with the bottom goals, but as for some top goals, it is hard to find one sentence to represent the goal. After all, it is hard to find abstract sentences from user reviews, and users prefer to report practical things. Furthermore, the AND refinements are not clear from reading the goal labels. For making more effective labels, we can make a sentence instead of selecting a sentence from reviews. Rolland et al. [18] provided a method in order to help developers understand long documents. In that method, some keywords and logical relationships are extracted and used to generate sentences with some rules. We believe that this is a feasible method to make labels. Keywords and logical relationships also can be found in user reviews. Keywords can help us to make labels while logical relationships can help us to clear refinements.

- **Other elements in the goal model need to be considered**:

  In the actual goal models, goals are not the only type of goal model elements. Functional requirements have been described as goals, while non-functional requirements can be described as soft goals. A functional requirement defines a system or its component whereas a non-functional requirement defines the performance attribute of a software system. The goal models generated by this method only have goals, which means that we cannot classify goals, soft goals and bug reports. The final objective of our research is to improve the automated goal modeling method and to visualize not only goals but also bugs and soft goals. To accomplish this objective, we should introduce a mechanism for visualizing the goal type, such as requirements or bug reports. Now we are trying to find a method to judge goal type based on word combinations. Maalej et al. [11] proposed the method to classify reviews into four types, i.e., bug report, user request, user experience, and rating. In users' views, some words combinations can help us to conduct this work. For example, the words combining "can" and "not" often in bug reports. If we can give enough weight to these word combinations, perhaps we can allow types of labels to be recognized more easily. To embed entities into the goal model and to classify goals into hard goals and soft goals, we plan to consider a goal model refinement process, such as one described in [15].

References

[1] Beel, J., Gipp, B., Langer, S., Breitinger, C.: Research-paper recommender systems : a literature survey. International Journal on Digital Libraries **17**(4), 305–338 (2016). https://doi.org/10.1007/s00799-015-0156-0

[2] Bettenburg, N., Just, S., Schröter, A., Weiss, C., Premraj, R., Zimmermann, T.: What makes a good bug report? In: Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering. pp. 308–318. ACM (2008)

[3] Bird, S., Loper, E., Klein, E.: Natural Language Processing with Python. O'Reilly Media Inc (2009)

[4] Chen, N., Lin, J., Hoi, S.C., Xiao, X., Zhang, B.: Ar-miner: mining informative reviews for developers from mobile app marketplace. In: Proceedings of the 36th International Conference on Software Engineering. pp. 767–778. ACM (2014)

[5] Conneau, A., Kruszewski, G., Lample, G., Barrault, L., Baroni, M.: What you can cram into a single vector: Probing sentence embeddings for linguistic properties. Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers) (2018). https://doi.org/10.18653/v1/p18-1198

[6] Dardenne, A., van Lamsweerde, A., Fickas, S.: Goal-directed requirements acquisition. Science of Computer Programming **20**(1-2), 3–50 (Apr 1993)

[7] Higashi, K., Nakagawa, H., Tsuchiya, T.: Improvement of user review classification using keyword expansion. In: The 30th International Conference on Software Engineering and Knowledge Engineering, Hotel Pullman, Redwood City, San Francisco Bay, California, USA, July 1-3, 2018. pp. 125–130 (2018)

[8] Hoon, L., Vasa, R., Schneider, J.G., Grundy, J., et al.: An analysis of the mobile app review landscape: trends and implications. Faculty of Information and Communication Technologies, Swinburne University of Technology, Tech. Rep (2013)

[9] Kaiya, H., Horai, H., Saeki, M.: Agora: attributed goal-oriented requirements analysis method. In: Proceedings IEEE Joint International Conference on Requirements Engineering. pp. 13–22 (2002)

[10] Lebret, R., Collobert, R.: Word embeddings through hellinger pca. Proceedings of the 14th Conference of the European Chapter of the Association for Computational Linguistics (2014). https://doi.org/10.3115/v1/e14-1051

[11] Maalej, W., Nabil, H.: Bug report, feature request, or simply praise? on automatically classifying app reviews. In: Proc. of the 23rd IEEE International Requirements Engineering Conference (RE). pp. 116–125 (Aug 2015). https://doi.org/10.1109/RE.2015.7320414

[12] Maalej, W., Kurtanović, Z., Nabil, H., Stanik, C.: On the automatic classification of app reviews. Requirements Engineering **21**(3), 311–331 (Sep 2016). https://doi.org/10.1007/s00766-016-0251-9

[13] Mikolov, T., Sutskever, I., Chen, K., Corrado, G., Dean, J.: Distributed representations of words and phrases and their compositionality. In: Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 2. pp. 3111–3119. NIPS'13, Curran Associates Inc., USA (2013)

[14] Mylopoulos, J., Chung, L., Nixon, B.: Representing and using nonfunctional requirements: a process-oriented approach. IEEE Transactions on Software Engineering **18**(6), 483–497 (June 1992)

[15] Nakagawa, H., Ohsuga, A., Honiden, S.: A goal model elaboration for localizing changes in software evolution. In: Proc. of 21st IEEE International Requirements Engineering Conference (RE'13). pp. 155 – 164. IEEE CS (2013)

[16] Pagano, D., Maalej, W.: User feedback in the appstore: An empirical study. In: 2013 21st IEEE international requirements engineering conference (RE). pp. 125–134. IEEE (2013)

[17] Palmieri, A., Collet, P., Amyot, D.: Handling regulatory goal model families as software product lines. In: International Conference on Advanced Information Systems Engineering. pp. 181–196. Springer (2015)

[18] Rolland, C., Achour, C.B.: Guiding the construction of textual use case specifications. Data & Knowledge Engineering **25**(1), 125 – 160 (1998). https://doi.org/https://doi.org/10.1016/S0169-023X(97)86223-4

[19] Shimada, H., Nakagawa, H., Tsuchiya, T.: Goal model construction based on user review classification. In: Joint Proceedings of REFSQ-2019 Workshops, Doctoral Symposium, Live Studies Track, and Poster Track co-located with the 25th International Conference on Requirements Engineering: Foundation for Software Quality (REFSQ 2019), Essen, Germany, March 18th, 2019 (2019)

[20] Tripathy, A., Agrawal, A., Rath, S.K.: Classification of sentimental reviews using machine learning techniques. Procedia Computer Science **57**, 821–829 (2015)

[21] van Lamsweerde, A.: Goal-oriented requirements engineering: a guided tour. In: Proceedings Fifth IEEE International Symposium on Requirements Engineering. pp. 249–262 (Aug 2001). https://doi.org/10.1109/ISRE.2001.948567

[22] Ward Jr., J.H.: Hierarchical grouping to optimize an objective function. Journal of the American Statistical Association **58**(301), 236–244 (1963)

[23] Yu, E.S.K.: Towards modelling and reasoning support for early-phase requirements engineering. In: Proceedings of the Third IEEE International Symposium on Requirements Engineering, 1997. pp. 226–235 (January 1997)

A Co-evolutionary Method Between Architecture and Code

Tong Wang*, Bixin Li†,Lingyuan Zhu†

*School of Computer Science and Technology, Anhui University of Technology, Maanshan, China
†School of Computer Science and Engineering, Southeast University, Nanjing, China

Abstract—Code evolution and architecture evolution are respectively related to functional requirements and non-functional requirements. According to the type of requirements, architects or developers evolve one of them. That causes the unevolved one is inconsistent with the evolved one. To solve the problem of inconsistency, we propose a co-evolutionary method to keep the consistency between architecture and code. In our method, two evolutional scenarios are considered, including co-evolving code based on evolved architecture and co-evolving architecture based on evolved code. In the former method, we first convert architecture change to code change based on mapping rules, then modify code to implement the corresponding code changes. In the latter method, we first modify the file dependency graph based on mapping rules, then recover architecture based on the modified file dependency graph. We conduct our experiments with eight open source projects, the experimental results indicate that our method can keep the consistency in the two evolutional scenarios, so that, our co-evolutionary method between architecture and code is effective.

Index Terms—Software architecture, code source, co-evolution

I. INTRODUCTION

In the software life cycle, architects and developers frequently evolve code and architecture to keep competitiveness and vitality of software [1]. Code is the actual implementation of software, so developers evolve it for meeting new functional requirements, such as new functions, high performance, and so on. Architecture is the high abstraction view of software, so architects evolve it for meeting new non-functional requirements, such as testability, maintainability [2], and so on [3].

According to the content of new requirements, developers and architects decide which one should be evolved. When developers or architects evolve code or architecture, another one is not consistent with the evolved one. The problem of inconsistent may cause the unevolved one to mislead developers and architects. To solve the problem, many co-evolutionary methods are proposed.

There are mainly four types of evolutionary methods, architecture recovery [4], code automatic generation [5], multi-view software evolution approach [6], and information fusion approach [7]. However, not all the above methods have taken all evolution scenarios into consideration, resulting in the limitation of effectiveness.

For resolving the above problems, we propose a co-evolutionary method to keep the consistency between architecture and code. The contributions of the paper are as follows:

- The method supports the two-way co-evolution, that is, it supports co-evolving code base on evolved architecture and co-evolving architecture base on evolved code.
- More types of co-evolutionary actions are taken into consideration to improve the effectiveness of co-evolution.
- We implement our method on eight open source projects, the experimental results indicate that our method can keep the consistency between code and architecture effectively.

The paper is organized as follows. Section II introduces the two-way co-evolutionary method. In Section III, we implement our method on open source projects to analyze the effectiveness of our method. Section IV introduces related work. Section V draws the conclusion and introduces the future work.

II. OUR METHOD

Code and architecture belong to different levels of granularity. According to new requirements, one of architecture and code will be evolved first. According to the type of the evolved object, our method is divided into two parts, the method of co-evolving code based on architecture and the method of co-evolving architecture based on code.

A. A Co-evolutionary Method of Code based on Evolved Architecture

Architecture is represented by CDG, and CDG consists of component nodes and dependency edges between nodes, so architecture changes are reflected by component node changes and dependency edge changes [8]. Code changes determine file changes and dependency changes, that is, code changes are reflected by FDG. So the relation between architecture and code can be converted to the relation between CDG and FDG.

To locate which parts of FDG need to be co-evolved, we propose mapping rules to convert CDG changes to FDG changes, then modifying code to implement these FDG changes.

Edge changes of CDG contain the following types: adding an edge, deleting an edge, increasing the weight of the edge, and reducing the weight of the edge.

Adding an edge indicates that some functions are invoked by another component, which can avoid the duplication of functions in multiple components. The corresponding code change is adding a dependency between files contained in the involved components.

Increasing the weight of the edge means that several functions are enhanced, and the corresponding code change is to increase the dependency intensity between the corresponding file sets.

Adding an edge is a special situation of increasing the weight of the edge, that is, the dependency intensity is increased from 0.

Deleting an edge means the relation between components is removed. The corresponding code change is deleting the corresponding dependency between corresponding sets of files.

Reducing the weight of the edge indicates that several functions should be reduced to avoid mutual influence. The corresponding code change is to find the corresponding file sets, then the dependency which has the lowest dependency intensity is deleted.

Deleting an edge is a special situation of reducing the weight of the edge, that is, the dependency intensity is reduced to 0.

According to the above analysis, edge changes are related to dependency intensity. Here, we introduce the definition of dependency point.

Dependency point: If file A has a dependency with file B, and the dependency is caused by that statement a of file A has a dependency with statement b of file B, then statement a and statement b are the dependency points.

We use slice technology detect the direct and indirect effects caused by dependency points. Program slicing is an important technology to analyze programs [9]. The core concern is the slice criterion which is usual a tuple $<s,v>$, herein, *s* means the location and *v* means a variable or a set of variables that is defined or used in *s*. The slice of a program about a criterion is the set of statements that potentially have influenced on *v* of *s* or be affected by *v* of *s* [10]. The dependency point as the slice criterion, then we use slice technology to extract the related statements.

Node changes of CDG contain the following types: adding a component-node, deleting a component node, merging component node, and splitting a component node.

Adding a component node indicates that the software needs to be added new functions where the corresponding code change is adding highly cohesive files. These highly cohesive files refer to increase the set of relevant files rather than the isolated and fragmented files. A new component indicates a new function, however, we cannot get the detail of code based on CDG, so it is implemented by adding a set of files, then the code is developed based on the detail of functions.

Deleting a component node indicates that a function needs to be deleted where the corresponding code change is deleting all files contained in the involved component.

Merging component nodes represent integrating related functions of the software, and the corresponding code change is increasing the dependency intensity between the corresponding sets of files. The higher dependency intensity indicates that the two sets of files can be clustered into a new component.

Splitting a component node indicates that the functions of the component need to be refined. The corresponding code change is splitting the set of files contained in the involved component into two sets.

This operation contains two steps, splitting a set of files into two sets of files, then reducing the dependency intensity between the two sets of files.

We split the set of files based on the loss function. The loss function is a function that maps random events or their related random variables to non-negative real numbers to represent the "risk" or "loss" of the random event. The coupling and cohesion are the important metrics for software, so we define the loss function based on cohesion and coupling as shown in Formula 1.

$$L(\theta, f) = \alpha * D(f) + \beta * S(f) \tag{1}$$

In Formula 1, $L(\theta, f)$ is the loss function, θ represents architecture change, f indicates a splitting scheme, $D(f)$ shows the impact of the splitting scheme on cohesion, $S(f)$ remarks the influence of the splitting scheme on the coupling, and α and β are the weights of cohesion and coupling.

Formula 1 is refined to Formula 2.

$$L(\theta, f) = \alpha * \sum_{p \in N} D_p(f_p) + \beta * \sum_{p,q \in N} S(f)(f_p \neq f_q) \tag{2}$$

In Formula 2, N represents the files contained in the split component. f indicates a splitting scheme for these files N. p and q are two files of N, f_p represents the set to which the split file p belongs, and $D_p(f_p)$ shows the loss resulting from splitting p by f, which is inversely proportional to the cohesion of the divided set of files, and $S(f_p \neq f_p)$ remarks the loss resulting from the different belonging of p and q, which is proportional to the coupling of the new sets after division.

The weight of the dependency edge is proportional to the dependency intensity. When the dependency intensity is low, the weight is low, that is, the related two files can be classified into different sets to obtain the minimal overall loss. The algorithm is shown in Algorithm 1.

B. A Co-evolutionary Method of Architecture based on E-volved Code

Architecture is a high abstraction view of FDG, and FDG reflects the attributes of files and the dependencies between files. So, the impact of evolved code on architecture depends on whether the evolved code changes the files or the dependencies between files. When the evolved code does not change files, architecture does not need to be evolved, otherwise, architecture needs to be modified based on code change. In this paper, we only need to consider the latter situation.

The method consists of three steps: (1) Obtaining code change by using a change detection method; (2) Modifying FDG based on code change; (3) Recovering architecture based on the modified file dependency graph.

We propose mapping rules from code change to FDG, then according to code change, we modify FDG based on mapping rules. Code changes can be divided into two types: edge changes and node changes. The following types of changes

Algorithm 1 The algorithm of calculating the approximate optimal solution of loss function

Input:
    let CDG be the component dependency graph
    let FDG be the file dependency graph
    let req be the code change
Output:
    let $result$ be the divided set of files
1: **Function** $partition(CDG, FDG, req)$
2: $node \leftarrow findNode(CDG, req.comp)$
3: $k \leftarrow req.comp$
4: /*Construct the local dependency graph named ldg formed by the files contained in the split components */
5: $ldg \leftarrow findLocalDG(FDG, node.files)$
6: /* Combine the outgoing and incoming edges of nodes in the graph named ldg to form an undirected dependency graph*/
7: $udg \leftarrow transfer(ldg)$
8: $result \leftarrow preclassify(udg, k)$
9: /*add label nodes according the pre-classification result*/
10: $addLabelNodes(udg, s1, s2, ..., sk, result)$
11: $curNum \leftarrow 0$
12: **while** $curNum < k$ **do**
13:     /* Renaming two label nodes with the largest weight by S and T respectively*/
14:     $< S, T > \leftarrow renameLabelNodeWithLargestWeight(udg, s1, s2, ..., sk)$
15:     $cutEdges \leftarrow mincut(udg, < S, T >)$
16:     $< n | n \in S, n | n \in S > \leftarrow restorePartition(udg, cutEdegs)$
17:     **if** $n | n \in S$ **then**
18:         $recordPartion(newComps, result)$
19:         **return** $newComps$
20:     **else**
21:         $removeNode(udg, n | n \in S)$
22:         $addPartition(newComps, n | n \in S)$
23:         $curNum = curNum + 1$
24:     **end if**
25: **end while**
26: **return** $newComps$
27: **Procedure** $mincut(udg, < S, T >)$
28: $cutEdges \leftarrow \varnothing$
29: $maxflow \leftarrow 0$
30: **while** $findAugPath(S, T) = true$ **do**
31:     $maxflow \leftarrow maxflow + maxFlowByDfs(S, T)$
32: **end while**
33: $cutEdges \leftarrow findCutEdgeByBfs(S, T)$
34: **return** $cutEdges$
35: **Procedure** $preclassify(udg, k)$
36: /* The sum of weight of edge of each node is calculated */
37: $Emap \leftarrow calWeight(udg)$
38: /* The first k nodes with the maximum weight are selected as central node */
39: $centers \leftarrow findKthNode(map)$
40: $disArray \leftarrow \varnothing$
41: $curNum \leftarrow 0$
42: **while** $curNum < k$ **do**
43:     $< center, node, dis > = calDis(udg, centers)$
44:     $curNum+ = < center, node, dis >.size$
45:     $disArray = disArray \cup < node, center, dis >$
46: **end while**
47: $result \leftarrow \varnothing$
48: **for** each $node \in udg.nodes$ **do**
49:     /* The sum of weight of edges belongs to the node is calculated as totalW*/
50:     $totalW = calWeight(udg, node)$
51:     $disMap \leftarrow \varnothing$
52:     **for** each $center \in centers$ **do**
53:         $dis \leftarrow findDisWithNodesBelongCenter(disarray, center, node)$
54:         $disMap \leftarrow disMap \cup < center, totalW - dis >$
55:     **end for**
56:     /*Find the target center with the minimum distance*/
57:     $targetCenter \leftarrow findTargetCenter(disMap)$
58:     $result \leftarrow result \cup < node, center >$
59: **end for**
60: **return** $result$

are related to nodes. In this paper, we use a multiple-level change detection method to extract changed code [11].

Adding a file corresponds to adding a file node in FDG. The statements belong to the added file are the slice criteria, and we obtain which files have dependencies with the new file, then we add related dependency edges between the added file node with other file nodes.

Deleting a file corresponds to deleting a file node from FDG, and its related edges are deleted.

The following types of changes are related to edges.

Increasing dependency intensity between files corresponds to increasing the weight of the edge, and the weight of the edge is assigned based on the dependency intensity.

Adding a dependency between files corresponds to adding an edge between two file nodes.

Reducing dependency intensity between files corresponds to reducing the weight of the edge, and the weight of the edge is assigned based on the dependency intensity.

Deleting a dependency between files corresponds to deleting a dependency edge between two file nodes.

After modifying FDG, we use cluster methods to obtain new architecture based on modified FDG. In this paper, we adopted a cohesive hierarchical clustering method [12]. This clustering method initially treats each file in the file dependency graph as a cluster, and then continuously merges the clusters with a small distance between clusters, and updates the distances between the new clusters and other clusters. The algorithm is shown in Algorithm 2. Finally, we obtain new architecture based on the evolved code by implementing the above algorithm.

III. EXPERIMENTS AND EVALUATION

A. Experiment Setup

In the section, we conduct experiments to evaluate the effectiveness of our method. We conduct our method with eight open source programs to answer the following research questions.

RQ1: is the co-evolutionary method of code based on evolved architecture effective?

RQ2: is the co-evolutionary method of architecture based on evolved code effective?

We randomly selected eight open source projects as the experimental cases, and these projects contain Java projects, C projects, and C++ projects. The information about these projects is listed in Table I.

TABLE I
THE INFORMATION ABOUT EXPERIMENTAL CASES

Project	Evolution process	Language	LOC
Apns	0.1.5→0.2.0	Java	3K
La4j	0.5.0→0.5.5	Java	9K
AssertJ	3.2.0→3.3.0	Java	19K
GoogleMock	1.5.0→1.6.0	C++	16K
Filezilla	3.30.0→3.31.0	C++	128K
Lua	5.0.0→5.0.1	C	11K
Libev	1.3.2→1.4.8	C	25K
Bash	4.4.12→ 4.4.18	C	103K

Algorithm 2 The hierarchical clustering algorithm

Input:
    Let fg be the file dependency graph after evolution
    Let $reqs$ be the code change requirements
    Let $compLocsb$ be the collection of the lines of code for every components
    Let $locb$ be the total lines of code before evolution
Output:
    Let cg be the component graph after evolution
1: **Function** $clustering(fg, reqs, compLocsb, locb)$
2:   $cg \leftarrow clone(fg)$
3:   $n_b \leftarrow size(compLocs_b)$
4:   /* Calculate the number of components before evolution*/
5:   $loc_a \leftarrow loc_b$
6:   /* Calculate the expected number of components after evolution as stop condition*/
7:   $stopc \leftarrow generateStopCondition(cg, reqs, compLocsb, locb, nb)$
8:   $n_a \leftarrow n_b$
9:   **while** $n_a > stopc$ **do**
10:     /*Find the edge with min distance from component diagram*/
11:     $< src, dest > \leftarrow findMinDPair(cg)$
12:     $merge(cg, < src, dest >)$
13:     $n_a \leftarrow n_a - 1$
14:   **end while**
15:   **return** cg
16: **Procedure** $generateStopCondition(cg, reqs, compLocs_b, loc_b, n_b)$
17:   $average_b \leftarrow loc_b/n_b$
18:   **for** each $req \in reqs$ **do**
19:     **if** $req instance of AddComp$ **then**
20:       $loc_a \leftarrow loc_a + average_b$
21:     **else**
22:       **if** $req instance of RemoveComp$ **then**
23:         $loc_a \leftarrow loc_a - compLoc_b.get(req.comp)$
24:       **end if**
25:     **end if**
26:   **end for**
27:   **if** $loc_a \neq loc_b$ **then**
28:     $stopc \leftarrow loc_a/average_b$
29:   **end if**
30:   **return** $stopc$
31: $merge(cg, < src, dest >)$
32: /* Find the node whose id is src from component graph remove the graph*/
33:   $nodeSrc \leftarrow findNode(cg, src)$
34:   $nodeDest \leftarrow findNode(cg, dest)$
35:   $inEdgeSrc \leftarrow findInEdges(cg, src)$
36:   $outEdgeSrc \leftarrow findOutEdges(cg, src)$
37:   $inEdgeDest \leftarrow findInEdges(cg, dest)$
38:   $outEdgeDest \leftarrow findOutEdges(cg, dest)$
39:   $nodeSrc.name \leftarrow nodeSrc.name + nodeDest.name$
40:   $inEdgeSrc \leftarrow inEdgeSrc \cup EdgeDest$
41:   $outEdgeSrc \leftarrow outEdgeSrc \cup EdgeDest$
42:   $cg \leftarrow cg - nodeDest$
43: /*Remove the cluster named nodeDest from component graph*/

In our experiments, we take actual history versions as experimental cases. V_b and V_a are two software versions before and after evolution. The actual code and actual architecture of V_b are respectively denoted in V_{b_c} and V_{b_a}. Similarly, the actual code and actual architecture of V_a are respectively denoted in V_{a_c} and V_{a_a}.

B. Results and Evaluation

RQ1: Is the co-evolutionary method of code based on evolved architecture effective?

We conduct the following two experiments to analyze the effectiveness of our method: (1) Is the co-evolved code consistency with the actual architecture after evolution? (2) Is the co-evolutionary content of co-evolved code consistency with the actual code changes?

The first experiment is that, we analyze the consistency between the actual code V_{a_c} with the co-evolved code which is obtained by using our method base on V_{a_a}.

Code and architecture belong to different granularity levels, and they cannot be compared directly. We measure the effec-

tiveness of our method based on architectural similarity. The details of the experiments are as follows:

- Co-evolving code by using our method to obtain the new FDG.
- Recovering architecture based on the new FDG to obtain the new architecture.
- Calculating the architectural similarity between the actual architecture and the new architecture.

We use component similarity to assess the similarity between the actual architecture and the new architecture. It is calculated as Formula 3.

$$ComSimilarity(C_i, C_j) = \frac{|F_i \cap F_j|}{|F_i \cup F_j|} \tag{3}$$

In Formula 3, $ComSimilarity(C_i, C_j)$ is the component similarity between the i_{th} component C_i and the j_{th} component C_j, F_i is the set of files which belongs to the i_{th} component, and $|X|$ is the number of elements of the set X. Architecture similarity is the average value of all components similarity.

The component similarity and the architecture similarity are shown in Table II.

TABLE II
THE COMPONENT SIMILARITY AND THE ARCHITECTURE SIMILARITY

Project	Actual component	New componet	Component similarity	Architecture similarity
La4j	data1	decomposition	0.563	
	data2	matrix/source	0.619	
	operation	operation	1	0.836
	linear	linear	1	
	matrixOperation	matrixOperation	1	
Filezilla	interface1	interface#85	1	
	interface2	interface\setting	1	
	interface#1	interface#86	0.882	
	engine	engine	0.961	0.978
	putty	putty	1	
	dbus	dbus	1	
	interface#2	interface#5	1	
Bash	readline1	readline#13	1	
	readline2	termcap	1	
	[ROOT]#1	[ROOT]#32	0.931	
	[ROOT]	[ROOT]#29	0.885	0.977
	support	suppport	1	
	examples\loadables	examples\loadables	1	
	lib\sh	lib\sh	1	
	lib\intl	lib\intl	1	
Libev	libcork1	libcork\config	1	
	libcork2	libcork#50	1	
	libipset	libipset	0.625	
	\	libipset\map	0	0.703
	\	libipset\set	0	
	libcork#1	libcork#52	1	
	libev#47	libcork#52	1	
	libev	libev#47	1	
GoogleMock	gmock1	gmock#10	1	
	gmock2	gmock#15	1	1
	src	src	1	
	internal	internal	1	
Bash	readline1	readline#13	1	
	readline2	termcap	1	
	[ROOT]#1	[ROOT]#32	0.931	
	[ROOT]	[ROOT]#29	0.885	0.977
	support	suppport	1	
	examples\loadables	examples\loadables	1	
	lib\sh	lib\sh	1	
	lib\intl	lib\intl	1	

Architecture change	The co-evolutionary content of code	Actual code changes
Adding component named libipset	Adding highly cohesive files	Adding two directories named libipset\include\ipset and libipset
Adding method invocation dependencies between the two components named src and libipset	Adding method invocation dependencies between the files contained in the two components named src and libipset	Adding method invocation dependencies between src\acl.c and libipset\include\ipset\ipset.h.
Increasing include reference dependencies between the two components named src and libev	Increasing include reference dependencies between the files contained in the two components named src and libve	1) Adding include reference dependencies between src\tunnel.h and libev\ev.h. 2) Adding include reference dependencies between src\udprelay.h and libev\ev.h.
Splitting the component core into core#1 and core#2	Splitting the files contained in the component named core into two parts and reducing the dependencies between the two parts.	1) Reducing dependencies between org.assertj.core.api and org.assertj.core.condition. 2) Reducing dependencies between org.assertj.core.api and org.assertj.core.util.

As Table II shows that, the average architecture similarity is 0.899. The similarity indicates that our method can co-evolve architecture based on evolved code effectively.

Here, we take `Libev` as an example to show the corresponding relations between the co-evolutionary content of code and the actual code change, the corresponding relations are shown in Table III.

According to Table III, the co-evolutionary content of code is consistent with the actual code change, so our method is effective for co-evolving code.

RQ2: Is the co-evolutionary method of architecture based on evolved code effective?

The co-evolutionary change method of architecture based on evolved code aims at keeping the consistency between architecture and the evolved code. We evaluate the effectiveness of the method by analyzing the consistency between the co-evolved architecture and the actual architecture change.

We conduct the following two experiments: (1) Is the co-evolved architecture consistency with the actual architecture after evolution? (2) Are the details of co-evolved architecture consistency with the actual architecture changes?

The experiment is performed in the following steps:

- Obtaining the actual code change by using the change detection method.
- Obtaining the co-evolved architecture by using our method.
- Obtaining the actual architecture change by comparing CDG before and after evolution.
- Analyzing the consistency between the co-evolved architecture and the actual architecture change.

The co-evolved architecture and the actual architecture changes are about CDG, and the graph consists of the component and the dependency edges between components, that is if the co-evolved architecture is consistent with actual changes, the number of changed nodes and changed edges of them are the same. We summarize the number of changed nodes and the number of changed edges of each project to analyze the consistency, and the information is shown in Table IV.

Table IV shows statistical information about the number of changed nodes and dependency edges. As the table shows that the number of co-evolved objects is equal to the number of actual changes, that is, our method can co-evolve architecture which is consistent with the actual architecture changes. So

TABLE IV
THE NUMBER OF CHANGED NODES AND CHANGED EDGES.

		Type	La4j	AssertJ	Lua	Libev	Apns
Co-evolved objects	Node	Adding	2	23	4	2	6
		Deleting	29	0	2	4	2
	Edge	Adding	30	231	10	124	15
		Deleting	274	10	2	0	5
		Increasing	3	44	9	8	13
		Reducing	3	6	5	3	2
Actual change	Node	Adding	2	23	4	2	6
		Deleting	29	0	2	4	2
	Edge	Adding	30	231	10	124	15
		Deleting	274	10	2	0	5
		Increasing	3	44	9	8	13
		Reducing	3	6	5	3	2

the co-evolutionary method of architecture based on evolved code is effective.

In the second experiment, we further analyze the detail between actual architecture.

Here, we take `La4j` as an example to analyze the consistency between the co-evolutionary content of architecture and the actual change. The details are shown in Table V.

As Table V shows that, actual architecture change in the actual evolution is "Add org.la4j.Matrix node", the corresponding co-evolutionary content of the architecture is "Add org.la4j.Matrix node", so the co-evolutionary content matches the actual architecture change. The evolution of dependency is mainly reflected in the addition or deletion of dependency edges between nodes or changes in the weights of edges. According to Table V, we know that, the actual architecture change is consistent with the co-evolved architecture.

IV. RELATED WORK

At present, there are mainly four co-evolutionary methods which are shown in the first column of Table VI.

Direction. The first two methods only support one-way co-evolution, so the two methods support fewer application scenarios than the other methods.

Representation. Most of these methods use the component dependency graph as the representation of architecture, except the multi-view method. The component dependency graph is a widely acceptable representation of architecture. Considering architecture and code may be used in other researches, we think that the component dependency graph is more applicable for representing architecture.

TABLE V
THE CO-EVOLVED ARCHITECTURE AND THE ACTUAL ARCHITECTURE CHANGES IN LA4J PROJECT

Actual Code change	The co-evolutionary content of architecture	Actual architecture change
Adding a file Matrix in the package org.la4j.	Adding a new node org.la4j.Matrix.	Adding a new node org.la4j.Matrix.
Deleting the file CCSFactory of the package org.la4j.factory.	Deleting the node org.la4j.factory.CCSFactory.	Deleting the node org.la4j.factory.CCSFactory org.la4j.factory.
Adding 4 method invocation dependencies between org.la4j.LinearAlgebra and org.la4j.Matrix.	Adding a new dependent edge between org.la4j.LinearAlgebra and org.la4j.Matrix, and the weight is 4.	Adding a new dependent edge between org.la4j.LinearAlgebra and org.la4j.Matrix, and the weight is 4.
Deleting the generalization dependencies between the two files ArrayVectorSource and VectorSource which are contained in the package org.la4j.vector.source.	Deleting the dependent edge between the two nodes org.la4j.vector.source. ArrayVectorSource and org.la4j.vector.source. VectorSource.	Deleting the dependent edge between the two nodes org.la4j.vector.source. ArrayVectorSource and org.la4j.vector.source. VectorSource
Increasing the parameter dependencies between the two files org.la4j.linear.JacobiSolver and org.la4j.Matrix, and the increased dependencies are 2 times.	Increasing the weight of edge between the two nodes org.la4j.linear. JacobiSolver and org.la4j.Matrix by 2.	Increasing the weight of edge between the two nodes org.la4j.linear.JacobiSolver and org.la4j.Matrix by 2.
Reducing the method invocation dependencies between the two files AbstractSolver and LinearSystemSolver which are contained in the package org.la4j.linear.	Reducing the weight of edge between the two nodes org.la4j.linear.AbstractSolver and org.la4j.linear. LinearSystemSolver by 1.	Reducing the weight of edge between the two nodes org.la4j.linear.AbstractSolver and org.la4j.linear. LinearSystemSolver by 1.

TABLE VI
THE COMPARISON BETWEEN RELATED METHODS

Method	Direction	Representation	Technology	Implementation
Architecture recovery	One-way	Component dependency graph	Mapping rules, clustering algorithm	Automatic
Generate code	One-way	Component dependency graph	Mapping rules, clustering algorithm	Automatic
Multi-View	Two-way	UML	Mapping relations	Automatic
Information fusion	Two-way	Component dependency graph	Logic element programming, metadata, language development	Artificial

Technology. The table shows that the mapping rules are widely used in many methods. It indicates that the mapping rules are effective for co-evolutionary methods.

Implementation. Most of these methods are implemented automatically, but the information fusion method is implemented artificially. So the information fusion method is not suitable for large-scale programs.

According to the above analysis of related work, we know that the above four methods do not support the two-way co-evolution automatically between component dependency graph and code automatically. So, we propose a co-evolutionary method to solve these problems.

V. CONCLUSION AND FUTURE WORK

In the paper, we propose a co-evolutionary method between architecture and code, including the co-evolutionary method of code based on evolved architecture and the co-evolutionary method of architecture based on evolved code. In our method, more types of change actions are taken into consideration, and the changes are converted based on mapping rules, then we use FDG as the intermediate level between architecture and code to perform co-evolutionary algorithms. We conduct the experiments with eight open source projects, and the experimental results indicate that the co-evolutionary method of architecture based on evolved code and the co-evolutionary of code based on evolved architecture are all effective. In our future work, we will combine our method with architecture quality, then we improve architecture quality automatically.

REFERENCES

[1] Umberto Azadi, Francesca Arcelli Fontana, and Davide Taibi. Architectural smells detected by tools: a catalogue proposal. In *2019 IEEE/ACM International Conference on Technical Debt (TechDebt)*, pages 88–97. IEEE, 2019.
[2] Daniel Link, Pooyan Behnamghader, Ramin Moazeni, and Barry Boehm. The value of software architecture recovery for maintenance. In *Proceedings of the 12th Innovations on Software Engineering Conference*, pages 1–10, 2019.
[3] Ana Paula Allian, Bruno Sena, and Elisa Yumi Nakagawa. Evaluating variability at the software architecture level: an overview. In *Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing*, pages 2354–2361, 2019.
[4] Burak Uzun and Bedir Tekinerdogan. Domain-driven analysis of architecture reconstruction methods. In *Model Management and Analytics for Large Scale Systems*, pages 67–84. Elsevier, 2020.
[5] Eric Bruneton, Thierry Coupaye, Matthieu Leclercq, Vivien Ma, and Jean Bernard Stefani. The fractal component model and its support in java. *Software Practice & Experience*, 36(11-12):1257–1284, 2010.
[6] R. France and J. M. Bieman. Multi-view software evolution : A uml-based framework for evolving object-oriented software. In *Proceedings IEEE International Conference on Software Maintenance. ICSM 2001*, pages 386–395, 2001.
[7] Pooyan Jamshidi and Claus Pahl. Business process and software architecture model co-evolution patterns. In *International Workshop on Modeling in Software Engineering*, pages 91–97, 2012.
[8] Mitchell A Potter and Kenneth A De Jong. Cooperative coevolution: An architecture for evolving coadapted subcomponents. *Evolutionary Computation*, 8(1):1–29, 2014.
[9] Amir Ngah and Siti Aminah Selamat. Using object to slice java program. *Journal of Engineering and Applied Sciences*, 13(6):1320–1325, 2018.
[10] Yeounoh Chung, Tim Kraska, Neoklis Polyzotis, Ki Hyun Tae, and Steven Euijong Whang. Slice finder: Automated data slicing for model validation. In *2019 IEEE 35th International Conference on Data Engineering (ICDE)*, pages 1550–1553. IEEE, 2019.
[11] Tong Wang, Dongdong Wang, Ying Zhou, and Bixin Li. Software multiple-level change detection based on two-step mpat matching. In *2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 4–14. IEEE, 2019.
[12] Nenad Medvidovic and Vladimir Jakobac. Using software evolution to focus architectural recovery. *Automated Software Engineering*, 13(2):225–256, 2006.

An Empirical Study of Maven Archetype

Xinlei Ma[†] Yan Liu[†] *
[†]School of Software Engineering, Tongji University, Shanghai, China
* Corresponding Author
{1831592, yanliu.sse}@tongji.edu.cn

Abstract—Archetype is a Maven project templating toolkit. An archetype is defined as an "original pattern" of the representative Maven project. Using archetypes enables Maven developers to quickly work in a way consistent with the best practices which demonstrate many Maven features and usage patterns of Maven components. Nowadays, more and more developers utilize archetypes to standardize development within their organizations. Despite the ever-growing use, there are still limited experimental evidence and guidance on how to leverage the power of archetype. Meanwhile, because of the enormous scale and spotty quality of projects, it is incredibly challenging to perform analysis on the whole Maven central repository. As the simple "artifacts" of the Maven best practices in many diverse domains, Maven archetypes are ideal for studying the "Maven Way" of configuration and the usage pattern of Maven libraries. Therefore, we perform the first empirical study on 2,326 archetypes retrieved from the Maven central repository to discover the archetype characteristics. Our results identify the configuration schema patterns, structural patterns, the uses of dependencies/plugins in archetypes, and summarize some evolution characteristics of archetypes as well. The primary archetype characteristics capture the potential research value of archetypes. The guidance on how to configure the archetype and utilize Maven libraries can be leveraged to maintenance, automatic completion both for archetypes, and Maven projects.

Index Terms—Maven; Archetype; Configuration; Software Analytics

I. INTRODUCTION

Apache Maven is a software project management and comprehension tool. Based on the concept of a project object model (POM), Maven can manage a project's build, reporting, and documentation from a central piece of information [1], [2].

Archetype is a Maven project templating toolkit. It helps archetype authors create Maven project templates for Maven developers and provides developers with the means to generate Maven projects from those project templates. An archetype is defined as an "original pattern" or "model", which helps Maven developers get started as quickly as possible. More encouragingly, it will introduce new users to the best practices employed by Maven [3]. Nowadays, more and more developers adopt the archetypes to standardize development within their organizations. Despite the ever-growing use, there are still limited experimental evidence and guidance on how to configure archetypes.

Maven central repository is one of the most popular and widely used repositories of Maven projects, which contains

more than 2.8M Java Maven projects [4]. However, those who want to analyze the whole Maven central repository face challenges on an enormous scale and spotty quality of projects [4]. As the simple "artifacts" of Maven best practices, archetypes also present the "Maven Way" [5] of configuration and some usage patterns of Maven libraries. Therefore, we perform the empirical study on Maven archetypes in the Maven central repository instead of all Maven projects. To our knowledge, we are the first to perform the empirical study on Maven archetypes [6]. Previous archetype-related researches [7] paid more attention to its practical usage but failed to leverage its hidden knowledge.

An archetype consists of the archetype metadata which describes the contents of archetype and Maven project templates which can generate a working project [3]. These project templates include *pom.xml* files, Java files, and other resource files. In concept, the archetype is a code "skeleton" or a very simple "artifact" [3]. Maven archetypes and projects are configured by a POM, which is stored in a *pom.xml* file. Since our research object is studying archetype configuration and usage patterns of Maven libraries, we focus on POMs in *pom.xml* files.

This work start with four research questions on the configuration levels, design appropriate analysis process to explore archetype patterns, uses of dependency/plugin, evolution, and obtain some interesting conclusions finally. The main contributions of our work are as follows:

- Gaining novel insight into configuration-level analysis across representative Maven projects.
- Presenting a first complete process on archetype analysis with appropriate static and quantitative methodologies.
- Releasing a pre-processed datasets in https://zenodo.org/record/3702349#.Xmd21JMzZQI, including all 2,326 archetypes which can be used for further research.

The remainder of this paper is organized as follows: Section 2 presents the terminologies related to this paper. Section 3 describes the methodology applied to conduct the analysis. Section 4 presents the experimental results of the four proposed questions. Section 5 presents the conclusions of the study. Due to space limitations, all graphs and Table II and Table III are available at https://drive.google.com/file/d/1EhqU1GY1KBS5xipo4DkUtCiurMmE2pdS/view.

II. TERMINOLOGY

Maven artifacts such archetypes, dependencies, and plugins are identified by the 3-tuple "GroupId:ArtifactId:Version", where GroupId is the organization of this project, ArtifactId is the name of this project, and Version is the version for this project respectively. We provide some terminologies of POM here to make readers understand better.

POM Tree. A POM is stored in a well-defined eXtensible Markup Language (XML) file consisting of element tags, which can be modelled as a rooted tree with element nodes: the POM XML tree [8].

POM Relationship. POM relationships generally include dependencies, inheritance, and aggregation [2]. POM Inheritance means that a POM extends its parent POM directly and can inherit and override its parent. A Maven project can be represented as a sub-module with its own POM. Through POM, a multi-module or aggregator project can group many sub-module projects [2].

Then, for better analysis, we give detailed definitions of essential concepts in this paper.

POM XML Element Sequence. In a POM Tree, each element node has a unique path from the root ("project" element) to itself. Given a specific element Ele, we can represent it as a tuple $<Ele, S_{ele}>$ where Ele is the tail element of the sequence, and S_{ele} is the prefix of this element in the POM tree.

POM Inheritance Depth. It defined as the depth of the given POM in the POM inheritance chain. In other words, it is the number of POMs preceding a given POM in the POM inheritance chain.

Evolution Type of Dependency and Plugin. We define the following kinds of dependency changes: *dep_add*, *dep_del*, *dep_add_del*, *dep_v*, *dep_scope*, and *dep_exclutions*. The evolution type of plugin is similar. We use *dep*, *plu*, *v*, and *del* to denote dependency, plugin, version, and deletion.

III. METHODOLOGY

In this empirical study, firstly, we give some definitions for some concepts related to our study. Then we propose four archetype-based research questions, collect the dataset, and design corresponding experiments. Based on the findings of each experiment, we provide conclusions for each research question or explanations of experimental results. The overall workflow is presented in Fig. 1.

A. Research Questions

We propose four questions from the aspects of patterns, uses of dependency/plugin, and evolution. The questions are:

- RQ0: What are typical configuration schema patterns in archetype POMs?
- RQ1: How are structural patterns used in archetypes?
- RQ2: What is the usage pattern of dependencies and plugins in archetypes?
- RQ3: How archetypes evolve during the lifecycle?

B. Dataset Preparation

1) Collecting raw dataset: We collect all the second latest archetypes in JAR format from the Maven central repository, which include 6,432 POM files and 11,050 Java code file. Moreover, for archetype evolution study, our dataset contains 10,184 archetypes releases for a total of 2,326 archetypes, which gives an average of 4 releases per archetype.

2) Processing raw dataset: In Maven projects or archetypes, the *pom.xml* file specifies the project structure, settings for different build steps, and libraries on which the project depends. We decompress the JAR archetypes, retrieve *pom.xml* files from decompressed archetypes, and parse them to extract useful data (such as adopted dependency and pom relationships). In this work, we mainly focus on the root *pom.xml* files in archetypes except for the POM relationships (which are extracted from the *pom.xml* file for archetype generation process).

C. Technologies

Apriori [9] frequent itemset mining algorithm is used to investigate what type of dependencies or plugins always work along, with Confidence level (C), Support (S), and Lift (L) as metrics.

IV. OBSERVATIONS AND FINDINGS

A. RQ0: What are typical configuration schema patterns in archetype POMs?

As a configuration file, a *pom.xml* file should base on the POM schema. *Configuration Schema Pattern*s of POM are useful for POM automatic completion. We define *Configuration Schema Pattern* as a combination of frequent element sequences to implement a configuration concern in POM. To answer this question, we propose three subquestions:

- RQ0.1 Which element tags are frequently used in root *pom.xml* files?
- RQ0.2 Which element sequences are frequently used in root *pom.xml* files?
- RQ0.3 Which configuration schema patterns are frequently used in root *pom.xml* files?

Figure 3 and Figure 4 illustrate the top frequently used element tags and element sequences in the root *pom.xml* files separately. Referring to these element sequences with corresponding frequency, we summarize several typical schema patterns by combining relational element sequences into an element tree. Figure 2 displays some typical schema patterns, and the root element of these patterns are *build* (the top-level build element under project), *dependency*, *plugin*, and *profile*. Regarding each element in the element tree, we manually divide their direct suffix element tags into two sets according to frequency. In Figure 2, the dotted lines with boxes mean the set of highly frequently used element tags, and the original lines with boxes mean the set of less frequently used element tags. The orange box means the non-leaf node, and the green box means the leaf node.

RQ0 We analyze the XML schema patterns used in POM files to detect the recurring element tags and sequences. For POM automatic completion, several configurations patterns are detected and concerns principally the tags *build*, *dependency*, *plugin*, and *profile*.

B. RQ1: How are structural patterns used in archetypes?

Our investigation of structural patterns in archetypes focuses on aggregation and inheritance. POM aggregation is used to manage a complex system which involves hundreds of interrelated sub-modules. POM inheritance can effectively reduce the repetition of configuration code and make it easier to reuse configuration code. Some typical "Maven Ways", such as centralized management of dependencies, predefining of public components, can be used through POM aggregation and POM inheritance. In this part, we propose two subquestions:

- RQ1.1 Which proportion of archetypes adopts POM inheritance or POM aggregation in our dataset? Is the POM Inherited Depth usually not more than one?
- RQ1.2 Are archetypes which adopt POM inheritance and POM aggregation simultaneously in proportion in our dataset?

TABLE I: Uses of POM inheritance and aggregation in primary dataset and extended dataset

Item	Num	Avg	Mid	Max	Scope
inheritance	1,817	/	/	/	primary
aggregation	1	/	/	/	primary
inheritance depth	/	2.18	2.00	7.00	primary
aggregation	491	/	/	/	extended
aggregation submodule	/	10.03	6.00	62.00	extended
inheritance & aggregation	1229	/	/	/	extended

For RQ1.1, we find that 1,817 archetypes have the specified parent, while only one archetype has sub-modules(it has specified parent as well). The latter phenomenon is unreasonable, and we infer it is because these archetypes are sub-modules of other archetypes which are out of our study range. Therefore, besides previous basic 2,326 archetypes, we expand our study scope to parent archetypes upward POM inheritance chains and find that 491 archetypes are sub-module projects of their direct parent. Moreover, Table I shows the average, median, and max POM Inherited Depth for 1,817 child archetypes, and the average, median, and max sub-module number for 1,296 archetypes which are both multi-module and parent projects.

RQ1 *Pom Inheritance* is adopted by more than 2/3 of the studied archetypes, and POM Inherited Depth is always more than 1. About 1/5 of the studied archetypes are sub-modules of their direct parent, which means POM aggregation and inheritance are often used simultaneously. When developers desire to scale up their archetypes, POM aggregation and inheritance are encouraged to be utilized.

C. RQ2: What is the usage pattern of dependencies and plugins in archetypes?

According to the principle about the wisdom of the crowds in software engineering, referring to other developers' decisions on the library can avoid some pitfalls experienced by other developers [10]. Therefore, we try to explore the following subquestions:

- RQ2.1 What is the utilization distribution of dependencies and plugins?
- RQ2.2 Which dependencies and plugins in archetypes are frequently adopted, and in which scenario?
- RQ2.3 Which dependencies and plugins are frequently adopted at the same time and why?

Figure 5a and Figure 5b present the utilization distribution of dependencies and plugins, respectively, and each one of distributions is right-skewed. The x-axis shows the number of archetypes using a type from the dependency or plugin. The y-axis shows the fraction of total dependencies or plugins falling within that utilization number bin. It indicates that the majority of dependencies or plugins show low utilization, and this result is similar to the study about utilization distribution of Maven Components among open-source Java projects [11].

For RQ2.2, Figure 6a and Figure 6b illustrate the top 20 frequently used dependencies (from overall 3,188 distinct dependencies) and the top 20 frequently used plugins (from overall 444 distinct plugins) at the level of GroupId. The most popular dependency is junit:junit appearing in 536 archetypes, and the most frequently used plugin is org.apache.maven.plugins:maven-compiler-plugin appearing in 643 archetypes.

The result of frequent item-set mining shown in Table III presents what types of dependencies or plugins are utilized together in high frequency (RQ2.3). We remove the most frequently used dependencies or plugins in this result like junit since they nearly appear in every archetype. In Table III, the tag means the category of dependency or plugin. And S, C, L, D, P means support, confidence level, lift, dependency, and plugin, respectively.

Moreover, we summarize three relationships based on the mining result to explain why they always work along, which are also indicated in Table III:

- **Functionally Related.** Functionally related dependencies or plugins usually belong to the same Java function module, such as log and test.
- **Tool.** Some dependencies (or plugins) may support others as tools.
- **Up-Down-Stream.** The downstream dependencies (or plugins) usually depend on the upstream dependencies (or plugins) to realize interfaces.

155

RQ2 Very few dependencies or plugins are frequently used, while most of them are hardly used. We make a two-level classification for popular dependencies according to their function and usage scenarios. The result of frequent item-set mining gives a primary answer to what types of dependencies or plugins always work along, and we summarize the relationships to explain the mining result.

D. RQ3: How archetypes evolve during the lifecycle?

Configuration management includes controlling the changes to the items such as dependencies in the system throughout their life cycle [12]. In order to analyze the evolution of archetypes during the life cycle of POM files, we try to answer several subquestions as follows:

- RQ3.1 How often do archetypes release a new version?
- RQ3.2 Which are the most frequently changed items when archetypes evolve?

TABLE IV: Overview of dependency/plugin addition and deletion

Item	Median	Mean
Dependency Add Rate	0.25	0.46
Dependency Del Rate	0.25	0.31
Plugin Add Rate	0.50	0.79
Plugin Del Rate	0.50	0.50

Item	Number
Dependency Addition	932
Dependency Deletion	683
Dependency Addition and Deletion	539
Plugin Addition	461
Plugin Deletion	298
Plugin Addition and Deletion	193

An archetype version number composes of major, minor, incremental version and qualifier. We regard only major and minor version number change as an iteration in this question for incremental numbers and qualifiers are optional. In our dataset, archetypes release a new version every 150 days on average and have 7,018 iterations in total.

From the results in Figure 7, we can find that the majority of change types are associated with dependencies and plugins, especially their version. Fig. 8 shows the occurring ratio for each evolution type. (In one iteration, it will be accumulated when calculating, if the same item changes.)

We also notice that the high frequency of dependency and plugin addition or deletion, so we calculate the added or deleted number of dependencies/plugins and their rate for each iteration. The added dependency rate is calculated as the following equation:

$$rate_{added\_dependency} = \frac{Cnt_{added\_dependency}}{Cnt_{total}} \quad (1)$$

where $Cnt_{added\_dependency}$ is the number of added dependencies and Cnt_{total} is the number of total dependencies in the

old version of archetype. The other three rates are similar to this equation.

As shown in the first part of Table 9, the number of added or deleted plugins is less than that of dependencies in general, while the add rate (or del rate) of plugins is slightly higher than that of dependency as shown in Table IV. The second part of Table IV gives an overview of the total number of addition and deletion behaviors for dependency and plugin. It shows that almost half of addition and deletion appear together. Figure 10a and Figure 10b show the time of dependency/plugin addition and deletion behaviors. Dependency addition and deletion become frequent since 2007, while plugins since 2009.

RQ3 In general, archetypes release a new version every 2 to 5 months. For archetype iterations, the items about dependencies and plugins change most often. The reason is that archetype developers always update the dependency and plugin information and add/delete dependencies and plugins. This reflects the high-level attention on the dependency/plugin management from archetype authors.

V. THREADS TO VALIDITY

In our study, we use the second latest version of archetype to carry out our research, which may cause subtle deviations in our research results. Another threat to validity concerns the definitions of the archetype iteration, which may slightly influence our findings. However, this will not change the trend of overall results on how archetypes evolve.

VI. CONCLUSIONS

In general, Maven archetypes are the best Maven practices which are well worth of research. This paper presents an empirical analysis of 2,326 Maven archetypes hosted by the Maven central repository. We focus on the Maven archetype configuration from the aspects of patterns, uses of dependency/plugin, and evolution, and provide some interesting experimental results. We public our datasets, including 2,326 archetypes, 10,184 releases, and result data in this study. We hope this paper will benefit further study on this topic.

REFERENCES

[1] Wikipedia, "Apache maven," https://en.wikipedia.org/wiki/Apache_Maven, 2019.
[2] A. S. Project, "Apache maven," https://maven.apache.org/pom.html#What_is_the_POM/, 2019.
[3] ——, "Apache maven archetype," https://maven.apache.org/guides/introduction/introduction-to-archetypes.html, 2019.
[4] A. Benelallam, N. Harrand, C. Soto-Valero, B. Baudry, and O. Barais, "The maven dependency graph: a temporal graph-based representation of maven central," in *2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)*. IEEE, 2019, pp. 344–348.
[5] T. O'Brien and M. V. S. Inc., *Maven: the definitive guide*. O'Reilly, 2008.
[6] M. Allamanis, E. T. Barr, P. Devanbu, and C. Sutton, "A survey of machine learning for big code and naturalness," *ACM Computing Surveys (CSUR)*, vol. 51, no. 4, p. 81, 2018.
[7] T. J. Speicher and Y. Cheon, "Composing a cross-platform development environment using maven," in *Proceedings of the RCCS+ SPIDTEC2 Workshop on Regional Consortium for Foundations, Research and Spread of Emerging Technologies in Computing Sciences, Juarez, Mexico*, 2018, pp. 68–80.

[8] Y. Lu, J. Liang, Y. Xiao, S. Huang, D. Yang, W. Wang, and H. Lin, "Xmlvalue: Xml configuration attribute value recommendation," in *2017 IEEE 41st Annual Computer Software and Applications Conference (COMPSAC)*, vol. 1. IEEE, 2017, pp. 202–207.

[9] R. Srikant, "Fast algorithms for mining association rules and sequential patterns," Ph.D. dissertation, Citeseer, 1996.

[10] Y. M. Mileva, V. Dallmeier, M. Burger, and A. Zeller, "Mining trends of library usage," in *Proceedings of the joint international and annual ERCIM workshops on Principles of software evolution (IWPSE) and software evolution (Evol) workshops*, 2009, pp. 57–62.

[11] H. Sajnani, V. Saini, J. Ossher, and C. V. Lopes, "Is popularity a measure of quality? an analysis of maven components," in *2014 IEEE international conference on software maintenance and evolution*. IEEE, 2014, pp. 231–240.

[12] I. of Electrical and E. Engineers, *IEEE Standard for Software Configuration Management Plans*. IEEE, 1990.

Evaluating the Usefulness and Ease of Use of an Experimentation Definition Language

Florian Auer, Michael Felderer
University of Innsbruck
Innsbruck, Austria
florian.auer@uibk.ac.at, michael.felderer@uibk.ac.at

Abstract—Before any online controlled experiment, a hypothesis has to be formulated. Moreover, the design, execution, and analysis have to be planned. Given that the definition of an experiment varies considerably amongst experimentation platforms, no common experiment definition exists. Furthermore, there is to the best of the authors' knowledge no platform-independent experiment definition model proposed in the literature.

Thus, we aim to propose an experimentation definition language and evaluate its usefulness and ease of use. Therefore, we developed a domain-specific language based on the results of a previous study and conducted a technology acceptance model study with 30 participants. It revealed that the proposed experiment definition language is considered useful amongst the majority of participants. Moreover, most of the participants rated the language easy to use. Participants without prior knowledge of the domain-specific language's host language (JSON – JavaScript Object Notation) rated the language considerably less easy to use.

To conclude, the proposed experimentation definition language supports practitioners in their experimentation process by providing them a structure and pointing them out to experiment characteristics that could be considered. Furthermore, the machine-readable definition of experiments represents a first step for many research directions, like the automated verification of experiments, or the development of an experiment knowledge base.

Index Terms—continuous experimentation, domain-specific language, technology acceptance model, online controlled experiment definition

I. INTRODUCTION

Online controlled experimentation of software features allows to quickly assess ideas and to make data-driven decisions about them [1]. Furthermore, the technique of deploying a change, exposing it to a subset of the users and collecting telemetry about it, has the potential to be a vehicle for software quality assurance [2] of modern technologies like machine learning or the internet of things (IoT) that are challenging for offline software quality assurance techniques [3] like traditional software testing. Given the potentially large impact of decisions that are based on experiments, the correct execution and therefore, the reliability of experiments are of great importance to organizations. Recent empirical studies [4], [5] found that the majority of practitioners use in-house built experimentation platforms. Thus, they use self-built tools to execute their experiments. Although this seems to be a reasonable choice to adapt the experimentation process to the

respective needs of an organization, it complicates the knowledge and experience exchange within the community and the development of platform-independent techniques. A platform-independent experimentation definition language would represent a foundation for a more structured experiment definition, static experiment verification, and software quality assurance for modern technologies like machine learning or IoT – independent of the concrete used experimentation platform. Thus, this paper presents a tool-independent experimentation definition language (EDL) based on the findings of [5]. The language is evaluated on its usefulness and ease of use using the technology acceptance model by Davis et al. [6].

The remainder of this paper is organized as follows. Section II describes the proposed experimentation definition language. Section III discusses the applied research methods and their threats to validity. Section IV presents the findings and discusses them. Finally, Section V concludes the paper.

II. AN EXPERIMENTATION DEFINITION LANGUAGE (EDL)

A domain-specific language to define online controlled experiments needs to describe an experiment through all its lifecycle phases to provide a complete definition of an experiment. Fabijan et al. [4] present an experiment lifecycle with three phases (see Fig. 1).
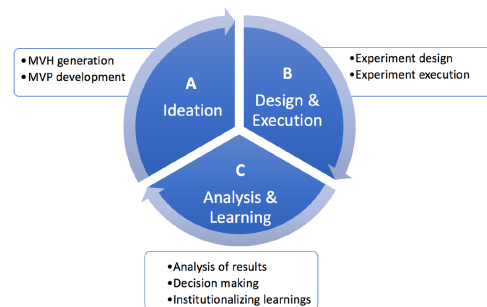
Fig. 1. Experiment lifecycle by Fabijan et al. [4].

In the first phase called the ideation phase, ideas are formalized as treatment descriptions and their success criteria (minimal viable hypotheses). Based on a hypothesis the treatment is developed as a minimal viable product. Thus, the implementation is tailored to meet the experiment's requirements.

158
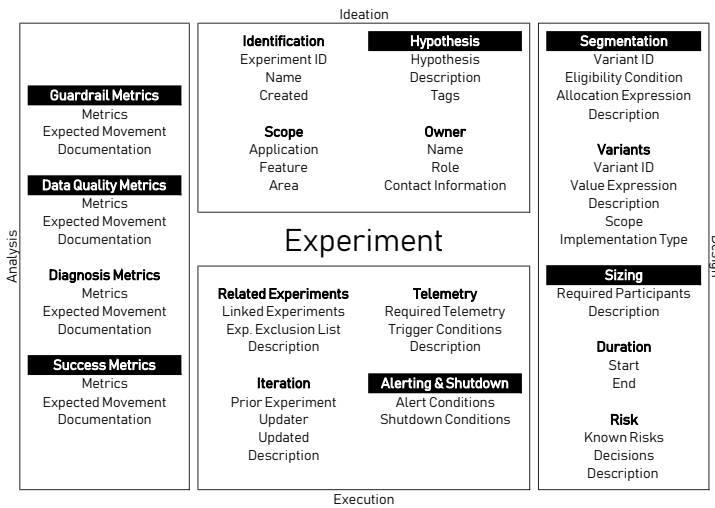
Fig. 2. Experiment definition characteristics taxonomy [5].

In the next phase, the design and execution of an experiment are addressed. The design of an experiment includes amongst others, the segmentation of the customers and the size and duration required for reliable findings. During the execution of an experiment monitoring is necessary to detect harmful experiments (e.g. through alert conditions).

Finally, in the last phase, the results are analyzed, decisions based on the findings are made and lessons from an experiment are shared. Therefore, the collected data of an experiment is analyzed and metrics are computed. After the comparison of the success criteria with the outcome of the experiment, data-driven decisions are made. Finally, the lessons learned about the experimentation process (e.g. execution) and the influence of the change on metrics are shared within an organization.

Authors in [5] assembled a taxonomy of characteristics (see Fig. 2) that is used to describe online controlled experiments in each phase of the experimentation lifecycle. It contains in comparison to other experiment models (e.g. [4]), properties that define the experiment design for each phase. The study considered the literature on online controlled experimentation, experimentation platforms, and the opinions of industrial experts. The developed taxonomy consists of 17 characteristics (e.g. experiment owner) and their properties (e.g. name, role, contact). Based on these findings and the observations made during the research, an experimentation definition language was developed. The described taxonomy of experimentation characteristics served as a domain model for the development of the domain-specific language.

The analysis of open-source as well as proprietary experimentation platforms in [5] revealed that the data exchange format JSON was the preferred data format of most platforms. Thus, it seemed reasonable to use JSON as host language given its widespread use in the community and its mature support by all common programming languages. As a result, the domain model based on the taxonomy was translated into a JSON schema that describes the experimentation definition language,

its objects, and properties. This decision has the advantage that definitions written in the domain-specific language (DSL) are not only machine-readable, but they can be validated with standard JSON tools too. Listing 1 of an experiment defined in the language shows that the structure follows the taxonomy closely.

Listing 1. Structure of an experiment written in EDL.

```
{
  "Ideation":{
    "Hypothesis":...,
    "Owners":...
  },
  "Design":{
    "Variants":...,
    "Segmentation":...
  },
  "Execution":{
    "AlertingAndShutdown":...
  },
  "Analysis":{
    "SuccessMetrics":...,
    "GuardrailMetrics":...
  }
}
```

Given that the taxonomy already considers the experimentation lifecycle, the structure of the taxonomy was transferred to the language. Furthermore, the lifecycle-oriented arrangement of the characteristics gives guidelines during the planning of an experiment.

The second main technology used for the implementation of the language, is the — according to the Stack Overflow Developer Survey[1] — popular code editor Microsoft Visual Code[2]. It has a mature support for both JSON and JSON schemas, including additional JSON schema constructs to add online documentation, auto-completion of predefined partial templates and instant validation of the JSON document according to a JSON schema. Hence, in addition to the JSON schema, online documentation, and partial templates for all properties were developed. As a result, the editor allows language users to explore the language through auto-completion (e.g. what could be written at this location?) and provides rapid feedback on the syntactical validity of the defined experiment (e.g. is the experiment valid against the schema?).

As a result, the experiment definition language has the following key characteristics:

- *Human-readable*: Given that experimentation involves stakeholders from different functional expertise (e.g. UX design, software engineering, business), the experiment definition has to be easy to understand and readable independent of the reader's expertise.
- *Machine-readable*: The definition of an experiment is based on a common data exchange format that can be processed with standard tools.
- *Living documentation*: The definition of an experiment should be the single point of truth concerning the respective experiment. Thus, it should serve the stakeholders as a discussion base and plan of action, and it should provide

---

[1]https://insights.stackoverflow.com/survey/2019#development-environments-and-tools, accessed 2020-02-25.

[2]https://code.visualstudio.com/

all necessary properties for the experiment platform to execute the experiment.

- *Knowledge sharing*: Lessons learned from the design, execution, and analysis of the experiment are valuable knowledge that should be captured and shared. Therefore, the definition of an experiment includes comments, decisions, and the reasoning behind them. Moreover, properties like tags improve the structured archival of experiment definitions.

III. RESEARCH METHOD

To evaluate the proposed language, the technology acceptance model (TAM) by Davis et al. [6] was applied. It is a model based on the theory of reasoned action (TRA) [7] that allows to assess the user's technology acceptance behavior. The used sets of questions (see Table I) were adapted from [6], [8], and [9]. They measure the three main constructs perceived usefulness, ease of use, and self-predicted future use (see Fig. 3).
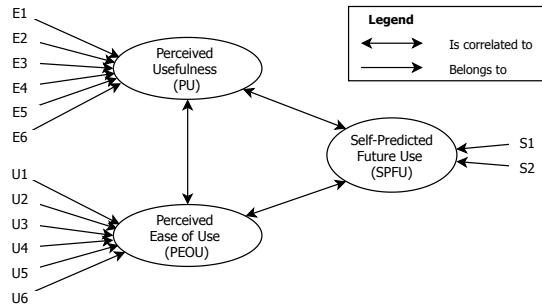
Fig. 3. Model of usefulness, ease of use, and self-predicted future usage (TAM). Abbreviations are defined in Table I.

A. Objective

The experiment was performed for two reasons. First, to evaluate the usefulness of a platform-independent and machine-readable experimentation definition language. Second, to evaluate the ease of use of the proposed experimentation definition language.

B. Variables

The following variables were considered in this study:

- *Perceived usefulness* is the degree users expect that the system will improve their job performance.
- *Perceived ease of use* is the degree users expect that the system will be free of effort.
- *Self-predicted future usage* is the degree users expect to use the system in the future.

C. Subjects

The subjects of the experiments were 30 graduate students of the University of Innsbruck (Austria) that were enrolled in a course on advanced concepts and techniques of software engineering. The course is part of the computer science as well as information systems master's (where some students did a

Perceived Usefulness	
U1	Using the software in my job would enable me to accomplish tasks more quickly.
U2	Using the software would improve my job performance.
U3	Using the software in my job would increase my productivity.
U4	Using the software would enhance my effectiveness on the job.
U5	Using the software would make it easier to do my job.
U6	I would find the software useful in my job.
Perceived Ease of Use	
E1	Learning to operate the software would be easy for me.
E2	I would find it easy to get the software to do what I want it to do.
E3	My interaction with the software would be clear and understandable.
E4	It was easy to become skillful using the software.
E5	It is easy to remember how to perform tasks using the software.
E6	I would find the software easy to use.
Self-predicted future usage	
S1	Assuming the software would be available on my job, I predict that I will use it on a regular basis in the future.
S2	I would prefer using the software to other forms for defining experiments.

bachelor in business administration) program. Thus, the students had a mixed background in computer science and business. Given that online controlled experimentation involves multiple stakeholders with different professional backgrounds, either more technical or more business-oriented, the setting seemed to be advantageous to represent potential users of the language. To further classify the subjects, some demographic questions about their prior knowledge on key technologies used by the experimentation definition language were asked (see Fig. 4). The results indicate that most participants (76%) mentioned that they know JSON. In contrast, only a quarter of all participants stated that they know the code editor Microsoft Visual Code.
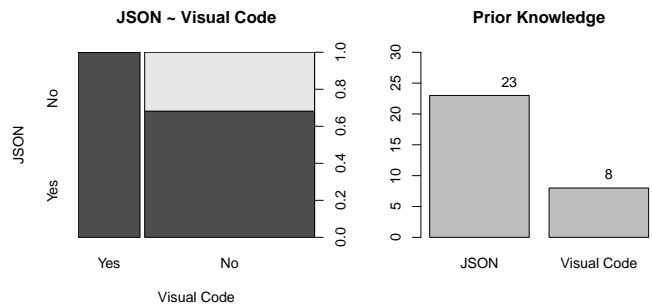
Fig. 4. Participants prior knowledge about the technologies JSON and Microsoft Visual Code.

D. Running the experiment

Given that the students already learned about experimentation in software engineering as part of the lecture, only a quick repetition of this topic was given. Thereafter, current approaches to experimentation (e.g. experimentation platforms, best practices) were discussed. After that, the development

and purpose of the experimentation definition language was presented. Finally, a short introduction of the tool (Microsoft Visual Code) was given. The language itself was introduced by some example experiment definitions. Next, all students received the same task to model an experiment in the language based on a written description of an experiment. Finally, the students submitted their written experiment definition and filled out a questionnaire containing demographic questions, the adapted TAM questions (see Table I), and a comment question.

E. Threats to validity

Potential threats of the study's validity [10] were considered and minimized whenever possible.

Construct validity was improved by considering the technology acceptance model [6] to evaluate the usefulness and ease of use of the experimentation definition language. Moreover, best practices and lessons learned from similar technology acceptance model studies (e.g. [8]) were considered.

Internal validity threats are caused by faulty conclusions that could for example happen because of mistakes in the statistical analysis. To mitigate this, the statistical analysis considers more than one measure (e.g. Pearson product correlation and Spearman rank-order correlation). Besides, the measurement instrument itself was analyzed regarding its reliability (Cronbach's alpha). Furthermore, the participants' demography was considered during the statistical analysis to draw more detailed conclusions of observations.

External validity covers to which extend the generalization of the results is justified. Given that experiments with students are usually considered to be of low external validity [11], the possibility to generalize the results of the study is limited. Nevertheless, this threat was slightly mitigated by considering graduate students from a course that addresses the subject online controlled experimentation and that have varying backgrounds (computer science, business) that reassembles teams in practice.

Reliability was improved by recording every step and decision of this study carefully and reporting the most important decisions as well as the reasoning behind them. Moreover, all questions of the study are reported.

IV. Results and discussion

A. Reliability

To test the reliability of the questionnaire, Cronbach's alpha is used commonly in empirical studies. Values above 0.8 are typically considered indicating the questionnaire as a highly reliable measurement instrument [12]. The analysis of the answers revealed a Cronbach's alpha of 0.967 for usefulness and 0.962 for ease of use. This indicates that our questionnaire is a reliable measurement instrument.

B. Factorial validity

In factor analysis it is examined whether the two factors ease of use and usefulness form distinct constructs (see Table II). The analysis clusters the variables together that tend to be

TABLE II
FACTORIAL ANALYSIS.

	Usefulness	Ease of use
Quick (U1)	0.92	0.03
Performance (U2)	0.96	-0.01
Productivity (U3)	0.92	-0.08
Effectiveness (U4)	0.91	0.03
Easy (U5)	0.91	-0.04
Useful (U6)	0.84	0.12
Easy to learn (E1)	-0.05	0.91
Easy to do (E2)	0.00	0.89
Clear (E3)	0.13	0.90
Skillful (E4)	0.24	0.81
Remember (E5)	-0.17	0.94
Easy to use (E6)	-0.03	0.95

correlated and assigns them a factor loading that describes the correlation of the variable to the factors. In Table II the results for all variables are given. Like correlations, the factor loading ranges from -1 (negative correlation) to +1 (positive correlation). A variable should have at least a factor loading of 0.7 to be a meaningful factor [13]. The results in Table II indicate that there are two factors on which the variables load. All variables associated with questions about usefulness tend to be loaded on the factor usefulness. The same can be observed for ease of use. The questions about ease of use tend to load on the factor ease of use. Thus, the result confirms that there are two factors ease of use and usefulness in the data.

C. Usefulness

In Fig. 5 the results for usefulness are shown. It presents a box plot for each usefulness variable by the Likert scale value that ranges from 1 (extremely unlikely) to 7 (extremely likely). Note that according to the suggestion of Laitenberger et al. [8] the middle option 4 (neither) was omitted because this answer option would not give any information about the direction a participant leans to. The summative results range between 21 and 42 (ignoring the two outliers with the sums 6 and 7) with a mean and median of about 33. Given that the maximum possible rating is 42, the results suggest that the participants consider the definition language useful. Figure 5 gives a more detailed picture of the ratings. It shows that across all six variables (U1 - U6) the median rating is between 5 and 6. There are also some outliers visible that show ratings below the Likert score value of 3 (slightly unlikely). A more detailed analysis of these answers revealed that two participants rated all variables with 1 (extremely unlikely) or 2 (quite unlikely). All other participants' ratings were 3 (slightly unlikely) or above. In the final comments question one of the two participants mentioned problems with understanding the task description. The other participants provide unfortunately no final comments. Given that their responses differ considerably from the majority of participants, the problem with the task description is not considered a possible threat to all answers.
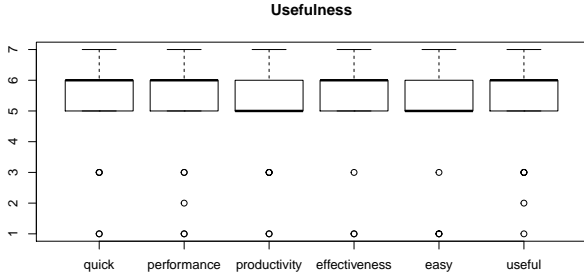
Fig. 5. Usefulness results.

D. Ease of use

The results for ease of use are summarized in Fig. 6. The summative results (excluding two outliers with a sum of 6 and 9) range between 15 and 42 (maximum score) with a median of 34 and a mean of 31. These findings suggest that the participants consider the presented solution easy to use. However, the widespread of the summative results indicate that the participants' opinion on this varies.
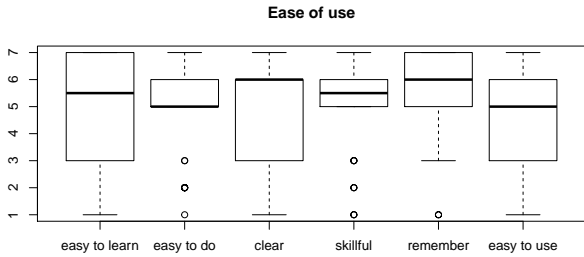
Fig. 6. Ease of use results.

The lowest rating were given easy to learn (E1), clear (E3), and easy to use (E6). Reasons for this could be the short introduction of the language by examples before to the experiment.

Nevertheless, the majority of participants (see Fig. 6) gave high ratings for the language to be easy to express experiments (E1), easy to become skillful in it (E4) and easy to remember how to perform tasks (E5). This seems to suggest that the domain model fits well with the domain concepts.

A more detailed analysis of the answers revealed that the ease of use ranking correlates with the participants' prior knowledge on the data interchange format JSON. Calculating the correlation coefficients (see Table III) revealed that there is a strong positive correlation between the participant's prior knowledge of JSON and the ease of use. Figure 7 shows the influence of the participants' prior knowledge of JSON on the ease of use rating. Participants with prior JSON knowledge rated every variable with a median of 6, whereas participants without prior knowledge of JSON rated with a median of 3. Also, the large spread of the ratings for participants without JSON knowledge could indicate that there is some unknown variable (maybe demographic) that could explain the data

TABLE III
PEARSON PRODUCT / SPEARMAN RANK ORDER CORRELATION
COEFFICIENTS OF THE RESPECTIVE SUMMATIVE RESULTS.

	Usefulness	Ease of use	Self-pred.	JSON
Usefulness	1.00 / 1.00	0.23 / 0.30	0.60 / 0.56	0.43 / 0.30
Ease of use	0.23 / 0.30	1.00 / 1.00	0.62 / 0.68	0.56 / 0.57
Self-pred.	0.60 / 0.56	0.62 / 0.68	1.00 / 1.00	0.75 / 0.67
JSON	0.43 / 0.30	0.56 / 0.57	0.75 / 0.67	1.00 / 1.00

more. Thus, it seems that the selection of the host language (in our case JSON) has a strong influence on the user's perceived ease of use of the developed DSL. In the case of JSON the researchers could observe during the experiment that the concept of the brackets was difficult to some participants, despite the provided auto-completion.
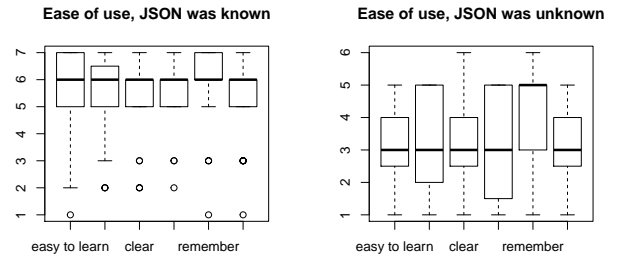
Fig. 7. Ease of use results by the participant's knowledge about JSON. The variables are from left to right: easy to learn (E1), easy to do (E2), clear (E3), skillful (E4), remember (E5) and easy to use (E6).

E. Correlations

The analysis of usefulness (see Section IV-C) and ease of use (see Section IV-D) showed that the participants seem to consider the language useful and most of them consider it easy to use too. Next we want to investigate the users' acceptance of the language by considering the correlations between the factors' usefulness, ease of use, and self-predicted future usage (see Fig. 3). In addition, we considered the influence of the participants' prior knowledge of JSON on those factors. Therefore, we want to investigate the correlation between the summative results of usefulness, ease of use, self-predicted future usage, and JSON knowledge. Table III shows the results of this analysis. It contains the Pearson product correlation coefficients together with the Spearman rank-order correlation coefficients between the summative results.

Usefulness is relatively low positively correlated with ease of use, which could mean that users that find the language useful not necessarily find it easy to use. This interpretation is supported by the observations made in Section IV-C and Section IV-D that although almost all find the language useful, not all participants find it easy to use.

Both factors usefulness and ease of use are similarly strong positively correlated with self-predicted future usage. One interpretation could be that both factors, the usefulness of the software and its ease of use are strong drivers for the participants to adapt the language. It seems that participants

TABLE IV
SIGNIFICANCE OF THE CORRELATIONS REPORTED BY THE *P*-VALUES
(PEARSON PRODUCT MOMENT CORRELATIONS / SPEARMAN RANK
ORDER CORELLATION COEFFICIENT). COMMON STAR NOTATION [15].

	U(sefulness)	E(ase of use)	S(elf-pred.)	J(SON)
U	-	ns/ns	***/**	*/ns
E	.202 / .102	-	***/***	**/***
S	<.001 / .001	<.001 / <.001	-	***/***
J	.015 / .105	.001 / <.001	<.001 / <.001	-

can find the language useful, but not easy to use (and vice versa) and still give a high rating for self-predicted future usage. Thus, usefulness could compensate faults in ease of use and vice versa. This would also explain the still high rating of self-predicted future usage, although some participants did not find the language easy to use.

An interesting fourth factor to include in the correlation analysis is the participants' prior knowledge of JSON. The analysis showed that there is a positive correlation between JSON knowledge, usefulness and ease of use. One explanation for this result could be that the usefulness of a language is more independent of the participants' prior knowledge of JSON than the ease of us. However, it also indicates the strong influence of a host language for an internal DSL project. The strong correlation between JSON and self-predicted future usage further supports this interpretation.

In addition to the calculation of the correlation coefficients, the significance of correlation among the four factors were analyzed (see Table IV). They indicate that the correlation between usefulness and ease of use is not significant, which is consistent with the results of King et al. [14]. Furthermore, all correlations to self-predicted future usage were statistically significant. Thus, it seems that these factors are indicators of the users' acceptance of the language. Moreover, it supports further the impact of the host language on the users' acceptance of an internal DSL.

As a result the analysis of the answers suggests that the participants consider the language useful and most of them easy to use too. Besides, most of the participants would use the language in the future and prefer it over other solutions.

F. Limitations

Even though possible threats to validity were considered during the design and execution of the study, the findings of this experiment have to be interpreted within their limitations. The main limitation of the study is the selection of students instead of practitioners as participants. This is in general considered to make results less generalizable. However, in the case of this experiment the particular students have considerable similarities with relevant industrial stakeholders, given their mixed background of computer science and business as well as their prior knowledge about experimentation in general and the domain of experimentation in particular.

V. CONCLUSIONS

For organizations that practice experimentation, the reliability of its experiments are of great importance, given the large potential impact their results can have. Therefore, the definition of an experiment as documentation and plan of action is an important artifact for reliable experimentation. Therefore, we proposed an experiment definition language based on recent research about the definition of experiments. Moreover, we conducted a technology acceptance study to validate the usefulness and ease of use of it. The results show that the majority of participants found the language useful and most found it easy to use too. Interesting future research directions are the development of a tool-chain for this language. These tools could apply static analysis on an experiment definition (e.g. answering whether for every variant a segmentation is properly defined), synchronize the definition with an experimentation platform, and even conduct some dynamic analysis like interference with other experiments on the experimentation platform.

REFERENCES

[1] R. Kohavi and R. Longbotham, "Online controlled experiments and a/b testing," *Encyclopedia of machine learning and data mining*, vol. 7, no. 8, pp. 922–929, 2017.

[2] F. Auer and M. Felderer, "Shifting quality assurance of machine learning algorithms to live systems," *Software Engineering und Software Management 2018*, 2018.

[3] M. Felderer, B. Russo, and F. Auer, "On testing data-intensive software systems," in *Security and Quality in Cyber-Physical Systems Engineering, With Forewords by Robert M. Lee and Tom Gilb*, S. Biffl, M. Eckhart, A. Lüder, and E. R. Weippl, Eds. Springer, 2019, pp. 129–148. [Online]. Available: https://doi.org/10.1007/978-3-030-25312-7_6

[4] A. Fabijan, P. Dmitriev, H. H. Olsson, and J. Bosch, "The online controlled experiment lifecycle," *IEEE Software*, 2018.

[5] F. Auer, C. S. Lee, and M. Felderer, "Characteristics of continuous experimentdefinitions: Results from a systematic literaturereview, a tool review and an expert survey," submitted to 46th EUROMICRO SEAA Conference, unpublished.

[6] F. D. Davis, R. P. Bagozzi, and P. R. Warshaw, "User acceptance of computer technology: a comparison of two theoretical models," *Management science*, vol. 35, no. 8, pp. 982–1003, 1989.

[7] M. Fishbein and I. Ajzen, "Belief, attitude, intention, and behavior: An introduction to theory and research," 1977.

[8] O. Laitenberger and H. M. Dreyer, "Evaluating the usefulness and the ease of use of a web-based inspection data collection tool," in *Proceedings Fifth International Software Metrics Symposium. Metrics (Cat. No. 98TB100262)*. IEEE, 1998, pp. 122–132.

[9] I. Steinmacher, T. Conte, C. Treude, and M. A. Gerosa, "Overcoming open source project entry barriers with a portal for newcomers," 05 2016, pp. 273–284.

[10] S. Easterbrook, J. Singer, M.-A. Storey, and D. Damian, "Selecting empirical methods for software engineering research," in *Guide to Advanced Empirical Software Engineering*. Springer London, 2008, pp. 285–311.

[11] D. Falessi, N. Juristo, C. Wohlin, B. Turhan, J. Münch, A. Jedlitschka, and M. Oivo, "Empirical software engineering experts on the use of students and professionals in experiments," *Empirical Software Engineering*, vol. 23, no. 1, pp. 452–489, 2018.

[12] E. G. Carmines and R. A. Zeller, *Reliability and validity assessment*. Sage publications, 1979, vol. 17.

[13] J.-O. Kim, O. Ahtola, P. E. Spector, C. W. Mueller *et al.*, *Introduction to factor analysis: What it is and how to do it*. Sage, 1978, no. 13.

[14] W. R. King and J. He, "A meta-analysis of the technology acceptance model," *Information & management*, vol. 43, no. 6, pp. 740–755, 2006.

[15] J. O. Berger and J. Mortera, "Interpreting the stars in precise hypothesis testing," *International Statistical Review/Revue Internationale de Statistique*, pp. 337–353, 1991.

Time-Aware Models for Software Effort Estimation

Michael Franklin Bosu[1], Stephen G. MacDonell[2], Peter Whigham[2]

[1]Centre for Information Technology, Waikato Institute of Technology, New Zealand

{stephen.macdonell, peter.whigham}@otago.ac.nz

[2]Department of Information Science, University of Otago, New Zealand

Abstract—**It seems logical to assert that the dynamic nature of software engineering practice would mean that software effort estimation (SEE) modelling should take into account project start and completion dates. That is, we should build models for future projects based only on data from completed projects; and we should prefer data from recent similar projects over data from older similar projects. Research in SEE modelling generally ignores these recommendations. In this study two different model development approaches that take project timing into account are applied to two publicly available datasets and the outcomes are compared to those drawn from three baseline (non-time-aware) models. Our results indicate: that it is feasible to build accurate effort estimation models using project timing information; that the models differ from those built without considering time, in terms of the parameters included and their weightings; and that there is no statistical significance difference as to which of the two model building approaches is superior in terms of accuracy.**

Keywords-empirical software engineering, software engineering decision support, software effort estimation, time-aware models

I. INTRODUCTION

Contemporary research efforts to address software effort estimation (SEE) typically develop and evaluate models using one, sometimes more, random split(s) of a secondary dataset of project observations into training and testing sets. Models are built using the training set and model accuracy is assessed on the testing set. In practice, however, organizations accumulate data over time as projects are worked on and are (hopefully) completed. It could be expected, then, that this accumulating data set would be the 'training set', used to build models to estimate the effort of *future* projects as each new project is proposed. Thus we have a disconnect between research and practice. Most effort estimation models developed by the research community disregard project start and/or completion dates [1]; as a result, data from 'future' projects can be used to build predictive models of effort for projects that occurred before them in time. To some extent this may be due to the absence of the necessary time-oriented features in the datasets [1]; however, even for datasets that include timing information, this is widely ignored in SEE research as only two (ISBSG and Finnish datasets) of the six datasets in the public domain with timing information have so far been used in developing software effort estimation models that considers time. To the best of our knowledge, this is the first study to develop time-aware effort estimation models using the NASA93 and Desharnais datasets. In these datasets, the completion dates of projects represent the timing information. This study therefore explicitly considers the year of project completion and uses only data from completed projects to develop models to estimate the effort of projects completed in subsequent years. Two time-aware approaches; Time-Aware Sequential Accumulation (TASA) and Time-Aware Moving Window (TAMW) are used in model development (see section III for details).

The performance of these time-aware models are then assessed in an absolute sense and in a relative sense against three baseline 'models' – leave-one-out, mean and median.

To the best of our knowledge, this study differs from all previous effort estimation time-aware studies as this study applies the TAMW approach and considers the stability of the models. Our research questions are expressed as follows:

RQ1: Is it feasible to develop accurate effort estimation models using project completion dates?

RQ2: Are the parameters and coefficients of time-aware models stable or volatile?

RQ3: Which of the two time-aware modelling approaches, if either, is superior in terms of accuracy?

The rest of this paper is organized as follows. Section II presents the related work, our research method is presented in section III, in section IV we present our results, section V reports threats to the validity of our study, and section VI comprises a discussion and draws conclusions.

II. RELATED WORK

Though numerous SEE models have been proposed (see [2]) the number of studies that have considered project timing information in effort estimation is negligible and attributed to very few researchers. This section summarizes the few studies that are related to this research.

Lokan and Mendes [3] applied a moving window of the most recently completed projects to new projects in their effort estimation studies. Their results indicated that the use of a moving window of the most recently completed projects contributed significantly to the accuracy of models. In a recent study, Amasaki and Lokan [4] proposed a method that is able to select whether to build a model based on time or to use the growing portfolio of projects. MacDonell and Shepperd [5] applied two timing methods – sequential accumulation of project data over time and constant moving window of size 5 – on a proprietary dataset and obtained improved results over managers' estimates, especially for the moving window approach compared to a LOO approach.

This paper applies the two approaches used by MacDonell and Shepperd [5] to two publicly available datasets, except that the moving window approach, presented in the next section, is dynamic as compared to the fixed window size used in [5] and in earlier similar studies.

III. RESEARCH METHOD

Data Grouping

For each of the two datasets used in this study, an attempt is first made to work with the entire dataset before consideration is given to splitting the data into homogeneous subsets with a view to developing models for each partition. The division of datasets into homogeneous subsets is intended to enable us to identify whether specific partitions of the data exhibit trends that are different from those evident for other partitions, or across the entire dataset. Partitions are typically based on factors such as the type of application, the application domain of the project, and/or the unit or department responsible for development.

Partitions such as these are formed by relying on the visualization of boxplots and the use of Mann-Whitney tests to assess whether observations belong to the same distribution. In this study, data that fell outside the boxplot whiskers of distributions were considered as outliers and were not used in

model building. A significance level of 0.05 is used for the Mann-Whitney tests, so groupings that have a p-value greater than 0.05 are taken to belong to the same distribution. Use of these partitions will ensure that models are developed for datasets that as far as possible share similar characteristics.

A. Datasets

NASA93 Dataset

This dataset was collected by NASA and it comprises 93 projects undertaken between 1971 and 1987 (as downloaded from the PROMISE Repository http://openscience.us/repo/). The dataset is structured according to the Constructive Cost Model (COCOMO81) developed by Barry Boehm [6]. It comprises 24 attributes of which 15 are the mandatory effort multipliers.

Preliminary analysis indicated that, due to the diversity of the NASA93 projects, it was neither feasible nor sensible to build time-aware models for the entire dataset, and as such the dataset was split into four subsets. These four subsets are: NASA82, comprising projects developed in 1982 and beyond; Center 2 (C2) and Center 5 (C5) subsets, comprising projects developed at NASA's Center 2 and Center 5, respectively; and Semidetached (SD), which includes projects of the semidetached development mode. Due to space limitations the boxplots are not shown, but they can be found at this link[1]. In addition to outliers being evident in the boxplots, three other projects with atypical characteristics were also not used – two projects with size values greater than their effort values, and a project with a productivity rate (i.e., effort divided by size) more than twice as high as that for the project with the next highest productivity rate, and almost eleven times the mean productivity rate.

Desharnais Dataset

The Desharnais dataset was collected by Jean-Marc Desharnais from ten organizations in Canada [7]. The projects in this dataset were undertaken between 1983 and 1988. The dataset consists of 81 records and twelve attributes, including size measured in function points and effort measured in person- hours. We used the version comprising of 77 projects as has been done by most studies that used this dataset because there are four missing records in the original Desharnais dataset. The Desharnais dataset, like the NASA93 dataset, contains only the year of project completion, and as such the training and test sets were formed in the same way as for the NASA93 dataset (i.e., by using the year of project completion).

According to Mann-Whitney analysis and associated boxplots, the Desharnais dataset forms a single distribution. Models were therefore built for the entire dataset along with a subset which also forms a single distribution developed using a programming language termed 'Advanced Cobol' (herein referred to as the Adv.Cobol dataset). This subset is made up of 23 projects and is identified in the Desharnais dataset as "category 2" under the language attribute.

B. Effort Estimation Model Development

In software effort estimation modelling (as in other fields) the dataset is usually split into two, forming a training set and a test set. The training set is used to develop the model and the performance of that model is then evaluated on the test set. This study follows a similar approach (see Analysis 1 and Analysis 2 in this section for model development algorithms). All models in this study are developed using the statistical package R, v.3.5.2.

All models are developed using linear regression which has enjoyed widespread use in software effort estimation studies. In

order to accommodate the diverse nature of the two datasets being used in this study, especially in regard to the number of variables, specific linear regression models are applied to each dataset (or partition) as described in the respective datasets section.

It should also be noted that the models developed in this study are all well-formed models. That is, the degrees of freedom are considered whereby a training set is formed only when the number of projects is at least two plus the number of explanatory variables being used for model construction. Maxwell's proposal [8] to identify influential observations using Cook's distance during model building was also adopted for this study.

NASA93 Models

In estimating effort for projects completed in a given year, equation 1, the COCOMO81 equation for effort estimation, is used for all four partitions of the NASA93 dataset.

$$effort(personmonths) = a * (KLOC)^b * (\textstyle\prod EM_j)\dots(1)$$

In order to develop a regression model, as in other COCOMO81 effort estimation studies [6] [9], equation (1) is linearized by logarithmic transformation, as indicated in equation (2).

$$\ln(effort) = \ln(a) + b * \ln(KLOC) + \ln(EM_1) + \cdots (2)$$

Backward stepwise regression is applied in order to support the inclusion or exclusion of variables, as previous studies have established that not all the effort multipliers of the NASA93 COCOMO81 format dataset are influential in model building [9].

Desharnais Models

Desharnais himself [7] identified the size and language attributes as those that are influential in a regression model. Kitchenham and Mendes [10] supported Desharnais' claim by proposing the use of the language attribute as a dummy variable. This approach has been adopted here for the models developed for this dataset, as shown in equation (3).

$$\ln(effort) = \ln(size) + language\dots\dots(3)$$

This study used the adjusted function point value as the most complete size attribute and treated the three-value language attribute as a dummy variable, with the reference dummy value being the Basic Cobol projects indicated as "1" in the Desharnais dataset. The smaller Adv.Cobol dataset only uses size as an explanatory variable in model development.

C. Analysis Procedure

The following procedures are applied to all datasets modelled in this study.

Analysis 1: Time-Aware Sequential Accumulation (TASA)

1. For each dataset with timing information, select the first year in which projects were completed as the training set – if the first year of projects comprises fewer than the number of observations needed to build a well-formed model, add the next year(s) of projects, until the minimum requirement for a well-formed model is satisfied. The subsequent year of projects is then used as the test set.
2. Check for normality (using the Shapiro- Wilk test of normality) in the distributions of the training data– if data follow a normal distribution go to step 3 else step 2.1
2.1 Apply the appropriate transformation to make the data normal and recheck normality for verification as in step 2 above.
3. Build a regression model using the training data (where the form of the regression model will be specific to each dataset).
4. Apply the model obtained in step 3 to predict the effort values in the test set.

---

[1] http://tinyurl.com/SEKE2020-TIME

5. Calculate the accuracy measures (see below) for the formulae.
6. Add the test year's data to the training set; the subsequent year's data becomes the new test set.
7. Repeat steps 2 to 6 through to the estimation of the last year of projects.

Analysis 2: Time-Aware Moving Window (TAMW)
This algorithm applies a moving window to the dataset used in Analysis 1 thus accounting for the longevity of the projects in the training set.
1. For each dataset used in Analysis 1, drop the oldest year's projects.
2. The 'new' oldest year's projects now become the first year of projects; apply step 1 of Analysis 1.
3. Apply steps 2-6 from Analysis 1.
4. Repeat steps 3 to step 6 of Analysis 1 until the training set comprises projects from all years except the last year of projects.
5. Remove the oldest year's projects from the training set.
6. Repeat steps 1 to 5 until there is only one year of projects in the training set or until there is not enough data in the training set to build a well-formed model.

Baseline Models
Three baseline models are developed for each dataset/subset used in this study and their performance is compared with that of the time-aware models. The baseline models are a leave-one-out holdout (LOO – note that the 'one' in this case refers to all projects in one year rather than a single project), the mean and the median of the training set data. The mean and median effort values are calculated over the training data and become the effort estimates for the projects in the test set.

D. Measures of Accuracy

Accuracy measures used to evaluate the performance of the effort prediction models are relative error, mean squared error and total absolute error. Note that in all three cases lower values are preferable.

***Relative Error (RE)* -** The relative error is computed using the following equation:

$$RE = \text{variance(residuals)/variance(measured)},\text{where}$$

measured is the test data. The relative error measure accounts for the variability in data and as such it is robust to outlier data points.

***Mean Squared Error (MSE)* -** *MSE* is defined as:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(actual - estimate)^2$$

where *n* is the total number of test data points, *actual* is the recorded effort used in developing the project and *estimate* is the effort predicted by the model. The MSE measures the general quality of the prediction model across all data points and accounts for projects of varying size. It can be susceptible to outliers; however, if a data set is largely free of outliers it can provide a useful indication of a model's overall accuracy.

***Total Absolute Error (TAE)* -** *TAE* is defined as:

$$TAE = \sum_{i=1}^{n}|(actual - estimate)|$$

IV. RESULTS

The results of applying the modelling approaches to the two datasets and their partitions are now presented. Due to space constraints we include only some of the results – the complete set of results may be found at the link specified previously.

NASA93 Dataset
It is evident from Table I that the accuracy measures are themselves not consistent in terms of model performance. That aside, it does seem to be feasible to build time-aware models for this dataset based on projects completion dates, as the worst model performance recorded (excluding the models with the large prediction errors) in terms of relative error is 0.26 which is quite satisfactory. Also, in just two instances the median baseline results are better than the time-aware models; in all other cases the models are better than both the mean and median baseline results.

TABLE I. NASA93 EFFORT ESTIMATION RESULTS

	Year	Time-Aware Sequential Accumulation			Time-Aware Moving Window		
		RE	MSE	TAE	RE	MSE	TAE/AE
NASA82	1985	0.06	2136.52	243	-	-	-
	1986	1717.8	3.4E+07	18151	-	-	-
	1987	0.19	61.87	15	0.11	113.38	21
C2	1987	0.26	73.12	16	0.26	73.12	16
C5	1983	-	-	302	-	-	302
	1984	0.12	6256.95	278	0.12	6256.94	278
	1985	-	-	12	-	-	8
	1985*	-	-	12	-	-	2
SD	1984	7.5E+05	6.1E+09	174344	-	-	-
	1985	0.02	199.98	64	-	-	-
	1986	849.93	1.7E+07	13996	1.6	8353.57	926
	1986*	849.93	1.7E+07	13996	2738	5.4E+07	22513
	1987	0.19	75.92	17	0.2	72.17	16
	1987*	0.19	75.92	17	0.19	85.47	17

* number of additional TAMW models built for that particular year
'-' indicates no computation of a result for a specific accuracy measure

The LOO baseline results, however, is better than all the models developed for this dataset (see previous link). The highlighted results in Table I indicate large prediction errors. Manual inspection of the NASA82 and Semidetached (SD) datasets revealed that the effort multipliers of the training projects were quite different from those of the projects being estimated.

To formally gauge whether one of the time-aware models resulted in more accurate effort predictions, a two-tailed paired samples Wilcoxon test was applied. The p-value results are 1 (due mainly to the ties), 0.5839 and 0.5839 for RE, MSE and TAE, respectively. This indicates that the differences in prediction accuracy for the two models are not statistically significant. Therefore, for this dataset, we conclude that either time-aware approach could be used to estimate effort. The two time-aware models consistently included size as an explanatory variable. Beyond that, however, both the variables included in the effort estimation models and their coefficients were quite dynamic, as the models differed from one time period to another (see previous link). There was no consistent pattern as to a decrease or increase in the values of the coefficients of both model types developed for the NASA93 dataset. All the predictive models developed for the NASA93 datasets can be termed as sufficiently accurate as the Adjusted R^2 values fell between 0.89 and 0.98 (see previous link).

Desharnais Dataset
It is evident from Table II that it is again feasible to build time-aware models for this dataset using projects completion dates, with some of the results in terms of RE reaching 0.01. The corresponding TAE results are equally satisfactory. Though the worst result, for 1986 at 40727 hours, might appear large, it equates to an average of 36 weeks per project since 28 projects were completed in 1986. There are four instances where the model results are better than their corresponding LOO(available at the previous link) baseline models (2 for MSE and 2 for RE). A two-tailed paired samples Wilcoxon test was applied to determine the superior modelling method. The p-value results are 0.6698, 0.5566 and 1 for RE, MSE and TAE, respectively, indicating that the two models are not significantly different. Therefore for this dataset, either of the time-aware approaches could be used to

develop effort estimation models.

The models' explanatory variables and coefficients are consistent, as shown in Table III (TASA model) and Table VI (TAMW model). All of the models built have Adjusted R^2 values of between 0.60 and 0.88 and as such could be termed as reasonably accurate models.

TABLE II. DESHARNAIS EFFORT ESTIMATION RESULTS

	Year	Time-Aware Sequential Accumulation			Time-Aware Moving Window		
		RE	MSE	TAE	RE	MSE	TAE/AE
Desharnais	1986	0.65	4953913.7	39911	0.67	4964415.7	40727
	1987	0.71	837535.2	7267	0.77	914156.4	7473
	1987*	0.71	837535.2	7267	0.69	816717.2	7153
	1987**	0.71	837535.2	7267	0.69	846571.1	8348
	1988	0.02	196984.4	1326	0.02	153573.1	1182
	1988*	0.02	196984.4	1326	0.01	134684.7	1293
	1988**	0.02	196984.4	1326	0.01	102903.6	1394
	1988***	0.02	196984.4	1326	0.15	1437442	3765
Adv. Cobol	1987	0.13	1393102.6	6515	0.13	1372071	6388
	1987*	0.13	1393102.6	6515	0.13	1364820	6251
	1988	-	-	1205	-	-	1015
	1988*	-	-	1205	-	-	1102
	1988**	-	-	1205	-	-	525

TABLE III. COEFFICIENTS OF TIME-AWARE SEQUENTIAL CCUMULATION MODELS - DESHARNAIS DATASET

Dataset	Year	Intercept	Size	Lang2	Lang3	Adj.R^2
Desharnais	1986	5.65	0.50	-0.50	-1.66	0.68
	1987	3.78	0.82	-0.04	-1.49	0.74
	1988	3.89	0.80	-0.04	-1.44	0.74
Adv. Cobol	1987	2.66	1.03			0.84
	1988	2.62	1.04			0.83

TABLE IV. COEFFICIENTS OF TIME-AWARE MOVING WINDOW MODELS - DESHARNAIS DATASET

Dataset	Year	Intercept	Size	Lang2	Lang3	Adj.R^2
Desharnais	1986	5.65	0.51	-0.55	-1.71	0.71
	1987	3.67	0.85	-0.05	-1.50	0.76
	1988	3.81	0.82	-0.05	-1.45	0.75
	1987*	3.59	0.85	0.001	-1.37	0.74
	1988*	3.78	0.82	-0.002	-1.35	0.74
	1987**	2.91	0.96	0.17	-1.12	0.88
	1988**	3.65	0.83	0.10	-1.24	0.85
	1988***	4.76	0.62	-0.007	-1.06	0.60
Adv. Cobol	1987	2.74	1.02			0.84
	1988	2.96	0.98			0.83
	1987*	3.01	0.98			0.84
	1988*	3.37	0.92			0.86
	1988**	3.32	0.92			0.75

V. THREATS TO VALIDITY

The first threat to the validity of this study is the generalization of our results, as the datasets used are convenience sampled from the PROMISE repository. Though these datasets cannot be representative of the entire software industry they have become benchmarks datasets in software effort estimation research. The age of the datasets might also raise concern, however, these datasets are still increasingly being used in recent software effort estimation studies. Another threat to validity is due to the bias that could be introduced by considering only the completion dates, however, we had little choice as these datasets only have completion dates.

VI. DISCUSSION AND CONCLUSIONS

The results presented for the two datasets examined here indicate that *it is feasible to develop accurate effort estimation models that are also time-aware based on projects completion dates*, positively answering RQ1. In most instances, the performance of the models developed for the NASA93 dataset was acceptable, with Adjusted R^2 between 0.89 and 0.98 except for the large errors shown in Table I.

The Adjusted R^2 for the models built for the Desharnais in this study all exceeded 0.60 (better than the models built by Desharnais himself [7] which had Adjusted R^2 of 0.54), most were greater than 0.70, and the highest Adjusted R^2 was 0.88. These results suggest that performance improvements can potentially be gained by building effort estimation models that are time-aware. The results of this study also support Amasaki and Lokan [3] notion that it is not in all cases that time-aware models are superior. In the case of the NASA93 dataset, the LOO baseline was in fact superior to all the time aware models whilst for the Desharnais dataset, the result was mixed as the time-aware models were superior to the LOO baseline in some cases and vice-versa.

Our results regarding model stability were mixed. The variables and coefficient values for the Desharnais dataset models were generally stable, in sharp contrast to our results for the NASA93 models. The dynamic nature of the NASA93 models can be attributed to the greater heterogeneity in the NASA93 dataset – it consists of 14 different application types, developed for 5 different NASA centers, principally by a number of external vendors who may themselves have had varied development practices. The relative stability of the models built for the Desharnais dataset is somewhat surprising because this dataset was collected from ten different organizations in Canada over a period of 6 years. However, the project types and development languages used were few. This implies that it is possible that organizations working at the same time on similar projects may well have similar practices, and as such, models that are built to characterize their practice may be more homogeneous than heterogeneous. Thus, in relation to RQ2 we must conclude that *the stability of the parameters and coefficients of time-aware models largely depend on the diversity of the dataset.*

In terms of answering RQ3 as to *which of the two time-aware modelling approaches, if either, is superior in terms of accuracy*, the Wilcoxon tests indicate that there is no significant difference in performance for either time-aware modelling approach. Our results therefore indicate that, for these two datasets, neither method is superior, and so either approach may be used to create sufficiently accurate time-aware models.

REFERENCES

[1] M.F. Bosu and S.G. MacDonell, "A taxonomy of data quality challenges in empirical software engineering," 22nd Austral. Softw. Eng. Conf., ASWEC13, pp.97-106, 2013.

[2] M. Jorgensen and M. Shepperd, "A systematic review of software development cost estimation studies," IEEE Trans. Softw. Eng., vol. 33, no. 1, pp. 33–53, Jan. 2007.

[3] C. Lokan and E. Mendes, "Applying Moving Windows to Software Effort Estimation," Third Int. Symp. Empir. Softw. Eng. Meas., ESEM09, pp. 111–122, 2009.

[4] S. Amasaki, and C. Lokan, "An Evaluation of Selection Methods for Time-Aware Effort Estimation." In 24th Asia-Pacific Software Engineering Conference, pp. 624-629, 2017.

[5] S. G. MacDonell and M. Shepperd, "Data Accumulation and Software Effort Prediction," Proc. 2010 ACM-IEEE Int. Symp. Empir. Softw. Eng. Meas., ESEM10, pp. 31–34, 2010.

[6] B. W. Boehm, "Software Engineering Economics," IEEE Transactions on Software Engineering, vol. SE-10, no. 1. Prentice-Hall, Englewood Cliffs, NJ, Jan-1981.

[7] J.-M. Desharnais, "Statistical Analysis on the Productivity of Data Processing with Development Projects using the Function Point Technique," Université du Québec à Montréal., 1988.

[8] K. Maxwell, Applied Statistics for Software Managers. Englewood Cliffs, NJ,: Prentice-Hall, 2002.

[9] B. K. Singh, S. Tiwari, K. K. Mishra, and a. K. Misra, "Tuning of Cost Drivers by Significance Occurrences and Their Calibration with Novel Software Effort Estimation Method," Adv. Softw. Eng., vol. 2013, no. 1, pp. 1–10, 2013.

[10] B. A. Kitchenham and E. Mendes, "Why Comparative Effort Prediction Studies may be Invalid," Proc. 5th Int. Conf. Predict. Model. Softw. Eng., 2009.

An Empirical Study on Issue Knowledge Transfer from Python to R for Machine Learning Software

Wenchin Huang[1,2], Zhenlan Ji[3], Yanhui Li[1,2,*]

1. State Key Laboratory for Novel Software Technology, Nanjing University, China
2. Department of Computer Science and Technology, Nanjing University, China
3. School of Management and Engineering, Nanjing University, China
*Corresponding author: yanhuili@nju.edu.cn

Abstract—Background: With the blowout of programming languages, developers employ different languages to solve similar problems (e.g., to implement machine learning algorithms) separately and frequently, which gives rise to knowledge transfer across different language development. Since GitHub provides an issue tracking system for developers and users to follow with issues, knowledge about how to deal with issues is a main part of available knowledge on GitHub. Such issue knowledge could be directly transferred to help developers handle new issues on current projects from similar projects in different languages.

Aims: Inspired by a large amount of developed and developing machine learning software written in Python and R on GitHub, we aim to discover how much issue knowledge can be transferred from Python projects to R projects.

Method: We investigate totally 1161 issues from 15 popular machine learning projects in R and 7496 issues from Scikit-Learn in Python on GitHub. After computing the text similarity between issues from R and Python projects, we match top 5 similar Scikit-Learn issues for each R issue and manually judge 1161×5 issue-pairs to label and group them.

Results: We observe that a) 13% (149/1161) of R issues can refer to related Python issues; b) 47% (71/149) of related R issues can be linked to Python issues by the text mining technique BM25 at the very early stage; c) 83% (124/149) of related Python issues support code and description about the similar machines learning problems; d) reference knowledge is considered as the most useful knowledge from Python issues.

Conclusion: We put forward the following suggestions: a) referring to the corresponding cross languages issues is an efficient way for developers, especially there is the lack of related information in current language; b) the text mining technique BM25 is helpful for developers to start earlier for searching similar issues cross languages.

I. Introduction

With the blowout of programming languages and the widespread use of GitHub, software development has evolved from a single language development to socio-technical ecosystems, within which developers from different language communities solve similar problems separately and frequently [1]. Knowledge gained across different language development assists developers to find out how different languages implement the same requirements, which greatly contributes to knowledge acquisition about current requirements, and subsequently to decisions about design, coding, test and maintenance [2].
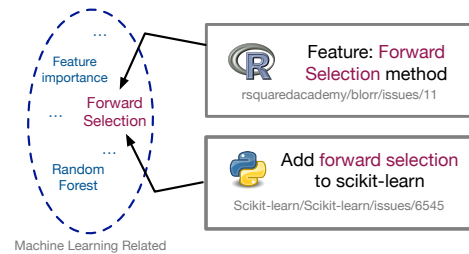
Fig. 1. Two issues with similar topics from R and Python projects

Machine learning software is a category of libraries to implement machine learning algorithms that allows users to accurately predict outcomes without explicit programming (e.g., Scikit-Learn[1]), which consists of algorithm implementations with *similar topics* and *different languages*: (a) machine learning algorithms (e.g., Random Forest) have clear specifications of the functionality regardless of language implementation [3]; (b) for most of machine learning algorithms, there are existing libraries written in popular languages (e.g., Python) with reliable performance for predictive modeling [4]; (c) developers from a different language community (e.g., R) may face the same or very similar requirements, and consequently develop libraries to implement the same or very similar functionality of algorithms in new languages [5]. During the new language implementation of algorithms, knowledge transferred (e.g., from Python to R) would be useful to accelerate current software development.

Issue is a critical way for software projects to track the progress of problems reported by developers and users during developing, maintaining, or using, which could be a bug report, a code document, a feature enhancement request, a task and so on [6], [7]. Since GitHub provides an issue tracking system for developers and users to handle issues, knowledge about how to deal with issues is a main part of available knowledge extracted from current projects on GitHub, which can be transferred into the development of the similar projects in different languages. Here we illustrate an example for a pair of issues sharing the similar topics from the Python project and the R project in Figure 1. From the topics of the issue

---

[1]Scikit-Learn is one of the most famous machine learning software in Python. http://scikit-learn.github.io.

#6545 in Scikit-learn/Scikit-learn and the other issue #11 in rsquaredacademy/blorr, we can observe that they both aim to implement the specific algorithm of "forward selection". Obviously, when dealing with the new issue in R, developers can gain knowledge from the old issue in Python.

Inspired by a large amount of developed and developing machine learning software written in Python and R on GitHub, we aim to discover how much issue knowledge can be transferred from Python to R. In this paper, we select 15 popular R machine learning repositories and Scikit-Learn in Python to extract their issue contents. By computing the text similarity between 1161 R issues and 7496 Python issues, we manually browse and label the issue-pairs (i.e., the older issue in Python and the new one in R) to check similarity with a top-5 similarity list. Finally, we extract 149 related issue-pairs for further research.

To examine how knowledge can be transferred from Python issues to R issues, we structure our study by addressing the following three research questions (RQs):

RQ1 (Linking issues from R to Python): how can developers link to the related Python issues when R issues are just created? By experimenting three strategies to search related Python issues at the very early stages of R issues, we observe that BM25 search performs the best, which successfully searches over 47% corresponding Python issues for the R issues by only using the initial information (e.g., title and body) from issues.

RQ2 (Types of Knowledge): what kinds of knowledge can developers learn from related Python issues? We observe that related Python issues offer 6 types of knowledge for those R issues: description, reference, outer link, related issue, code and code document. Among them, the most popular kinds of knowledge are description and code.

RQ3 (Helpfulness of Knowledge): which type of knowledge from Python issues is most helpful? Based on the comparison of spearman correspondence and weighted mean helpfulness, we observe that reference knowledge type turns out to be the most helpful knowledge.

Our study makes the following contributions.

- Dimension. This study opens a new dimension in knowledge transfer from Python to R in cross-language software development.
- Study. This study includes an empirical study of cross language knowledge transfer on 1161 issues from 15 repositories in R and 7496 issues from Scikit-Learn in Python.
- Strategy. This paper puts forward BM25 search as an efficient strategy to search for corresponding issues at the early stage for cross language knowledge transfer.

The rest of this paper is organized as follows. Section II describes our research methodology. Section III, Section IV, and Section V present the results of three RQs above. Threats to validity is discussed in Section VI. Finally, the conclusion and future work is put forward in Section VII.

II. Our Approach

In this study, we collect issues from 15 R repositories as the newer language repositories and the famous Python repository Scikit-Learn as the older language repository. Our approach comprises three steps: first we collect the issues from these repositories on the GitHub; after that, we compute the similarity between issue-pairs by using the word embedding technique; finally, we extract the related issue-pairs by manually judging their relationship.

A. Issues Collecting

We choose one of the most representative machine learning packages Scikit-Learn as our studied repository in Python. Besides, we take 15 open sources R packages into consideration, which are all involved in machine learning and highly recommended. We choose these R repositories according to the following four principles:

- they implement widely used algorithms, such as Random Forest, Naive Bayes and K-Nearest Neighbors.
- they are open source repositories on GitHub.
- they are listed on CRAN[2].
- the time of first commit in these repositories should be later than the time of first commit in Scikit-Learn.

Table I illustrates the detail of 15 R repositories we study. The first column shows the names of R repositories. The URL links to the repository are listed in the second column. The numbers of issues and the first commit time are listed in the third and fourth columns, while the algorithms implemented are listed at the last. Though there are some repositories with little stars or forks, even little issues and pulls, we still choose them for the following two reasons: (a) to ensure the variousness of algorithms, (b) to observe issue knowledge transfer for these repositories from the very beginning.

To sum up, we collect 1161 issues from the R repositories, including both open and closed issues, and 7496 issues from Scikit-Learn.

B. Similarity Computing

After choosing the repositories and collecting the issues, we compute the text similarities between 1161 issues from R and 7496 issues from Scikit-Learn by employing the word embedding dataset offered and pre-trained by Google [8]. It is worthwhile pointing out that, we use the **whole issue content**, including topic, question and discussion in each issue for similarity computing. We only consider the natural language text, and exclude the other parts, e.g., code. For issue pairs $\langle \mathcal{I}_P, \mathcal{I}_R \rangle$ from **P**ython and **R** correspondingly, we consider $\mathcal{I}_P$ and $\mathcal{I}_R$ as the sets of words appearing in them, and calculate the similarity of them as follows.

(a) Given two words w_P and w_R appearing in $\mathcal{I}_P$ and $\mathcal{I}_R$, their semantic similarity is defined as the cosine similarity by using their word embeddings:

$$sim\left(w_P, w_R\right) = \frac{w_P^T w_R}{\|w_P\| \|w_R\|}$$

---

[2]CRAN is the most popular online repository that store up-to-date versions of R packages, including code and documentation. https://cran.r-project.org

R repository	URL	#Issues	First Commit	#Watch/#Star/#Fork	Classifier(Algorithms)
Arborist	https://github.com/suiji/Arborist	42	31 Jan 2013	15/68/12	RF
benchm-ml	https://github.com/szilard/benchm-ml	56	28 Mar 2015	155/1734/326	LR/SVM/RF/boosting/...
bigrf	https://github.com/aloysius-lim/bigrf	20	15 Feb 2013	11/91/26	RF
blorr	https://github.com/rsquaredacademy/blorr	71	15 May 2017	2/9/1	LR
classyfire	https://github.com/eaHat/classyfire	17	11 Jul 2014	3/8/0	SVM
edarf	https://github.com/zmjones/edarf	57	4 Sep 2014	12/61/10	RF
forestFloor	https://github.com/sorhawell/forestFloor	33	5 Jul 2015	5/36/7	RF
ggRandomForests	https://github.com/ehrlinger/ggRandomForests	32	4 Jan 2013	8/107/23	RF
grf	https://github.com/grf-labs/grf	332	28 Jul 2014	41/359/99	RF
kknn	https://github.com/KlausVigo/kknn	17	20 Apr 2015	3/15/5	KNN
lumberjack	https://github.com/neurodata/lumberjack	88	15 Feb 2017	9/54/35	RF
naivebayes	https://github.com/majkamichal/naivebayes	5	3 Jun 2017	2/14/4	NB
randomForestSRC	https://github.com/kogalur/randomForestSRC	22	18 Nov 2016	8/48/8	RF
ranger	https://github.com/imbs-hl/ranger	366	28 Jul 2014	42/507/114	RF
TFG	https://github.com/Dani-Basta/TFG	3	31 Oct 2017	5/3/0	KNN
TOTAL		1161			

which is calculated by the Euclidean norm of their vectors using inner product.

(b) In order to compute the similarity between the issues, we introduce the similarity calculation approach proposed by Ye et al. [9], which modified the text-to-text similarity [10]. To calculate the similarity between a word w and the whole context of the issue $\mathcal{I}$, we compute the maximum similarity between w and w' in $\mathcal{I}$:

$$sim\,(w, \mathcal{I}) = \max_{w' \in \mathcal{I}}\{sim\,(w, w')\}$$

(c) Both the words with no word embedding and the words not appearing in the target issues $\mathcal{I}^*$ are ignored in the following calculation. The asymmetric similarity from $\mathcal{I}$ to $\mathcal{I}^*$ can be computed as:

$$sim\,(\mathcal{I} \to \mathcal{I}^*) = \frac{\sum_{w \in P(\mathcal{I} \to \mathcal{I}^*)} sim\,(w, \mathcal{I}^*)}{|P\,(\mathcal{I} \to \mathcal{I}^*)|}$$

where $P(\mathcal{I} \to \mathcal{I}^*) = \{w \in \mathcal{I}|sim(w, \mathcal{I}^*) \neq 0\}$.

(d) The final symmetric similarity $sim(\mathcal{I}_P, \mathcal{I}_R)$ between two issues $\mathcal{I}_P$ and $\mathcal{I}_R$ can be computed as the sum of two asymmetric similarity.

$$sim\,(\mathcal{I}_P, \mathcal{I}_R) = sim\,(\mathcal{I}_P \to \mathcal{I}_R) + sim\,(\mathcal{I}_R \to \mathcal{I}_P)$$

For each R issue $\mathcal{I}_R$, we rank the issue pairs $\langle \mathcal{I}_P, \mathcal{I}_R \rangle$ from large similarity values to small ones in order to find the most related Python issue from Scikit-Learn. We remain the top 5 of the most similar issue pairs (totally 1161×5 issue-pairs), which will be filtered to pick out the real related pairs by the following manual check.

C. Manual Judgement

1) Preprocessing Candidate Issue-Pairs: Among these 1161×5 issue-pairs, some of the issues describe the issue confusedly, some of them are simply notes for recording the updating of repositories, and some of them are just discussing the details in the code (e.g., XX lines in Y documents). These issues are not included in our following consideration.

Before manual judgement, we divide the issue context into two parts, the question part and the discussion part for the following description. The **question part** contains the topic and the body of the issues submitted by issue reporter on the top of the issue content. The **discussion part** is the content followed by the question part. Generally, we classify an issue by browsing the whole issue context, including both question part and discussion part.

2) Checking Related Issues: Our manual judgement is conducted by three members of our research group. Each of them scans the issue-pairs independently, and labels the similarity as "related" or "unrelated". Once we get different results on classification, we will make a double check to ensure whether the issue-pairs are related or not. If the results of the double check are still different, the final label is determined by voting. For example, if two participants vote "related" and one votes "unrelated" for an issue-pair, we judge the issue-pair as a related issue-pair. After manual judgement, there are 149 issue-pairs labeled as related. Notice that we exclude the issue-pairs, in which the creation time of Python issues are later than the creation time of R issues. To ensure the reliability of human labeling, we also calculate the inter-rater (i.e., Cohen's Kappa [11]) for manual labeling, by comparing the final result after voting and the independent result from each member. The results show that for each member, the value of Cohen's Kappa is larger than 0.6, which means the related/unrelated results are substantial [12].

3) Evaluating Related levels: Besides, we label the related levels of these 149 issue pairs. In detail, we group the knowledge transferred between these 149 issue pairs into three levels manually.

- **Direction-related level (88 issue-pairs).** the corresponding Python issues offer knowledge related to the R issues in the same or very similar *directions*.
- **Problem-related level (32 issue-pairs).** In this level, the two issues are probably discussing on the very similar *problems*, however, Python issues have not given direct solutions to R issues.
- **Solution-related level (28 issue-pairs).** The Python issues offer direct *solutions* (e.g., pseudo-code and reference), which are most helpful knowledge to the newer R

issue. This kind of knowledge offered by Python issues can almost be used in dealing with R issues directly.

III. LINKING ISSUES FROM R TO PYTHON

"How can developers link to the related Python issues when facing R issues" may become a critical question of knowledge transfer from Python to R. No matter in which situations, developers desire to gain useful information to solve issues as soon as possible. In this RQ, we focus on the **question part** (see Section II-C1) of the 149 R issues, which contains the topic and the body of issues submitted by issue reporters. The question part of R issues represents the early information we can get at the report time of R issues. Based on the question part of R issues, we employ different strategies to search for their related Python issues, by their topics and two popular text mining methods TF-IDF and BM25[3].

Topic Search. Because of the limitation of searching APIs available on GitHub, we can hardly search the corresponding issues by using all the content of question parts. We select the topics from the issues, which represent the main ideas of issue reporters. We search for the corresponding issues in Scikit-Learn repository by using searching API from GitHub and check the top 10 results on the return list.

TF-IDF search. From two strategies mentioned above, we observe that the available searching tool from GitHub can hardly fulfill our needs. Therefore, we introduce a classic strategy in natural language process, TF-IDF [13]. We abstract word embeddings by TF-IDF with the question part of the R issues and use these word embeddings vectors to match 10 most similar corresponding issues. Different with the text similarity search we used before, we set up word embeddings by TF-IDF instead of using the open source data [8]. In detail, we implement the TF-IDF method by Scikit-learn[4].

BM25 search. Though the strategy above improve a lot, we still consider finding a more suitable way for raising successfully searched rate. BM25 [14] is usually used in evaluating the relationship between query and documents. This strategy mainly computes the similarity by 3 parts, which are the weight of words, similarity between words and documents, similarity between words and queries. We employ BM25 to compute the similarity between question parts from R issues and Python issues. In detail, we reuse an implementation of BM25 algorithm in the Python library Gensim[5] with the

---

[3]Our dataset is constructed via cosine distance and the word embedding set offered and pre-trained by Google (see Section II-B). To avoid the bias, we introduce two different text mining methods TF-IDF and BM25 here.

[4]https://scikit-learn.org/stable/modules/feature_extraction.html#text-feature-extraction

[5]http://pydoc.net/gensim/3.2.0/gensim.summarization.bm25/

default parameter settings.

For each search strategy, we check the top 10 results on the return list. Once the related Python issue from the issue-pair occurs on the top 10 list, we count it as a "successful search" and record the rank. Table II represents the results of the above three search strategies. We can find that in Table II, BM25 performs much better than the other strategies, which can successfully detect almost half of the issue-pairs.

> Answer to RQ1: by conducting three strategies to search related issues at the very early stage of R issues, we observe that BM25 search performs the best, which successfully searches over 47% related Python issues for the 149 R issues by only using the initial information (e.g., title and question body).

IV. TYPES OF KNOWLEDGE

In the RQ above, we start from R issues of the related issue-pairs, and find out how to link to Python issues. In the following two RQs, we will focus on Python issues of the related issue-pairs. Specifically, we are going to discover what type of knowledge we can get from the Python issues in this RQ. In order to make a summary of knowledge in Python related issues, we classify the knowledge into 6 groups, which are **description**, **reference**, **outer link**, **related issue**, **code** and **code document**, as illustrated in Table III with numbers of issues containing such kind of knowledge, brief introduction and typical examples.

We choose an interesting example containing all 6 knowledge types, as shown in Figure 2, which mainly discusses about the implementation of Balanced Random Forest. Next, we will present the description of these 6 kinds of knowledge with help of the example.

Description type (147 Python issues): Almost all of the issues are considered to have description, which offer information in natural language directly. For example in Figure 2, the orange box is labeled as the description part. However, not all issue with natural language are included, we exclude two issues which only contain "thanks" in the content.

Reference type (28 Python issues): This type is the most recognizable in these 6 groups. It offers the research papers, related tutorial, etc. As shown in the green box of Figure 2, the Python issue offers a research paper's link, which mainly talks about using random forest to learn imbalanced data. Though this kind of knowledge may sometimes be complex, it is worth reading for developers, which usually offers the original idea of an algorithm.

Outer link type (97 Python issues): Python issues contain various kinds of url links, which offer related packages or datasets, or connect to different repositories on GitHub and other open source platform. The purple box in Figure 2 gives an example of outer link, which offers dataset.

Related issue type (74 Python issues): Referring to another issue is also quite usual in issues. Those links with different issue numbers in current repository or other repositories are all

TABLE III
TYPE OF KNOWLEDGE IN 149 RELATED PYTHON ISSUES

Group	#Issues	Brief introduction	Example Issue
Description	147	Natural language description	https://github.com/scikit-learn/scikit-learn/issues/1454
Reference	28	Research paper etc.	https://github.com/scikit-learn/scikit-learn/issues/6545
Outer Link	97	A link to outer website e.g Wikipedia	https://github.com/scikit-learn/scikit-learn/issues/448
Related Issue	74	Another linked issue	https://github.com/scikit-learn/scikit-learn/issues/6473
Code	124	More than 5 lines of code/ Pulls	https://github.com/scikit-learn/scikit-learn/issues/2089
Code Document	58	Tutorial of code	https://github.com/scikit-learn/scikit-learn/issues/3735
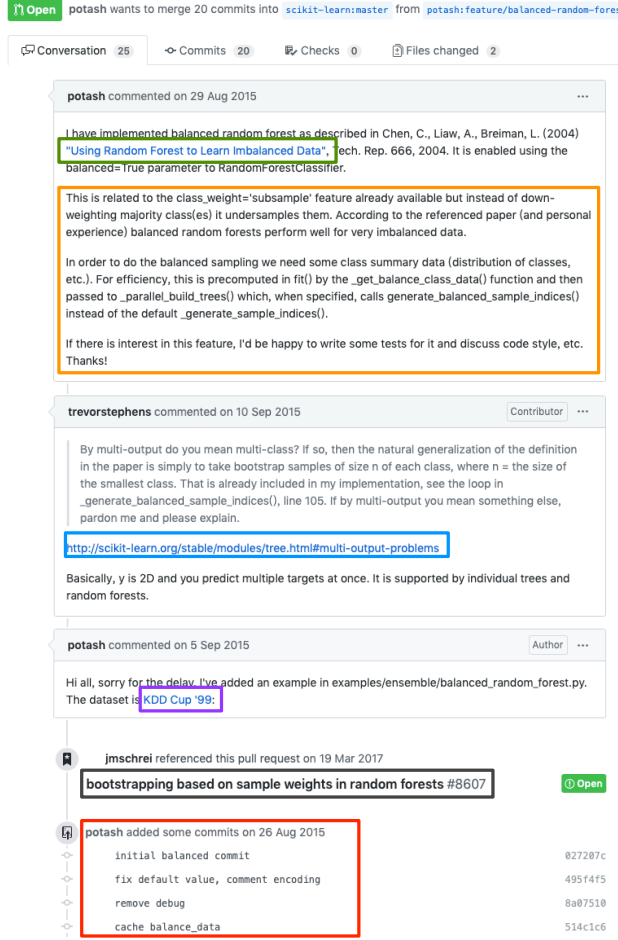
Fig. 2. An example of issues with multiple kinds of knowledge from Scikit Learn Project

involved in this type of knowledge. The black box in Figure 2 is an example of related issue knowledge.

These 4 types of knowledge described above can be learned language-independently, which are more general. The following two kinds of knowledge may need Python skill to understand and apply.

Code type (124 Python issues): Once a code part with more than 3 lines occurs in the issue, we count it as an instance of code knowledge. As illustrated in Figure 2, the red box is a typical example of code part. Due to the different grammar of different languages, this type of knowledge may be more useful for those developers who can skillfully use Python.

Code document type (58 Python issues): Besides code type, code document type will sometimes occur in the issues. As shown in Figure 2, the blue box is an example of code document.

> Answer to RQ2: we observe that related Python issue can offer 6 types of knowledge for those R issues: description, reference, outer link, related issue, code and code document. Among them, the most popular kinds of knowledge are description and code.

V. HELPFULNESS OF KNOWLEDGE

In the above RQ, we discuss various types of knowledge from Python issues. "Which type of knowledge from Python issues is most helpful?" is the question we need to solve next. Based on the related levels of the issue-pairs (see Section II-C3), we mark the helpfulness $H(\mathcal{I}_P)$ of corresponding Python issues $\mathcal{I}_P$ in the range from 1 to 3:

$$H(\mathcal{I}_P) = \begin{cases} 1, & \text{if } \langle \mathcal{I}_P, \mathcal{I}_R \rangle \text{ is labeled as direction-related;} \\ 2, & \text{if } \langle \mathcal{I}_P, \mathcal{I}_R \rangle \text{ is labeled as problem-related;} \\ 3, & \text{if } \langle \mathcal{I}_P, \mathcal{I}_R \rangle \text{ is labeled as solution-related.} \end{cases}$$

and use the helpfulness of Python issues to evaluate the most useful knowledge type.

Spearman correspondence: First, we conduct a Spearman corresponding test [15] between whether containing the knowledge type and the helpfulness of Python issues, to select the most useful knowledge type. The larger Spearman correspondence shows the knowledge type with higher helpfulness. As illustrated in Table IV, we can find that the reference type gets the highest Spearman correspondence among 6 types of knowledge. Besides, related issue type gets the second place.

Weighted mean helpfulness: Besides, we judge the helpfulness of knowledge type by the following formulas. Let $T =$ {description, reference, outer link, related issue, code, code document}. For $t \in T$, $S(\mathcal{I}, t)$ implies the set of Python issues containing the knowledge in type t. $H(\mathcal{I}_P)$ implies the helpfulness ($H(\mathcal{I}_P) \in \{1, 2, 3\}$) of the Python issue $\mathcal{I}_P$. The weighted mean helpfulness ($H_w(t)$) of type t are computed as follows.

$$H_w(t) = \frac{\sum\limits_{\mathcal{I}_P \in S(\mathcal{I},t)} H(\mathcal{I}_P)}{|S(\mathcal{I},t)|}$$

Finally, we rank the value of $H_w(t)$ for each knowledge type in Table IV. We can find that reference type gets the first place again. Also, the related issue type still follows.

Knowledge Type	Spearman	$Rank_{sp}$	$H_w(t)$	$Rank_{sc}$
Reference	0.16	1	1.89	1
Related issue	0.11	2	1.70	2
Description	0.09	3	1.60	5
Code document	0.04	4	1.61	4
Outer link	0.01	5	1.67	3
Code	-0.02	6	1.58	6

According to the above results, we conclude that the reference part is the most important part in the related Python issues of the cross language knowledge transferring. Once we search for the cross language issues, scanning for the reference type of knowledge is highly recommended.

> Answer to RQ3: based on the comparison of spearman correspondence and weighted mean helpfulness, we observe that reference knowledge type turns out to be the most useful knowledge.

VI. THREATS TO VALIDITY

We select 15 of the open source R repositories from GitHub which all come from machine learning classifiers category. They do not cover the issues in all kinds of R repositories, and new issues submitted after November 2018 are not included. Nonetheless, these 15 repositories are popular and well known which cover several classifiers. Furthermore, it is recommended that more projects with more issues in R and Python should be tested using our approach, and the result may vary.

For each issue-pair, we browse their content, try to understand their ideas, carefully judge the similarity levels and issue groups. We then gather all the result from 3 members and do the double-check job to ensure their correctness. Thus, we believe that all the issue-pairs we extracted are true positive. However, we compute the similarities by only using natural language text, and excluded the code, chart, etc., which may lead to lose some issue-pairs that might also be helpful for cross-language referring.

Text similarity search uses the similar text mining method with dataset extraction. Though we employ two different text mining methods while doing the searching job in RQ1, there might be some coincidence that some issues have no discussion part, which may cause little higher of the successfully searched rate.

VII. CONCLUSIONS AND FUTURE WORKS

Different languages are used to solve similar problems, which gives rise to knowledge transfer across different languages development. In this paper, we discover knowledge transfer between 15 machine learning R repositories and Scikit-Learn by analyzing their issues on GitHub.

We extract 149 related issue-pairs from 1161 R issues and 7496 Scikit-Learn issues manually. We experiment and observe that text similarity search gains high successfully searched rate and the perfect performance on ranking on searching corresponding issues for just created issues. Then, we abstract 6 types of knowledge from cross language issue, which are presented in 149 issue-pairs. Finally, in order to calculate the helpfulness, we rank the knowledge type by two indicators.

In the future, we will extend the study by enlarging the datasets from more different languages to conduct a more complete investigation. At the same time, we will also improve our similarity approach by recording more information (like code), which is considered to be more helpful.

REFERENCES

[1] Y. Zhang, D. Lo, P. S. Kochhar, X. Xia, Q. Li, and J. Sun, "Detecting similar repositories on github," in *2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, Feb 2017, pp. 13–23.

[2] C. McMillan, M. Grechanik, and D. Poshyvanyk, "Detecting similar software applications," in *Proceedings of the 34th International Conference on Software Engineering*, ser. ICSE '12. Piscataway, NJ, USA: IEEE Press, 2012, pp. 364–374. [Online]. Available: http://dl.acm.org/citation.cfm?id=2337223.2337267

[3] S. Athey, J. Tibshirani, S. Wager *et al.*, "Generalized random forests," *The Annals of Statistics*, vol. 47, no. 2, pp. 1148–1178, 2019.

[4] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg *et al.*, "Scikit-learn: Machine learning in python," *Journal of machine learning research*, vol. 12, no. Oct, pp. 2825–2830, 2011.

[5] M. N. Wright and A. Ziegler, "ranger: A fast implementation of random forests for high dimensional data in c++ and r," *Journal of Statistical Software*, vol. 077, no. 1, 2015.

[6] T. F. Bissyande, D. Lo, L. Jiang, L. Reveillere, J. Klein, and Y. L. Traon, "Got issues? who cares about it? a large scale investigation of issue trackers from github," in *IEEE International Symposium on Software Reliability Engineering*, 2013.

[7] E. Kalliamvakou, G. Gousios, K. Blincoe, L. Singer, D. M. German, and D. Damian, "An in-depth study of the promises and perils of mining github," *Empirical Software Engineering*, vol. 21, no. 5, pp. 2035–2071, 2016.

[8] Google, "A pre-trained dataset from google news, google-news-vectors-negative300," https://code.google.com/archive/p/word2vec.

[9] X. Ye, H. Shen, X. Ma, R. Bunescu, and C. Liu, "From word embeddings to document similarities for improved information retrieval in software engineering," in *Proceedings of the 38th international conference on software engineering*. ACM, 2016, pp. 404–415.

[10] R. Mihalcea, C. Corley, and C. Strapparava, "Corpus-based and knowledge-based measures of text semantic similarity," *Unt Scholarly Works*, vol. 1, pp. 775–780, 2006.

[11] L. M. Hsu and R. Field, "Interrater agreement measures: Comments on kappan, cohen's kappa, scott's π, and aickin's α," *Understanding Statistics*, vol. 2, no. 3, pp. 205–219, 2003.

[12] J. R. Landis and G. G. Koch, "The measurement of observer agreement for categorical data," *biometrics*, pp. 159–174, 1977.

[13] X. Yang, D. Lo, X. Xia, L. Bao, and J. Sun, "Combining word embedding with information retrieval to recommend similar bug reports," in *IEEE International Symposium on Software Reliability Engineering*, 2016.

[14] C. Sun, D. Lo, S. C. Khoo, and J. Jiang, "Towards more accurate retrieval of duplicate bug reports," in *IEEE/ACM International Conference on Automated Software Engineering*, 2011.

[15] D. J. Best and D. E. Roberts, "Algorithm as 89: The upper tail probabilities of spearman's rho," *Journal of the Royal Statistical Society*, vol. 24, no. 3, pp. 377–379, 1975.

Quantifying the Relationship Between Health Outcomes and Unhealthy Habits

Swapna S. Gokhale
Dept. of Computer Science & Engg.
Univ. of Connecticut, Storrs, CT 06269
{swapna.gokhale}@uconn.edu

Abstract

Chronic health outcomes impact the quality of life of affected individuals and their families and also lead to huge health care costs. Most of the chronic health outcomes can be attributed to few unhealthy behaviors, however, the extent to which these behaviors can explain the variation in the common outcomes is not known. This paper explores the relationship between: (i) unhealthy behaviors using principal components analysis; and (ii) unhealthy behaviors and chronic health outcomes using multiple linear regression. The 500 Cities data, released by the Center for Disease Control, forms the basis of this investigation. PCA suggests that the unhealthy behaviors can be projected along two dimensions, each punctuated by the common age of occurrence. The results of linear regression are consistent with expectations for some outcomes, but reveal unexpected trends for the others.

1   Introduction & Motivation

Chronic diseases are broadly defined as conditions that last longer than a year or more and require ongoing medical attention or limit activities of daily living or both. Chronic conditions such as heart disease, cancer, and diabetes impact the quality of lives of the affected individual as well as their families. Moreover, they are the leading causes of death and disability, and drivers of the nation's $3.3 trillion in annual health care costs. The CDC estimates that six in ten adults in the U.S. have one chronic disease, and four in ten adults have two or more [6].

Many chronic diseases may be attributed to a short list of risky behaviors: (i) tobacco use and exposure to secondhand smoke; (ii) poor nutrition, including diets low in fruits and vegetables and high in sodium and saturated fats; (iii) lack of physical activity; and (iv) excessive alcohol use [14]. The association between these risk factors and chronic diseases is known qualitatively. What is not known, however, is the relationship of these unhealthy behaviors with each other,

and the extent to which these behaviors contribute to specific chronic health outcomes. It is crucial to quantify the level of variance in the different health outcomes that can be explained by risky behaviors; because then the search for what leads to unexplained or residual variance can begin in earnest. These additional causes, beyond unhealthy or risky behaviors, may be found in other factors such as environmental stressors and genetic predisposition.

In this paper, we explore the relationship among the unhealthy behaviors themselves, and between unhealthy behaviors and chronic health outcomes. The 500 cities data [7], which provides city and census-tract level small area estimates for chronic disease risk factors, health outcomes, and clinical preventive service use for the largest 500 cities in the United States forms the basis of our investigation. Principal components analysis is used to study how unhealthy behaviors cluster together, and multiple linear regression is used to relate these behaviors to the health outcomes. Our results suggest that the five unhealthy behaviors can be mapped to two dimensions. The first dimension accounts for approximately 68% of the variation and comprises of habits that may mostly develop around the middle age, whereas the second dimension includes only binge drinking which is more prevalent among the younger population. The results of multiple linear regression confirm that a large percentage of variation in coronary heart disease, stroke, high cholesterol, COPD, and diabetes can be attributed to unhealthy behaviors. However, a relatively lower percentage of variation in high blood pressure, which is viewed as a risk factor for heart disease and stroke, and asthma which is considered a risk factor for COPD can be explained by unhealthy habits. Moreover, it appears surprising that over 80% of the variation in arthritis and teeth loss, two conditions that co-exist with aging-related deterioration, is attributable to unhealthy habits. Finally, only about 50% of the variance in cancer is explainable by unhealthy behaviors, suggesting the presence of strong genetic and/or environmental influences.

The rest of the paper is organized as follows: Section 2 summarizes the 500 cities data. Section 3 and Section 4

1

discuss principal components and linear regression analysis respectively. Section 5 compares related research. Section 6 offers concluding remarks and future research directions.

2 The 500 Cities Data

The 500 Cities Project is a collaboration between the Center for Disease Control (CDC), the Robert Wood Johnson Foundation, and the CDC Foundation. The purpose of the 500 Cities Project is to provide city and census-tract level small area estimates for 5 unhealthy behaviors, 13 health outcomes, and 11 clinical preventive service use for the largest 500 cities in the United States [7]. These measures include major risk behaviors that lead to illness, suffering and early death related to chronic diseases and conditions, as well as the conditions and diseases that are the most common, costly, and preventable of all health problems [9]. These measures are estimated using the raw data from the CDCs Behavioral Risk Factor Surveillance System [8], using a multi-level statistical modeling framework [10].

In this paper, we considered the 13 health outcomes and 5 unhealthy behaviors from the 500 Cities Project. Tables 1 and 2 offer a brief summary, significance, and mean prevalence of these measures. In Table 1, all the health outcomes, except for mental and physical health, are formally diagnosed by medical professionals whereas estimates of (lack of) mental and phsyical health are self-reported [9].

3 Principal Components Analysis

Principal Components Analysis (PCA) uses an orthogonal transformation to convert a set of observations with correlated variables into a set of values of linearly uncorrelated variables called principal components [12]. The first principal component has the largest possible variance, that is, it accounts for as much variability in the data as possible. Each succeeding principal component in turn has the highest variance possible under the constraint that it is orthogonal to the preceding components. PCA creates as many new independent variables as there exist in the original data. Usually, however, the first few independent variables can explain a large percentage of the variation in the data and are retained for analysis, while the others that contribute very little to the variability are eliminated in favor of model parsimony. PCA is therefore also referred to as a feature extraction or dimensionality reduction procedure.

We apply PCA to uncover the relationships among the 5 unhealthy behaviors. The cumulative variability explained by the principal components is represented using a scree plot in Figure 1, which shows that the first two dimensions account for about 84% of the variation. Focusing on these two dimensions, our next step was to investigate the contribution of unhealthy behaviors to each as shown in Figures 3

and 4. The contribution of each variable is represented as a percentage, where the red dashed lines are reference lines that correspond to the expected contribution if each variable pitched uniformly. With 5 original variables, the reference lines are shown at 20%. Variables with contributions above the reference line are considered important for that dimension. According to this heuristic, in the figure, three variables, namely, lack of physical activity, obesity, and smoking are important contributors to the first dimension. Of these, lack of physical activity and obesity contribute predominantly, while smoking is just barely above the reference line. Because lack of activity and obesity usually develop around middle age, we label this dimension as "Midlife Crisis". For the second dimension, only binge drinking contributes more than 20%, which tends to occur in younger adults, and hence, we label this dimension as "Youthful Adventures". The graph of PCA variables shows the orthogonal projection of the five behaviors along the two dimensions as shown in Figure 2. Figures 5 and 6 show that the top 20 cities contribute more than the uniform 0.2% towards each dimension. Contributors to Midlife Crisis concentrate in the Midwest and Mountain States, whereas Youthful Adventures cluster along the East and West coasts as shown in Figure 7.
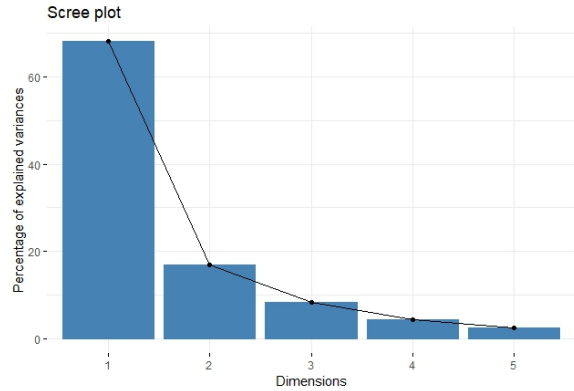
Figure 1. Scree Plot

4 Multiple Linear Regression

We postulate a linear relationship between health outcome i, and unhealthy behaviors $UB_1, \ldots, UB_5$ given by:

$$HO_i = \beta_{i,0} + \sum_{j=1}^{5} \beta_{i,j} * UB_j + e_i \qquad (1)$$

The key assumption underlying least squares linear regression models is homoskedasticity, which implies that the variations for all the observations in a data set are equal.

2

Table 1. Chronic Health Outcomes: Significance & Prevalence

	Health Outcome	Mean
HO_1	**Arthritis:** Reduces physical function, quality of life.	22.39
HO_2	**Asthma:** ED visits, hospitalizations, missed work, comorbid depression.	9.18
HO_3	**Cancer:** Still a leading cause of death, second to heart disease.	5.98
HO_4	**Chronic Kidney Disease:** Ninth leading cause of death, but most affected don't know.	2.75
HO_5	**Chronic Obstructive Pulmonary Disease (COPD):** Impaired pulmonary function, which often goes undiagnosed.	6.05
HO_6	**Coronary Heart Disease (CHD):** Common form of heart disease, leading cause of death	5.73
HO_7	**High BP:** Responsible for $20-30\%$ CHD, $20-50\%$ Stroke, cardiovascular complications.	30.39
HO_8	**High Cholesterol:** Responsible for $30-40\%$ CHD, $10-20\%$ strokes.	31.35
HO_9	**Diabetes:** Impaired glucose function, complications if not managed.	10.25
HO_{10}	**Mental Health:** Not good for more than 14 days. Related to diabetes, cancer, cardiovascular disease, asthma, obesity. Many risk factors; physical inactivity, smoking, binge drinking, insufficient sleep also contribute to mental illness.	12.44
HO_{11}	**Physical Health:** Not good for more than 14 days. Related to health-related quality of life.	12.57
HO_{12}	**Stroke:** 1 out of 20 deaths, serious long-term disability.	3.05
HO_{13}	**Teeth Loss:** Reduces quality of life, self-image, and daily functioning (>65 years old).	14.51

Table 2. Unhealthy Behaviors: Significance & Prevalence

	Unhealthy Behavior	Mean
UB_1	**Current Smoking:** Greater than 100 cigarettes and smoke every day or most days. Increases the risk for heart disease, stroke, multiple types of cancer, and chronic lung disease.	17.58
UB_2	**Binge Drinking:** Five or more drinks (men), four or more drinks (women) at one time. Accounts for over 40,000 deaths and 1 million years of potential life lost annually. Health and social problems such as motor-vehicle crashes, violence, suicide, hypertension, acute myocardial infarction, STDs, unintended pregnancies, fetal alcohol spectrum disorders, sudden infant death syndrome.	16.53
UB_3	**No Leisure Time Physical Activity (LoPA):** Other than their regular job, did not participate in any physical activities or exercises such as running, calisthenics, golf, gardening, or walking. Improve the health and quality of life of all ages, regardless of chronic disease or disability. Lower the risk for early death, coronary heart disease, stroke, high blood pressure, type 2 diabetes, breast and colon cancer, falls, and depression.	25.86
UB_4	**Obesity:** Body mass index (BMI) greater than $30.0\ kg/m^2$. Increases the risk for multiple chronic diseases, including heart disease, stroke, hypertension, type 2 diabetes, osteoarthritis, and certain cancers.	29.31
UB_5	**Sleeping less than 7 hours (LoS):** Insufficient sleep (< 7 hours), on an average, during a 24-hour period. Associated with chronic conditions such as diabetes, cardiovascular disease, hypertension, obesity, and depression. May cause motor vehicle crashes and industrial errors, causing substantial injury and disability. Reduces productivity and quality of life.	35.69

Most real-world data sets will probably be heteroskeastic [15], but it is possible to use the least squares model for large enough sample sizes, which is the case here. In Equation (1), HO_i's are the predicted or response variables, and $UB_1, \ldots, UB_5$ are the independent or predictor variables, often known as regressors. The coefficients $\beta_{i,j}, j = 0, \ldots, 5$ are estimated by minimizing the sum of squared unexplained parts. The coefficient of determination R^2 is given by Equation (2), where $H\hat{O}_{i,k}$ is the estimate of the health outcome i for the kth city produced by the model, and $\bar{HO}_i$ is the mean value of the outcome i across all the 500 cities. R^2 measures the proportion of variation in HO_i that can be explained by the regressors $UB_1, \ldots, UB_5$.

$$R_i{}^2 = \frac{ModelSS}{TotalSS} = \frac{\sum_{k=1}^{500}(\hat{HO_i} - \bar{HO}_{i,k})^2}{\sum_{k=1}^{500}(HO_{i,k} - \bar{HO_i}^2)} \quad (2)$$

For each health outcome, a p-value is also estimated by

3
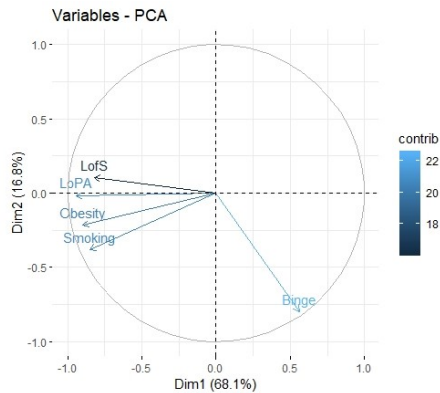
Figure 2. Graph of Variables – PCA

Figure 3. Unhealthy Behaviors ==> Midlife Crisis

Figure 4. Unhealthy Behaviors ==> Youthful Adventures

Figure 5. Cities ==> Midlife Crisis

Figure 6. Cities ==> Youthful Adventures
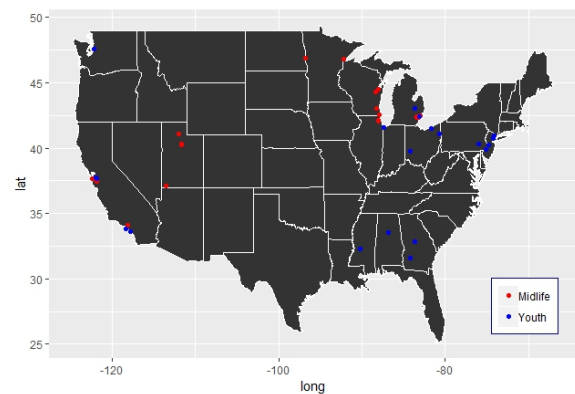
Figure 7. Cities ==> Dimensions, Geographical Spread

the model for all unhealthy behaviors, which is compared against the typical level of significance $\alpha = 0.05$. If the p-value is less than α, then the effect of that specific behav-

ior on the particular health outcome is significant. Table 3 shows the results of the regression model. For each health outcome, the table lists the t-statistic and p-values for each unhealthy behavior, and R^2 which explains the total varia-

4

tion that can be attributed collectively to these behaviors. We divide the outcomes into two groups I and II, these groups respectively comprise of the outcomes for which over 80% and less than 80% is explained by the unhealthy habits. The results confirm certain expectations, but also reveal anomalies. Only for about 50% of the outcomes, all the five unhealthy behaviors are statistically significant. These include CHD, stroke, teeth loss, diabetes, cancer, asthma, and physical health. High blood pressure and high cholesterol, are the two common precursors to CHD and stroke, however, these belong to first and second groups respectively. Thus, although high blood pressure may be mostly attributed to unhealthy behaviors making up the midlife crisis, high cholesterol may have additional origins. Transitioning from high cholesterol which is relatively benign, to life threatening conditions such as stroke and CHD, however, may be precipitated by lifestyle choices. A similar relationship can be seen between COPD and asthma, COPD belongs to the first group, but asthma which is considered a risk factor belongs to the second. Although all the unhealthy behaviors are statistically significant for cancer and asthma, collectively they explain only about 50% and 73% variation respectively. This suggests that in cancer and asthma genetics and the environment [5] may interplay with lifestyle choices. A few additional significant and interesting observations include: Smoking is significant for all the health outcomes except high cholesterol. Binge drinking is not significant for high blood pressure and COPD. Lack of mental health is not influenced by either binge drinking or lack of sleep. Obesity is not significant for kidney disease. Finally, although teeth loss and arthritis are mainly dominant in aging populations, they are also members of the first group. This indicates that the influence of lifestyle choices is not limited to metabolic conditions of high blood pressure, high cholesterol and diabetes.

5 Related Research

The association between chronic diseases and lifestyle choices is generally known, however, very few studies have sought to quantify this association. Adaji *et. al.* [1] use logistic regression to identify the risk factors associated with some common chronic conditions (arthritis, angina, stroke, diabetes, and chronic lungs disorder) among people over 50 years in India. The model includes socioeconomic and demographic factors and the interplay between the conditions. A similar study by Ismail *et. al.* [11] is conducted for the younger Indian population but only for coronary heart disease. Zhao *et. al.* [16] estimate the prevalence and correlates of chronic diseases in an elderly population in Haikou. Four major chronic conditions, namely, hypertension, diabetes, COPD and stroke and sociodemographic characteristics and lifestyle factors are considered in the study. Regres-

sion analysis has been used in the context of chronic conditions to estimate the various types of burdens, including health care costs, absenteeism and employer costs associated with these conditions [3, 13]. In contrast, our research analyzes how the variance in a variety of chronic conditions can be attributed to five core unhealthy behaviors, regardless of the other socioeconomic and demographic factors.

6 Conclusions and Future Research

This paper explores the relationship among common unhealthy behaviors, and their influence on prevalent chronic health outcomes quantitatively. The analysis uses the 500 Cities data, which provides small area estimates of 27 health-related measures for 500 largest cities in the United States. PCA is used to map the unhealthy behaviors to orthogonal dimensions to understand their co-occurrence, and multiple linear regression is used to explore how these unhealthy behaviors relate to chronic health outcomes. PCA dimensions can be readily interpreted within the context of age. However, the results of multiple linear regression expose some interesting, and unexpected tendencies.

Our future research involves relating the health outcomes at the level of census tracts to demographic and socioeconomic data available from the U.S. Census Bureau [4].

References

[1] E. E. Adaji, A. S. Ahankari, and P. R. Myles. "An Investigation to Identify Potential Risk Factors Associated with Chronic Diseases Among the Older Population in India". *Indian J. Community Med*, 42(1):46–52, 2017.

[2] E. C. Alexopoulos. "Introduction to Multivariate Regression Analysis". *Hippokratia*, 14(Suppl 1):23–28, December 2010.

[3] G. R. B. Asay, K. Roy, J. E. Lang, R. L. Payne, and D. H. Howard. "Absenteeism and Employer Costs Associated with Chronic Diseases and Health Risk Factors in the US Workforce". *Prev Chronic Dis*, 2016.

[4] United States Census Bureau. "Census Data API User Guide". https://www.census.gov/data/developers/guidance/api-user-guide.html, June 2017. Accessed: 2019-01-21.

[5] Illinois Disability and Health Program. "What is Chronic Disease? Important Things to Know About Chronic Diseases for Persons with Disabilities". http://www.idph.state.il.us/idhp/idhp_ChronicDisease.htm. Accessed: 2020-01-21.

5

Table 3. Results of Multiple Linear Regression Model

HO	T-statistic, p-values					F-statistic	R^2
	Smoking	Obesity	LoS	LoPA	Binge		
Group I: Over 80% Variation							
Arthritis	$1.04e-08$	$< 2e-16$	$< 2e-16$	0.948	$< 2e-16$	$< 2.2e-16$	0.8389
High BP	$6.29e-14$	$6.86e-08$	$< 2e-16$	$< 2e-16$	0.267	$< 2.2e-16$	0.8621
CHD	$< 2e-16$	$< 2e-16$	$< 2e-16$	$< 2e-16$	0.00016	$< 2.2e-16$	0.8706
COPD	$3.43e-15$	$< 2e-16$	0.00101	0.02320	0.4887	$< 2.2e-16$	0.903
Diabetes	$3.29e-12$	$1.43e-09$	$< 2e-16$	$< 2e-16$	$< 2e-16$	$< 2.2e-19$	0.8223
Stroke	$< 2e-16$	$< 2e-16$	$< 2e-16$	$< 2e-16$	0.0263	$< 2.2e-19$	0.8442
Teeth Loss	$< 2e-16$	$< 2e-16$	0.00281	0.00180	$1.06e-09$	$< 2.2e-19$	0.8669
Group II: Less than 80% Variation							
Cancer	0.04591	$< 2e-16$	0.00312	$2.34e-15$	$< 2e-16$	$< 2.2e-16$	0.517
Asthma	0.000209	$< 2e-16$	$3.07e-05$	$5.78e-12$	$2.62e-15$	$< 2.2e-16$	0.641
High Chol.	0.1504	0.0673	$< 2e-16$	0.4080	$1.81e-07$	$< 2.2e-16$	0.6354
Kidney	$1.71e-11$	0.517	$1.65e-13$	$2.32e-09$	$8.59e-05$	$< 2.2e-16$	0.7317
Mntl. Hlth	$5.89e-08$	$< 2e-16$	0.694	$< 2e-16$	0.140	$< 2.2e-16$	0.7394
Phys. Hlth	$3.65e-11$	$2.13e-10$	0.0205	$1.34e-06$	$2.99e-08$	$< 2.2e-16$	0.7372

[6] Center for Disease Control and Prevention. "About Chronic Diseases". https://www.cdc.gov/chronicdisease/about/index.htm. Accessed: 2019-01-21.

[7] Center for Disease Control and Prevention. "500 Cities: Local Data for Better Health, About the Project". https://www.cdc.gov/500cities/about.htm, November 2017. Accessed: 2019-01-21.

[8] Center for Disease Control and Prevention. "Behavioral Risk Factor Surveillance System". https://www.cdc.gov/brfss/index.html, April 2017. Accessed: 2019-01-21.

[9] Center for Disease Control and Prevention. "The 500 Cities Project: Local Data for Better Health, Measures Definition". https://www.cdc.gov/500cities/measure-definitions.htm, November 2017. Accessed: 2019-01-21.

[10] Center for Disease Control and Prevention. "The 500 Cities Project: Local Data for Better Health, Methodology". https://www.cdc.gov/500cities/methodology.htm, November 2017. Accessed: 2019-01-21.

[11] B. Ismail and M. Anil. "Regression Methods for Analyzing the Risk Factors for a Life Style Disease Among the Young Population of India". *Indian Heart J.*, 66(6):587–592, November-December 2014.

[12] K. Khan. "Principal Component Analysis - An Introduction with R Implementation". https://rpubs.com/koushikstat/pca. Accessed: 2020-01-21.

[13] H. H. Konig, H. Leicht, H. Bicket, A. Fuchs, and et. al. J. Genischen. "Effects of Multiple Chronic Conditions on Health Care Costs: An Analysis based on an Advanced Tree-Based Regression Model". *BMC Health Services Research*, 2013.

[14] W. C. Willett, J. P. Koplan, R. Nugent, C. Dusenbury, P. Puska, and T. A. Gaziano. "Prevention of Chronic Disease by Means of Diet and Lifestyle Changes". In D. T. Jamison, J. G. Breman, and A. R. Measham, editors, *Disease Control Priorities in Developing Countries, 2nd Edition*, chapter 44. he International Bank for Reconstruction and Development / The World Bank, 2006.

[15] C. Yobero. "Methods for Detecting and Resolving Heteroskedasticity". https://rpubs.com/cyobero/187387, June 2016. Accessed: 2020-01-21.

[16] C. Zhao, L. Wong, Q. Zhu, and H. Yang. "Prevalence and Correlates of Chronic Diseases in an Elderly Population in an Elderly Population: A Community Survey in Haikou". *PLoS One*, June 2018.

6

The Reaction of Open Source Projects to C++ Templates and Lambdas: An Empirical Replication Study

Donghoon Kim
Department of Computer Science
Arkansas State University
Jonesboro, AR, USA
dhkim@astate.edu

Loc Ho
Department of Computer Science
Arkansas State University
Jonesboro, AR, USA
loc.ho@smail.astate.edu

Abstract—New language features are added into a programming language to make high quality software. However, new features are not always welcomed in the programming community since they have pros and cons. Templates and lambdas have benefits so both features were added in many programming languages, such as C++, Java, and C#. To find improvements, the programming community wants to know how these features are actually being used in projects. Researchers have conducted these studies in different languages and different experimental environments. In this study, we conduct an empirical replication study with C++ open source projects in the same experimental environment we've conducted with Java and C# to ascertain what the community wants. Our framework of the static analysis tool for Java and C# has been extended to analyze C/C++ open source projects with quantitative data. We investigate how two language features—templates and lambdas—are used in C++ open source projects. We found that C++ templates are used widely by projects and developers, but C++ lambdas are not used widely. These results of this study are similar to those in other programming languages and in other experimental environments.

Index Terms—programming language, language feature, static analysis tool, template, lambda expression, open source project, quantitative data

I. INTRODUCTION

Programming languages have many language features for producing good software. Whenever a new feature is introduced for a language, it is thoroughly tested to see if it is needed for the language [1], [2]. New features can bring efficiency to programmers, improve program performance, and provide benefits in many ways [3], [4]. On the other hand, there are disadvantages to new features. Thus, new features are not always welcomed in the programming community [5]–[7].

Generics feature (templates in C++) offers code reuse without compromising static type checking so it can avoid code duplication [8]. In addition, type checking in C++ templates can be conducted at compile time, rather than runtime. However, C++ templates still have some limits with low readability which can lead to difficulty for debugging the codes when a project uses many templates [9]. Lambda

expression (lambda), a core feature of functional programming, makes concise syntax which may lead to reduce LOC (Lines of Code) [10]. However, it may impact performance for executable size and execution time as well as making the code less readable [10]. These features—generics (templates in C++) and lambdas—have clear benefits, but they also have disadvantages that can make them difficult to use [11]. Thus, the programming community, including programming language designers and educators, wonders how these features are actually being used to find improvements for quality software, and teaching programming languages [11]–[15].

This paper presents an empirical study of C++ templates and lambdas. The work is a replication of previous studies with C# and Java [5], [6], [16], [17]. Our previous studies analyzed the usage of generics and lambdas, who used them, and how their benefits appeared in open source projects. In this study, the usage of two programming language features—templates and lambdas—in 20 open source projects are investigated. The results will be discussed by comparing the previous studies, including our studies and other studies, such as C++ templates by Chen *et al.* [14] and C++ lambdas by Uesbeck *et al.* [11]. More specially, the following two research questions (RQ) are created to investigate each feature:

- **RQ1:** Is a language feature widely used in open source projects?
- **RQ2:** Do project members in each project broadly use a language feature after introduction into the project?

The contributions of this paper are as follows:

- **Analysis of the usage of templates and lambdas in C++:** the open source projects written in C++ were analyzed to determine if the features are used widely. The following facts have been discovered: (1) the usage of templates is much higher than that of lambdas, (2) In many projects, templates were not used long after the project started, and (3) For projects that don't use lambdas, the introduction of templates tends to be late.
- **Analysis of the developers in open source projects:** this analysis results show that one or two developers used

a lot of templates in the projects. Developers who used lambdas also used templates, but not necessarily those who used templates used lambdas.

- **An empirical replication study in major programming languages:** this study in C++ provides a consistent experimental environment with those in the other two programming languages—Java and C#. Thus, it is possible to consistently compare how language features were used in the three programming languages—C++, Java, and C#. The results in this study were also compared with other studies; similar conclusions were drawn.

The paper is organized as follows. Section II illustrates related works. Section III describes research questions and our methodologies to conduct this study. Section IV answers to the research questions with quantitative data. Finally, Section V concludes this paper.

II. RELATED WORK

Researchers have investigated how language features are used with a variety of methods [18], [19]. Asaduzzaman *et al.* [18] discovered many developers, regardless of experience, misuse exception handling in open source Java projects.

Siek and Taha [20] addressed that templates are a powerful but poorly understood feature of the C++ language. Kim *et al.* [5] conducted an empirical study of C# generics feature in open source projects. They found that C# generics are used widely. They compared the results with Java generics [17] and explained several reasons for the different adoption rate of generics feature between C# and Java. Wu *et al.* [12], [14] analyzed how library templates influenced C++ programming in open source systems. They listed most commonly-used template libraries for C++ novices. Chen *et al.* [14] analyzed how C++ templates are used in 50 open source systems. They found that templates are useful for reducing code and C++ developers who prefer templates have no other programming experience. This work is most related to our work. They focused on the adoption of the different type of templates. Our work focuses on how the features are adopted over time and embraced by developers in open source projects.

Uesbeck *et al.* [11] conducted an empirical study of C++ lambdas and programmer experience. They analyzed participants' behaviors to solve programming tasks using lambda expressions and iterators. They found that the students have difficulty to use lambdas in programs and no benefits of lambdas have been made. Likewise, our quantitative results show that few developers are using C++ lambdas. Mazinanian *et al.* [13] analyzed how lambda expression is adapted by Java programmers. They found the reason why Java developers use lambdas. The reasons are (1) making existing code more succinct and readable and (2) avoiding code duplication. They investigated the introduction rate of lambdas over time, starting from the first commit and the last commit of the project. Lambda expression in Java has an increasing trend in the open source community. We also conduct similar methods with C++ open source projects. Our results show similar trends in C++ projects. However, not many C++ projects used lambdas

yet. Nielebock *et al.* investigated the adoption of lambdas in 2,923 open source projects and in three programming languages—C#, C++, and Java [19]. They found that developers are significantly used more lambdas in C# than C++ and Java, but the lambdas are not predominantly applied in concurrent code. Like other studies [5], their study allows us to analyze how a language feature affects usages when implemented differently in different languages.

III. RESEARCH APPROACH

In this section, we explain the approach in this study. Section III-A introduces our research questions. Section III-B introduces the characteristics of 20 projects collected for this study. Section III-C introduces the framework and the procedure to analyze those projects with quantitative data.

A. Research Questions (RQ)

First, we investigate the adoption rate of each language feature. As we explained the pros and cons in the Introduction Section, two language features—templates and lambdas—have similar pros and cons. Developers are responsible for using language features to enhance the project's performance if they are useful. If language features are beneficial, their adoption rate to be chosen by the developers must be high, which thus leads to our first research question:

> **Research Question 1 (RQ1):** Is a language feature widely used in open source projects?

Our speculation was that more software developers use templates rather than lambdas due to several reasons: (1) The template feature was added to C++ much longer than lambda expression; and (2) The syntax of lambda expression is not that easy.

C++ templates were added in 1998 as the standard C++ feature, which is relatively earlier than other major programming languages such as Java (2004) and C# (2005). Lambda expression has been adopted in many programming languages, such as C# (2007), C++ (2011), and Java (2014). Many software developers may know how to use lambda in C++. After a project decides to use a compiler that supports these features, the team can use the features or some individuals may take the initiative on their own [5]. Thus, we want to analyze how many project members actually use templates and lambdas practically in project, which thus leads to our second research question:

> **Research Question 2 (RQ2):** Do project members in each project broadly use a feature after introduction into the project?

B. Projects Studied

To find out the answers for RQs listed above, we downloaded 20 open source projects from `Black Duck Open Hub` (formerly `Ohloh` website). All of the projects have to satisfy these requirements:

---

https://www.openhub.net/

- Each project should have a high level of activity.
- Each project should have at least 100,000 lines of code in C++ programming language.
- Each project should begin before C++ 11 (2011) was released, but 'xLights' is an exception because we couldn't find a project that meets the first two conditions.

Table I describes the information of 20 selected open-source projects. Twenty projects seems small, but more than several billions LOC has been analyzed in each project since we analyzed the projects over time (e.g., each commit from developers). For example, 'Google' has 1,322,907 LOC and about 56,000 commits. For 'Google', more than 300 billions LOC (= 0.65 millions LOC × 56,000 commits) has been analyzed. We assume that the average LOC for 'Google' is the half (0.65 millions LOC) of the last number of LOC (1.3 millions LOC). The table includes the name of each project, and the number of total lines of code written in C++ measured by `Black Duck Open Hub` on the date we downloaded for analysis. The name in brackets is a short name for each project that will be used in this paper.

Project	LOC (C++)
Appleseed	410,994
Boost C++ Library (Boost)	3,222,155
deal.II (deal)	1,899,336
digikam	793,638
Dlib C++ Library (Dlib)	309,498
Fawkes Robot Software Framework (Fawkes)	475,294
Google V8 JavaScript Engine (Google)	1,322,907
ICU for C/C++/Java (ICU)	779,201
KDE Frameworks 5 (KDE)	1,037,464
libc++: The LLVM C++ Standard Library (libc++)	551,586
libMesh: A C++ Finite Element Library (libMesh)	685,658
LLVM/Clang C family frontend (LLVM)	1,276,127
mangos-classic (mangos)	350,519
Mantid	1,376,385
MITK	1,162,057
MongoDB	746,533
OpenMS	428,914
Orfeo ToolBox (Orfeo)	364,063
Point Cloud Library (Point)	1,035,913
xLights	343,268

TABLE I: The 20 C/C++ projects under investigation

C. Procedure

We have a framework of the static program analysis tool for Java and C# [5]. The tool is used to analyze how language features are used in open source projects. Recently, we extended our existing framework to analyze C/C++ open source projects with quantitative data. Our framework is written with several programming languages such as python and C++ with `llvm` library and the `ws2_32` library. The new tool can extract the information to answer our research questions, such as the number of templates, lambdas, and developers. The following is the overall steps to analyze open source projects. After choosing projects that meet the requirements based on the information from `Black Duck Open Hub`, we cloned each project from its remote repository using Git and Subversion to a local machine, check out every version of every file from a project's repository and store the different file revisions in an intermediate format, and transfer this information to a database; extract language features information from each file revision and populate the information in the database server; finally analyze the data in the database to answer each research question.

IV. Experimental Results

To get an overview of adoption of the template and lambda expression features, we investigate the usage of both features in the 20 selected projects. We measure the number of both features to observe how those features are adopted. Table II shows the overall data on how templates and lambdas are used by developers. The title in each column indicates as follows:

- **Start Date**: Date the project started
- **Developers**: the number of developers involved in the project
- **First Date in Template and Lambda expression**: the first date when the first template or lambda was used by a developer (N/A means no one used lambda expression in the project.)
- **Developers in Template and Lambda expression**: the number of developers who used templates or lambdas in the project
- **Usage in Template and Lambda expression**: the number of templates or lambdas used by developers in the project

We observe that:
- The total number of templates is much higher than lambdas in all projects. Lambdas are used in only 10 projects (out of 20 projects).
- Templates were used (adopted) in one year after the projects started in 10 projects. On average, templates were used on approximately 6.5 years after the project started. On the other hand, lambdas are used (adopted) in approximately five years after lambdas were added in C++11 in the year 2011. For projects that don't use lambdas, such as *deal, Fawkes, ICU, KDE*, and *libc++*, the introduction of templates tends to be late.
- The total number of developers who used templates is higher than the total number of developers who used lambdas.

A. RQ1: Usage of Language Features

RQ1: Is a language feature widely used in open source projects?
Templates: We extracted the usage of templates over time. Figure 1(a) shows the number of templates over time. *libc++* used the most templates with 19,012 and *Appleseed* used the second most templates with 3,930 in 20 projects. In most of the projects, the numbers of templates increase sequentially.

As in Table II, you can find relatively large numbers in templates' usage rather than lambdas' usage. However, it is not easy to draw a conclusion if templates are used 'widely' or vice versa because there is a lack of clear criteria to discern between 'widely' and 'not widely' with the number of usage. For this reason, we investigate the usage of other language

Project	Start Date	Developer	Template			Lambda expression		
			First Date	Developer	Usage	First Date	Developer	Usage
Appleseed	7/3/2010	39	7/3/2010	23	3,930	3/18/2018	3	20
Boost	7/7/2000	19	3/4/2004	15	154	7/6/2016	1	1
deal	11/24/1997	10	12/11/2009	3	190	N/A	0	0
digikam	5/5/2004	140	1/9/2007	22	155	N/A	0	0
Dlib	5/2/2008	25	5/2/2008	10	440	10/18/2015	7	40
Fawkes	1/3/2004	25	11/8/2011	4	39	N/A	0	0
Google	6/30/2008	7	10/9/2008	2	8	N/A	0	0
ICU	8/16/1999	72	10/3/2009	20	35	N/A	0	0
KDE	9/28/1999	47	10/18/2009	7	8	N/A	0	0
libc++	5/11/2010	102	5/11/2010	78	19,012	7/31/2015	20	109
libMesh	1/9/2003	4	7/17/2012	3	959	N/A	0	0
LLVM	7/11/2007	181	8/7/2012	88	1,855	9/15/2014	31	130
mangos	10/13/2008	45	10/14/2008	27	166	N/A	0	0
Mantid	4/4/2007	137	10/26/2007	83	907	2/8/2016	32	29
MITK	9/6/1997	218	11/16/2002	77	840	6/3/2016	3	4
MongoDB	10/19/2007	283	12/11/2008	171	2,401	4/9/2015	127	307
OpenMS	6/11/2006	60	2/7/2014	30	128	12/13/2018	3	2
Orfeo	1/5/2006	33	5/15/2006	10	168	N/A	0	0
Point	3/2/2011	244	3/3/2011	82	382	7/4/2019	1	1
xLights	1/29/2013	31	9/26/2014	11	32	N/A	0	0

TABLE II: Software developers involved in the projects and the total usage of template and lambda expression
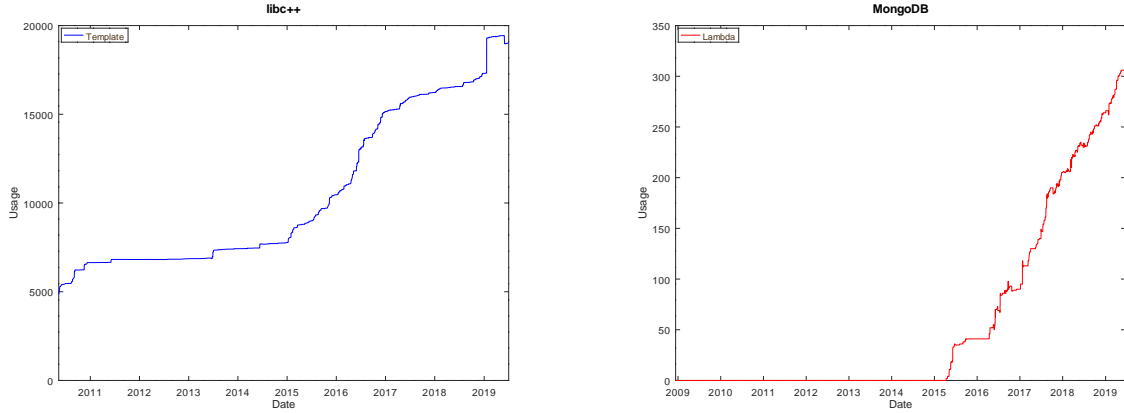
Fig. 1: Usage of features over time: (a) Templates in libc++ (left), (b) Lambdas in MongoDB (right)

Project	Template	If	Switch	Lambda
Appleseed	3,930	7,027	56	20
Boost	154	38	0	1
deal	190	1,774	26	0
digikam	155	6,985	63	0
Dlib	440	581	17	40
Fawkes	39	1,968	31	0
Google	8	406	18	0
ICU	35	7,566	62	0
KDE	8	28	0	0
libc++	19,012	1,722	20	109
libMesh	959	1,185	0	0
LLVM	1,855	1,308	58	130
mangos	166	7,690	48	0
Mantid	907	4,993	47	29
MITK	840	5,241	84	4
MongoDB	2,401	32,857	287	307
OpenMS	128	1,660	2	2
Orfeo	168	7,073	64	0
Point	382	12,472	126	1
xLights	32	1,898	32	0

TABLE III: Comparing the usage of language features with *if* and *switch* statements

features to compare the relative numbers. `if` and `switch` statements have been selected [21]. The `If` statement is the most popular feature in projects from GitHub repositories while the `switch` statement is less popular feature [21]. Table III shows the numbers of language features (i.e., Template, If, Switch, and Lambda) used in the projects. The number of templates used by the *Appleseed* project is 3,930. The *libc++* project is the highest number with 19,012 which is quite higher than the number of `if` statement with 1,722. The template feature can be one of popular features since the numbers of templates have a similar relationship with `If` statement. **Overall, our results suggest that templates are widely used in open source projects.**

Lambdas: The numbers of projects using lambdas are much smaller than the numbers of project using templates. 10 projects (out of 20 projects) used lambdas. *MongoDB* used the most lambdas with 307 and *LLVM* used the second most lambdas with 130 in 10 projects. Figure 1(b) shows the number of lambdas over time in *MongoDB*. As mentioned earlier, `switch` statements are a less popular feature [21]. When lambdas are compared with `switch` statements, the number of lambdas is less than the number of switch statements for most of the projects except *Dlib*, *libc++*,

Project	All	None	Template	Template only	Intersection	Lambda only	Lambda
Appleseed	39	16 (41.0 %)	23 (59.0 %)	20	3	0	3 (7.7 %)
Boost	19	4 (21.1 %)	15 (78.9 %)	14	1	0	1 (5.3 %)
deal	10	7 (70.0 %)	3 (30.0 %)	3	0	0	0 (0.0 %)
digikam	140	118 (84.3 %)	22 (15.7 %)	22	0	0	0 (0.0 %)
Dlib	25	15 (60.0 %)	10 (40.0 %)	3	7	0	7 (28.0 %)
Fawkes	25	21 (84.0 %)	4 (16.0 %)	4	0	0	0 (0.0 %)
Google	7	5 (71.4 %)	2 (28.6 %)	2	0	0	0 (0.0 %)
ICU	72	52 (72.2 %)	20 (27.8 %)	20	0	0	0 (0.0 %)
KDE	47	40 (85.1 %)	7 (14.9 %)	7	0	0	0 (0.0 %)
libc++	102	22 (21.6 %)	78 (76.5 %)	60	18	2	20 (19.6 %)
libMesh	4	1 (25.0 %)	3 (75.0 %)	3	0	0	0 (0.0 %)
LLVM	181	92 (50.8 %)	88 (48.6 %)	58	30	1	31 (17.1 %)
mangos	45	18 (40.0 %)	27 (60.0 %)	27	0	0	0 (0.0 %)
Mantid	137	53 (38.7 %)	83 (60.6 %)	52	31	1	32 (23.4 %)
MITK	218	140 (64.2 %)	77 (35.3 %)	75	2	1	3 (1.4 %)
MongoDB	283	89 (31.4 %)	171 (60.4 %)	67	104	23	127 (44.9 %)
OpenMS	60	30 (50.0 %)	30 (50.0 %)	27	3	0	3 (5.0 %)
Orfeo	33	23 (69.7 %)	10 (30.3 %)	10	0	0	0 (0.0 %)
Point	244	162 (66.4 %)	82 (33.6 %)	81	1	0	1 (0.4 %)
xLights	31	20 (64.5 %)	11 (35.5 %)	11	0	0	0 (0.0 %)
SUM	1,722	928 (53.9 %)	766 (44.5 %)				228 (13.2 %)

TABLE IV: Analysis of Developers

LLVM, and *MongoDB*. **Overall, our results suggest that lambdas are not widely used in open source projects**. Similar results were found in Java lambda expression [22]. Uesbeck *et al.* observed that lambdas don't benefit developers in terms of time to completion, or compiler errors [11]. However, as Mazinanian *et al.* found an increasing trend in the adoption rate of lambdas [13], the adoption rate of C++ lambdas also shows an increasing trend in some projects which used lambdas 1(b). This indicates that if we analyze C++ lambda expression again in a decade, the answer may be changed.

B. RQ2: Who used Language Features

RQ2: Do project members in each project broadly use a language feature after introduction into the project?
Templates: As the answer to RQ1 , C++ templates are used by most projects. However, few developers may take major usage of templates or vice versa. To evaluate RQ2, we examined how many developers used templates. Table IV shows the number of developers who used templates and lambdas. Each title in Table IV represents:

- **All**: the total number of developers involved in the project
- **None**: the number of developers who didn't use both template and lambda
- **Template (Lambda)**: the number of developers who used templates (lambdas)
- **Template (Lambda) only**: the number of developers who only used templates (lambdas)
- **Intersection**: the number of developers who used both templates and lambdas

In SUM in Table IV, 44.5% (766 out of 1,722) developers used templates. More than 50% developers used templates in 8 projects. *Boost* is the highest project with 78.9% (15 out of 19) of developers using templates. *libc++* is the second highest project with 76.5% (78 out of 102) of developers using templates. These facts indicate that most developers understand the benefits of the template feature. *digikam* is the lowest project with 15.7% (22 out of 140) of developers using

templates. *Fawkes* is the second lowest project with 16.0% (4 out of 25) of developers using templates. In 13 projects, more than 50% developers did not use both templates and lambdas.

Figure 2 shows the introduction and removal of both templates and lambda expressions by the top 5 developers per project. Top 5 developers means 5 developers using the most templates in a project. A dashed line represents the number of templates while a solid line represents the number of lambdas. We observed that one or two developers show higher usage of templates in the projects. For example, Figure 2(a) shows that two developers used significantly higher templates than other developers; *Al**** used more than 300 templates and *Jo**** used more than 150 templates. In the *appleseed* project which is not in Figure 2, one developer (*Fr****) used around 3,700 templates (out of 3,930) while other developers used less than 100 templates. This pattern was observed in many of the other projects such as *appkeseed*, *Boost*, *digikam*, *dlib*, *Fawkes*, *ICU*, *libMesh*, and *Orfeo*. **Overall, our results indicates that templates are used by a small pool of developers.** These results are similar to other previous studies [5], [14], [17].

Lambdas: As can be seen by the analysis results in RQ1, we recognize that not many project members used lambdas. Table IV shows that 13.2% (228 out of 1,722) used lambdas. In *LLVM* (Figure 2(a)), *Al**** (who used the most templates) used the most lambda. *Jo**** (who used the second most templates) used the second most lambdas. In *MongoDB*, 44.9% (127 out of 283) used lambdas. Several developers (*Be****, *Ma****, *Ka****) used almost 100 templates; *Be**** used 75 lambdas. In most projects, developers who used lambdas also used templates, except *MongoDB*; In *MongoDB*, 23 developers only used lambdas, not templates. **Overall, C++ lambdas are used by a very small number of project members.** Nine years have passed since lambda expression was added in C++ in 2011. We need to look at whether each project uses C++11 compiler (or higher version) to enable lambdas and when each project begins to use C++11 compiler, but for now only a few developers use C++ lambdas.

| (a) LLVM | (b) MongoDB |
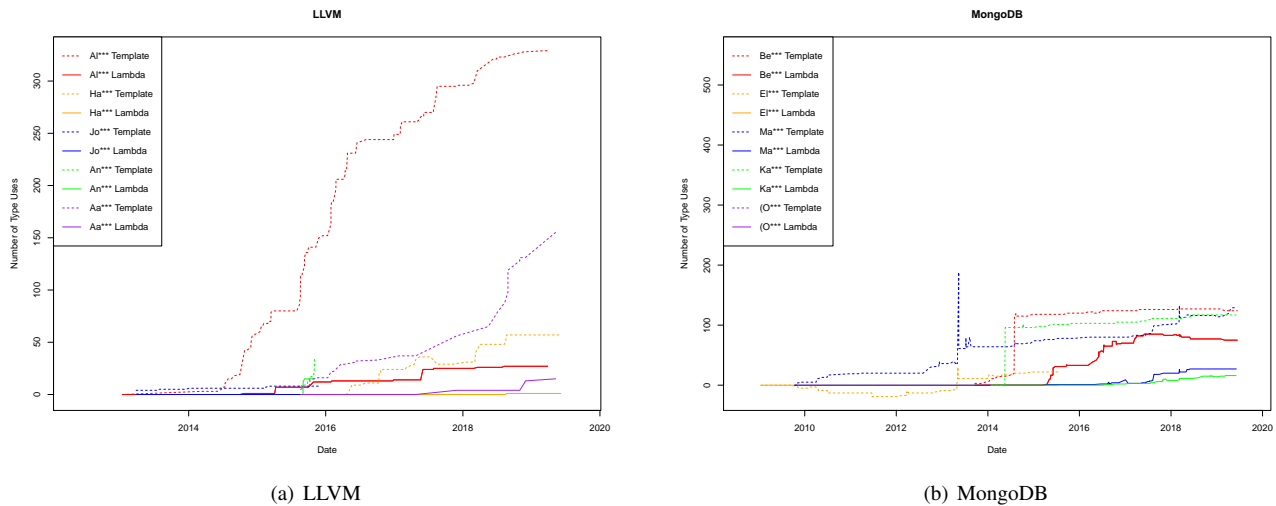
Fig. 2: Individual developers' usage of templates and lambdas over time

V. CONCLUSION

Template and lambda expression features have benefits. Throughout the empirical study with C++ open source projects, we investigated the usage of templates and lambdas and how these features are used by developers. We found that templates are used widely by many developers in open source projects. However, lambdas are not used widely in open source projects, and not many developers use lambdas. These results from quantitative data will help you create a new programming language, add new language features into the programming language, or teach programming with what priorities. Since language features were added at different times, we recognize that quantitative comparisons are limited. Further research with qualitative data may be needed to analyze the reasons for these conclusions.

REFERENCES

[1] G. Bracha, N. Cohen, C. Kemper, S. Marx, M. Odersky, S.-E. Panitz, D. Stoutamire, K. Thorup, and P. Wadler, "Adding generics to the java programming language: Participant draft specification," 2001.

[2] J. Järvi, J. Freeman, and L. Crowl. (2008, Feb) Lambda functions and closures for C++ (Revision 4), Technical Report N2550=08-0060, ISO/IEC JTC 1, Information technology, Subcommittee SC 22, Programming Language C++. [Online]. Available: http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2008/n2550.pdf

[3] B. Stroustrup, "Evolving a language in and for the real world: C++ 1991-2006," in *Proceedings of the third ACM SIGPLAN conference on History of programming languages*. ACM, 2007, pp. 4–1.

[4] R. Dyer, H. Rajan, H. A. Nguyen, and T. N. Nguyen, "Mining billions of ast nodes to study actual and potential usage of java language features," in *Proceedings of the 36th International Conference on Software Engineering*. ACM, 2014, pp. 779–790.

[5] D. Kim, E. R. Murphy-Hill, C. Parnin, C. Bird, and R. Garcia, "The reaction of open-source projects to new language features: An empirical study of c# generics." *Journal of Object Technology*, vol. 12, no. 4, pp. 1–1, 2013.

[6] C. Liddell and D. Kim, "Analyzing the adoption rate of local variable type inference in open-source java 10 projects," *Journal of the Arkansas Academy of Science*, vol. 73, no. 1, pp. 51–54, 2019.

[7] D. Kim and G. Yi, "Measuring syntactic sugar usage in programming languages: an empirical study of c# and java projects," in *Advances in Computer Science and its Applications*. Springer, 2014, pp. 279–284.

[8] M. H. Austern, *Generic programming and the STL: using and extending the C++ Standard Template Library*. Addison-Wesley, 1999.

[9] Á. Sinkovics and Z. Porkoláb, "Implementing monads for c++ template metaprograms," *Science of computer programming*, vol. 78, no. 9, pp. 1600–1621, 2013.

[10] J. Järvi and J. Freeman, "C++ lambda expressions and closures," *Science of Computer Programming*, vol. 75, no. 9, pp. 762–772, 2010.

[11] P. M. Uesbeck, A. Stefik, S. Hanenberg, J. Pedersen, and P. Daleiden, "An empirical study on the impact of c++ lambdas and programmer experience," in *Proceedings of the 38th International Conference on Software Engineering*. ACM, 2016, pp. 760–771.

[12] D. W. 0014, L. Chen, Y. Zhou, and B. Xu, "An empirical study on the adoption of c++ templates: Library templates versus user defined templates." in *SEKE*, 2014, pp. 144–149.

[13] D. Mazinanian, A. Ketkar, N. Tsantalis, and D. Dig, "Understanding the use of lambda expressions in java," *Proceedings of the ACM on Programming Languages*, vol. 1, no. OOPSLA, p. 85, 2017.

[14] L. Chen, D. Wu, W. Ma, Y. Zhou, B. Xu, and H. Leung, "How c++ templates are used for generic programming: An empirical study on 50 open source systems," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 29, no. 1, pp. 1–49, 2020.

[15] A. P. Black, K. B. Bruce, M. Homer, and J. Noble, "Grace: the absence of (inessential) difficulty," in *Proceedings of the ACM international symposium on New ideas, new paradigms, and reflections on programming and software*. ACM, 2012, pp. 85–98.

[16] C. Saldivar, R. Clayton, and D. Kim, "The adoption rate of lambda expressions in java open source projects," *31st Annual National Conference on Undergraduate Research*, 2017.

[17] C. Parnin, C. Bird, and E. Murphy-Hill, "Java generics adoption: how new features are introduced, championed, or ignored," in *Proceedings of the 8th Working Conference on Mining Software Repositories*. ACM, 2011, pp. 3–12.

[18] M. Asaduzzaman, M. Ahasanuzzaman, C. K. Roy, and K. A. Schneider, "How developers use exception handling in java?" in *Proceedings of the 13th International Conference on Mining Software Repositories*. ACM, 2016, pp. 516–519.

[19] S. Nielebock, R. Heumüller, and F. Ortmeier, "Programmers do not favor lambda expressions for concurrent object-oriented code," *Empirical Software Engineering*, vol. 24, no. 1, pp. 103–138, 2019.

[20] J. Siek and W. Taha, "A semantic analysis of c++ templates," in *European Conference on Object-Oriented Programming*. Springer, 2006, pp. 304–327.

[21] M. J. Lemay, "Understanding java usability by mining github repositories," in *9th Workshop on Evaluation and Usability of Programming Languages and Tools (PLATEAU 2018)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2019.

[22] C. Mcdougal, B. Staufer, J. Reach, and D. Kim, "The evolution of java involving lambda," *Fifteenth Annual Consortium for Computing Sciences in Colleges Mid-South Conference In Cooperation With ACM/SIGCSE*, 2017.

Analyzing the Performance of Apps Developed by using Cross-Platform and Native Technologies

Lucas Pugliese Barros
Informatics Coordination
Federal Institute of Alagoas
Maceió, Alagoas, Brasil
lucas.pugliese.barros@gmail.com

Flávio Medeiros
Informatics Coordination
Federal Institute of Alagoas
Maceió, Alagoas, Brasil
flavio.medeiros@ifal.edu.br

Eduardo Moraes
Informatics Coordination
Federal Institute of Alagoas
Maceió, Alagoas, Brasil
ecmoraes@gmail.com

Anderson Feitosa Júnior
Informatics Coordination
Federal Institute of Alagoas
Maceió, Alagoas, Brasil
andersonmfjr@gmail.com

Abstract—The number of mobile devices are increasing worldwide and there are a number of new mobile apps being delivered daily. When developing mobile applications, developers need to support a number of platforms, such as *iOS* and *Android*. Many cross-platform technologies appeared, including *Flutter* and *React Native*, to avoid the need of developing different applications for every platform. In this context, companies and developers have to analyze different issues when selecting a cross-platform or native technology. Both strategies have positive and negative points, and one important aspect when choosing a technology is performance. In this study, we perform a comparative study to analyze the performance of mobile applications developed by using *Flutter*, *React Native*, and *iOS* and *Android* native technologies. The results show that native technologies are still a little faster for most functionalities, but there are also a number of cases in which *Flutter* and *React Native* perform statically equivalent when compared to native technologies. Our study complements previous work by including these two modern cross-platform technologies that have not be considered in comparative studies previously.

Index Terms—Mobile Development, Performance Analysis

I. INTRODUCTION

Mobile devices are currently a crucial part of our daily life. We find mobile applications for a wide range of domains, such as health, tourism, planning, and sports. There are many a number of platforms that uses different Software Development Kits (SDK) to build mobile applications, including *iOS*, and *Android*. In this context, developers should provide their solutions across those platforms. A number of cross-platform technologies appeared, allowing developers to support different platforms by using a single source code, such as *React Native*, and *Flutter* [1–4].

In particular, when using native development, companies need to develop different mobile applications for every supported platform. Thus, the cost of native development gets higher. For this reason, cross-platform development has emerged as a potential alternative. On the other hand, cross-platform technologies also have negative points [3, 5, 6]:

- Performance: it is one of the most important requirement of an application, and it might be slower in applications developed by using cross-platform technologies;

- Harder code design: developers have to adapt their design and functionalities to handle the specific peculiarities of every platform;
- Longer time for new features: every new feature for *Android* or *iOS* takes some time to be available in cross-platform frameworks.

To help developers and companies to select a technology for their mobile applications, we performed a comparative study to analyze the performance of cross-platform applications developed by using *Flutter* and *React Native* when compared to native applications written in *Java* and *Swift*, for *Android* and *iOS* devices respectively. This study complements previous work by including two modern cross-platform technologies not considered before [6–8]. The first on is *Flutter*, which has appeared as a promising cross-platform technology that aims to bundle mobile applications with performance improvements. This technology has been used by many companies, including *Alibaba*, and *Google Ads*. The second is *React Native*, which is the most popular technology currently and it is also used by many enterprises, such as *Facebook*, *Airbnb*, *Tesla*, *Walmart*, and *Uber*.

To perform this comparative study, we define a set of functionalities that are present in most mobile applications. Then, we implemented these functionalities by using different languages and technologies: (1) in *Flutter* by using the *Dart* language; (2) in *JavaScript* with *React Native*; (3) in *Java* for *Android* devices; and (4) in *Swift* for *iOS* devices. In all implementations we use the same functionalities and layout, and we strictly follow the documentation of each technology to get the best benefits of each one. The functionalities implemented include storing and retrieving information of local storage, making *HTTP* calls, and rendering data as lists. To answer our research questions, we use User Interface (UI) Test [9, 10] to automatically run every implementation and execute each functionality 100 times to compute the time of execution in the different technologies.

Our study reveals that the applications developed by using native technologies, i.e., *Swift* and *Java*, are still a little faster for most functionalities. However, there are also cases in which *Flutter* and *React Native* are statistically equivalent in terms of performance, such as storing and retrieving data by using local storage in *Android* and *iOS* devices.

The remainder of this paper is organized as follows. In Section II, we present the settings of our comparative study to analyze the performance of mobile applications. Section III shows the results of our study with regards to *Flutter*, *React Native*, and *iOS* and *Android* development. In Section IV, we depict the threats to validity of our study, and Section V discusses some implications to practice. In Section VI, we present the related work, and we discuss the concluding remarks in Section VII.

We provide all information about this comparative study, including the source codes, and the result data in a complementary website.[1]

II. STUDY SETTINGS

In this section, we present the settings of our comparative study to analyze the performance of mobile applications developed by using cross-platform and native technologies. To better structure our study, we use the Goal, Question, and Metrics (GQM) approach [11].

A. Definition

The goal of this comparative study is to analyze implementations of a mobile application for the purpose of evaluation with respect to verifying the performance of cross-platform and native mobile technologies in the context of the languages *Dart* (*Flutter*), *JavaScript* (*React Native*), *Java*, and *Swift*. In particular, this study addresses the following research questions:

- **RQ1.** What is the difference in terms of performance for *Android* applications developed by using *Java*, *Flutter* (*Dart*), and *React Native* (*JavaScript*)?
- **RQ2.** What is the difference in terms of performance for *iOS* applications developed by using *Swift*, *Flutter* (*Dart*), and *React Native* (*JavaScript*)?

To answer these two research questions included in our study, we define four metrics:

- **REMOTE:** this metric computes the processing time to make a request to an external Application Programming Interface (API). It computes the time of the *HTTP* request, that is, the time to make the request and receive all data by using the JavaScript Object Notation (JSON);
- **RENDER:** it computes the time to render the first five items of a list by showing them on the screen for the users;
- **STORE:** this metric computes the time to save one item of a list in the local database;
- **RETRIEVE:** it computes the time necessary to retrieve the information of five items from the local database.

B. Planning and Operation

In this section, we describe the subjects, instrumentation, and operation of our study.

Our study uses *Flutter* (Dart) and *React Native* (JavaScript) as the cross-platform technologies, and *Java* and *Swift* as

---

[1]http://cpsoftware.com.br/performance-study

---

the natives ones. The reason to select *Flutter* is because it has appeared as a promising cross-platform technology that aims to bundle mobile applications with performance improvements. As performance is a negative point of cross-platform applications, it makes sense to include *Flutter* in our study. We choose *React Native* because it is the most common framework in practice. In addition, both frameworks are used by known companies, such as *Alibaba*, *Facebook*, *Airbnb*, *Walmart*, and *Uber*. We select *Android* and *iOS* because they are the two most common platforms for mobile devices.

To perform this comparative study, we developed four implementations of an application by using different technologies: (1) *Flutter*; (2) *React Native*; (3) *Java*; and (4) *Swift*. The implementations contain the same functionalities and layout, and we developed the applications by strictly following the documentation of each technology.

To compute metric *REMOTE*, we select the *Google APIs Explorer* service because there is no authentication restriction to access it, and because it is possible to return a considerable amount of information (258 items) through a single request. As the local database, we use the *SQLite* database because it is compact as well as available in the *Android*, *IOS*, *Flutter*, and *React Native* SDKs. In addition, the documentations of the technologies recommend to use this database in practice.

In Figure 1, we can see one implementation of the application written in *Swift* for *iOS*. On the left-hand side, we see the list of items returned by the API. On the right-hand side, we present the items retrieved from the *SQLite* database. In Figure 2, we show the screens but considering the implementation for *Android*, written in *Java*.

Fig. 1. Interface of one implementation of the application in *iOS*.

To answer our research questions, we created user interface tests to execute the applications 100 times and to compute the time to perform each functionality in the different technologies. The user interface tests repeat a sequence of steps 100 times on real devices, simulating the behavior of users. Thus, instead of run the experiment manually, it was possible to automate the complete process.

To compute the processing time for each metric, we create a class in *Java*, *Swift*, *JavaScript*, and *Dart*, respectively for *Android*, *iOS*, *React Native*, and *Flutter*. So that, the logic was the same for all four implementations with the purpose of avoiding influences on the processing time of the metrics. These classes are responsible for managing all metrics, calculating the processing time in milliseconds, and formatting and printing the processing time of the metrics. With the data collected, the next step was to format the data in the Comma Separated Values (CSV) format to run the statistics by using the *R Project for Statistical Computing*.

We execute the study on a MacBook Pro 2.4GHz dual-core Intel Core i5 8GB, running Mac OS X 10.8 Mountain Lion. Furthermore, we run the applications by using an *iPhone* 7 32GB 3GB RAM and a *Samsung Galaxy* S8 64GB 4GB RAM. That is, we run the *Android* native application and the *Flutter* and *React Native* versions for *Android* by using the *Samsung Galaxy*, and the *iOS* native application and the *Flutter* and *React Native* versions for *iOS* by using the *iPhone*. We perform all the analyses by using the same network connection.

To run the experiment, we need several tools. We use *Android Studio* 3.6, *Android* 10, *Java* 1.8, *Flutter* version 1.12.13, *Xcode* 11.3.1, *Visual Studio Code* 1.42, *React Native* 0.61, *Volley* 1.1.1, *SQLite* 3.31.1, *Espresso* 3.1.1, *XCTest* 5.2, and *Detox* 15.4.2.
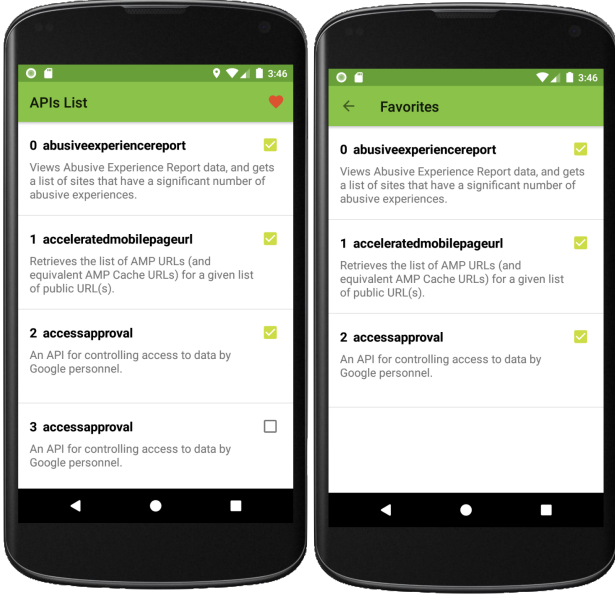
Fig. 2. Interface of one implementation of the application in *Android*.

Next, we interpret and discuss the results of this study to analyze the performance of mobile applications developed by using cross-platform and native technologies.

III. RESULTS

In the next subsections, we present the results of our comparative study. Section III-A presents the results related to the *Android* platform, and in Section III-B, we show the results of the *iOS* platform.

A. Android

In this section, we answer **RQ1** by presenting the results for each metric by considering the applications running on the *Android* device.

REMOTE: In Figure 3a, we show the mean and the confidence interval (95%) for each technology regarding metric *REMOTE*. The means are 1772, 1818, and 2295 milliseconds for *React Native*, *Android* native, and *Flutter* respectively.

To check if the data is normal distributed, we use the *Shapiro-Wilk* normality test (95%). The null-hypothesis of this test is that the population is normally distributed. Thus, if the *p-value* is less than the chosen alpha level (0.05), then the null hypothesis is rejected and there is evidence that the data tested are not normally distributed [12, 13].

- Null Hypothesis (H_0): it tests if the population is normally distributed;
- Alternative Hypothesis (H_1): there is evidence that the population is not normally distributed.

By running the *Shapiro-Wilk* normality test (95%), we find that the data is not normal. Thus, we compare the groups of data by using the *Wilcoxon* test [12, 13], which considers:

- Null Hypothesis (H_0): there is not any difference in the data sets, that is, the medians are equal;
- Alternative Hypothesis (H_1): there is evidence that the data sets are different in terms of median.

The results show that the performances of *Android* native and *React Native* are statistically faster than the performance of the application developed by using *Flutter*. That is, the median values are different according to the statistical test. In Table I, we present the statistical tests we run and their respective results.

Test	p-value	(H_0)
Shapiro-Wilk for *Android*	$9.9e^{-13}$	rejected
Shapiro-Wilk for *Flutter*	$2.9e^{-08}$	rejected
Shapiro-Wilk for *React Native*	$5.7e^{-11}$	rejected
Wilcoxon for *Android* and *Flutter*	$2.3e^{-13}$	rejected
Wilcoxon for *Android* and *React*	$6.9e^{-04}$	rejected
Wilcoxon for *Flutter* and *React*	$2.2e^{-16}$	rejected

TABLE I
STATISTICAL TESTS FOR METRIC REMOTE IN ANDROID APPLICATIONS.

RENDER: In Figure 3b, we present the mean and the confidence interval (95%) for each technology regarding metric *RENDER*. The means are 173, 247, and 277 for *Android* native, *Flutter*, and *React Native* respectively.

The data is not normal here also according to the *Shapiro-Wilk* test, as we can see in Table II. *Android* native is the technology with better performance with statistical significance. It has the lowest median, which is different from the other data sets according to the *Wilcoxon* test. On the other hand, there is no statistical difference between the performance of the technologies *React Native* and *Flutter*.

STORE: In Figure 3c, we show the results for metric *Store*. We present the mean and the confidence interval (95%) for
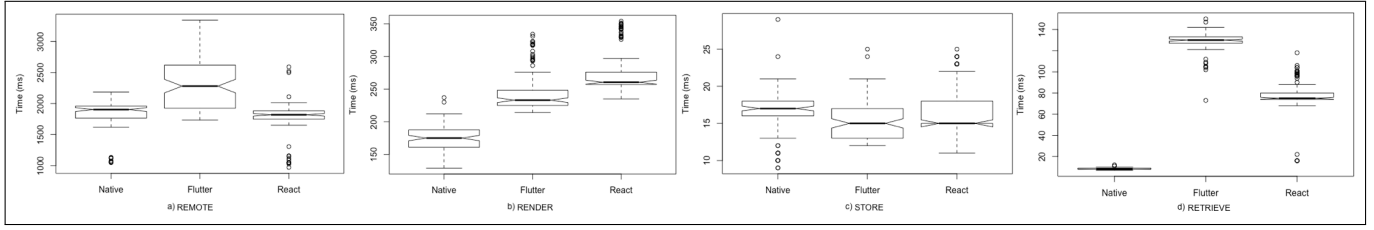
Fig. 3. Results for metrics in *Android*.

Test	p-value	(H₀)
Shapiro-Wilk for *Android*	0.3	accepted
Shapiro-Wilk for *Flutter*	$9.2e^{-12}$	rejected
Shapiro-Wilk for *React Native*	$1.1e^{-12}$	rejected
Wilcoxon for *Android* and *Flutter*	$2.2e^{-16}$	rejected
Wilcoxon for *Android* and *React*	$2.2e^{-16}$	rejected
Wilcoxon for *Flutter* and *React*	$1.3e^{-14}$	rejected

TABLE II
STATISTICAL TESTS FOR METRIC RENDER IN ANDROID APPLICATIONS.

each technology. The means are 15.59, 16.43, and 16.96 for *Flutter*, *Android* native, and *React Native* respectively.

In Table III. we present the statistical tests and their results. There is no statistical significance in the data sets. That is, the performance of all technologies is statistically equivalent.

Test	p-value	(H₀)
Shapiro-Wilk for *Android*	$2.97e^{-10}$	rejected
Shapiro-Wilk for *Flutter*	$2.486e^{-5}$	rejected
Shapiro-Wilk for *React Native*	$4.038e^{-10}$	rejected
Wilcoxon for *Android* and *Flutter*	0.003335	rejected
Wilcoxon for *Android* and *React*	0.03211	rejected
Wilcoxon for *Flutter* and *React*	0.0096	rejected

TABLE III
STATISTICAL TESTS FOR METRIC LIKE IN ANDROID APPLICATIONS.

RETRIEVE: In Figure 3d, we show the results for metric *Retrieve Data*. We present the mean and the confidence interval (95%) for each technology. The means are 8.35, 77, and 128.6 for *Android* native, *React Native*, and *Flutter* respectively.

In Table IV, we present the results of metric *RETRIEVE*. The *Android* native technology is the fastest one with statistical significance. On the other hand, the performance of the technologies *React Native* and *Flutter* is statically equivalent.

Test	p-value	(H₀)
Shapiro-Wilk for *Android*	$6.24e^{-12}$	rejected
Shapiro-Wilk for *Flutter*	$1.074e^{-11}$	rejected
Shapiro-Wilk for *React Native*	$7.78e^{-13}$	rejected
Wilcoxon for *Android* and *Flutter*	$2.2e^{-16}$	rejected
Wilcoxon for *Android* and *React*	$2.2e^{-16}$	rejected
Wilcoxon for *Flutter* and *React*	$2.2e^{-16}$	rejected

TABLE IV
STATISTICAL TESTS FOR METRIC LIKE IN ANDROID APPLICATIONS.

SUMMARY

Regarding data rendering and retrieving data from local storage, Android native is faster than Flutter and React Native. However, performance is statistically equivalent when considering accessing remote data and storing data in local storage.

B. iOS

In this section, we answer **RQ2** by showing the results of our experiment in *iOS*. Next, we present the results for each metric.

REMOTE: In Figure 4a, we show the mean and the confidence interval (95%) for each technology regarding metric *REMOTE*. The means are 45.35, 1,802.46, and 2,474.89 milliseconds for *iOS* native, *React Native*, and *Flutter* respectively.

In Table V, we present the statistical tests and their results. In this metric, *iOS* native is statically faster than *Flutter* and *React*. Further, *React* is statistically faster than *Flutter*.

Test	p-value	(H₀)
Shapiro-Wilk for *Android*	0.002446	rejected
Shapiro-Wilk for *Flutter*	$7.975e^{-11}$	rejected
Shapiro-Wilk for *React Native*	$6.888e^{-15}$	rejected
Wilcoxon for *Android* and *Flutter*	$2.2e^{-16}$	rejected
Wilcoxon for *Android* and *React*	$2.2e^{-16}$	rejected
Wilcoxon for *Flutter* and *React*	$5.805e^{-12}$	rejected

TABLE V
STATISTICAL TESTS FOR METRIC REMOTE IN *i*OS APPLICATIONS.

RENDER: In Figure 4b, we present the mean and the confidence interval (95%) for each technology regarding metric *RENDER*. The means are 15.79, 56.71, and 72.91 for *iOS* native, *React Native*, and *Flutter* respectively.

In Table VI, we present the statistical tests and their results. In this metric, *iOS* native is also statically faster than *Flutter* and *React*. Further, *React* is statistically faster than *Flutter*.

Test	p-value	(H₀)
Shapiro-Wilk for *Android*	$7.206e^{-15}$	rejected
Shapiro-Wilk for *Flutter*	$2.2e^{-16}$	rejected
Shapiro-Wilk for *React Native*	$2.042e^{-15}$	rejected
Wilcoxon for *Android* and *Flutter*	$2.2e^{-16}$	rejected
Wilcoxon for *Android* and *React*	$2.2e^{-16}$	rejected
Wilcoxon for *Flutter* and *React*	$2.2e^{-16}$	rejected

TABLE VI
STATISTICAL TESTS FOR METRIC RENDER IN *i*OS APPLICATIONS.

STORE: In Figure 4c, we show the results for metric *STORE*. We present the mean and the confidence interval (95%) for each technology. The means are 2.89, 5.73, and 6.23 for *React Native*, *iOS* native, and *Flutter* respectively.

In Table VII, we present the statistical tests and their results. In this metric, *React Native* is statically faster than *iOS* Native and *Flutter*.
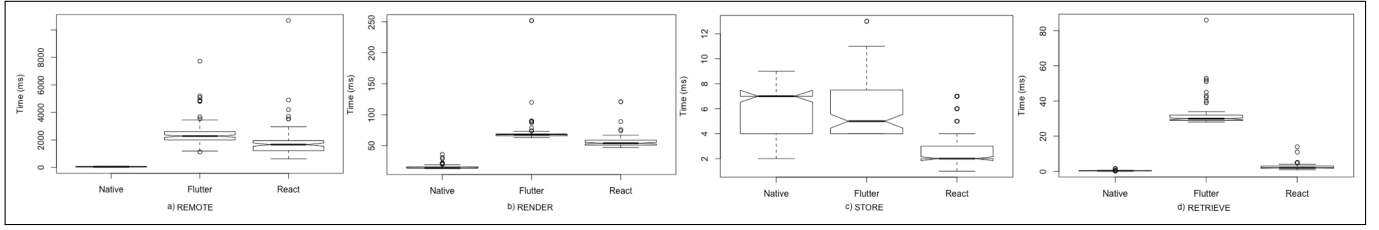
Fig. 4. Results for metrics in *iOS*.

Test	p-value	(H$_0$)
Shapiro-Wilk for *Android*	$1.346e^{-11}$	rejected
Shapiro-Wilk for *Flutter*	$4.202e^{-08}$	rejected
Shapiro-Wilk for *React Native*	$1.94e^{-13}$	rejected
Wilcoxon for *Android* and *Flutter*	0.5488	rejected
Wilcoxon for *Android* and *React*	$2.2e^{-16}$	rejected
Wilcoxon for *Flutter* and *React*	$2.2e^{-16}$	rejected

TABLE VII

STATISTICAL TESTS FOR METRIC RENDER IN *i*OS APPLICATIONS.

RETRIEVE: In Figure 4d, we show the results for metric *RETRIEVE*. We present the mean and the confidence interval (95%) for each technology. The means are 0.38, 2.82, and 32.18 for *Android* native, *React Native*, and *Flutter* respectively.

Here, the hypotheses of all statistical tests were rejected with $p-value = 2.2e^{-16}$. It is the lowest number that R can represent, that is, the null hypotheses (H$_0$) are all rejected with a small *p-value*, as shown in Table VIII.

Test	p-value	(H$_0$)
Shapiro-Wilk for *Android*	$2.2e^{-16}$	rejected
Shapiro-Wilk for *Flutter*	$2.2e^{-16}$	rejected
Shapiro-Wilk for *React Native*	$2.2e^{-16}$	rejected
Wilcoxon for *Android* and *Flutter*	$2.2e^{-16}$	rejected
Wilcoxon for *Android* and *React*	$2.2e^{-16}$	rejected
Wilcoxon for *Flutter* and *React*	$2.2e^{-16}$	rejected

TABLE VIII

STATISTICAL TESTS FOR METRIC RENDER IN *i*OS APPLICATIONS.

SUMMARY

Regarding data rendering, accessing remote data and retrieving data from local storage, iOS native is faster than Flutter and React Native. However, performance is statistically equivalent or faster when considering storing data into local storage.

IV. THREATS TO VALIDITY

Construct Validity. It refers to whether the functionalities that we choose are indeed relevant for mobile applications. We minimize this threat by analyzing existing mobile applications. The functionalities that we select, i.e., call remote API, render items as lists, and store and retrieve data from database, are present in almost all applications that we analyzed in our study.

Internal Validity. In our study, we implemented a specific class in different languages, i.e., *Java*, *Swift*, *JavaScript*, and *Dart*, to measure the time of performing the functionalities in the exact way for each technology. Furthermore, we select a stable internet connection to avoid differences when running

the applications in the different technologies. However, notice that we may still face network differences. To minimize this threat, we run the experiment for all technologies in an interval of three hours.

External Validity. We analyze four functionalities that we commonly find in existing applications. However, we have not considered functionalities that use resources, such as GPS and camera. Thus, we cannot generalize the results of this study to this context.

V. IMPLICATIONS TO PRACTICE

The difference in terms of performance among *React Native* and *Flutter* when compared to native technologies can be explained based on how these cross-platform technologies work. In *React Native*, the source code is not compiled to *C/C++* or other native languages. Instead, the user interface components are compiled into native equivalents and *JavaScript* is executed on a separate thread (called bridge). In *Flutter*, the source code is compiled to native language directly, but it also uses an engine that goes together with the source code of the mobile applications, and a platform channel is necessary to access native resources, such as camera and GPS. That is, they work as intermediates that may decrease performance also. However, these technologies are evolving, they are used in practice by many companies, and they can perform as fast as native technologies for specific functionalities, such as storing and retrieving data by using local storage, as we shown in our comparative study.

Another important point to take into account is the costs for development. When using native technologies, such as *Java* and *Swift*, developers need to develop the same application for every supported platform, making the development more expensive and time-consuming. Thus, when performance requirements are not extremely important, it makes sense to use cross-platform technologies, as their performance is closer to natives ones to most functionalities, as we have shown in our current study.

VI. RELATED WORK

In this section, we compare our study with previous work. Sommer and Krusche [8] performed a comparative study with a number of technologies, including *Titanium*, *Rhodes*, and *PhoneGap*. The authors recommend to use cross-platform technologies in general, but they alert for high requirements with regards to performance issues, usability or native user experience. In [7], the authors compared the same three cross-platform technologies and measured performance in terms of

memory, CPU usage and power consumption. The former provided a similar result when comparing to our study in terms of performance. In the latter study, the authors used a different strategy to measure performance, as we computed the time of execution in our study.

Heitkötter et al. [14] performed a study by comparing the *PhoneGap* and *Titanium Mobile* technologies. The authors compared the technologies by considering the native look and feel, supported platforms, license and costs, as well as the application speed at start-up and run-time. Different from *React Native* and *Flutter*, the technologies used do not generate native code, they use a *Web app* approach knows as *WebView* [15], which let the application slower, as discussed by the authors in their results. Furthermore, technologies that use *WebViews* may cause security issues [16].

In the study of Xiaoping et al. [6], the authors compared the *Apache Cordova*, *Microsoft Xamarin*, and *Appcelerator Titanium* against the native technologies, that is, *Android* and *iOS*. The study is similar to ours, it presents trade-offs of different technologies and offer guidance in selecting an appropriate technology based on performance requirements. Further, the results are similar with our results, but the results of *Flutter* and *React Native* seem to be closer to the performance of native technologies.

In [17], the authors discussed the different strategies used by cross-platform technologies, and mention the advantages in productivity when using a model-driven approach. Our study complements all these related work by considering two modern cross-platform technologies for mobile development.

VII. CONCLUSIONS

In this study, we present a comparative study to analyze the performance of cross-platform and native technologies for mobile development. To run this study, we developed the same application by using different technologies: *Android (Java)*, *iOS (Swift)*, *Flutter (Dart)*, and *React Native (JavaScript)*. Our results reveal that native technologies are faster, as we expected, but the performance of the current cross-platform technologies are pretty closer. In a number of cases, we could not show statistical different among the performance of the applications written in different technologies. For instance, when storing and retrieving data by using local storage in *Android* and *iOS* devices. As future work, we plan to implement more functionalities in the applications to run the study again considering different aspects, such including functionalities that use geolocation, camera, and accelerometer.

REFERENCES

1. Latif, M., Lakhrissi, Y., Nfaoui, E. H. & Es-Sbai, N. *Cross platform approach for mobile application development: A survey* in *International Conference on Information Technology for Organizations Development* (IEEE, 2016), 1–5.
2. Palmieri, M., Singh, I. & Cicchetti, A. *Comparison of cross-platform mobile development tools* in *International Conference on Intelligence in Next Generation Networks* (IEEE, 2012), 179–186.
3. Javeed, A. *Performance Optimization Techniques for ReactJS* in *Int. Conf. on Electrical, Computer and Communication Technologies* (IEEE, 2019), 1–5.
4. Serrano, N., Hernantes, J. & Gallardo, G. Mobile Web Apps. *IEEE Software* **30,** 22–27 (2013).
5. Lin, H. & Lee, G. *Building a Secure Cross Platform Enterprise Service Mobile Apps Using HTML5* in *International Conference on Network-Based Information Systems* (IEEE, 2015), 162–166.
6. Jia, X., Ebone, A. & Tan, Y. *A Performance Evaluation of Cross-Platform Mobile Application Development Approaches* in *Int. Conf. on Mobile Software Engineering and Systems* (Association for Computing Machinery, 2018), 92–93.
7. Dalmasso, I., Datta, S. K., Bonnet, C. & Nikaein, N. *Survey, comparison and evaluation of cross platform mobile application development tools* in *Int. Wireless Communications and Mobile Computing Conference* (IEEE, 2013), 323–328.
8. Sommer, A. & Krusche, S. *Evaluation of cross-platform frameworks for mobile applications* in *Software Engineering - Workshopband* (eds Wagner, S. & Lichter, H.) (Gesellschaft für Informatik e.V., 2013), 363–376.
9. Daniel, F. *et al.* Understanding UI Integration: A Survey of Problems, Technologies, and Opportunities. *IEEE Internet Computing* **11,** 59–66 (2007).
10. Tirodkar, A. A. & Khandpur, S. S. *EarlGrey: iOS UI Automation Testing Framework* in *International Conference on Mobile Software Engineering and Systems* (IEEE/ACM, 2019), 12–15.
11. Basili, V., Caldiera, G. & Rombach, D. H. in *Encyclopedia of Software Engineering* (Wiley, 1994).
12. Kanji, G. K. *100 statistical tests* 3rd ed. (Sage Publications, 2006).
13. Boslaugh, S. & Watters, P. A. *Statistics in a nutshell - a desktop quick reference.* (O'Reilly, 2008).
14. Heitkötter, H., Hanschke, S. & Majchrzak, T. A. *Comparing Cross-platform Development Approaches for Mobile Applications* in *Int. Conf. on Web Information Systems and Technologies* (2012).
15. Shin, D., Yao, H. & Rosi, U. *Supporting Visual Security Cues for WebView-Based Android Apps* in *Proceedings of the Annual Symposium on Applied Computing* (ACM, 2013), 1867–1876.
16. Bao, W., Yao, W., Zong, M. & Wang, D. *Cross-Site Scripting Attacks on Android Hybrid Applications* in *Proceedings of the International Conference on Cryptography, Security and Privacy* (ACM, 2017), 56–61.
17. Gaouar, L., Benamar, A. & Bendimerad, F. T. *Model Driven Approaches to Cross Platform Mobile Development* in *Proceedings of the International Conference on Intelligent Information Processing, Security and Advanced Communication* (Association for Computing Machinery, 2015).

On the Use of Support Mechanisms to Perform Experimental Variables Selection

Lilian P. Scatalon[*], Rogério E. Garcia[†], and Ellen F. Barbosa[*]

[*]University of São Paulo (ICMC-USP), São Carlos-SP, Brazil
[†]São Paulo State University (FCT-Unesp), Presidente Prudente-SP, Brazil
lilian.scatalon@usp.br, rogerio.garcia@unesp.br, francine@icmc.usp.br

Abstract

The selection of variables in a given experiment is crucial, since it is the theoretical foundation that guides how data should be collected and analyzed. However, selecting variables is an intricate activity, especially considering areas such as Software Engineering and Education, whose studies should also consider human-related variables in the design. In this scenario, we aim to investigate how a support mechanism helps on the variables selection activity of the experiment process. To do so, we conducted a preliminary study on the use of an experimental framework composed by a catalog of variables. We explored the domain of the integration of software testing into programming education. Participants were divided into two groups (ad hoc and framework support) and asked to select variables for a given experiment goal. We analyzed the results by identifying threats to validity in their experimental design drafts. Results show a significant number of threats of type inadequate explication of constructs for both groups. Nonetheless, the framework helped to increase the clarity of concepts selected as variables. The cause of most raised threats, even with the framework support, was an inaccuracy in selecting the values of such variables (i.e. treatments and fixed values).

Keywords: Experimental design, Variables selection, Support mechanisms and Experimental framework.

1 Introduction

The central idea in an experiment is a cause-effect relationship of a given phenomenon. Naturally, the phenomenon of interest may be affected by a sheer number of variables, but the researcher is supposed to select the ones related to the hypothesized cause-effect relationship. Such variables make up much of experimental design, i.e. the plan to conduct the experiment [8, 19].

Researchers usually rely on their personal experience and the empirical literature as a source to help designing their experiments [6]. In this sense, Borges et al. [4] identified several support mechanisms present in the literature of Software Engineering.

Some mechanisms provide help in terms of method, by delineating the experiment process and guidelines on how to conduct its composing activities, such as [19, 8] and [9]. Still, an experimental framework is another kind of support mechanism that provides help in the sense of promoting better study designs.

An experimental framework usually includes models of the domain of interest, providing the basic structure of experiments in such domain [2, 7]. In this sense, it can help to design new studies as a support mechanism to define domain-specific elements.

In this paper we investigated the support provided by an experimental framework to study designing. We explored the domain of software testing integration into programming courses, with a framework that we created in previous works [16], named Step. More specifically, we were interested in evaluating the support of Step while researchers conduct the variables selection activity, which is part of the planning phase in the experimental process.

The remainder of this paper is organized as follows. Section 2 describes the variables selection activity and the experimental framework Step. Section 3 presents other existing frameworks in the literature and similar studies that also investigated research activities. The study protocol and the obtained results are described in sections 4 and 5, respectively. We discuss threats to validity in Section 6. Finally, Section 7 presents conclusions and future work.

1

2 Background

The **variables selection** activity involves representing the investigated cause and effect constructs as experiment variables. The cause is represented by *independent variables* and the effect by *dependent variables*.

Juristo and Moreno [8] indicate in details the rationale to select the variables of a given experiment. For all the identified "input" variables representing the cause construct, i.e. *independent* or *context variables*, the researcher has to determine whether each one is a *factor*, *parameter* or *blocking variable*.

Similarly, the identified "output" variables representing the effect construct are *dependent variables*. Such variables hold quantitative result values and should be operationalized by means of a *metric*, if not directly measurable. Still, the *hypothesis* states in a testable way the researcher's guess about how these selected variables will behave during the experiment.

An experimental framework can provide support to identify and properly select such variables [2, 7, 17, 10]. In this scenario, we created an experimental framework, named **Step**, for studies on the integration of **s**oftware **te**sting into **p**rogramming education [16].

Step includes the model depicted in Figure 1, which consists in a catalog of variables on such research domain. The idea is to support researchers in the planning phase of the experiment process [19], more specifically in the activities of context selection, hypothesis formulation and variables selection.
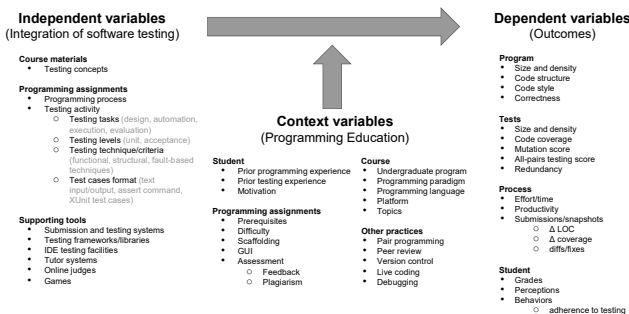
Figure 1. Experimental framework Step

3 Related Work

There are several proposals of frameworks in the Software Engineering literature, all aiming to incorporate domain models. Authors use different names to refer to this kind of frameworks: *organizational framework* [2], *research framework* [7] and *evaluation framework* [18, 11]. Nevertheless, all of them include models of domain-specific elements (e.g. variables) that should be defined when designing an experimental study.

It is worth mentioning that the Computer Science Education area presents an experimental framework for algorithm visualization [12]. In this case, the authors explore one independent variable of the domain, i.e. students' engagement with visualization tools.

The aforementioned frameworks have been used as a reference to design studies in each respective domain, as their authors demonstrate. In this work, we were interested in evaluating the support of this kind of framework while researchers conduct experimental activities.

To this end, we followed a similar approach to Rainer et al. [14], Neto and Conte [13] and Ribeiro et al. [15], which also evaluated researchers conducting research activities, such as applying guidelines, performing validity evaluation and conducting systematic reviews.

4 Method

We conducted an exploratory study on the use of Step by researchers that were not involved with the framework creation. Our goal, expressed using the GQM template [1], is as follows:

Analyze *the use of Step*
for the purpose to *characterize*
with respect to *validity of variables selection*
from the point of view of the *researcher*
in the context of *graduate students selecting variables and formulating a hypothesis for a given research goal*

As stated in the goal, we investigated the use of Step as a support mechanism during the experiment process. We focused on the variables selection activity, providing an experiment goal on our domain of interest (i.e. software testing in programming education) as a starting point to participants.

4.1 Participants

There were seven participants in total, all graduate students that completed the Experimental Software Engineering course at ICMC-USP. Hence, they all had knowledge on the basics of experimentation and the experiment process.

We characterized their background experience both in terms of experimentation and our area of interest (programming education). Firstly, Figure 2 shows how many experiments they have conducted (including definition, planning, execution and analysis). Note that every participant conducted at least one experiment and most were involved in two or more experiments.

Since we aim to evaluate the use of Step during the experiment process, we asked participants what other support
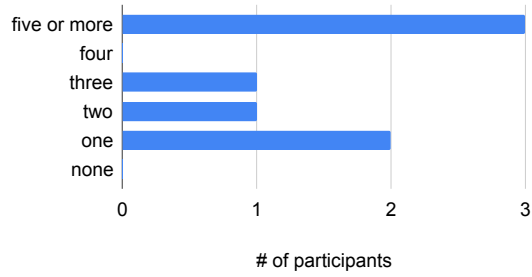
Figure 2. Number of experiments conducted by the participants

mechanisms they usually consult while conducting experiments. Borges et al. [4] identified several mechanisms used by researchers to conduct empirical studies. We presented the ones related to experiments as options to participants, namely: [19], [9], [1], [8], and [2]. Nonetheless, they could indicate other sources of information as well. There was only one mention to another paper [3].

Figure 3 provides an overview of responses. We refer to the options as the first author's name, distinguishing the repeated ones by adding the main subject next to it. The book of Wohlin et al. [19] is the most consulted one, followed by the GQM model [1]. On the other hand, it is interesting to note that nobody indicated the book of Juristo and Moreno [8], especially considering that such book provides detailed rationale on designing experiments.
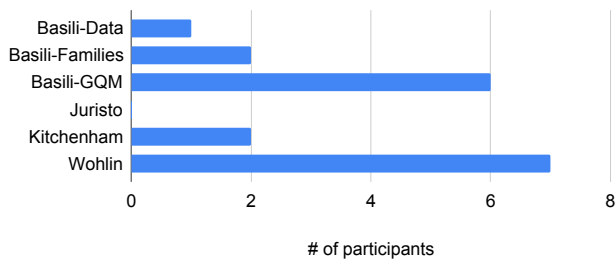
Figure 3. Support mechanisms used by the participants to conduct experiments

Regarding their background on our domain of interest, we asked participants about their experience in programming education. We provided options in terms of the roles in which they could have performed activities in this area, namely instructor, teaching assistant (TA), researcher and none. As Figure 4 shows, most (85.71% – 6) had some kind of experience in the area, whether in practice, as instructor or TA of programming courses, or in theory, as researchers.

We also asked about their familiarity with the integration
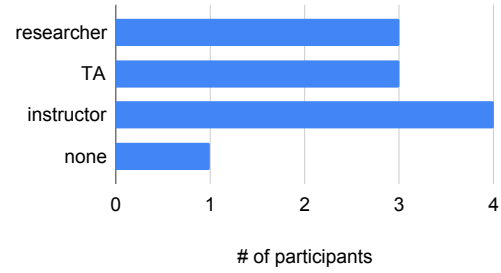
Figure 4. Experience in programming education

of software testing into programming education. All had some kind of familiarity on the domain, since the option "none" was not selected by anyone. Some (42.86% – 3) even have conducted empirical studies on the domain.
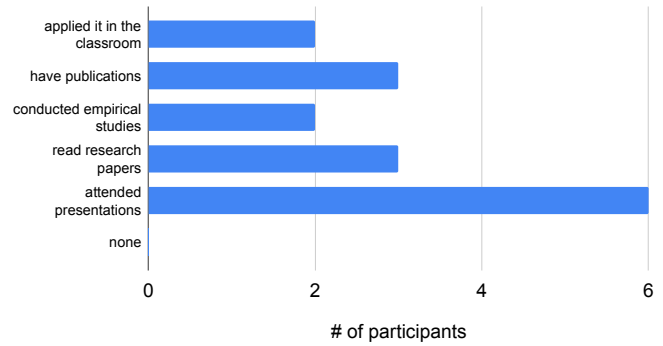
Figure 5. Familiarity with the integration of software testing into programming education

4.2 Procedures and materials

The study involved participants performing variables selection in our domain of interest with two different approaches. They were thus divided into two groups: one using an ad hoc approach (G1) and the other consulting Step (G2).

Firstly, participants filled out a consent form and then received training on study designing, to recall basic concepts such as independent, context and dependent variables.

Moreover, we reinforced the rationale to select which input variables should be factors, parameters or blocking variables and which output variables should be the investigated dependent variables in a given experiment. Only participants in G2 received additional training on the experimental framework Step.

We handed out the training materials on experiment designing to both groups and the overview of Step only to G2. Then, we asked participants to fill out the characterization form, which provided us information on participants' background, as discussed in Section 4.1.

Finally, participants undertook the study task, generating experimental design drafts. We asked them to perform variables selection and formulate a hypothesis for the following experiment goal:

Analyze *progressive assignments*
for the purpose to *evaluate*
with respect to *student's testing performance*
from the point of view of the *researcher*
in the context of *introductory programming courses of Computer Science at ICMC-USP*.

We discussed the underlying scenario with participants, highlighting the motivation to explore this particular goal, which is the following. If students had a greater incentive to ensure quality in their programs, maybe they would feel the need to conduct software testing. One way to lead students in this direction is conducting progressive assignments [5].

A sequence of assignments could be formulated in such a way to configure the progression, i.e. all assignments, except the first one, should have as a prerequisite the solution of the previous one. Students would have to maintain their code from previous solutions, instead of starting them all from scratch.

To complete the study task, participants were supposed to fill out a form, resulting in an experimental design draft. The form was composed by the elements required in the experiment activities we explored: hypothesis, independent and context variables (factors with respective treatments, parameters with respective values, blocking variables with respective blocks) and dependent variables with respective metrics/description.

We evaluated each experimental design draft by means of the number of identified threats to validity. To do so, we used the threats to validity presented by Wohlin et al. [19] as a checklist. However, we only considered threats due to the activities that participants performed, namely hypothesis formulation and variables selection.

Furthermore, the selected threats have to do with how well experiment variables represent the theory constructs (construct validity) and whether the investigated relationship is indeed causal (internal validity).

The remaining types are related to representativeness of subjects and objects (external validity) and issues of the statistical analysis (conclusion validity). Such threats are raised due to decisions of other activities outside the scope of this study (i.e. selection of subjects, instrumentation, execution, hypothesis testing and so on).

5 Results

Table 1 shows occurrences of threats to validity for individual participants. Each type of threat is labeled with an id (T1, T2, and so on). A dash in a cell indicates that the corresponding threat in the row had no occurrence for the participant in the column. Similarly, each value indicates the number of threat occurrences for a given participant.

These same results are summarized in Table 2 with the average (Avg.) and standard deviation (SD) for each group. Considering all threats to validity (i.e. total in the last row), participants using Step presented less threats in average than the ones selecting variables ad hoc, respectively 3.25 against 5.66.

Looking at each threat that had occurrences, T1 was the most frequent one, for both groups. In particular, participants s3 and s5, from distinct groups, presented high values for this threat. Still, in average, Group 2 (2.75) presented less T1 threats than Group 1 (5.00).

One example of such threat found in s2's dependent variables, "student program quality", whose description/metric was "student programs will be assessed by instructors' test cases". It is possible to note that it is not clear how quality is going to be measured.

Another example, now on Group 2, from s6's context variables, "student previous knowledge" was selected as a blocking variable, with blocks defined as "different levels of knowledge in Java development". However, it is not clear what levels are these.

Next, T3 had one occurrence in Group 1 for s2, whose selection of dependent variables included only one variable, thus configuring mono-method bias. Again, the selected variable was "student program quality", without defining how quality should be assessed.

All participants in Group 2 selected more than one dependent variable. Moreover, s4, s5 and s6 selected many dependent variables (more than five) without necessarily having a direct relation with the hypothesis they formulated.

Threat T4 was the second most frequent one. For Group 1, s3 indicated that "C++ or Java" would be a context variable, when in fact the correct construct would be "programming language" instead.

For Group 2, s4 indicated "prior programming experience" as a blocking variable, whose blocks would be "up to four completed programming courses" and "more than four completed programming courses". Such division seems arbitrary, since, until completing four programming courses, students can present very different levels of programming experience.

Table 1. Occurrences of threats to validity

id	Threat	Group 1 Ad hoc			Group 2 Step			
		s1	s2	s3	s4	s5	s6	s7
T1	Inadequate preoperational explication of constructs	4	4	7	1	7	3	-
T2	Mono-operation bias	-	-	-	-	-	-	-
T3	Mono-method bias	-	1	-	-	-	-	-
T4	Confounding constructs and levels of constructs	-	-	1	1	-	1	-
T5	Interaction of testing and treatment	-	-	-	-	-	-	-
T6	Restricted generalizability across constructs	-	-	-	-	-	-	-
T7	Ambiguity about direction of causal influence	-	-	-	-	-	-	-
	Total	4	5	8	2	7	4	0

Table 2. Summary of threats to validity occurrences

id	Threat	Group 1 Ad hoc		Group 2 Step	
		Avg.	**SD**	**Avg.**	**SD**
T1	Inadequate preoperational explication of constructs	5	1.73	2.75	3.09
T2	Mono-operation bias	-	-	-	-
T3	Mono-method bias	0.33	0.57	-	-
T4	Confounding constructs and levels of constructs	0.33	0.57	0.5	0.57
T5	Interaction of testing and treatment	-	-	-	-
T6	Restricted generalizability across constructs	-	-	-	-
T7	Ambiguity about direction of causal influence	-	-	-	-
	Total	5.66	2.08	3.25	2.98

6 Threats to validity

Regarding the external validity of our preliminary study, we had a small sample size and the study task may not represent how researchers conduct study designing in real-world conditions.

Firstly, they worked alone only in a limited part of the experiment process, when usually researchers conduct experiments in a holistic and collaborative way. Nevertheless, we believe it was thereby possible to isolate the use of Step.

Another threat to validity that could be raised is limiting researchers to the concepts present in the experimental framework. However, participants in Group 1 (ad hoc) presented design drafts that tended to be more incomplete than Group 2 (Step), what can be verified by the higher number of threats T1 (inadequate preoperational explication of constructs) in Group 1.

Also, it is important to highlight that threats to validity are expected in a human-centered experiment. The presence of a threat does not necessarily invalidate an experimental design. Hence, only the number of threats may not fully capture its quality. Nonetheless, we focused on two types of validity (construct and internal), both related to theory representation, whose corresponding threats should be carefully analyzed and, whenever possible, mitigated.

Finally, only one person, i.e. the first author, analyzed the experimental design drafts and identified the threats to validity. Therefore, we cannot address inter-rater reliability nor discard the presence of bias in the results. Despite this, the findings shed light on how researchers use an experimental framework, along with possible benefits and drawbacks in doing so.

7 Conclusion

In this paper we reported an exploratory study on the use of the framework Step as a support mechanism to select experimental variables. Despite preliminary, we can point out some interesting findings. The overall number of threats to validity in average was lower for participants using Step, what suggests that it does help to perform variables selection while designing an experiment.

However, there was a considerable amount of threats T1 (inadequate preoperational explication of constructs) for both groups. Looking at specific occurrences of this threat, it is possible to observe different trends for each group.

Group 1 (ad hoc) tended to present less clarity when selecting both the concepts for the variables and their respective values (i.e. treatments, values, blocks and metrics). On the other hand, Group 2 (Step) tended to only present difficulties on selecting the latter. Indeed, Step does not provide much support towards defining variables' values.

Another aspect that drew attention in results is that some participants in Group 2 (Step), namely s5, s6 and s7, seem to have aimlessly selected several variables from Step, in an attempt to form a experimental design. However, not every selected variable in this way had a clear relation with their formulated hypothesis or the provided goal.

Hence, results suggest that Step helps to select variables on the domain with more clarity, but does not provide enough support to select their corresponding values and metrics. Also, we observed a possible side effect of novice researchers using an experimental framework, which is the selection of unnecessary variables, in a kind of trial and error behavior.

The purpose of Step is to help researchers to have an overview of what has been done in the domain research, allowing them to borrow useful concepts and to explore "new" ones. More importantly, the organized overview of concepts can help the researcher to clearly see the boundaries of the study being conducted in terms of domain concepts. In this sense, the results on the use of Step allowed us to observe how researchers not involved with its creation would use it and their resulting experimental designs.

As future work, we intend to further investigate how experimental designing is done in practice, especially by novices. Also, we aim to propose a mechanism in terms of method to support the conduction of variables selection and validity evaluation in parallel.

Acknowledgments

References

[1] V. R. Basili, G. Caldiera, and H. D. Rombach. Goal question metric paradigm. In *Encyclopedia of Software Engineering*, pages 528–532. John Wiley & Sons, 1994.

[2] V. R. Basili, F. Shull, and F. Lanubile. Building knowledge through families of experiments. *IEEE Transactions on Software Engineering*, 25(4):456–473, 1999.

[3] V. R. Basili and D. M. Weiss. A methodology for collecting valid software engineering data. *IEEE Transactions on Software Engineering*, SE-10(6):728–738, Nov 1984.

[4] A. Borges, W. Ferreira, E. Barreiros, A. Almeida, L. Fonseca, E. Teixeira, D. Silva, A. Alencar, and S. Soares. Support mechanisms to conduct empirical studies in software engineering. In *Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, ESEM '14, pages 50:1–50:4, New York, NY, USA, 2014. ACM.

[5] H. B. Christensen. Systematic Testing Should Not Be a Topic in the Computer Science Curriculum! In *Proceedings of the 8th Annual Conference on Innovation and Technology in Computer Science Education*, ITiCSE '03, pages 7–10, New York, NY, USA, 2003. ACM.

[6] L. Fonseca, S. Soares, and C. Seaman. Describing what experimental software engineering experts do when they design their experiments: A qualitative study. In *Proceedings of the 11th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, ESEM '17, pages 211–216, Piscataway, NJ, USA, 2017. IEEE Press.

[7] H. Gallis, E. Arisholm, and T. Dyba. An initial framework for research on pair programming. In *International Symposium on Empirical Software Engineering (ISESE 2003)*, pages 132–142, Sept 2003.

[8] N. Juristo and A. M. Moreno. *Basics of Software Engineering Experimentation*. Springer Publishing Company, Incorporated, 1st edition, 2010.

[9] B. A. Kitchenham, S. L. Pfleeger, L. M. Pickard, P. W. Jones, D. C. Hoaglin, K. E. Emam, and J. Rosenberg. Preliminary guidelines for empirical research in software engineering. *IEEE Transactions on Software Engineering*, 28(8):721–734, 2002.

[10] P. Morrison. A security practices evaluation framework. In *Proceedings of the 37th International Conference on Software Engineering - Volume 2*, ICSE '15, pages 935–938, Piscataway, NJ, USA, 2015. IEEE Press.

[11] P. Morrison. *A Security Practices Evaluation Framework*. PhD thesis, North Carolina State University, 2017.

[12] T. L. Naps, G. Rossling, V. Almstrum, W. Dann, R. Fleischer, C. Hundhausen, A. Korhonen, L. Malmi, M. McNally, S. Rodger, and J. A. Velazquez-Iturbide. Exploring the role of visualization and engagement in computer science education. In *Innovation and Technology in Computer Science Education (ITiCSE*, pages 131–152, New York, NY, USA, 2002. ACM.

[13] A. A. Neto and T. Conte. Identifying threats to validity and control actions in the planning stages of controlled experiments. In *26th International Conference on Software Engineering and Knowledge Engineering*, 2014.

[14] A. Rainer, T. Hall, and N. Baddoo. A preliminary empirical investigation of the use of evidence based software engineering by under-graduate students. In *Proceedings of the 10th International Conference on Evaluation and Assessment in Software Engineering*, EASE'06, pages 91–100, Swindon, UK, 2006. BCS Learning & Development Ltd.

[15] T. V. Ribeiro, J. Massollar, and G. H. Travassos. Challenges and pitfalls on surveying evidence in the software engineering technical literature: An exploratory study with novices. *Empirical Software Engineering*, 23(3):1594–1663, 2018.

[16] L. P. Scatalon. *A framework for experimental studies on the integration of software testing into programming education*. PhD thesis, University of São Paulo (ICMC-USP), 2019.

[17] L. Williams, W. Krebs, L. Layman, A. Anton, and P. Abrahamsson. Toward a framework for evaluating extreme programming. In *Assessment in Software Engineering (EASE)*, 2004.

[18] L. Williams, L. Layman, and P. Abrahamsson. On establishing the essential components of a technology-dependent framework: A strawman framework for industrial case study-based research. In *Proceedings of the 2005 Workshop on Realising Evidence-based Software Engineering*, REBSE '05, pages 1–5, New York, NY, USA, 2005. ACM.

[19] C. Wohlin, P. Runeson, M. Host, M. Ohlsson, B. Regnell, and A. Wesslen. *Experimentation in Software Engineering*. Springer Publishing Company, Incorporated, 2 edition, 2012.

Testing the Stationarity Assumption in Software Effort Estimation Datasets

Michael Franklin Bosu[1], Stephen G. MacDonell[2], Peter Whigham[2]
[1]Centre for Information Technology, Waikato Institute of Technology, New Zealand
{stephen.macdonell, peter.whigham}@otago.ac.nz
[2]Department of Information Science, University of Otago, New Zealand

Abstract—Software effort estimation (SEE) models are typically developed based on an underlying assumption that all data points are equally relevant to the prediction of effort for future projects. The dynamic nature of several aspects of the software engineering process could mean that this assumption does not hold in at least some cases. This study employs three kernel estimator functions to test the stationarity assumption in three software engineering datasets that have been used in the construction of software effort estimation models. The kernel estimators are used in the generation of non-uniform weights which are subsequently employed in weighted linear regression modeling. Prediction errors are compared to those obtained from uniform models. Our results indicate that, for datasets that exhibit underlying non-stationary processes, uniform models are more accurate than non-uniform models. In contrast, the accuracy of uniform and non-uniform models for datasets that exhibited stationary processes was essentially equivalent. The results of our study also confirm prior findings that the accuracy of effort estimation models is independent of the type of kernel estimator function used in model development.

Keywords—Software effort estimation, software processes, stationarity, kernel estimators, weighted linear regression

1 INTRODUCTION

Software engineering datasets emanate from a complex and dynamic ecosystem that involves numerous actions and interactions of people and technologies over time. Data collected about software projects are used to support decision making during software development and the planning of future projects. This paper focuses specifically on software development effort data that may be used in the ongoing management of the cost and/or schedule of current projects as well as in the estimation of the effort required in future projects. One such aspect is project timing – that is, when in time a project and its constituent activities were undertaken. In ignoring the timing of projects most current effort estimation practices implicitly assume the underlying development processes to be stationary over time. The adoption of the stationarity assumption in SEE has culminated in the treatment of all past data as equally relevant during the modeling process. The key objective of this paper is to test the validity of this stationarity assumption in the context of SEE.

The range of factors that can affect the effort required in software development is vast such as the competence and experience of the developers, the participation of the customer, the commitment of top management, requirements ambiguity, adequacy of tools support and communication among the development team. The list of potential influences is practically endless as demonstrated by the following studies.

Ten factors that have significant influence on the development cost and productivity of software projects were identified when 50 projects were analyzed in a Swedish bank [1]. Wagner and Ruhe [2] divided software productivity factors into two groups; soft factors are deemed to be attributes that are influential over the way people work and technical factors relate to the software itself. Maxwell and Forselius [3] assessed the productivity factors of 206 software projects from twenty-six Finnish companies and found the company and the type of business of the client organization as being the most influential factors.

A potentially important additional aspect missing from the above analyses is that which is in focus here – that is, the *stationarity* of the development process. It is the contention of this study that over some (unknown) period of time, an organization's software development processes will not remain static. In this paper we therefore assess three software effort estimation datasets to determine whether or not their underlying processes remain stationary over time. The rest of the paper is presented as follows. In Section 2 we consider related work. Section 3 describes our research design. Our analysis and results are presented in Section 4, and in Section 5 is the threat to validity of the study. Section 6 is the discussion and conclusion.

2 RELATED WORK

Although numerous SEE models have been proposed (see [4]) the number of studies that have considered project timing information in effort estimation is negligible. This section summarizes the few studies that are directly related to the research reported here.

MacDonell and Shepperd [5] assessed the efficacy of two time-aware estimation methods – sequential accumulation of projects over time and a constant moving window of size five – when applied to a proprietary dataset. They obtained improved results over project managers' effort estimates, especially for the moving window approach [5].

Geographically Weighted Regression (GWR) is a method to manage non-stationarity in spatial data. GWR was applied to capture the non-stationarity of relationship in a landscape fragmentation study [6]. GWR derives non-uniform estimates in spatial data; that is, relationships are established in data that belong to a specified (non-uniform) area, as opposed to ordinary least squares regression (OLS) which outputs the estimates of the average or uniform relationships among all observed data.

GWR relies on the assumption that entities that are near to each other in a geographical area are more likely to exhibit similar properties than those that are more distant. This assumption is acted on by weighting nearer areas more than distant areas.

The study here employs a procedure similar to GWR wherein non-uniform weightings are applied to software effort estimation data over time. The use of kernel bandwidth values also enables the determination of the stationarity of the process underlying the data, except that instead of being applied to parameters of space, the approach is applied to the parameters of software projects.

In spite of the proposals of numerous estimation techniques, process (non-)stationarity and its effect on SEE has received minimal attention as reported by Smartt and Ferreira [7].

To the best of our knowledge there are just three prior studies [8], [9], [10] in the software effort estimation domain that have employed kernel estimators in a manner similar to that reported in this paper.

The study presented here differs from that reported by Kocaguneli, Menzies and Keung [9] in that a wider range of kernel bandwidth values (between 1 and 100) is used in order to discover the stationarity properties of the datasets, whereas five selected kernel bandwidth values were used in [9]. In addition, this study employs weighted linear regression to build models based on the sequential accumulation of projects according to their completion dates, while [9] used analogy-based estimation and did not address data accumulation over time. The work presented in this paper has greater similarities with that of Amasaki and Lokan [8] in that it applies linear regression to a growing portfolio of projects using the same set of kernel functions; however, it differs in the use of a wider range of kernel bandwidth values, as they are being applied in this study to assess the stationarity of the datasets, and the processes underpinning the data. The study reported here also employs three datasets exhibiting different characteristics whereas [8] used an extract from the ISBSG repository. Angelis and Stamelos [10] also employed the kernel estimator in software effort estimation based on analogies. They used the kernel function in order to identify the distributions of effort estimates that are not obvious (such as Normal or Lognormal). They [10] used a fixed bandwidth whilst this study uses a range of bandwidths.

The following specific research questions are addressed by this study:

RQ1. Is there only a stationarity process underlying software effort estimation datasets?

RQ2. Does non-stationarity of software effort estimation datasets affect the accuracy of effort estimation models when applied over time?

RQ3. Does kernel type affect the accuracy of software effort estimation models?

3 RESEARCH DESIGN

In this section we first describe each of the three datasets to be analyzed along with the particular computation of effort estimation used in each case. We then describe our model development and evaluation process before specifying how the various kernel weightings are determined.

3.1 Dataset Descriptions

NASA93 Dataset

The NASA93 dataset was collected by NASA from five of its development centers and it collectively represents fourteen different application types. The entire dataset comprises 93 projects undertaken between 1971 and 1987. Projects were completed in the years indicated in the version of the dataset that is available from the PROMISE Repository http://openscience.us/repo/. The dataset is structured according to the Constructive Cost Model (COCOMO81) format developed by Boehm [9]. It comprises 24 attributes of which 15 are the mandatory effort multipliers. Effort multipliers and development modes are describe in detail in [9]. Effort multipliers are assigned a range of predefined values which were obtained from regression analysis of the original COCOMO81 data. The other attributes of relevance are product size, measured in thousands of lines of code (KLOC), and effort, measured in calendar months (where one calendar month is said to be equivalent to 152 person-hours of effort). The computation of effort for COCOMO81 projects is given by equation (1).

$$effort(personmonths) = a * (KLOC^b) * (\prod_i EM_i) \qquad (1),$$

where KLOC is size measured in thousands of lines of code and EM represents the effort multipliers. COCOMO81 projects are classified into three development modes that each requires the use of certain parameter values in the model the values of a and b are domain-specific values dependent on the mode of the project being developed.

Desharnais Dataset

The Desharnais dataset was collected from ten organizations in Canada by Jean-Marc Desharnais. The projects in this dataset were undertaken between 1983 and 1988. The dataset consists of 81 records and twelve attributes, including size measured in function points and effort measured in person-hours. In most studies that employ this dataset, 77 of the 81 records are used because of missing data in four records [11]. In this study, the version with the 77 projects is therefore also used. The Desharnais dataset, like the NASA93 dataset, contains only the year of project completion and, as such, the training and test data sets are formed in the same way as the NASA93 dataset (i.e., by using the year of project completion).

Though there are twelve attributes in the Desharnais dataset, analysis carried out by Desharnais identified the size and language attributes as those that are influential in a regression model. Kitchenham and Mendes [12] supported Desharnais' claim by proposing the use of the language attribute as a dummy variable. This approach has therefore been adopted in this study for the models developed for this dataset, as shown in equation (2).

$$\ln(effort) = \ln(size) + language \qquad (2)$$

This study used the adjusted function points value as the most complete size attribute (rather than the raw function point count) and treated the three-value language attribute as a dummy variable, with the reference dummy value (being the Basic Cobol projects) indicated as "1" in the Desharnais dataset.

Kitchenham Dataset

The Kitchenham dataset [13] was collected from the American-based multinational company Computer Sciences Corporation

(CSC). This dataset contains information about 145 software development and maintenance projects that CSC undertook for several clients. There are 10 attributes considered, the size attribute was measured in function points, and effort was measured in person-hours. The attributes also include start date and estimated completion dates, and the projects were undertaken between 1994 and 1999. The attributes useful for effort modeling (based on prior research evidence) are the size attribute and the application type attribute. This study used the application type attribute as a dummy variable with the reference value being the "Development" type. Again following prior work this study uses 105 records related to projects developed for so-called 'client 2' [13].

As this dataset includes information about the actual start date of projects and their duration in days, these values are used to compute each project's completion date. Training sets are formed based on the years in which projects were completed, as was done for the NASA93 and Desharnais datasets. Composition of the test data sets follows a slightly different process, however, because of the availability of actual start dates: a test set consists of projects completed in the subsequent year and started after the date the last project in the training set was completed. This dataset consists of 67 perfective maintenance projects and 38 development projects. The model formulation is shown as equation (3).

$$\ln(effort) = \ln(size) + type \qquad (3)$$

3.2 Effort Estimation Model Development

In software effort estimation modeling, as in many other fields, the (secondary) dataset is usually split into two, forming a larger training set and a smaller test set. Models are then built using the training set, and the unbiased performance of the models is evaluated on the test set. This study follows a similar approach; the specifics of how the training and test sets are formed are described in the modeling algorithm subsection below. All models in this study are developed using the statistical package R (version 3.5.2). In preparatory testing the Shapiro-Wilk test of normality was applied to the numeric variables in the training sets. All such variables that failed the normality test were logarithm transformed, meaning that in the associated models developed, log(effort) (shown as ln(effort) in the equations) would be the dependent variable and log(size) (ln(size)) one of the explanatory variables. The estimated (natural log) effort values are back-transformed to unscaled values prior to the computation of any accuracy measures. All models are developed using linear regression, considered to be a widely used modeling approach in effort estimation [4]. The actual linear regression equations for each dataset have been presented in subsection 3.1. It should also be noted that the models developed in this study are all *well-formed models*, that is, the degrees of freedom are considered whereby a training set is formed only when the number of projects is at least two plus the number of explanatory variables being used for model construction.

Modeling Algorithm

This paper generally follows the sequential accumulation approach used by MacDonell and Shepperd [5] in forming the training sets for the effort estimation models. As such, the following procedures are applied to all datasets

modelled in this study:

1. For each dataset with timing information, select the first year in which projects were completed as the training set – if the first year of projects comprises fewer than the number of observations needed to build a well-formed model, add the next year(s) of projects until the minimum requirement for a well-formed model is satisfied. The subsequent year of projects is used as the test set.
2. Check for normality in the distributions of the training data – if data follow a normal distribution go to step 3 else
2.1. Apply the appropriate transformation to make the data normal and recheck normality for verification as above.
3. Build a regression model using the training data.
4. Apply the model obtained in step 3 to predict the values in the test set.
5. Calculate the accuracy measures (see below) for the prediction model.
6. Add the test year's data to the training set, and the subsequent year's data becomes the new test set.
7. Repeat steps 2 to 6 through to the estimation of the last year of projects.

Model Evaluation

We employ the relative error (RE) measure in evaluating each of the models developed in this study. This is because the relative error measure accounts for the variability in data and as such it is robust to outlier data points [14]. Values of RE equal to or greater than 1 indicate that the model is performing no better than the prediction of a constant value [14], while values approaching zero indicate an increasingly accurate prediction. The relative error is computed using equation (4):

$$RE = variance(residuals) / variance(measured) \qquad (4),$$

where measured is test data

3.3 Generation of Kernel Weights

In order to apply a consistent approach to our analysis the completion date of each project in the three datasets is the only property of time considered in the determination of the kernel weights in this section (even though the Kitchenham datasets include project start *and* completion dates).

Table 1. Formulae of Kernel Types

Kernel Type	Formula
Uniform	$W_{ij} = 1$, $|t| < 1$
Gaussian	$W_{ij} = \exp(-0.5 * t^2)$, $|t| < 1$
Epanechnikov	$W_{ij} = 1 - t^2$, $|t| < 1$
Triangular	$W_{ij} = 1 - |t|$, $|t| < 1$

Table 1 shows the four kernel estimators used in generating weights applied to the datasets (where the Uniform kernel serves as a non-weighted baseline). To find t, we used formula (5):

$$t_{ij} = (t_j - t_i)/b \qquad (5)$$

where t_{ij} is the period, or in this case, the number of years that have elapsed between project i and the target project j (that is, the project being estimated). W_{ij} is the weight applied to project(s) completed in year i with reference to projects in a target year j, and b is the kernel bandwidth (discussed later in this section).

The value of 1 is assigned to the oldest completion period in each dataset and a yearly increment of 1 is applied thereafter.

The elapsed time periods are determined between a specific year and the target year to be used in the application of the formulae in Table 4 to derive the weights for projects in specific years; each past year is subtracted from the target year and the results indicate the elapsed time (in years) from the target year. For instance, given two projects developed in different years; tij = j – i. The weight is 1 when i is equal to j. The bandwidth controls the weighting contribution of neighboring projects, that is, projects from specific years [8].

Fig. 1 depicts the weights that are generated for selected bandwidth values for the datasets based on the Gaussian kernel used in this paper (Note that for clarity, it is impractical to show all the bandwidth values between 1 and 100). For this study, the bandwidths are set between 1 and 100 at increments of 1. Fig. 1 shows that, as the bandwidth value increases, the weights applied to all projects in the training set approach 1. Older projects have smaller weights because the assumption is that the underlying software process used in generating the data is different to that used for current projects. It is also evident in Fig. 1 that small bandwidth values such as 1 and 2 lead to a rapid decline in the weights that are assigned to projects that occur later in time from the target year.
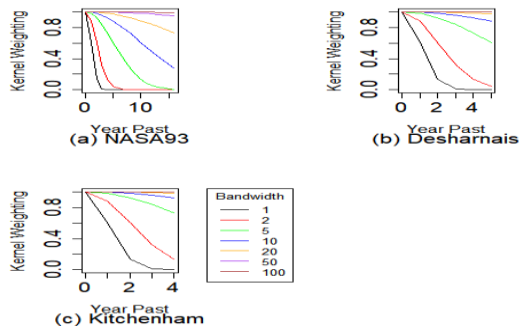
Fig. 1. Weights generated for datasets using the Gaussian Kernel

However, the weight for larger bandwidth values declines gradually and as such the weights for the data in the training set become nearly the same irrespective of the completion date of a project. Due to lack of space, all other graphs generated for selected bandwidth values for all the datasets and kernel types are not shown, however, they are available at this link[1]. The concave nature of the Epanechnikov kernel for the NASA93 dataset curves corresponds to the expected shape of this particular kernel [8]. In comparison to the Gaussian kernel curves the weights decrease a little more gradually, for all bandwidth values across the periods of project completion. Finally, the weights generated for selected bandwidth values for the datasets based on the Triangular kernel are linear for all bandwidth values and across all periods. Just like the Gaussian and the Epanechnikov kernels, the weights for larger bandwidth values decline in a more gradual manner.

4 Analysis and Results

The kernel weights generated as per the procedure described in subsection 3.3 are applied to effort estimation models for the

three datasets. The relative errors of the models are computed over the specified range of bandwidth. Use of the kernel functions enables the application of non-uniform weights to the projects in these datasets as they are used to develop effort estimation models. In order to determine the stationarity or otherwise of these datasets, effort estimation models are developed according to the modeling algorithm of subsection 3.2. The modeling equations derived for each of the datasets in subsection 3.1 are subsequently applied.

In order to determine whether or not a model exhibits a stationary process, the weight graph (Fig. 1) above should be considered alongside the graphs depicting prediction errors on Fig. 2 and others available at the previously specified link. For example, in the case of the Gaussian kernel, Fig. 1 is read in combination with graphs of the models developed for each of the three datasets that used the Gaussian kernel in weight generation, shown in Fig. 2. The bandwidth at which stationarity was attained is identified on the graph of the respective dataset and then this bandwidth value is mapped onto the corresponding Fig. 1 curve to determine the year at which the models remained stationary. This process is repeated for all kernel types and datasets (available at previously specified link) in the interpretation of the results.

The accuracy measure of the models built using the weights generated by the kernel estimators are shown on the plots as 'train', which is effectively the non-uniform model (applying non-uniform weighting). The non-uniform model is then used to predict the effort of projects in the test set, indicated as 'test' on the graphs. Similarly, the result of the uniform model (where no weighting is applied) is indicated on the plot as 'train global', and the model is then used to predict the effort of projects in the test set, indicated as 'test global'. The results are shown on each graph to aid comparison of the models and to enable the identification of models that are stationary or otherwise. It is worth noting that, in presenting the results, emphasis is placed on the training model outcomes because the intention is to identify the stationarity status in the data. The results are subsequently presented for each of the datasets in this section, however, only Gaussian kernel for the NASA93 datasets will be illustrated in detail due to lack of space. The other datasets and kernel types follow the same procedure outlined in section 4.1.

4.1 NASA93 Dataset

The results of the models developed for the Gaussian kernel modeling of the NASA93 dataset are shown in Fig. 2. The graphs show the relative error against bandwidth values for models built over the various time periods under consideration. In Fig. 2(a), at approximately bandwidth 5, the non-uniform model and the uniform model converge, meaning a stationary process is achieved at this point. Looking at a bandwidth of 5 on Fig. 1(a) indicates that convergence would occur at about the 15th year of projects in the training set. Given that the training set for this model is made up of only 7 years of projects, this means there is effectively no convergence, implying that these projects exhibit a non-uniform process. The underlying process can therefore be said to be non-stationary. The results of the model depicted in Fig. 2(c) are similar to those shown in Fig. 2(a). These two models, Fig. 2(a) and Fig. 2(c), converge at

---
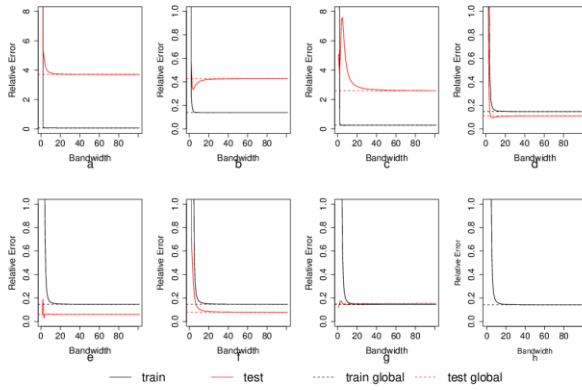
Fig. 2. Gaussian Models - Relative Error against Bandwidth for the NASA93 Dataset.

about a bandwidth value of 5. According to Fig. 1(a), a bandwidth value of 5 converges beyond the number of years that constitute the entire NASA93 dataset, implying that the model of Fig. 2(c) also exhibits a non-stationary process.

Fig. 2(d) indicates that at about bandwidth 14 the model started converging and that actual convergence occurred at bandwidth of 25, which according to Fig. 1(a) is well beyond the number of years of projects that constitute the training set, implying that all of the projects that constitute the non-uniform model exhibited a non-stationary process.

The non-uniform model of Fig. 2(e) started approaching a stationary process at a bandwidth value of about 17. If this is mapped onto Fig. 1(a), it is beyond the number of years for which convergence can be attained based on the training set, implying that the model exhibits non-stationary characteristics.

The non-uniform models of Figs. 2(f) and 2(g) both started approaching the curve of the uniform model at a bandwidth value of around 20. The actual convergence of the non-uniform models to the uniform models occurred at bandwidth of 30 and 35 respectively on Fig. 2(f) and Fig. 2(g). This again occurs beyond the number of years of projects in the datasets (as indicated on Fig. 1(a)) which implies that the projects used in building the models exhibited non-stationary characteristics.

A model developed using the entire NASA93 dataset, as shown in Fig. 2(h), started approaching the uniform model curve at bandwidth 15 and actually converged to that of the uniform model at about bandwidth 18. This convergence value according to Fig. 1(a) requires more than the 14 years of projects that constitute the NASA93 dataset, implying that the process underlying this model is non-stationary.

Overall Fig. 2 indicates that the accuracy of the uniform models is better than (that is, they exhibit lower relative error values) the non-uniform models for the NASA93 dataset. The curves also show the existence of non-stationary processes underlying the projects of the NASA93 data set across the different projects over time, evident in the rapid decline of the relative error of the non-uniform models as the bandwidth value increases.

4.2 Desharnais Dataset

The results for the Desharnais dataset using the Gaussian kernel function indicate that, in general, the uniform models are nearly the same as the non-uniform models in terms of their accuracy, though the non-uniform models are marginally better in two cases. For the model built with the entire Desharnais dataset, the non-uniform and the uniform model results are nearly the same, with both exhibiting an underlying stationary process. Taken overall, the results of the Desharnais model analysis generally indicate a nearly stationary process across the different bandwidths and across time. For this dataset, the non-uniform model and uniform model predictions are nearly the same, for all the models. The predictions based on the models (non-uniform and uniform) is similar to that described for the NASA93 dataset in section 4.1, as some of the models' predictions are better in terms of accuracy than others across time. The results obtained for Epanechnikov kernel and the Triangular kernel are largely similar to those obtained for the Gaussian models.

The relative stationarity of the models built for the Desharnais dataset is somewhat surprising because this dataset was collected from 10 different organizations in Canada over a period of 7 years. However, the project types and development languages used were few. This perhaps implies that it is possible that organizations working at the same time on similar projects may well have similar practices and, as such, models that are built to characterize their practices may be more homogeneous than heterogeneous.

4.3 Kitchenham Dataset

The models developed using the Gaussian kernel when applied to the Kitchenham dataset depicts a near stationary process. The first models exhibits non-stationarity at lower bandwidth values until they converge to a stationary process at bandwidth values of between 4 and 10. Mapping these bandwidth values onto Fig. 1(c) indicates that the models will not attain stationarity in time – thus the process underlying the second to fourth models are interpreted as being non-stationary. The results for this dataset are therefore mixed – there is evidence of a stationary process in one of the models while the other three imply a non-stationary process. The predictions based on the Gaussian model are relatively good for this dataset as they all attained a relative error of less than 1.

The results of the models based on the Epanechnikov and Triangular kernels are similar to their equivalent Gaussian curves based on exact bandwidth values comparisons (they exhibit similar stationarity and non-stationarity at the same bandwidth values respectively). The accuracy of the models of all three kernel estimators are similar for the Kitchenham dataset. Both the respective non-uniform models and the uniform models generated similar results in terms of the RE measure. The predictions based on the models differ across time as was observed for the previous two datasets.

The mixed results (both stationary and non-stationary models) obtained for the Kitchenham dataset could be attributed to the different practices associated with the development types: it seems likely that the organization would have applied different processes to new software development projects as compared to their perfective maintenance projects. This could explain the non-stationarity of some of the models. On the other hand, the stationary model could be due to the fact that all projects were developed by one organization for a single client, and as such, similar general (organization-level) procedures

could have been applied.

5 THREATS TO VALIDITY

The first threat to the validity of this study is to the generalization of our results, as the datasets used are convenience sampled from the PROMISE repository. Though these datasets cannot be considered as representative of the entire software industry, those stored in the PROMISE repository have rather become benchmark datasets in empirical software engineering. Moreover, the three datasets were selected in terms of their possessing different characteristics. As such these results provide promising insights into the derivation of the nature of processes underlying software engineering datasets, and the effect of stationarity on the effectiveness of non-uniform or uniform estimation models.

Another threat is the lack of detailed information for the publicly available datasets. The absence of data detailing the composition of the development teams, the experience of the team and manager, the tools that supported the software development process, the procedures applied at the different development phases, and so on, means that we characterized models as stationary or non-stationary due to the nature of their curves when plotted. Whether the underlying datasets are truly in keeping with this characterization cannot be determined from the limited data available.

6 DISCUSSION AND CONCLUSION

Three kernel estimator functions have been applied to three datasets in developing non-uniform models that identifies the stationarity and/or non-stationarity process underlying SEE datasets. Based on the results presented, the research questions are answered as follows.

RQ1. Is there only a stationarity process underlying software effort estimation datasets?
The result of this study indicates that for the datasets used in this study, both stationary and non-stationarity processes might be present in software effort estimation datasets. The result further establishes that, it is even possible for one dataset to exhibits both stationary and non-stationary process over time as evidenced by the Kitchenham datasets.

RQ2. Does non-stationarity of software effort estimation datasets affect the accuracy of effort estimation models when applied over time?
In considering the above results we determine that the answer to research question RQ2 is yes. For all datasets that exhibited non-stationary processes the models (non-uniform models) resulted in relatively large relative errors especially prior to convergence to the uniform models. In contrast, the estimation accuracy for datasets that exhibited stationarity is in almost all cases the same as that obtained from the uniform models. These results are observed for all kernel types. Thus, we would conclude that the accuracy of effort estimation models is indeed affected by the stationarity of the datasets.

RQ3. Does kernel type affect the accuracy of software effort estimation models?
For the datasets that have been analyzed in this study the evidence indicates that the type of kernel does not affect model accuracy. The accuracy of the models as measured by the relative error were mostly the same for the respective datasets for all kernel types. The estimations based on the models using

the test sets were also the same for each dataset irrespective of the kernel type that was used in the generation of the weights. This study therefore reaffirms the result of the Kocaguneli, Menzies and Keung [9] study that did not find variation in model accuracy due the type of kernel. In terms of using different kernel types to assess the stationarity of a dataset there were just a few occurrences where the different kernel types generated contrasting results, as presented in Section 4.

This study found that there is the possibility of both stationarity and non-stationarity processes present in SEE datasets. A further finding is that the stationarity or otherwise of a datasets impacts on model prediction accuracy. The evidence drawn from this study further suggests that the accuracy of models is independent of the kernel type used in the generation of weights for the non-uniform models. This is observed in the fact that, for each dataset, all three kernel estimators resulted in the same relative errors for all equivalent models and their estimates for the test set observations.

Future work will apply the kernel estimators to other datasets as well as assess the effect of bandwidth values on model accuracy.

REFERENCES

[1] R. Lagerström, L. M. von Würtemberg, H. Holm, and O. Luczak, "Identifying factors affecting software development cost and productivity," Softw. Qual. J., vol. 20, pp. 395–417, 2012.

[2] S. Wagner and M. Ruhe, "A Systematic Review of Productivity Factors in Software Development," Proc. 2nd Int. Work. Softw. Product. Anal. Cost Estim., pp. 1–6, 2008.

[3] K. D. Maxwell and P. Forselius, "Benchmarking Software Development Productivity," IEEE Softw., vol. 17, no. 1, pp. 80–88, 2000.

[4] M. Jørgensen and M. Shepperd, "A Systematic Review of Software Development Cost Estimation Studies," IEEE Trans. Softw. Eng., vol. 33, no. 1, pp. 33–53, 2007.

[5] S. G. MacDonell and M. Shepperd, "Data accumulation and software effort prediction," Proc. 2010 ACM-IEEE Int. Symp. Empir. Softw. Eng. Meas. - ESEM '10, p. 1, 2010.

[6] J. Gao and S. Li, "Detecting spatially non-stationary and scale-dependent relationships between urban landscape fragmentation and related factors using Geographically Weighted Regression," Appl. Geogr., vol. 31, no. 1, pp. 292–302, 2011.

[7] C. Smartt and S. Ferreira, "Advancing Systems Engineering in Support of the Bid and Proposal Process," Syst. Eng., vol. 14, no. 3, pp. 305–326, 2011.

[8] S. Amasaki and C. Lokan, "On the effectiveness of weighted moving windows: Experiment on linear regression based software effort estimation," J. Software-Evolution Process, vol. 27, no. 7, pp. 488–507, 2015.

[9] E. Kocaguneli, T. Menzies, and J. W. Keung, "Kernel methods for software effort estimation. Effects of different kernel functions and bandwidths on estimation accuracy," Empir. Softw. Eng., vol. 18, no. 1, pp. 1–24, Dec. 2013.

[10] L. Angelis and I. Stamelos, "A Simulation Tool for Efficient Analogy Based Cost Estimation," Empir. Softw. Eng., vol. 5, no. 1, pp. 35–68, 2000.

[11] M. Shepperd and C. Schofield, "Estimating Software Project Effort Using Analogies," IEEE Trans. Softw. Eng., vol. 23, no. 12, pp. 736–743, 1997.

[12] B. Kitchenham and E. Mendes, "Why Comparative Effort Prediction Studies may be Invalid," Proc. 5th Int. Conf. Predict. Model. Softw. Eng., 2009.

[13] B. Kitchenham, S. Lawrence Pfleeger, B. McColl, and S. Eagan, "An empirical study of maintenance and development estimation accuracy," J. Syst. Softw., vol. 64, no. 1, pp. 57–77, Oct. 2002.

[14] P. A. Whigham, C. A. Owen, and S. G. Macdonell, "A Baseline Model for Software Effort Estimation," ACM Trans. Softw. Eng. Methodol., vol. 24, no. 3, p. 20., 2015.

Graph Machine Learning for Anomaly Prediction in Distributed Systems

Sheyda Kiani Mehr, Wenting Sun, Xuancheng Fan, Nikita Butakov, Nicolas Ferlans

Ericsson, Santa Clara, USA

Email: sheyda.kiani.mehr@ericsson.com, wenting.sun@ericsson.com, xuancheng.fan@ericsson.com

nikita.butakov@ericsson.com, nicolas.ferland@ericsson.com

Abstract—Machine-learning based anomaly detection in distributed systems is a challenging task. With thousands of dynamically changing parameters, prediction of events requires considerable effort on feature selection, feature engineering, and training. For the unstructured and multi- dimensional data in IT infrastructure, many traditional machine-learning methods are unsuitable and perform poorly. Graph-based machine learning is a powerful tool capable of representing this data without losing the temporal and logical connections that are critical in making accurate predictions. Furthermore, graph-based embedding can effectively reduce the data complexity, without losing key relations. In this paper, we investigate the use of graphs for representing IT infrastructure data, in particular with the node2vec algorithm, and evaluate the performance of a random forest model. The results show that the embedded graph representation improves the precision and AUC, compared to other conventional approaches, while significantly reducing the memory needed for training and validation, thereby making it more suitable for inference on edge devices, where compute resources could be limited.

Index Terms—Graph, Embedding, Node2Vec, Random Walk, Machine Learning

I. INTRODUCTION

Any distributed system must ensure high availability and high consistency. To resolve performance degradations, which cause dissatisfaction to users, detection and prediction of anomalies can help engineers provide more reliable services, and thereby save money and time. Anomaly detection has a wide range of applications in detecting cyber-intrusion, fraud, industrial damage, and safety issues . There are various approaches including classification, clustering, statistical, information-theoretic, and spectral [1] approaches.

To effectively detect anomalies, data must accurately represent the logical dependencies, attributes, and domain characteristics [2]. Graph data structures are a powerful tool to represent data with strong inter-dependencies. For a system that can be represented as a graph, the connectivity and interaction between data points make the graph turn into a number of linked paths for finding out these relations, to detect and analyze an anomaly. Traditional graph analysis methods based on graph statistics and features are helpful, but designing them can be a time-consuming and expensive process [3].

Alternatively, a graph embedding approach can be utilized, which automatically transforms graph into feature vectors.

This embedding will ideally contain all the relevant graph information. After optimizing the embedding space, the learned embedding can be used as feature inputs for downstream machine learning tasks [3].

In this work we present a graph-based machine learning anomaly detection approach for identifying anomalies in the response times of APIs in a distributed system. We propose a Node2Vec approach for graph embedding, with Random Forest Classifier method for anomaly prediction. We find the graph-based ML approach shows improved performance, compared to traditional approaches, with a significant reduction in required compute resources.

Overall the contribution of this work includes:

- The graph representation of the data that keeps all the knowledge intact.
- The graph dimensionality reduction, which enables faster development of models
- Classification and labeling for an anomaly prediction problem.
- Prediction performance improvement, compared to conventional data approaches.

In Section 2, related work is reviewed. In Section 3, our new methodology is presented. In Section 4, the implementation process is explained. In Section 5, the approach is evaluated. And in Section 6, we provide the conclusion.

II. RELATED WORK

There has been much recent work in anomaly detection on graph-like data. Kandel et. al. [4] detect anomalies by using the node attribute information, together with the structural connectivity. To preserve closeness information, they consider similarities between node values with multiple attributes. To discover the similarity between attributes, they use discretization, distance- based similarity measures, and a k-means clustering approach.Ahmed et. al. [5] propose neighborhood-structure-assisted negative-matrix-factorization (NMF) and its application in unsupervised point anomaly detection. They consider the incorporation of the neighborhood structural similarity information, within the NMF framework, by modeling the data with a minimum spanning tree.Tosyali et. al. [6] proposed a cluster-based outlier score function to identify outliers in citation networks based on NMF. They represent citation data as a directed graph and cluster it into logical

groupings of nodes.Markovitz et. al. [7] propose pose clustering for anomaly detection of human actions. They apply a deep-embedded-clustering model with three parts: an encoder, decoder, and soft clustering layer. Venkatesan et. al. [8] propose graph-based unsupervised models for edge anomaly and node anomaly detection in social network data. They apply the HDBSCAN clustering algorithm for node anomaly detection with various dimensionality reduction algorithms. Lu et. al [9] proposes anomaly detection for container-based cloud environments by monitoring the response time and resource usage of components.

Our work is a supervised learning classification approach for anomaly prediction in distributed systems. We represent data with graphs and embed each to feature vectors. We define labels based on the response time thresholds. Many of these previous works use graph statistic information and dimension reduction methods to modify the data and feed it to an unsupervised learning algorithm for clustering anomaly purposes. However, so far we did not find any other work that use a graph embedding with neural networks [10] as input for ML algorithm and dimension reduction, which keep all the knowledge of the data. Most of the anomaly detection works used unsupervised learning approach, however we define it as a supervised learning problem with domain expert knowledge, help to achieve more reliable performance than unsupervised approaches.

III. METHODOLOGY

Distributed systems are time-sensitive. When a user makes a request to an API, a response should be quickly received. Identifying and predicting APIs with anomalous response times help operators quickly resolve and prevent anomalous behavior.

Our data is represented as a directed graph, with nodes being various servers and APIs, connected to each other with a weight equal to the response time (Fig. 1). Individual servers are not connected to other servers, and APIs are not connected to other APIs. As mentioned before, in this system, the definition of an anomaly is when an API fails to respond in a timely manner to a request. In the graph view, this corresponds to an edge weight which exceeds a certain threshold.

The differences between the graph embedding algorithms depend on the graph property being maintained. Different graph embedding algorithms have different definitions of the node, edge, substructure and whole-graph similarities. Node embedding represents each node as a vector in a low dimensional space. Nodes that are similar to each other in the graph have similar vector representations [3]. Our problem is a node embedding problem, as we want to identify which node is acting abnormally, and predict the anomalous behavior of that node in the future. Graph embedding techniques, such as Node2Vec, exploits the structure of the graph and can be used for transformation of graphs into the necessary feature vector space.
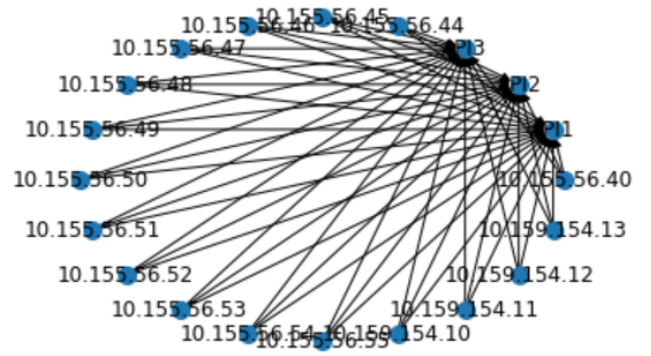
Fig. 1: A directed graph (of a two-minute snapshot) of a distributed system (three APIs, multiple servers).

A. Node2Vec

Node2Vec is one of the most common approaches for projecting nodes into feature vectors. In Node2Vec, nodes are mapped into a low-dimensional space of features that maximizes the likelihood of preserving network neighborhoods of nodes [10]. Node2Vec has two steps: random walk and word2vect. The former creates a corpus of acyclic subgraphs. The latter embeds this corpus to the feature vector space.

1) Random Walk: The parameters for the random walk step include: number of walks to be generated from each node, the number of nodes in each walk, the return parameter (p) and the out parameter (q). The random walk starts with a random node and proceeds through a path based on value of $weight * \alpha$ where α is $1/p$ or $1/q$, depending on if the path navigation is backward or forward.

2) Word2Vec: The output of the random walk step is a corpus of subgraphs. Each random walk corresponds to a sentence-like structure, in which each node corresponds to a word. The Word2Vec model transforms this corpus into an embedding, by using a SkipGram model with a neural-network layer into an N-dimensional embedding.

B. Anomaly Prediction

Anomaly Prediction can be implemented by either supervised classification algorithms or unsupervised clustering algorithms. Classification-based techniques can be thought of as operating in two phases. In the training phase the classifier learns using labeled training data. In the validation phase the classifier categorizes validation data as normal or abnormal. Classification based anomaly detection techniques operate under the general assumption that normal and anomalous classes can be distinguished in a given feature space [2]. Based on our data, we can define labels for each generated graph, and thereby develop a supervised algorithm for graph label classification. The random forest classifier is fast and easy to implement. It can produce highly accurate predictions, handle a very large number of input variables, and tolerate unbalanced or missing data [11].

IV. Implementation

A. Dataset

The dataset consists of two-minute snapshots, over two days of synthetic data, generated based on three months of real data, from a distributed system. In each snapshot, there are multiple servers and three APIs. Each API has its own response time threshold, based on domain knowledge. In some snapshots there may not be any response time, which means that API was not been called in the two-minute snapshot. In other words, there is no edge. For each series of graphs, their embeddings, labeling, and predictions will be separately implemented for each API.

To minimize the presence of overly-sparse graphs, we aggregate three two-minute snapshot graphs into six-minute graphs. These aggregated graphs are the input to Node2Vec, which outputs the embedded feature vectors. We feed these into the Python Sklearn Random Forest Classifier, which outputs predictions.

1) Nature of Dynamic graphs: One of the challenges in our work is the nature of time series data. The system gets a snapshot for every two minutes of the processes, requests and related latencies. In the graph representation, it will be translated to the nodes and the connectivities with weights on the edges, respectively. In each two-minute time slot, the numbers of nodes connected (involved servers and APIs) and latency of API requests (edge's weight) in the system are varying. In other words, the whole data set is a group of consecutive dynamic graphs. The graph of each time stamp might be very different compared to the graphs from the next or previous time stamp. In this case, the cosine similarity of an embedded node of the different graphs are high. With a high cosine similarity between each pair of graph embedding vectors, the prediction will not be accurate.

Graphs can be dynamic in number of nodes, connectivity/edges between nodes, edge directions, edge attributes, node attributes and etc (Fig. 2). In our dataset the changes over time in the graphs include the number of nodes, latency connectivity or weight value of edges. For example, in time stamp 00:00:00 we might have servers that are calling specific APIs with a very low latency and in the next time stamp 00:00:02, the same servers and APIs might have connection but with some different higher latency values on the edges which categorize some APIs of that graph as anomalous APIs. In addition, the variation of node numbers and connectivities in different graphs is eliminated with aggregating the graphs; therefore, all the six-minute graphs will have the same connectivity.
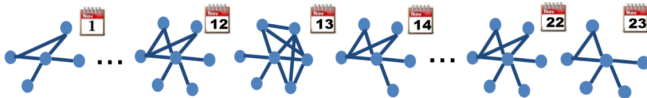
Fig. 2: Event detection in time Series of graph data [1].

Random walk is a good method if dynamic graphs with different connectivity are used as data. Weight is used for walking decision just per graph. In that case, random walks of a graph with very high weight edges might be equal to random walk of another graph with a way lower weight edges containing the same links. Therefore, the embedding of the two different graphs that has very different link weights will be the same although one might not have anomaly and the other one might have anomaly.

We make the data more informative when feeding it to the Random walk. We modify each six-minute graph, we consider an undirected graph and then we assumed all the edge weights as 1 or 0 based on the threshold values of the APIs in the data. The directed graph only guide each walk through the direction from an server to an API and will not cover all the possible nodes.

After implementing this method, we are able to lower the cosine similarity between graphs as the similarity of the graphs are not just based on the connectivity but also based on the weight on the edges, which plays an important role to capture the temporal dynamic information between graphs.

2) Two-minute graphs: We generate directed weighted graphs of two-minute snapshots and add all the nodes and edges with the assigned weights to each graph. In the case in which a graph contains multiple overlapping edges, the edge with the maximum weight (i.e. response time) is used. If any response time in the graph exceeds the threshold, it is labeled as 1, otherwise 0.

3) Six-miutes graphs: The two-minute snapshots are unnecessarily short intervals for anomaly prediction in this system. Therefore, we consider each aggregated undirected six-minute graph as the feature window, and the subsequent undirected six-minute graph as the prediction window. The label of each six-minute graph is the logistic OR of the label values in the next immediate three two-minute graphs. The weights of the edges in a six-minute graph is based on an API threshold, if the weight is more than the API threshold that the graphs have been generated and labeled for, then those edges will have weight value 1 otherwise 0.

We consider a rolling window for feature and prediction window, with no gap between feature windows and prediction windows. For example, the first six-minute feature graph will be the aggregation of three two-minute graphs from indexes 0 to 2 with the label, which is based on the prediction window from two-minute graphs with indexes 3 to 5. And the second six-minute feature graph will be the aggregation of two-minute graphs indexes from 1 to 3 with the label, which is based on the prediction window from two-minute graphs indexes from 4 to 6. And the last six-minute feature graph will be the aggregation of two-minute graphs from indexes n-5 to n-3 with the label, which is based on the prediction window from two-minute graphs indexes from n-2 to n. Mostly in the six-minute graph, we have multiple edges between the same pair of nodes, so we just consider edge with maximum weight between the same pairs.

B. Node2Vec Embedding

Node2Vec uses a random walk algorithm and then word2vec to create embedding of the random walks. We use the python Node2Vec package, which takes as input a graph, the dimensionality N for Word2Vec neural network layer, the walk length, the number of walks, the number of neural network workers, and the p and q parameter values. For the six-minute graphs, we find out the optimal values of these parameter by trying various measurements. The random walk is a flexible neighborhood sampling strategy which allows smooth interpolation between breadth-first-search and depth-first-search, in which the p and q parameters guide the walk [10].

The optimal values for parameters which we find through tuning are dimensions = 50, walklength = 20, numwalks = 20, p = 1, and q = 1. We should mention that although increasing dimension will improve the accuracy, it can also cause overfitting if the embedded dimension goes close or beyond the original dimension. Algorithm. 1 explains all the steps in more details.

V. RESULTS AND EVALUATION

Data is recorded every two minutes, with seventeen servers that call three different APIs (Fig. 3). We group data by two-minute snapshots in a way that each time stamp will have one row in the data frame. To mitigate issues with sparsity, we consider a six-minute feature and prediction window. The reason we choose a window size of six minutes and not larger, is that for larger sizes, the label does not have a reliable value, and the models overfit.

	_time	ip_address	API1	API2	API3
0	8/26/2019 3:02	10.155.56.40	28103.477840	2229.161672	1993.162746
1	8/26/2019 3:02	10.155.56.44	1203.942135	4192.376419	468.450000
2	8/26/2019 3:02	10.155.56.45	8589.461385	252.956551	2461.344498
3	8/26/2019 3:02	10.155.56.46	801.239658	5530.270823	1865.235112
4	8/26/2019 3:02	10.155.56.47	9564.414603	3818.368130	223.875846

Fig. 3: Initial data from distributed system with three APIs and multiple IPs in each two-minute snapshot.

A. Original Data

We flatten the latency values of the APIs for all the servers in each two-minute window to have one row, 51 columns for each row by grouping the snapshot data (17 servers x 3 APIs = 51 columns, Table. 4). For the six-minute duration feature window, we flatten all the rows of three sequential two-minute flattened rows to generate a row of six-minute snapshot (3 rows x 51 columns = 153 columns). The label column for a six-minute snapshot row depends on all the label values in the next three sequential two-minutes data. If any of the label values of the three two-minute snapshot that show up in the predict window is 1, then the label for six-minute snapshot feature is 1 otherwise 0 (Fig . 5).

We should mention that, another reason that we consider six-minute for feature and prediction window was that the

Algorithm 1 Anomaly detection with Graph Embedding for node API1

$Params$:df=$dataframe$,k=100,d=50,num_walk=50
$length\_walk$=50,p=1.5,q=0.5

  Function MAIN()
    $API1 \leftarrow threshold\_API1$
    $API2 \leftarrow threshold\_API2$
    $API3 \leftarrow threshold\_API3$
    $k \leftarrow 100$
    $G2\_list, label2\_list \leftarrow$ 2MIN-GRAPH$(API1, API2, API3, df)$
    $G6\_list, label6\_list \leftarrow$ 6MIN-GRAPH$(G2\_list, label2\_list)$
    $EMB\_list \leftarrow$ EMBED-GRAPH$(G10\_list)$
    $Accuracy \leftarrow$ RANDOM-FOREST$(EMB\_list, label6\_list, k)$
End Function

  Function 2MIN-GRAPH$(API1, API2, API3, df)$
    $time\_list \leftarrow$ GET-TIME(df)
    For $time$ in $time\_list$:
      $df2 \leftarrow df[time]$
      $API\_list \leftarrow$ GET-APIs$(df2)$
      $IP\_list \leftarrow$ GET-IPs$(df2)$
      For API in API_list:
        For IP in IP_list:
          $latency \leftarrow$ GET-LATENCY(IP, API)
          If $latency >= API1$:
            $G2 \leftarrow$ CREATE_GRAPH$(IP, API, wight = 1)$
            $G2\_list \leftarrow$ APPEND-TOLIST$(G2)$
      If $AP1\_threshold\_2min >= API1$:
        $label2 \leftarrow 1$
      else:
        $label2 \leftarrow 0$
          $label2\_list \leftarrow$ APPEND-TOLIST$(label2)$
    $return$ $G2\_list, label2\_list$
End Function

  Function 10MIN-GRAPH$(G2\_list, label2\_list)$
    For $(i = 0; i < size(G2\_list); i++)$:
      $G6 \leftarrow$ MERGE$(G2\_list[i]..G2\_list[i+3])$
      $G6\_list \leftarrow$ APPEND-TOLIST$(G6)$
      If $(OR(, label2\_list[i].., label2\_list[i+3]) == 1)$:
        $label6 \leftarrow 1$
      else:
        $label6 \leftarrow 0$
      $label6\_list \leftarrow$ APPEND-TOLIST$(label6)$
    $return$ $G6\_list, label6\_list$
End Function

  Function EMBED-GRAPH$(G6\_list)$
    For $G6$ in $G6\_list$:
      $EMB \leftarrow$ NODE2VEC$(G6, d, num\_walk, length\_walk, p, q)$
      $EMB\_list \leftarrow$ APPEND-TOLIST(EMB)
    $return$ EMB_list
End Function

flattening of all the values of 10 and 20 minute feature window is a much more computationally intensive aggregation for this particular data set. In this case the number of feature columns will be 5 rows x 51 columns = 255 rows and 10 rows x 51 columns = 510 rows for 10 minutes and 20 minutes feature window, respectively.

B. Graph Represented Data

We create a graph for each grouped two-minute data, with nodes including servers and APIs (Fig. 6). So, the type of the

Fig. 4: Flattened two-minute data

Fig. 5: Flattened six-minute data.

data is not a row of some values of response times anymore. Each row represent a two-minute graph that keeps all the connectivity and knowledge intact. All the represented links have weight value 1 and the pairs of nodes with no link have weight value 0 based on the threshold of the connected API (or there was no request between the pairs in the beginning at all). For six minutes duration, we aggregate three consecutive two-minute graphs to one graph (Fig. 7) with a reduced dimensionality compared to original flattened data (10, 20 and 50). The Six-minute graphs will be embedded by Node2Vec algorithm (Fig. 8). Then the feature vectors are ready to be fed to the prediction algorithm.
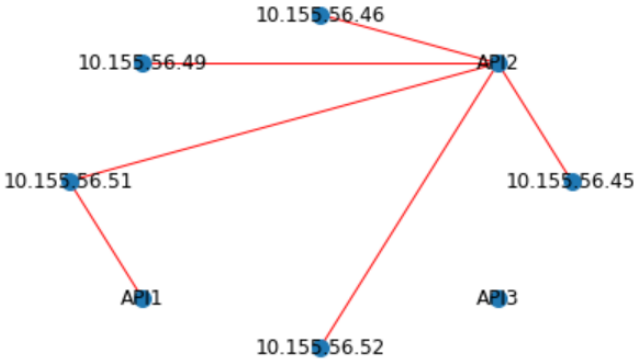
Fig. 6: A graph of a two-minute snapshot.

In the six-minute data aggregation, we shift the feature and prediction windows by 1. So, the feature windows' indices are

Approach	AUC
original	0.49514
Grpah d=10	0.54983
Grpah d=20	0.50564
Grpah d=50	0.59208

TABLE I: Evaluation of original data and graph representation of data for API1
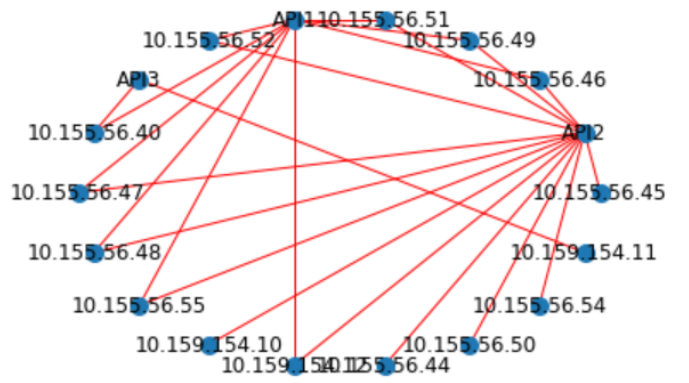
Fig. 7: A graph of a six-minute snapshot.

Fig. 8: Embedding feature vectors of a six-minute graph.

from 0 to n-1 and the labels' index are 1 to n. Then we applied k-fold cross validation with k=100. We measure the AUC for each API (Table. I). The threshold values of API1, API2, and API3 are 16770, 2184 and 4635 milliseconds, respectively.

The result can be improved significantly if the proposed approach is to be applied on the non-synthetic dataset. In the real dataset, the number of calls is more various in each snapshot window, and there are higher number of calls. The variations of latency are higher, and the threshold of values are more realistic.

This result also indicates that graph is a well-designed representation that improves predictability by 19.58% for graph d=50 compared to the original data (Table. I), by leveraging the temporal and dynamic additional information embedded in graph. We also present the training and inference time result for the original flattened and graph data with six minutes, ten minutes, and twenty minutes window times. The machine that is used to run these experiments has Windows 10 64-bit OS with core i7 CPU and 32GB RAM memory. For the largest dimension of the graph data, fifty, Table. II shows that the original data has higher training and inference time compared to the graph data. The consumed time has been decreased

Approach	6 minutes	10 minutes	20 minutes
Flattend	0.47	0.6	1
Grpah d=50	0.42	0.42	0.42

TABLE II: Training and inference time (seconds) of original and graph d=50 data for 6, 10 and 20 minute time window.

208

by 10.46% by using graph with six-minute window size. The consumed time with the ten and twenty minute window size decrease by 30% and 58% by using graph data, compared to original flattened data. The reason is that graph representation of the data always has 50 features, however, the original dataset has 153, 255 and 510 for 6 minutes, 10 minutes and 20 minutes time window, respectively.

VI. Conclusion

We implement node2vec graph embedding for our simulated distributed system data and compared it to the original data with a random forest classifier. Node2vec has two parts: random walk and word2vec. The random walk generates subgraphs of the original graph, and by considering it as a corpus, word2vec embeds the graph nodes to a desired dimension size. We achieve the dimensionality reduction in a way that all the relevant knowledge of the data remains intact. We used classification machine learning algorithm to predict a future anomalous event. The embedded graph shows better AUC than raw data. For future work, we could use graph representation to discover potential causality given that a proper DAG (directed acyclic graph) can be derived based on domain knowledge. In addition, edge and graph embedding and prediction can be further explored to identify the system dynamics from a different perspective. These can be interesting areas for future work. Based on our experiment, the random walk algorithm is not the best way to traverse a weight-based dynamic graph and the anomalous event has a direct relationship to an edge weight. We may be able to consider additional data with different features other than response time such as CPU, memory, and database event logs to enrich the graph design, in hope of improving both adaptability and prediction effectiveness.

References

[1] L. Akoglu, H. Tong, and D. Koutra, "Graph-based anomaly detection and description: A survey," 2014.

[2] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey," *ACM computing surveys (CSUR)*, vol. 41, no. 3, p. 15, 2009.

[3] W. L. Hamilton, R. Ying, and J. Leskovec, "Representation learning on graphs: Methods and applications," *arXiv preprint arXiv:1709.05584*, 2017.

[4] P. R. Kandel, *Node Similarity for Anomaly Detection in Attributed Graphs*. PhD thesis, Tennessee Technological University, 2019.

[5] I. Ahmed, X. B. Hu, M. P. Acharya, and Y. Ding, "Neighborhood structure assisted non-negative matrix factorization and its application in unsupervised point anomaly detection," *arXiv preprint arXiv:2001.06541*, 2020.

[6] A. Tosyali, J. Kim, J. Choi, Y. Kang, and M. K. Jeong, "New node anomaly detection algorithm based on nonnegative matrix factorization for directed citation networks," *Annals of Operations Research*, pp. 1–18.

[7] A. Markovitz, G. Sharir, I. Friedman, L. Zelnik-Manor, and S. Avidan, "Graph embedded pose clustering for anomaly detection," *arXiv preprint arXiv:1912.11850*, 2019.

[8] M. Venkatesan and P. Prabhavathy, "Graph based unsupervised learning methods for edge and node anomaly detection in social network," in *2019 IEEE 1st International Conference on Energy, Systems and Information Processing (ICESIP)*, pp. 1–5, IEEE, 2019.

[9] C. Lu, K. Ye, W. Chen, and C.-Z. Xu, "Adgs: Anomaly detection and localization based on graph similarity in container-based clouds," in *2019 IEEE 25th International Conference on Parallel and Distributed Systems (ICPADS)*, pp. 53–60, IEEE, 2019.

[10] A. Grover and J. Leskovec, "node2vec: Scalable feature learning for networks," in *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 855–864, ACM, 2016.

[11] G. Biau, "Analysis of a random forests model," *Journal of Machine Learning Research*, vol. 13, no. Apr, pp. 1063–1095, 2012.

F(X)-MAN: An Algebraic and Hierarchical Composition Model for Function-as-a-Service

Chen Qian* and Wenjing Zhu†

*School of Computer Science and Technology, Donghua University
†Institute of Scientific and Technical Information of Shanghai, Shanghai Library
Email: *chen.qian@dhu.edu.cn, †wjzhu@libnet.sh.cn

Abstract—Function-as-a-Service promises a new era in which functionalities are implemented, executed and managed on a cloud platform with the aim of developing and launching applications. This paper puts forward an algebraic and hierarchical model that has ability to construct composite services for cloud applications. Our model brings many advantages, which are also presented in this paper.

Keywords—Function-as-a-Service; algebraic composition; service model.

I. INTRODUCTION

Over the past decade, cloud computing has been developed at an unprecedented speed, whilst cloud applications have increasingly penetrated all areas of industry [1]. Cloud computing service model allows user to tailor the off-the-shelf services and adopt them nearly immediately, in spite of sometimes such a service does not completely customize to a user's application.

In the context of cloud computing, conventional services are developed and deployed in the form of monolithic computation units, in which all the code is interwoven into one large piece [2]. As a result, the monolith hinders the scalability and efficiency of client applications, especially when the number of participant services is increased. To address the issue, Function-as-a-Service (FaaS) is proposed, by means of structuring a *serverless architecture* that decomposes software applications into fine-grained functions [3]. Such functions are further invoked remotely at runtime.

At present, FaaS providers do not pay more attention to the service modeling. Unfortunately, the lack of modeling possibly (i) impairs the understanding of the system, (ii) blurs the details of software design, and (iii) obstructs quick and frequent changes in high levels of abstraction.

In this paper, we present a novel model specified for FaaS, which composes services in an algebraic and hierarchical manner.

II. RELATED WORK

With the growing of cloud computing, the concept of "everything is a service" has been formulated, such as *Platform-as-a-Service* (PaaS), *Infrastructure-as-a-Service* (IaaS) and *Software-as-a-Service* (SaaS) [4].

IaaS provides a base infrastructure for sale, e.g., virtual machines and repositories, on which user must configure and manage a platform before deploy applications on it. PaaS provides a platform which is already installed in the infrastructure. Hence, end user can develop and deploy the applications based on the platform. In comparison with the first two cloud services, SaaS is simpler. It enables cloud applications for direct use.

FaaS is even more flexible than PaaS. It allows developer to 'assemble' an application from functions on the cloud. However, current FaaS providers such as AWS Lambda, Google Cloud Functions, Microsoft Azure Functions and so forth focus on the language support, single function execution time and other properties, instead of the composition mechanism and underlying modeling methodology.

III. F(X)-MAN SERVICE MODEL

In this section, we put forward a service composition model, called F(X)-MAN, which is inspired by X-MAN component model and its extensions [5]–[9]. In F(X)-MAN, we regard both services and *exogenous* connectors as first-class entities. Figure 1 illustrates the F(X)-MAN constructs which we further describe below.

The F(X)-MAN model defines two types of services: *atomic* or *composite* service, which are demonstrated in Figure 1(a) and 1(b), respectively. An atomic service encapsulates a set of methods in the form of an input-output function with a purpose that different services can access, whereas a composite service consists of sub-services (atomic or composite) composed by exogenous connectors, which coordinate control flows between sub-services from the outside. Therefore, services are unaware they are part of a larger piece of behavior, which become perfectly suitable for FaaS. Ideally, the services do not have to be implemented by the same programming language.

Notably, F(X)-MAN model defines *algebraic service composition* [10]. This idea is enlightened by algebra where functions are composed hierarchically into a new function of the same type. Similarly, in F(X)-MAN, we utilize an exogenous connector as an operator to hierarchically compose multiple sub-services into a bigger service, while the resulting service can be further composed with other services, yielding a more complex one. The algebraic nature brings an advantage that F(X)-MAN services can be designed, implemented, deployed and remotely invoked with high flexibility. For instance, as

```
1    function run(args){
2        var a = args.param1;
3        var b = args.param2;
4        var x = firstMethod(a);
5        var y = secondMethod(x, b);
6        var feedback = {"result1":x, "result2":y};
7        return feedback;
8    }
9
10   function firstMethod(param){
11       //some code
12       return result1;
13   }
14
15   function secondMethod(param1, param2){
16       //some code
17       return result2;
18   }
```

args → i S_1 o → feedback

firstMethod;
secondMethond;
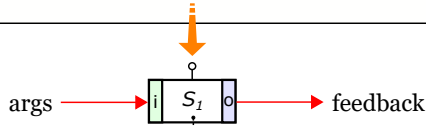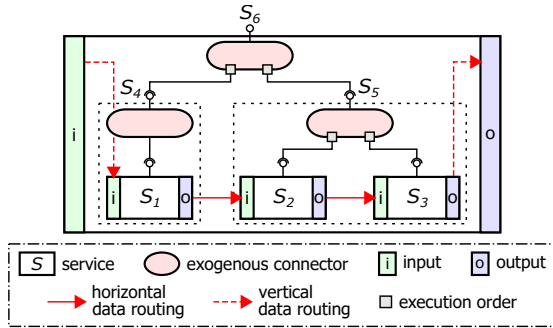
(a) Atomic service.

(b) Composite service.

Fig. 1. F(X)-MAN: Service model.

shown in Figure 1(b), when we deploy composite service S_6 in a network, in fact all its sub-services, no matter atomic (S_1, S_2, S_3) or composite (S_4, S_5), are exposed and ready for use.

In order to simulate the statements in computer programming, we define three categories of exogenous connectors in F(X)-MAN. *Composition connectors* compose multiple services by coordinating control flows among them, whereas *adaptation connectors* are applied to individual services with the aim of adapting the received controls. In addition, a *parallel connector* is specified to handle the concurrent invocation of services.

We hereby list the most commonly used statements along with their related connectors.

- **If statement.** The *if statement* (sometimes called *if-then statement*) is common across major programming languages [11]. If the expression is evaluated to true, statements inside the body of *if* are executed. Accord-

```
if (cond == true ) { execute S1; }
```
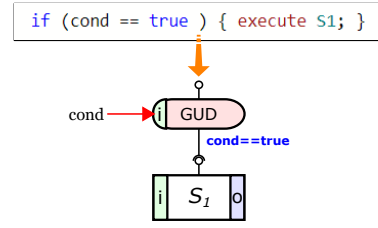

Fig. 2. F(X)-MAN: Guard.

ingly, we define an adaptation connector called *guard* that allows control to reach a service only if the condition is satisfied, as Figure 2 shows.

- **If-else statement.** In *if-else statement* (sometimes called *if-elif-else statement*), only the statements following the first expression that is evaluated to true will be executed [12]. Thus, we define a composition connector, namely *selector*, with the aim of branching. Figure 3 presents an example. If the input of selector has a value of 6, S_3 will be invoked. It is worth noting that *switch statement*

```
if (cond < 0) { execute S1; }
else if (cond < 5) { execute S2; }
else { execute S3; }
```
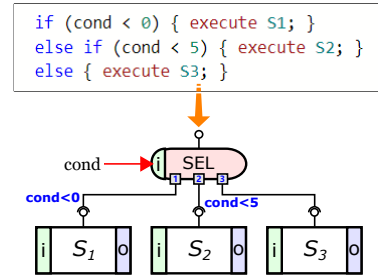

Fig. 3. F(X)-MAN: Selector.

usually can be converted to an equivalent if-else statement [13]. Therefore, we do not design another F(X)-MAN connector for it.

- **While statement.** The *while statement* presents the basic iteration [14]. In F(X)-MAN, we define an adaptation connector called *loop*, as illustrated in Figure 4. The loop connector repeatedly evaluates the expression, then invokes the service if the evaluation result is true, or stop the iteration if the evaluation result is false. Except for the while statement, *do-while statement* and *for statement* are also widely used, and both can be easily converted to equivalent while statements. Hence, we suggest to use the loop connector for all iterations.

- **Parallel statement.** The *parallel statement*, also known

```
while (cond < 10) { execute S1; }
```
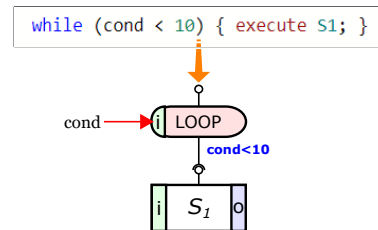

Fig. 4. F(X)-MAN: Loop.

as *concurrent statement*, indicates a certain synchronization of concurrent activities [15]. Figure 5 shows a composition connector called *parallel* that denotes contemporaneous execution of three services. Notably, when parallel connectors are used in an F(X)-MAN service, we must pay more attention to its underlying issues, e.g., deadlock [16] and race condition [17].
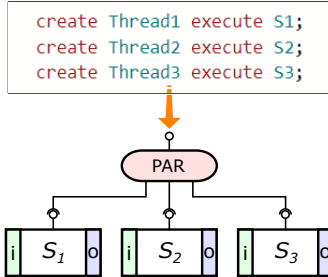
Fig. 5. F(X)-MAN: Parallel connector.

- **Block statement.** The *block statement*, also known as *compound statement*, is generally adopted to group a sequence of multiple statements [18]. Therefore, F(X)-MAN model provides a composition connector, namely *sequencer*, which allows user to determine the execution order.
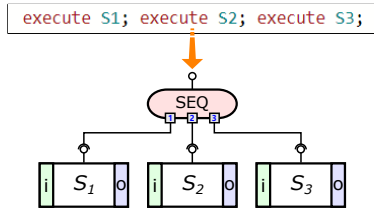
Fig. 6. F(X)-MAN: Sequencer.

While the exogenous connectors coordinate the control flows, the *data channels* coordinate the data flows. After a composite service is structured, we need to add data channels between services in order to define the direction of each data flow. Such a channel links the *input* and *output* of services. Figure 1(b) demonstrates two types of data channels in the composite service: *horizontal data routing* and *vertical data routing* [19]. The former is between two individual services, which indicates a service passes the outcome data to another, while the latter is data propagation between the services and its sub-services, which illustrates the data received by the composite service is passed to the first invoked sub-service, whereas the outcome data of last invoked sub-service becomes the output of the composite service. It is worth noting that we use JSON as the data interchange format, because (i) JSON has simple API that are available for many languages [20], (ii) the name-value pairs provides consistent patterns that are understandable by any user [21], and (iii) JSON provides faster object serialization and deserialization with less resources in comparison with other formats [22].

Next section presents how to use the F(X)-MAN composition semantic to construct FaaS.

IV. EXAMPLE

In this section, we present an example of using F(X)-MAN service model for application development based on FaaS.
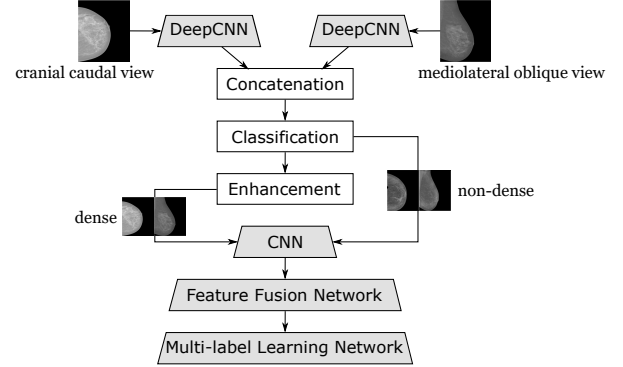
Fig. 7. A brief overview of mammography reading system.

Over the past decade, artificial intelligence (AI) technologies have been developed at an unprecedented speed, whilst AI applications have increasingly penetrated all areas of industry [23], e.g., intelligent healthcare [24] and E-commerce [25]. As a nature consequence, developing such a practical AI application commonly requires experts from various domains. Moreover, realizing the built-in AI algorithms such as training neural networks requires significant investment of time, effort, data and computing resource. Therefore, to the companies who want to focus on the core business whilst gain the benefits from data, FaaS looks like a promising solution, by means of a distribution model allowing AI services outsourcing offered by third-party vendors [26].

We hereby use F(X)-MAN service model to construct a *mammography reading system* (adapted from [27]) with the
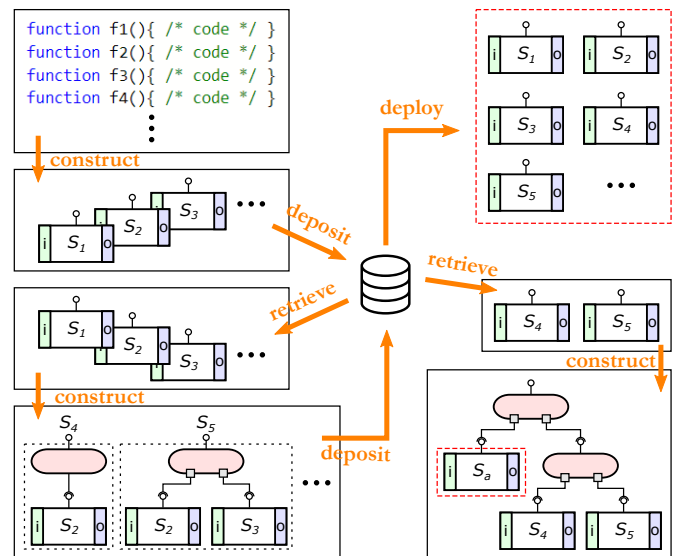
Fig. 8. F(X)-MAN: Workflow.
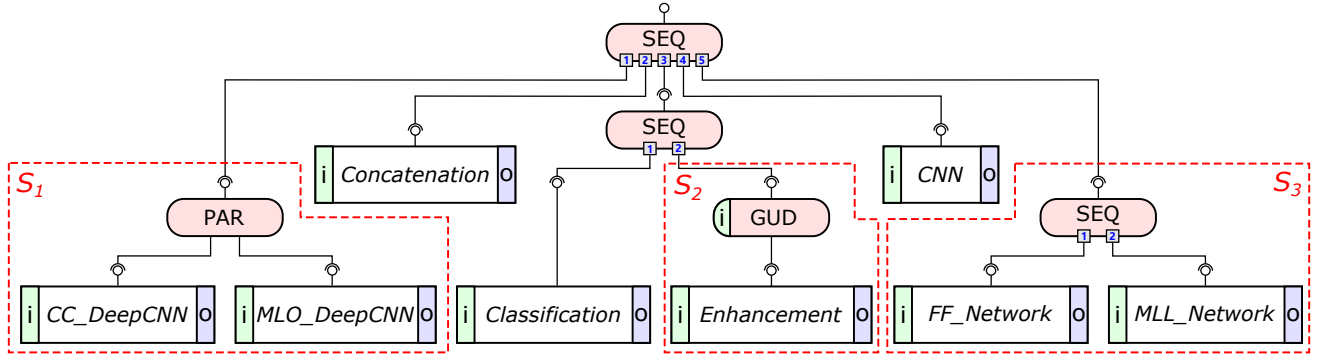
Fig. 9. F(X)-MAN: Mammography reading system.

ability to detect and label the lesions for breast cancer diagnosis. This system analyzes two-view mammography images through several well-trained deep learning models, each of which is a multi-layer neural network that processes medical data in a hierarchical fashion. Figure 9 demonstrates the primitive functional requirements.

At design-time, we firstly implement the relevant functions, and use them to create atomic services. Then we deposit the atomic services to a repository, and later retrieve them to construct composite services. The resulting services can be deposited again for further construction. After that, we can deploy the services (atomic or composite) from the repository to other developers and becomes a part of their programs at run-time, or use these services as building blocks to construct our own applications. The whole workflow of F(X)-MAN is illustrated in Figure 8.

Accordingly, we construct the mammography reading system, as shown in Figure 9. For the clarity, we omit the data channels. It is worth noting that S_1, S_2 and S_3 are services developed and deployed by third-parties, which are seamless connected to our system.

V. EVALUATION

This section provides an evaluation of our algebraic and hierarchical composition model via several quality attributes, i.e., low coupling, testability, scalability, reusability, maintainability and evolvability.

A. Low Coupling

In software evaluation, *coupling* is a term used to measure the degree of connection and the amount of interaction between modules [28]. The higher the coupling, the more likely it is that changes to the inside of a module will effect the original behavior of another one [29]. Thereby, low coupling is one of the ultimate pursuits for software engineering.

There are six levels of coupling, as enumerated in increasing order of malignity [30]: *data coupling*, *stamp coupling*, *control coupling*, *common coupling* and *content coupling*. Our F(X)-MAN service model only generates the loosest two couplings in a system:

1) In data coupling, the communication between services is limited, i.e., via scalar parameters, in which only simple

arguments are allowed to pass directly, e.g., variable and array. The passed data is always used for an invocation or a return of control [31].

2) Likewise, the communication in stamp coupling is also limited. But it passes composite data item, which usually is a entire data structure. Thus, sometimes a data structure may contain pieces of data that are unnecessary to the recipient module [32].

The coupling has a huge impact on testability, scalability, reusability, maintainability and evolvability.

B. Testability

Testability refers to the effort and time required to validate the software against its requirement. Thus, a system with better testability can be validated faster and easier [33]. However, perform testing in a serverless environment is never a simple task. An application using FaaS indicates that local code and foreign code are tangling together. It is difficult to run such
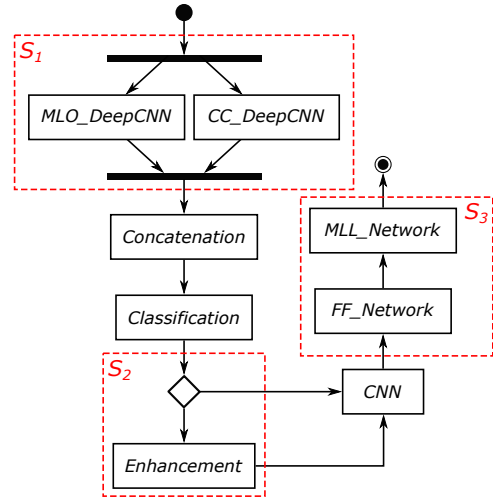
Fig. 10. F(X)-MAN: Deriving a statechart.

an application locally, unless the local environment can fully simulate the cloud environment [34].

Although the problem cannot be tackled once for all, we do facilitate the testability of systems build by F(X)-MAN service model. Firstly, such a system is completely modularized with

low coupling, which means every behavior implemented in the local environment can be examined in isolation by means of unit testing. Secondly, the control flows are coordinated by the exogenous connectors, i.e., outside of the services, which implies we can verify the system behavior through a statechart directly derived by following the control flows, without take the services (local and remote) into account. Figure 10 is the statechart derived from the mammography reading system in Figure 9.

C. Scalability

Scalability is a term that frequently appears in a variety of computer science domains. Hence, we must explicitly understand the scalability needed to be evaluated in the scope of FaaS. In order to avoid the ambiguity, we hereby define the scalability from two different aspects.

From the perspective of software engineering, scalability is a fundamental quality referring to the impact of code expansion [35]. In other words, the scalability of F(X)-MAN denotes the effectiveness of F(X)-MAN when used on differently sized problems. As presented in Section III and IV, F(X)-MAN provides outstanding mechanisms for partitioning, composition and visibility control, which result in great scalability [36]. For example, the mammography reading system constructed in Figure 9 can be regarded as another F(X)-MAN service, which can be composed with other services, such as a *mammography report generator*, to create a *breast cancer auxiliary diagnosis application*, which can be further composed again for a very large software.

On the other hand, scalability in the context of cloud computing describes the capability of a system to increase its throughput under an increased load, e.g., creating more service instances [37]. As a matter of fact, comparing to the traditional monolithic models, FaaS achieves much better scalability. Figure 11 makes a comparison. As Figure 11(a) shows, a monolithic model encapsulates all its functionalities into a single process, and scales by replicating the entire monolith. Contrariwise, current FaaS models put implemented functionalities into separate services, and replicate the desired services for scaling, as expressed in Figure 11(b). Apparently, in FaaS, only services with higher demand will be scaled, while the monolithic models anyhow waste resources.

Our F(X)-MAN model make a further improvement on scalability. Except the advantages brought by general FaaS models, F(X)-MAN also has a superb tailorability, which is another way of assessing scalability [36]. For example, we can directly instantiate the sub-services of a F(X)-MAN composite service, and use them for new compositions, as illustrated in Figure 11(c).

D. Reusability

Reusability is the capability of a previously implemented service to be used again or used repeatedly in part or in its entirely, with or without modification [38]. As aforementioned, due to the loose couplings among F(X)-MAN services, for a composite service, the reuse can happen at every level of
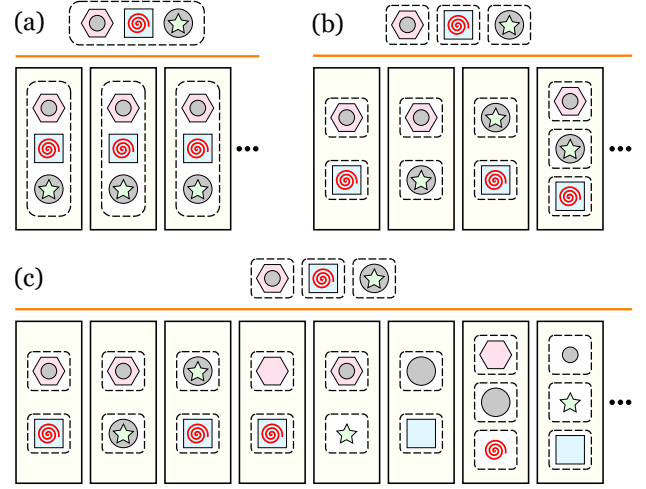
Fig. 11. Comparison of scalability.

granularity, from the atomic services to itself. Hence, the reusability of a concrete application also depends on the software design, e.g., how many methods should be put in a service. The details has been discussed earlier and demonstrated in Figure 8.

E. Maintainability

Maintainability is a composition of three main subcharacteristics: *analyzability*, *changeability*, and *understandability* [39]. So, we perform the evaluation of maintainability based on these three quality attributes, which are interpreted as follows:

- **Analyzability.** It describes the capability of model or source code of a software to be diagnosed for deficiency. In F(X)-MAN, a composite service can be analyzed from two perspectives: the control flow can be identified by its exogenous connectors, while the data flow can be observed by data channels.
- **Changeability.** It refers to the possibility and ease of modification in an application. Because of the low coupling, we can change any service in an application without effecting others, all we need to confirm is the related data channels.
- **Understandability.** It indicates how easy to understand an application by its developers and users. It becomes obvious that an application constructed by F(X)-MAN is a tree, whose structure visualizes the hierarchical composition of services. Moreover, every connector visualizes a fixed semantic that can form a statement.

F. Evolvability

Evolution of software is inevitable in industry, due to the changing requirements must be satisfied during the life cycle. Thus, the cost of software mainly depends on the evolution in long term. Thus, evolvability is the capability of an application to enable its own evolution. In comparison with changeability, evolvability refers to the change caused by new requirements. As we presented before, the computational nature of F(X)-MAN allows us replace any part of an F(X)-MAN architecture

with a new service in a simple fashion, while maintaining its architectural integrity.

VI. Conclusions and Future Work

This paper presents an algebraic and hierarchical composition model for FaaS. In the future, we plan to make an empirical study for the evaluation of F(X)-MAN model and implement a development tool.

References

[1] S. Marston, Z. Li, S. Bandyopadhyay, J. Zhang, and A. Ghalsasi, "Cloud computing—the business perspective," *Decision support systems*, vol. 51, no. 1, pp. 176–189, 2011.

[2] M. Stigler, "Understanding serverless computing," in *Beginning Serverless Computing*. Springer, 2018, pp. 1–14.

[3] L. Carvalho and A. P. F. de Araújo, "Framework node2faas: Automatic nodejs application converter for function as a service," in *Proceedings of the 9th International Conference on Cloud Computing and Services Science*, vol. 1, 2019, pp. 271–278.

[4] B. P. Rimal, E. Choi, and I. Lumb, "A taxonomy and survey of cloud computing systems," in *2009 Fifth International Joint Conference on INC, IMS and IDC*. Ieee, 2009, pp. 44–51.

[5] K.-K. Lau and S. di Cola, *An Introduction to Component-based Software Development*. World Scientific, 2017.

[6] S. di Cola, C. Tran, K.-K. Lau, C. Qian, and M. Schulze, "A component model for defining software product families with explicit variation points," in *Proceedings of 19th International ACM SIGSOFT Symposium on Component-Based Software Engineering*. ACM, April 2016, pp. 79–84.

[7] C. Qian and K.-K. Lau, "Enumerative variability in software product families," in *Computational Science and Computational Intelligence (CSCI), 2017 International Conference on*. IEEE, 2017, pp. 957–962.

[8] R. Arshad and K.-K. Lau, "Reverse engineering encapsulated components from object-oriented legacy code," in *Proceedings of The 30th International Conference on Software Engineering and Knowledge Engineering*. KSI Research Inc., July 2018, pp. 572–577.

[9] D. Arellanes and K.-K. Lau, "Exogenous connectors for hierarchical service composition," in *Proceedings of 2017 IEEE 10th International Conference on Service-Oriented Computing and Applications*. IEEE, 2017, pp. 125–132.

[10] D. Arellanes and K.-K. Lau, "Algebraic service composition for user-centric IoT applications," in *International Conference on Internet of Things*. Springer, 2018, pp. 56–69.

[11] I. Griffiths, *Programming C# 8.0: Build Cloud, Web, and Desktop Applications*. O'Reilly Media, 2019.

[12] I. Kalb, "If, else, and elif statements," in *Learn to Program with Python 3*. Springer, 2018, pp. 103–141.

[13] M. Ogihara, "The switch statements," in *Fundamentals of Java Programming*. Springer, 2018, pp. 245–261.

[14] S. Kedar, *Principles Of Programming Languages*. Technical Publications, 2008.

[15] C. Snow, *Concurrent Programming*, ser. Cambridge Computer Science Texts. Cambridge University Press, 1992.

[16] W. Haque, "Concurrent deadlock detection in parallel programs," *International Journal of Computers and Applications*, vol. 28, no. 1, pp. 19–25, 2006.

[17] R. H. Netzer and B. P. Miller, "What are race conditions? Some issues and formalizations," *ACM Letters on Programming Languages and Systems (LOPLAS)*, vol. 1, no. 1, pp. 74–88, 1992.

[18] E. Organick, A. Forsythe, and R. Plummer, *Programming Language Structures*. Elsevier Science, 2014.

[19] K.-K. Lau and C. Tran, "X-MAN: An MDE tool for component-based system development," in *Proc. 38th EUROMICRO Conference on Software Engineering and Advanced Applications*. IEEE, 2012, pp. 158–165.

[20] B. Smith, *Beginning JSON*. Apress, 2015.

[21] I. Baldini, P. Castro, K. Chang, P. Cheng, S. Fink, V. Ishakian, N. Mitchell, V. Muthusamy, R. Rabbah, A. Slominski *et al.*, "Serverless computing: Current trends and open problems," in *Research Advances in Cloud Computing*. Springer, 2017, pp. 1–20.

[22] N. Nurseitov, M. Paulson, R. Reynolds, and C. Izurieta, "Comparison of json and xml data interchange formats: a case study." *Caine*, vol. 9, pp. 157–162, 2009.

[23] S. J. Russell and P. Norvig, *Artificial intelligence: A modern approach*. Malaysia; Pearson Education Limited,, 2016.

[24] P. Hamet and J. Tremblay, "Artificial intelligence in medicine," *Metabolism*, vol. 69, pp. S36–S40, 2017.

[25] F. J. Martínez-López and J. Casillas, "Artificial intelligence-based systems applied in industrial marketing: An historical overview, current and future insights," *Industrial Marketing Management*, vol. 42, no. 4, pp. 489–495, 2013.

[26] S. Parsaeefard, I. Tabrizian, and A. Leon-Garcia, "Artificial intelligence as a service (AI-aaS) on software-defined infrastructure," in *2019 IEEE Conference on Standards for Communications and Networking (CSCN)*. IEEE, 2019, pp. 1–7.

[27] X. Wang, J. Li, and C. Qian, "Semantic label prediction of mammography based on CC and MLO views," in *Proceedings of 18th IEEE International Conference on Machine Learning and Applications*. IEEE, 2019, forthcoming.

[28] D. King, *Current practices in software development: a guide to successful systems*. Prentice Hall, 1984.

[29] F. Coda, C. Ghezzi, G. Vigna, and F. Garzotto, "Towards a software engineering approach to web site development," in *Proceedings of the 9th international workshop on Software specification and design*. IEEE Computer Society, 1998, p. 8.

[30] A. J. Offutt, M. J. Harrold, and P. Kolte, "A software metric system for module coupling," *Journal of Systems and Software*, vol. 20, no. 3, pp. 295–308, 1993.

[31] M. Hitz and B. Montazeri, "Measuring coupling and cohesion in object-oriented systems," 1995.

[32] G. Feuerlicht, "Simple metric for assessing quality of service design," in *International Conference on Service-Oriented Computing*. Springer, 2010, pp. 133–143.

[33] M. Mattsson, H. Grahn, and F. Mårtensson, "Software architecture evaluation methods for performance, maintainability, testability, and portability," in *Second International Conference on the Quality of Software Architectures*. Citeseer, 2006.

[34] P. Leitner, E. Wittern, J. Spillner, and W. Hummer, "A mixed-method empirical study of function-as-a-service software development in industrial practice," *Journal of Systems and Software*, vol. 149, pp. 340–359, 2019.

[35] M. Anastaopoulos and C. Gacek, "Implementing product line variabilities," Fraunhofer Institut Experimentelles software Engineering, Kaiserslautern, Germany, Technical Report IESE-Report No. 089.00/E, Version 1.0, Nov. 2000.

[36] M. Laitinen, M. E. Fayad, and R. P. Ward, "Thinking objectively: The problem with scalability," *Communications of the ACM*, vol. 43, no. 9, pp. 105–107, 2000.

[37] D. F. Garcia, G. Rodrigo, J. Entrialgo, J. Garcia, and M. Garcia, "Experimental evaluation of horizontal and vertical scalability of cluster-based application servers for transactional workloads," in *8th International Conference on Applied Informatics and Communications (AIC'08)*, 2008, pp. 29–34.

[38] S. Thakral and S. Sagar, "Reusability in component based software development - a review," *World Applied Sciences Journal*, vol. 31, no. 12, pp. 2068–2072, 2014.

[39] E. Bagheri and D. Gasevic, "Assessing the maintainability of software product line feature models using structural metrics," *Software Quality Journal*, vol. 19, no. 3, pp. 579–612, 2011.

Privacy-aware OrLa Based Access Control Model in the Cloud

Pengfei Shao, and Shuyuan Jin*

School of Data and Computer Science, Sun Yat-sen University, Guangzhou, China

GuangDong Key Laboratory of Information Security Technology, Sun Yat-sen University, Guangzhou, China

Email: shaopf@mail2.sysu.edu.cn, jinshuyuan@mail.sysu.edu.cn

Abstract—In recent years, cloud computing has been widely adopted by an increasing number of enterprises and individuals because of its attractive features, such as its large scale, low costs, and pay-per-use. Nevertheless, traditional access control models cannot satisfy the security requirements of complex cloud environments. In this paper, a privacy-aware access control model (Pa-OrLaBAC) is proposed that emphasizes privacy protection and flexibility. This model combines Organization based Access Control (OrBAC) model with Label-based access control (LaBAC) model and retains their respective advantages, making it more suitable for the cloud. By introducing the concept of purpose, the issue of lacking privacy protection is well addressed and the problem of the separation of control and ownership is alleviated to some extent. In order to get a more precise access purpose, two methods (static declaration and dynamic acquisition) and a negotiation module are also applied in this model. Finally, we illustrate the use of Pa-OrLaBAC with a case study and summarize this model.

Keywords—access control, cloud, privacy protection, flexibility

I. INTRODUCTION

There are many definitions of cloud computing, the most widely accepted of which was proposed by the National Institute of Standards and Technology (NIST): "Cloud Computing is a model for enabling convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction" [1]. In recent years, cloud computing has been adopted by an increasing number of enterprises and individuals as a new computing model because of its appealing characteristics, such as its ultra-large scale, high scalability, high dynamics, low costs and pay-per-use. In addition, users can obtain the required resources through the network without geographical limitations. Despite these benefits, the security of cloud computing remains a major factor hindering its development. Therefore, ensuring cloud security is one of the urgent tasks in cloud computing environments [2].

Access control [3] is the fundamental security method for the promotion and protection of network security and is used to confirm or deny a request for one subject to access an object. A proper access control model can prevent unauthorized users from maliciously or unintentionally obtaining data [4].

Moreover, there are a huge number of users and a huge amount of data in the cloud, making the traditional coarse-grained access models based on pre-defined rules inappropriate.

The cloud-specific outsourcing business model separates ownership and control. With the increasing number of companies and individuals migrating their data to the cloud, the protection of users' private information has become a major focus in this field. Privacy is defined as the rights of individuals to decide when, how, and to what extent their information could be shared with others [5]. The Organisation for Economic Co-operation and Development (OECD) Guidelines [6] are the most widely adopted principles of privacy protection, and they claim that purposes, conditions and obligations are the key elements of privacy preserving access control models. The primary concern of a privacy policy is the specific reason why the data have been collected or used [7]. However, the traditional access models are not designed to enforce privacy policies and satisfy privacy protection requirements [8]. That is because these models are based on guiding the operations of the user on relative data [9].

Nevertheless, for now, few existing studies on access control models focus on both privacy protection and flexibility. Considering the aforementioned reasons, the main aim of this paper is to address the problem of privacy disclosures in the cloud while ensuring the flexibility of the access control model.

This paper proposes an extensible access control model for privacy protection — the Privacy-aware OrLa based access control model (Pa-OrLaBAC) — for cloud environments. The features of this model are as follows.

1) Privacy-aware. By introducing the concept of purpose, privacy protection is strengthened and the shortcoming, which is the separation of ownership and control, of the cloud is alleviated to some degree.

2) Flexibility. The presented model preserves the extensibility of the Label-based access control model. In Pa-OrLaBAC, other proper labels could be added according to the specific requirements.

The rest of this paper is organized as follows: Section II summarizes related works. Section III presents necessary preliminaries to establish our model. Section IV details the Pa-OrLaBAC model. Section V introduces a usage scenario. Section VI concludes this paper and points out our future direction.

II. Related Work

Traditional access control models can be divided into three main classes: discretionary access control (DAC) [3], mandatory access control (MAC) [10], and role-based access control (RBAC) [9].

DAC restricts object access on the basis of the identity of the subjects or the groups to which they belong. For now, it is the most commonly used model in computer operating systems. However, it does allow legitimate users to pass permissions or rights to others, regardless of whether they are malicious or not.

In MAC, only the system administrator or central authority is responsible for designing and managing access control policies that cannot be changed or granted by the users. MAC is often used in military areas that require multi-level security. Thus, this centralized authorization approach can neither provide flexibility nor support the separation of duties or Least Privilege.

RBAC was designed to solve the shortcomings of the previous models. In RBAC, permissions are associated with roles, and users gain accesses to objects by acquiring the appropriate roles. RBAC satisfies the security needs of various organizations and also improves the efficiency and reduces the complexity of authorization management. However, due to the many-to-many mapping relationships between roles and users, when it is applied to the cloud, the role explosion problem may occur, and the flexibility may be affected by the millions of dynamic users and permissions that exist in commercial networks [11].

As is shown above, traditional access control models are not perfectly applicable to the cloud. Therefore, a variety of new access control modes have been proposed. Among which the attribute-based access control (ABAC) [12] model, the organization based access control (OrBAC) [13] model, and the label-based access control (LaBAC) [14] model are the most representative.

In ABAC, access is granted or denied according to a set of attributes that are associated with the subject, object and environment. It does possess more granularity and flexibility compared with traditional models. The main drawback of ABAC is how to accurately select the attributes for access decisions in a specific application environment such as the cloud [11]. Otherwise, designing a rich computational language to define attribute-based rules makes policy update and policy review NP-complete or even indeterminate problems [14].

Mustapha Ben Saidi et al. [15] proposed an access control model based on OrBAC that introduces the concept of the Trusted Third Party. Their goal is to better control the external connections of users with different accesses. To ensure a continuity of critical infrastructures, Nawal AIT AALI et al. [16] proposed an access control model based on trust management using the OrBAC model. This model can both manage different resource access policies from other organizations and keep the trust between collaborating organizations. By extending OrBAC with new entities and introducing a new trust relationship among tenants, MA Madani et al. [17] proposed an approach that ensures the access control to the shared resources in a collaborative session in cloud environments.

Roger E. Sanders [18] proposed a method for securing data using label-based access control (LBAC) in which data are protected by the security label. Only the administrator can modify the labels. In [19], access is managed based on the user label and the data label. Labels provide extra protection, especially for sensitive data such as credit cards and Social Security Numbers (SSNs). Chen et al. [20] proposed a novel framework, the multi label-based access control model, which uses different labels to provide access security for big data applications. Chinnasamy P et al. [21] proposed a solution to overcome data security defects by implementing multi label-based scalable access control as a service for the cloud. This model enables data owners to keep the authority over their resources.

Although the OrBAC model considers the context when making access requests and overcomes the limitation of directly binding permissions to roles, it is more suitable for centralized structure because of lacking flexibility [11] and it mainly restricts access control with respect to the subject.

LaBAC expresses authorization policies in the form of enumeration, and it is a variable-grained access control method that labels subjects and objects. Meanwhile the drawback, the separation of ownership and control, that is caused by cloud computing is alleviated. However, one concern about this model is that the costs of storing the potentially large number of enumerated tuples would be high [14]. Furthermore, neither of them takes both privacy protection and the flexibility of the access control model into account.

As we all know, utilizing multiple models with other enhancements may achieve a better result [22]. Inspired by this thought, in this paper, we propose a new access control model (Pa-OrLaBAC) for cloud computing. It could be an effective method to ensure the security of data in cloud environments.

III. Preliminaries

In this section, we introduce the required concepts that will be used in the Pa-OrLaBAC model.

A. Organization based Access Control (OrBAC)

The core feature of OrBAC [13] is the organization, and it defines a new level of abstract entities that are separated from concrete ones. The entities of subject, action and object are abstracted as role, activity and view, respectively. The other entity is the context, which is used to specify the concrete circumstances in which organizations grant role permissions to perform activities on views. Unlike RBAC, in OrBAC, after the subject is granted to the appropriate role, it no longer immediately obtains the access permission to the object. Instead, on the abstract level, the role obtains permission to perform an activity on the view in a certain context. Then, the access permission of the concrete level is derived from the abstract one. To make this transition, OrBAC also defines

some relationships that associate abstract entities with concrete ones. The framework of the model is shown as Figure 1.

- The *Employ* relationship
  In this model, a Subject is an active entity, i.e., a user. The entity Role indicates the status of the subject in the organization. If *org* is an organization, *s* is a subject and *r* is a role, then *Employ (org, s, r)* means that *org* employs subject *s* in role *r*.
- The *Use* relationship
  The entity Object is the resource being accessed. A View corresponds to a set of objects that satisfy a common property. If *org* is an organization, *o* is an object and *v* is a view, then *Use (org, o, v)* means that *org* uses object *o* in view *v*.
- The *Consider* relationship
  The entity Action contains computer actions such as read, write, and send. In some cases, different organizations may decide that the same action comes under different activities. Therefore, if *org* is an organization, α is an action and *a* is an activity, then *Consider (org, α, a)* means that *org* considers that action α falls within activity *a*.
- The *Define* relationship
  Contexts could be used to specify the concrete circumstances where organizations grant role permissions to perform activities on views. If *org* is an organization, *s* is a subject, *o* is an object, α is an action and *c* is a context, then *Define (org, s, o, α, c)* means that within organization *org*, context *c* is true among subject *s*, object *o* and action α.
- The *Permission* relationship
  This is the access authorization at the abstract level. If *org* is an organization, *r* is a role, *v* is a view, *a* is an activity and *c* is a context, then *Permission (org, r, v, a, c)* means that organization *org* grants role *r* permission to perform activity *a* on view *v* within context *c*.
- The *Is_permitted* relationship
  This is the concrete authorization that can be derived from the abstract one. *Is_permitted (s, o, α)* means that subject *s* is permitted to perform action α on object *o*.

The procedure through which a subject can obtain permission to perform an action on the object is as follows:

Employ (org, s, r) $\wedge$ Use (org, o, v) $\wedge$ Consider (org, α, a) $\wedge$ Define (org, s, o, α, c) $\wedge$ Permission (org, r, v, a, c) $\longrightarrow$ Is_permitted (s, o, α).
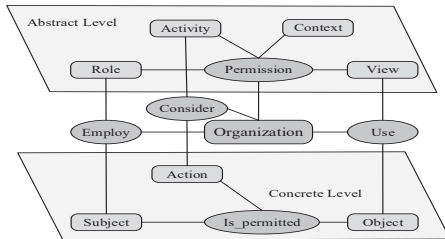
Fig. 1. Basic framework of OrBAC model.

B. Label-based access control (LaBAC)

The LaBAC [14] model expresses policies in the form of enumeration. Every subject and object is tagged using labels. A Label in LaBAC is a precise type of attribute. Values can be assigned by the administrator. The basic framework of LaBAC is shown in Figure 2.

In this model, the sets of users, objects and actions are denoted by U, O and A, respectively. Users are associated with a label function named uLabel, which maps the user to one or more values from the finite set UL (user label values). Similarly, the objects use oLabel to map the object label values (OL). A policy consists of a subset of tuples from the set of all tuples UL x OL. Only one policy can be defined for each action, which is denoted as Policy_a. If and only if the two-tuples group (ul, ol) $\in$ Policy_a is true will the related action be authorized.

An issue in LaBAC is that a complex access policy may need many or a significantly large number of enumerated policies to be defined. This may lead to a situation that the number of labels is greater than the number of entities in the system.
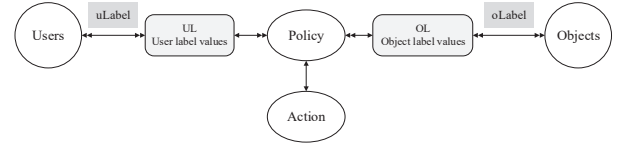
Fig. 2. Basic framework of LaBAC model.

C. Privacy protection

Typical privacy policies for data include purposes, conditions and obligations. The obligation designates the actions that must be followed after access is allowed. Conditions are prerequisites that should be satisfied when any action can be performed [23]. Purposes describe the reasons why the data are collected or used [24]. Platform for Privacy Preferences (P3P) defines the purposes as "the reason(s) for data collection and use" and specifies a set of purposes (World Wide Web Consortium). In commercial situations, purposes normally have hierarchical associations, i.e., generalization and specialization relationships. For instance, a group of purposes such as direct-marketing and third-party marketing can be represented by a more general purpose, marketing. We adopt the purpose definition from Byun et al. [7].

Definition 1 (Purpose and Purpose Tree): A purpose describes the reason(s) for data collection and data access. A set of purposes, which is denoted as Ω, is organized in a tree structure, which is referred to as a Purpose Tree and denoted as Φ. Each node in the Purpose Tree represents a purpose in Ω and each edge represents a hierarchical relation (i.e., specialization and generalization) between two purposes.

Figure 3 shows an example of a purpose tree. For instance, p_i and p_j are two purposes in Φ, and we say that p_i is an ancestor of p_j (or p_j is a descendent of p_i) if there is a downward path from p_i to p_j in Φ.

Fig. 3. Purpose Tree.

Fig. 4. The main framework of Pa-OrLaBAC.
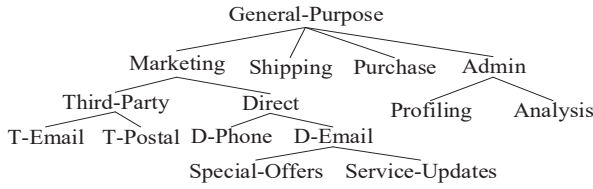
For a set of purposes, R, in Φ, the following notations will be used.

$R^\blacktriangle$ is the set of all nodes that are ancestors of the nodes in R, including the nodes in R themselves.

$R^\blacktriangledown$ is the set of all nodes that are descendants of the nodes in R, including the nodes in R themselves.

$R^\blacklozenge$ is the set of all nodes that are either ancestors or descendants of the nodes in R, that is, $R^\blacklozenge = R^\blacktriangle \cup R^\blacktriangledown$.

Definition 2 (Access Purpose, AP): An access purpose is used to access data objects, and it should be confirmed when data are requested.

Definition 3 (Intended Purpose, IP): The intended purpose is the data-related purpose that regulates data access. When access is requested, the access purpose is checked against the intended purposes. An intended purpose consists of two components: the Allowable Intended Purposes (AIP for short) and the Prohibited Intended Purposes (PIP for short).

Allowable Intended Purpose (AIP): Data providers explicitly allow data access for a particular purpose.

Prohibited Intended Purpose (PIP): Data providers strictly disallow data access for a particular purpose.

Therefore, an intended purpose (IP) is a tuple $\langle$AIP, PIP$\rangle$, where AIP $\subseteq \Phi$ and PIP $\subseteq \Phi$ are two sets of purposes. We adopt the denial-takes-precedence policy that PIP overrides AIP if there are conflicts between the AIP and the PIP for the same data element.

Definition 4 (Access Purpose Compliance): Let Φ be a purpose tree. IP = $\langle$AIP, PIP$\rangle$ be an intended purpose and AP be an access purpose that are defined over Φ, respectively. AP is said to be compliant with IP according to Φ if and only if the following two conditions are satisfied:

1. AP $\in AIP^\blacktriangledown$, and
2. AP $\notin PIP^\blacklozenge$.

IV. OUR PROPOSED MODEL

The access control model that is proposed in this work combines the OrBAC with LaBAC and introduces the concept of purpose. Its main framework is illustrated in Figure 4.

This paper only uses the "Purpose" label to protect privacy. By integrating the advantages of OrBAC and LaBAC, flexibility and fine-granularity can be achieved. It should be noticed that other proper labels could be added according to the specific requirements. In the following statement, the same parts that were previously depicted will not be described again, and new components that are extended or modified in Pa-OrLaBAC will be explained in detail.
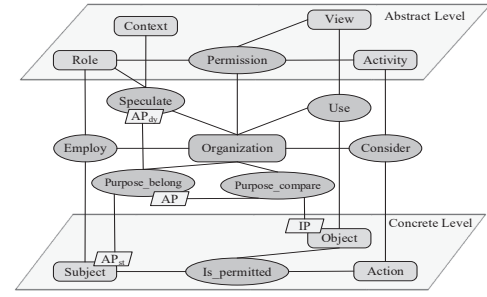
A. Context

In the traditional OrBAC, the actual circumstances in which organizations grant role permissions to perform activities on views could be clarified by using the entity context. In Pa-OrLaBAC, contexts will be used to specify the concrete conditions that organizations use to determine the dynamic access purpose (AP_{dy}) of the role. The process of inferring the AP_{dy} using the Context will be given later.

B. Access Purpose Authorization

a) Access Purpose: There are three possible ways to confirm the access purpose [7]. First, the users can be required to explicitly declare their access purposes along with the requests. Obviously, this method is the easiest to implement. Most privacy preserving access control models are based on it. Nevertheless, it demands the complete trust of the subjects, which is not suitable at all for an open environment. The second possible method is that the system registers a special access purpose for each application or stored procedure in advance. It may not be used in complex applications or stored-procedures scenarios in which subjects may access different objects for multiple access purposes. The third is that access purposes can be dynamically determined based on the current context.

Considering that not all the users in the cloud are absolutely trustworthy, in this paper, we use both the first and third methods to identify the access purpose. Those access purposes that are declared by users are named Static Access Purposes (AP_{st}). Similarly, those purposes that are dynamically determined are named Dynamic Access Purposes (AP_{dy}). One thing that should be noted is that only the AP_{st}s are dispensable.

Therefore, two new relationships and the Negotiation module are defined as follows.

- The *Speculate* Relationship

  This relationship is used to generate dynamic access purposes. *Speculate (org, c, r, AP_{dy})* means that if *org* is an organization, *c* is the current context, and *r* is a role, the dynamic access purpose AP_{dy} is true.

  Suppose that an employee of a delivery company is asking for access to a customer's address using a specific application during normal business time. We could

speculate that the AP_{dy} of this employee is shipping in such a situation.

- The *Purpose_belong* Relationship
  Purpose_belong(org, AP_{st}, AP_{dy}) $\rightarrow$ {True, False} is used to determine the affiliation between AP_{st} and AP_{dy}. If the user does not declare his own AP_{st}, we consider that *Purpose_belong* is always True. Under this circumstance, AP is AP_{dy}. Considering that this may lead to a situation in which some malicious users could intentionally hide their real access purposes, we introduce the Negotiation module as a reward mechanism. Otherwise, *Purpose_belong (org, AP_{st}, AP_{dy})* = True iff $AP_{st} \in AP_{dy}^{\blacktriangledown}$. Meanwhile, AP is AP_{st}.

- Negotiation module
  This module is activated only when the user declares his AP_{st} and $AP_{st} \notin AP_{dy}^{\blacktriangledown}$. In this case, the data request is not immediately terminated. Instead, the user can get a second chance to modify his AP_{st} or the Context, which means another opportunity to access the data item that he is requesting.

Example 1. Suppose AP_{dy} = "Third-Party" is defined over the purpose tree given in Figure 3.

Therefore, $AP_{dy}^{\blacktriangledown}$= {Third-Party, T-Email, T-Postal}.

1) If the user does not declare his AP_{st}, then his AP is "Third-Party" by default.
2) If AP_{st} = "Direct" and $AP_{st} \notin AP_{dy}^{\blacktriangledown}$, the Negotiation module is activated, and the user will get a second chance to modify his AP_{st} or the Context.
3) If AP_{st} = "T-Email", $AP_{st} \in AP_{dy}^{\blacktriangledown}$, then *Purpose_belong (org, AP_{st}, AP_{dy})* = True, the AP of the user would be "T-Email".

    b) Intended Purpose: Before migrating data to the cloud, a label named "intended purpose" is set for each item based on the data owner's privacy preferences. As described above, an intended purpose (IP) is a tuple $\langle$AIP, PIP$\rangle$.

Example 2. Suppose IP = ({Admin, D-Email}, {Third-Party}) is defined over the purpose tree that is given in Figure 3. Thus,

$AIP^{\blacktriangledown}$ = (Admin)$^{\blacktriangledown}$ $\cup$ (D-Email)$^{\blacktriangledown}$ = {Admin, Profiling, Analysis, D-Email, Special-Offers, Service-Updates}

$PIP^{\blacklozenge}$ = (Third-Party)$^{\blacklozenge}$ = {Third-Party, Marketing, T-Email, T-Postal, General-Purpose}

    c) Authorization: The relationship of *Purpose_compare (org, AP, IP)* $\rightarrow$ {True, False} is defined to determine the compliance between the user's AP and the object's IP. The access request could be allowed if and only if the AP satisfies the pre-set rules of the intended purpose. That is, AP $\in AIP^{\blacktriangledown}$ $\wedge$ AP $\notin PIP^{\blacklozenge}$.

Example 3. Suppose $AIP^{\blacktriangledown}$ and $PIP^{\blacklozenge}$ are discussed above. Then, the access purposes that meet the authorization conditions are {Admin, Profiling, Analysis, Special-Offers, Service-Updates}.

C. Security Policy

We can now give the security policies that apply to such an organization by adding our new entity AP to the access policy. The relationship *Permission(org, r, v, a, AP)* means that organization *org* grants role *r* permission to execute activity *a* on view *v* based on the access purpose *AP*.

D. Concrete authorization

In Pa-OrLaBAC, the procedure through which a subject can obtain permission to perform on the object is as follows:

    Employ (org, s, r) $\wedge$ *Use (org, o, v)* $\wedge$ *Consider (org, α, a)* $\wedge$ *Speculate (org, c, r, AP_{dy})* $\wedge$ *Permission (org, r, v, a, AP)* $\wedge$ *Purpose_belong (org, AP_{st}, AP_{dy})* $\wedge$ *Purpose_compare (org, AP, IP)* $\rightarrow$ *Is_permitted (s, o, α)*.

This means that if *org* employs subject *s* in role *r*, if *org* uses object *o* in view *v*, if *org* considers that action α falls within activity *a*, if organization *org* within the current context *c* speculates the dynamic access purpose of role r is AP_{dy}, if organization *org* grants role *r* permission to perform activity *a* on view *v* for access purpose AP, if *Purpose_belong* is true, and if *Purpose_compare* is true, then *s* has permission to perform α on *o*.

V. A Case Study

Medical informatization has become an inevitable trend of modern medical care. Electronic medical record (EMR), as the main carrier of medical information, plays an important role in modern medical treatment. The EMR itself contains a large amount of private information of the original owner, such as name, date of birth, home address, and sensitive information that is unwilling to be known to the outside world, such as marital status and disease information. The leakage and illegal use of this information may cause irreparable losses. However, patients have limited control over their medical data, which may lead to the disclosure of privacy information when EMR is consulted.

The proposed method can help solve the above problem.

An Application Scenario: Hospital *hosA* uses this method to manage its EMRs. John is treated in *hosA* whose attending internist is Tim. Figure 5 shows the purpose tree of *hosA*.
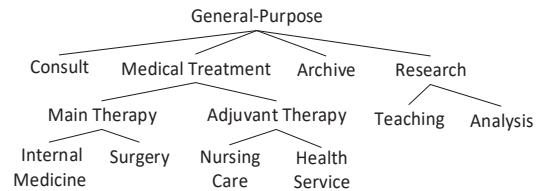
Fig. 5. The Purpose Tree of *hosA*.

Before submitting his EMR, John wanted to protect his "personal information" as much as possible, so he set the "intended purpose" label of it as ({Main Therapy, Archive}, {Research}). That is to say, for John's personal information,

$AIP^{\blacktriangledown}$ = (Main Therapy)$^{\blacktriangledown}$ $\cup$ (Archive)$^{\blacktriangledown}$ = {Main Therapy, Internal Medicine, Surgery, Archive}.

$PIP^{\blacklozenge}$ = (Research)$^{\blacklozenge}$ = {Research, Teaching, Analysis, General-Purpose}.

Tim is doing some researches at home. He requests access to John's EMR and declares his AP_{st} as "Medical Treatment".

Meanwhile, *hosA* determines Tim's AP_{dy} based on contextual information that whether John is receiving treatment right now or not, Tim's geographic location, etc.

Assuming that the inferred AP_{dy} of Tim is "Teaching". Obviously, $AP_{st} \notin AP_{dy}^{\blacktriangledown}$. Under this circumstance, the negotiation module is activated. Tim could have a second chance to modify his AP_{st}. If the condition is still not satisfied, access will be denied. In the meantime, John's "personal information" could not be accessed by Tim. In other words, John's "personal information" was protected as he wished.

The proposed method can also help us protect our privacy information from being leaked in other applications, e.g., banks and logistics. Depending on the actual requirement of different usage scenarios, this access control model can be adjusted dynamically by modifying the labels assigned to the data which could be validity, security level, risk value, etc.

VI. Conclusions And Future Work

For now, the access control models that are used by most Cloud Service Providers are based on RBAC. As an extension of RBAC, OrBAC overcomes the drawback that permissions are directly bound to roles. However, lacking flexibility makes OrBAC unsuitable for the cloud. In this paper, we presented Pa-OrLaBAC, which integrates OrBAC with LaBAC and introduces the concept of "purpose" as an effective means of privacy protection.

Before data are migrated to the cloud, a label named "intended purpose" is set for each data item based on the data owner's privacy preferences. Only when the access purpose of the subject is fully compliant with the intended purpose will the request be allowed. As for access purpose, two approaches and the Negotiation Module were applied to determine the most reliable one.

Compared with the traditional access control models, Pa-OrLaBAC alleviates the shortcomings of data control and ownership separation while ensuring flexibility by introducing purpose. Thus, it could be an effective method to protect resources in cloud environments. However, obviously, the Pa-OrLaBAC model still needs to be improved. In the future, we plan to formally analyse the properties of the proposed model compared to those of existing access control models, develop an XACML profile of the proposed model and enable this model to achieve dynamic access control.

VII. Acknowledgment

References

[1] P. Mell, T. Grance *et al.*, "The nist definition of cloud computing," 2011.

[2] M. A. AlZain, E. Pardede, B. Soh, and J. A. Thom, "Cloud computing security: from single to multi-clouds," in *2012 45th Hawaii International Conference on System Sciences*. IEEE, 2012, pp. 5490–5499.

[3] R. S. Sandhu and P. Samarati, "Access control: principle and practice," *IEEE communications magazine*, vol. 32, no. 9, pp. 40–48, 1994.

[4] R. Sandhu, "Engineering authority and trust in cyberspace: The om-am and rbac way," in *Proceedings of the fifth ACM workshop on Role-based access control*. ACM, 2000, pp. 111–119.

[5] R. Agrawal, J. Kiernan, R. Srikant, and Y. Xu, "Hippocratic databases," in *VLDB'02: Proceedings of the 28th International Conference on Very Large Databases*. Elsevier, 2002, pp. 143–154.

[6] OECD, "Oecd guidelines on the protection of privacy and transborder flows of personal data," 1980.

[7] J.-W. Byun and N. Li, "Purpose based access control for privacy protection in relational database systems," *The VLDB JournalThe International Journal on Very Large Data Bases*, vol. 17, no. 4, pp. 603–619, 2008.

[8] S. Fischer-Hübner, *IT-security and privacy: design and use of privacy-enhancing security mechanisms*. Springer-Verlag, 2001.

[9] R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman, "Role-based access control models," *Computer*, vol. 29, no. 2, pp. 38–47, 1996.

[10] D. E. Bell and L. J. LaPadula, "Secure computer systems: Mathematical foundations," MITRE CORP BEDFORD MA, Tech. Rep., 1973.

[11] J. Lopez and J. E. Rubio, "Access control for cyber-physical systems interconnected to the cloud," *Computer Networks*, vol. 134, pp. 46–54, 2018.

[12] V. C. Hu, D. Ferraiolo, R. Kuhn, A. R. Friedman, A. J. Lang, M. M. Cogdell, A. Schnitzer, K. Sandlin, R. Miller, K. Scarfone *et al.*, "Guide to attribute based access control (abac) definition and considerations (draft)," *NIST special publication*, vol. 800, no. 162, 2013.

[13] A. A. E. Kalam, R. E. Baida, P. Balbiani, S. Benferhat, F. Cuppens, Y. Deswarte, A. Miege, C. Saurel, and G. Trouessin, "Organization based access control," in *Proceedings POLICY 2003. IEEE 4th International Workshop on Policies for Distributed Systems and Networks*. IEEE, 2003, pp. 120–131.

[14] P. Biswas, R. Sandhu, and R. Krishnan, "Label-based access control: An abac model with enumerated authorization policy," in *Proceedings of the 2016 ACM International Workshop on Attribute Based Access Control*. ACM, 2016, pp. 1–12.

[15] M. B. Saidi, A. A. Elkalam, and A. Marzouk, "Torbac: A trust organization based access control model for cloud computing systems," *Int J Soft Comput Eng*, vol. 2, no. 4, pp. 122–130, 2012.

[16] N. A. Aali, A. Baina, and L. Echabbi, "Tr-orbac: A trust model for collaborative systems within critical infrastructures," in *2015 5th World Congress on Information and Communication Technologies (WICT)*. IEEE, 2015, pp. 123–128.

[17] M. A. Madani and M. Erradi, "How to secure a collaborative session in a single tenant environment," in *2015 International Conference on Protocol Engineering (ICPE) and International Conference on New Technologies of Distributed Systems (NTDS)*. IEEE, 2015, pp. 1–6.

[18] R. E. Sanders, "Securing data with label-based access control," http://www.tridug.org/wp-content/uploads/2012/05/Understanding LBAC.pdf, 2012.

[19] J. Leffler, "Label-based access control with ids cheetah," http://www.iiug.org/webcasts/replay/30may07.pdf, 2007.

[20] H. Chen, B. Bhargava, and F. Zhongchuan, "Multilabels-based scalable access control for big data applications," *IEEE Cloud Computing*, vol. 1, no. 3, pp. 65–71, 2014.

[21] P. Chinnasamy and P. Deepalakshmi, "A scalable multilabel-based access control as a service for the cloud (smbacaas)," *Transactions on Emerging Telecommunications Technologies*, vol. 29, no. 8, p. e3458, 2018.

[22] A. Li, Q. Li, and V. Hu, "Access control for distributed processing systems: Use cases and general considerations," in *2017 IEEE 3rd International Conference on Collaboration and Internet Computing (CIC)*. IEEE, 2017, pp. 117–125.

[23] Q. Ni, E. Bertino, J. Lobo, C. Brodie, C.-M. Karat, J. Karat, and A. Trombeta, "Privacy-aware role-based access control," *ACM Transactions on Information and System Security (TISSEC)*, vol. 13, no. 3, p. 24, 2010.

[24] E. Bertino, J.-W. Byun, and N. Li, "Privacy-preserving database systems," in *Foundations of Security Analysis and Design III*. Springer, 2005, pp. 178–206.

Formal Modelling and Verification of MCAC Router Architecture in ICN

Junya Xu[1], Huibiao Zhu[*1], Lili Xiao[1], Jiaqi Yin[1], Yuan Fei[*2], Gang Lu[1]

[1]Shanghai Key Laboratory of Trustworthy Computing,
East China Normal University, Shanghai, China
[2] School of Information, Mechanical and Electrical Engineering,
Shanghai Normal University, Shanghai, China

Abstract—As Information Center Network (ICN) becomes a candidate for the future Internet architecture, its security and privacy issues have aroused extensive attention. Mandatory Content Access Control (MCAC) router architecture, which extends the existing mainstream ones with hardware-rooted trust, is proposed to implement MCAC protocol to protect privacy. Therefore, it is necessary to study the security of this architecture from the perspective of formal methods.

In this paper, we use the process algebra Communicating Sequential Processes (CSP) to model and analyze MCAC router architecture in ICN. By adopting model checking tool Process Analysis Toolkit (PAT), we verify five important properties, namely *Deadlock Freedom, Key Faking, Level Mechanism, Data Availability* and *Data Leaking*. The results of verification show the correctness and security of MCAC router architecture, from which it can be concluded that this architecture is reliable.

Index Terms—MCAC Router Architecture, ICN, CSP, Modeling, Verification

I. Introduction

Information Centric Network (ICN) has been proposed as a candidate for future Internet architecture [?]. It breaks the host-centric pattern, replacing the traditional approach with end-to-end connectivity and unique named data based on content distribution architecture. Due to the fact that named data is isolated from physical locations in the network, caching and replication of data in ICN can more easily support network storage and forwarding [?], [?]. Since ICN is attracting more and more attention, security and privacy have also become the important concern [?]. Some security and privacy issues have been identified with current ICN architectures. For instance, as one of the ICN architectures, Content-Centric Network (CCN) [?] has been found that it still leads to data leaks, since it allows that routers in the forwarding path can cache contents.

The traditional Mandatory Access Control (MAC) [?] provides confidentiality for the network by security level labels mechanism. MAC is usually valid in a system, but not common in large-scale distributed systems. Thus, it is difficult to be applied in ICN. In order to address the issue, Li et al. [?] have proposed Mandatory Content Access Control (MCAC) to provide security protection for each component in ICN. Similar to MAC, MCAC also provides security level mechanism for different components to realize the security and privacy

for data in ICN. Subjects including processes in routers and objects such as contents and contest requests in MCAC, are respectively labeled into four levels $\{h,\ n,\ d,\ p\}$, where the relationship of label's level is defined: $h > n > d > p$. In other words, subjects in MACA can only read the objects with the labels equal or lower than themselves.

Since MCAC policies are implemented by content routers in ICN, Li et al. [?] proposed a design of MCAC router architecture. This router architecture is based on the existing router architecture and hardware-rooted trust, with the addition of an authentication protocol. In previous work, A. Datta et al. [?] have only verified the authentication protocol by using cord calculus, but this architecture has not yet been modeled and verified by using formal methods. In this paper, we propose a formal verification of the security-related properties of this architecture. First, we use CSP [?], [?] to model MCAC router architecture in ICN. Then, we choose PAT [?] to verify some properties of our system. The results of verification show that the security of MCAC router architecture is still guaranteed despite the presence of the intruder.

This paper is organized as follows. Section II gives a brief introduction to MCAC router architecture and CSP. In section III, we model the modules of one router in MCAC router architecture in CSP. In section IV, we use PAT to implement the model and verify five properties. Finally, we conclude this paper and make a discussion on the future work in section V.

II. Background

In this section, we give an overview of MCAC router architecture and a brief introduction to CSP.

A. MCAC Router Architecture

To implement MCAC policies, routers need to make forwarding decisions based on each content request and have the function of storage to implement content caching. It mainly consists of the following modules.

- *Trusted Storage Module* (TSM) is responsible for negotiating secret keys with TSM on neighbor routers and writing the secret keys into module TEM. In addition, TSM also handles caching contents.
- *Trusted Labeling Module* (TLM) checks whether each read operation and cache operation are legal by reading the labels in content packets.

- *Trusted Enforcement Module* (TEM) provides private protection for some contents by encrypting the contents. In addition, it also reclassifies the content labels according to the reclassification rules.

The procedures of router communication based on MCAC router architecture can be mainly divided into two stages: key negotiation (Fig. 1) and transmission of content packets (Fig. 2).
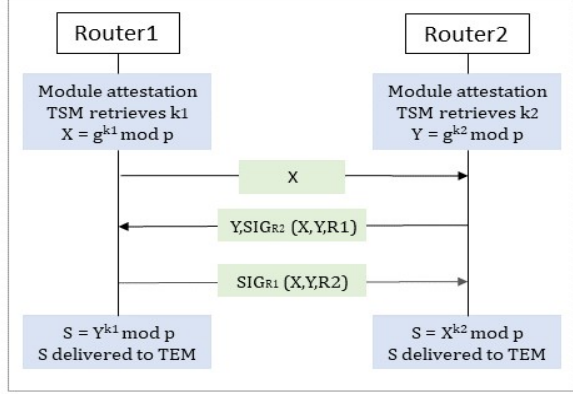
Fig. 1. Process of Key Negotiation

As shown in Fig. 1, identity authentication is added to M-CAC router architecture during the process of key negotiation. In the beginning, all the modules should be verified for integrity. Then TSM retrieves the private key k_i. R1 as an initiator sends the Diffie-Hellman exponent composed of private key to R2. After receiving the message, the responder R2 sends its own Diffie-Hellman exponent and signature including mutual exponent and R1's ID to R1. Then R1 replies a message with signature to complete the authentication. R1 and R2 can get the same key S by calculating the Diffie-Hellman exponents. After generating their session keys, routers deliver these keys to the corresponding TEM modules.
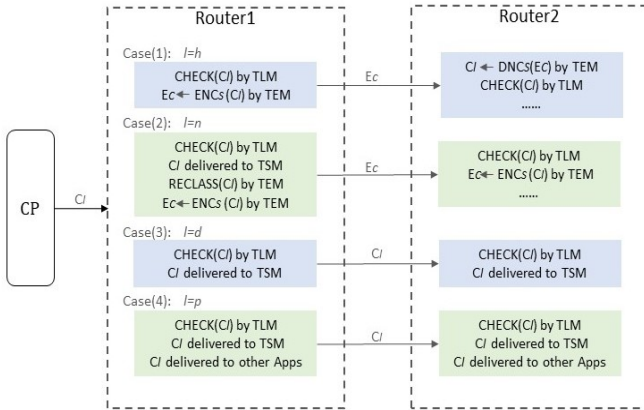
Fig. 2. Transmission of Content Packets

ContentProvider (CP) categorizes contents into four tag levels before sending them. Thus each router in MCAC router architecture needs to handle these contents in four situations as shown in Fig. 2.

- In case (1), CP generates the content with label h and sends the content to R1. After receiving the content, TLM

in R1 needs to check the label embedded in the content at first. Because the label is h, the content is not sent to TSM but directly forwarded to R2 after TEM encrypts it using the key S. When R2 receives the content, it firstly decrypts the content and repeats the process like R1.
- In case (2), the content labeled with n is provided by CP for R1. TLM checks the label and sends the content to TSM since the label is lower than h. Then, TEM reclassifies the content by changing the content label from n to h and encrypts the content before delivering to R2. That is to say, R2 receives the content with label h and performs actions like R2 in case (1).
- In case (3), CP generates the content with label d and sends it to R1. After checked the label by TLM, the content with label d means that the content is sent to TSM and directly forwarded to R2 without encryption. R2 performs the same actions in the process of the communication.
- Case (4) is similar to case (3), the only difference is that the content with label p can also be delivered to all the applications.

B. CSP

CSP is a process algebra proposed by C. A. R. Hoare. The language is mainly designed to describe and analyze the behavior of concurrent systems and processes, which has been successfully applied in modeling and verifying various concurrent systems and protocols [**?**], [**?**].

We give the syntax of the CSP language used to describe the process in this paper, where P and Q are processes, a denotes the event and c represents the name of channel.

$$P, Q = Skip \mid Stop \mid a \rightarrow P \mid c?x \rightarrow P \mid c!x \rightarrow P \mid$$
$$P \Box Q \mid P \parallel Q \mid P \lhd b \rhd Q \mid P; Q \mid P[|X|]Q$$

- $Skip$ represents the process which does nothing but terminates successfully.
- $Stop$ denotes that the process does nothing and it is in the state of deadlock.
- $a \rightarrow P$ describes an object which first performs the event a and then behaves like P.
- $c?x \rightarrow P$ receives a message through channel c and stores the value in variable x and then the behavior is like process P.
- $c!x \rightarrow P$ sends message x through channel c and then behaves like process P.
- $P \Box Q$ stands for the choice between process P and process Q. The election is decided by the environment.
- $P \parallel Q$ denotes that processes P and Q execute concurrently and are synchronized with the same communication events.
- $P \lhd b \rhd Q$ indicates if condition b is true, the process behaves like P, otherwise like Q.
- $P; Q$ describes that processes P and Q execute in sequence.
- $P[|X|]Q$ denotes that the parallel composition of P and Q performs the concurrent events on set X of channels.

III. MODELING

In this section, we formalize the model of MCAC router architecture in Information Centric Networks.

A. Sets, Messages and Channels

In order to model the communication of routers and the behaviors of the modules in a router, we give the definitions of sets, messages and channels we use in this paper.

First, we introduce some sets we use in the model. **Modules** set is composed of modules in the router including TSM, TEM and TLM. **Nonce** set represents Diffie-Hellman exponents. **Key** set consists of keys. **Content** set includes the contents transmitted between the modules or routers. **Label** set defines level of the content. **ID** set is the identity information of routers, and **Ack** set contains acknowledgements.

Then, we define an encryption function E and a decryption function D:

$$E(k, msg); \ D(k, emsg)$$

where the function E makes use of a key to encrypt the message msg while the function D uses a key to decrypt the encrypted message $emsg$. Therefore, we can get the following conclusion:

$$D(k, E(k, msg)) = msg$$

Based on the above sets and functions, we describe the following messages:

$$MSG_{datE} = \{msg_e.g, msg_e.c, msg_e.E(k,c), msg_e.E(k,g,r) \\ | \ g \in Nonce, \ k \in Key, r \in ID, c \in Content\}$$

$$MSG_{datM} = \{msg_m.m.n.k, \ msg_m.m.n.c, \ msg_m.m.n.l.c \ | \\ m, n \in Module, c \in Content, l \in Label\}$$

$$MSG_{ack} = \{msg_{ack}.x \ | \ x \in Ack\}$$

$$MSG_{pro} = \{msg_{pro}.E(k_1, g, r_1).k.r, \ msg_{pro}.E(k_1, c_1).k \ | \\ g \in Nonce, k \in Key, r \in Name, c \in Content\}$$

$$MSG_{in} = MSG_{ack} \cup MSG_{pro}$$

$$MSG = MSG_{datE} \cup MSG_{datM} \cup MSG_{in}$$

Here, MSG_{datE} represents messages transmitted between the modules in adjacent routers. MSG_{datM} consists of messages sent between the modules in the same router. MSG_{pro} denotes messages delivered to a processing process and MSG_{ack} represents the set of feedback information from the processing process.

Next, we give the definitions of channels in this paper.

- channels of honest routers, using COMR_PATH to represent:
  $$ComTEM, \ ComTSM,$$
- channels of honest modules in the same router, using COMM_PATH to represent:
  $$KeySet, \ ContentProcess, \ ContentCache$$
- channels of intruders who perform intercepting or faking behaviors, defined by INTR_PATH:
  $$FakeTEM, \ FakeTSM$$

- channels of processing messages and feedback messages, represented by PROC_PATH:
  $$CheckKey, \ GetData$$
- In addition, channels of normal communications, represented by COM_PATH:
  $$COM\_PATH = COMR\_PATH \cup COMM\_PATH$$

The declarations of channels are as follows:

Channel $COM\_PATH, INTR\_PATH$:
$$MSG_{datE} \cup MSG_{datM}$$
Channel $PROC\_PATH$: MSG_{in}

B. Overall Modeling

In this paper, we focus on the messages transmission of the modules inside a router and the external information transmission with the modules on neighbor routers. Meanwhile, we also consider the presence of intruders in normal communications. Note that in this paper, we allow intruders to eavesdrop on or intercept communication messages between routers, but these intruders are unable to obtain communication message between modules within the same router. Fig. 3 shows the communication between routers without intruders and Fig. 4 shows the communication with an intruder.
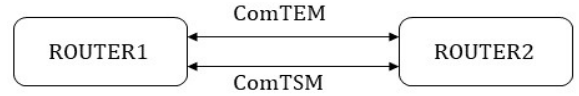
Fig. 3. Communication between Routers without Intruders
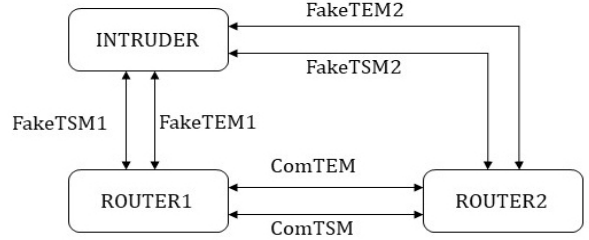
Fig. 4. Communication between Routers with an Intruder
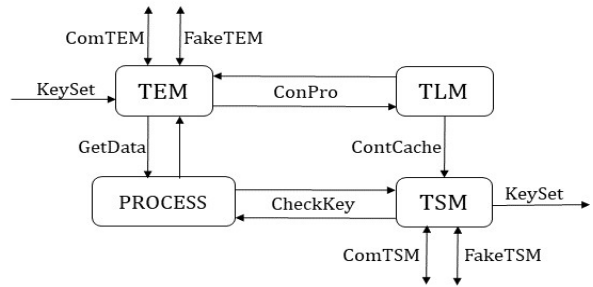
Fig. 5. Communication between Modules in a Router

Since the structure and communication function of each router are the same in MCAC router architecture, now we present the model of one router. Therefore as described in Fig. 5, any router can be abstracted as a system consisting of three modules including TSM, TEM, TLM and a process for internal information processing. That is to say, we do not consider other internal components of the router in this router architecture in this paper.

In order to take full account of the system security issues, we also define the process $INTRUDER$ to simulate intruders who eavesdrop and tamper messages.

Based on the above descriptions, we formalize the whole models $System$ and $System_I$ as follows.

$$System() =_{df} PROCESS() \ [|PROC\_PATH|] \ TEM_0()$$
$$[|COM\_PATH|] \ TSM_0() \ [|COM\_PATH|] \ TLM_0()$$
$$System_I() =_{df} System() \ [|INTR\_PATH|] \ INTRUDER$$

Process $System$ is composed of processes TSM_0, TLM_0, TEM_0 and $PROCESS$, which perform the concurrent events on the sets $PROC\_PATH$ and COM_PATH of channels. Process $System_I$ consists of processes $System$ and $INTRUDER$, which perform the concurrent events on the set $INTR\_PATH$ of channels.

C. TSM Modeling

First, we formalize process TSM_0 to describe the behaviors of Trusted Storage Module in a router.

$$TSM_0() =_{df} Init\{k = false, m = true\} \rightarrow$$
$$ComTSM!msg_e.g^{k_1} \rightarrow$$
$$ComTSM?msg_e.g^{k_2}.E(k_{RB}^{-1}, (g^{k_1}, g^{k_2}, RA)) \rightarrow$$
$$ComTSM!msg_e.E(k_{RA}^{-1}, (g^{k_1}, g^{k_2}, RB))\{k_{SA} := (g^{k_2})^{k_1}\}$$
$$\rightarrow KeySet!msg_m.s.e.k_{SA} \rightarrow$$
$$CheckKey!msg_{pro}.E(k_{RB}^{-1}, (g^{k_1}, g^{k_2}, RA)).k_{RB} \rightarrow$$
$$CheckKey?msg_{ack}.ack \rightarrow$$
$$\begin{pmatrix} (KeyFakingSuccess\{k = true\} \rightarrow Skip) \\ \lhd ack == YES \rhd \\ (KeyFakingError\{k = false\} \rightarrow Skip) \end{pmatrix};$$
$$ContCache?msg_m.l.s.c\{l := GetLabel()\} \rightarrow$$
$$\begin{pmatrix} (LevelMechanismErr\{m = false\} \rightarrow Skip) \\ \lhd l == h \rhd \\ (LevelMechanismCor\{m = true\} \rightarrow Skip) \end{pmatrix}$$
$$; \ TSM_0()$$

The initial state of TSM_0 is defined as $m = true$ and $k = false$, where m indicates whether the level mechanism is safe and k expresses whether key is fake. The following actions correspond to two parts including key negotiation and content caching. By channel CheckKey, we check if the session key is faked and GetLabel() is used to get the level labels of contents.

We need to take the possibility of intruder actions into consideration, therefore, the messages on channel $ComTSM$ can be faked or intercepted. We apply the renaming to process TSM_0. TSM_0 performs an action only on the channel $ComTSM$, but TSM can perform an action either on channel $ComTSM$ or on channel $FakeTSM$.

$$TSM() =_{df} TSM_0()[[$$
$$ComTSM?\{|ComTSM|\} \leftarrow ComTSM?\{|ComTSM|\},$$
$$ComTSM?\{|ComTSM|\} \leftarrow FakeTSM?\{|ComTSM|\},$$
$$ComTSM!\{|ComTSM|\} \leftarrow ComTSM!\{|ComTSM|\},$$
$$ComTSM!\{|ComTSM|\} \leftarrow FakeTSM!\{|ComTSM|\}]]$$

D. TLM Modeling

TLM is responsible for identifying the labels in different content packets. We formalize process TLM_0 to describe the behavior of Trusted Labeling Module in a router.

$$TLM_0() =_{df} ConPro?msg_m.e.l.c \ \{l := GetLabel()\};$$
$$\begin{pmatrix} ConPro!msg_m.l.e.h.c \\ \lhd l == h \rhd \\ \begin{pmatrix} ContCache!msg_m.l.s.c \rightarrow \\ \begin{pmatrix} ConPro!msg_m.l.s.n.c \\ \lhd l == n \rhd \\ ConPro!msg_m.l.s.c \end{pmatrix} \end{pmatrix} \end{pmatrix}$$
$$; TLM_0()$$

After receiving a content packet, TLM_0 makes a judgement about the label of the content packet and performs the corresponding actions. Here, GetLabel() is used to get the level labels of contents.

E. TEM Modeling

The process TEM_0 describes the behavior of Trusted Enforcement Module in a router which is formalized as follow.

$$TEM_0() =_{df} Init\{a = false\} \rightarrow KeySet?msg_m.s.e.k_{SA} \rightarrow$$
$$\begin{pmatrix} (ComTEM?msg_e.E(k_{SB}, c) \\ \{c := D(k_{SA}, E(k_{SB}, c))\} \rightarrow ConPro!msg_m.e.l.c) \\ \Box(ComTEM?msg_e.c \rightarrow ConPro!msg_m.e.l.c) \end{pmatrix};$$
$$GetData!msg_{pro}.E(k_{SB}, c) \rightarrow GetData?msg_{ack}.ack1 \rightarrow$$
$$\begin{pmatrix} (DataAcqcquisitionSuccess\{a = true\} \rightarrow Skip) \\ \lhd ack1 == YES \rhd \\ (DataAcqcquisitionError\{a = false\} \rightarrow Skip) \end{pmatrix};$$
$$\begin{pmatrix} (ConPro?msg_m.l.e.h.c \rightarrow \\ ComTEM!msg_e.E(k_{SA}, c)) \\ \Box(ConPro?msg_m.l.e.n.c \ \{n := h\} \rightarrow \\ ComTEM!msg_e.E(k_{SA}, c)) \\ \Box(ConPro?msg_m.l.e.c \rightarrow ComTEM! \ msg_e.c) \end{pmatrix}$$
$$; \ TEM_0()$$

We define the initial state of TEM_0 as $a = false$, where boolean variable a expresses data availability. At first, TEM_0 receives a message including the negotiated secret key k_S from TSM_0. Then if TEM_0 receives the encrypted message from its neighbor router, it makes use of the negotiated secret key to decrypt the message and sends the content in the message to TLM_0. If the content is not encrypted, it forwards the content directly to TLM_0. TEM_0 can also check whether the modules can get what it expects by channel $GetData$. In addition, TEM_0 needs to complete message encryption and reclassification functions.

Similarly for TEM, we do this renaming to process TEM_0.

$$TEM() =_{df} TEM_0()[[$$
$$ComTEM?\{|ComTEM|\} \leftarrow ComTEM?\{|ComTEM|\},$$
$$ComTEM?\{|ComTEM|\} \leftarrow FakeTEM?\{|ComTEM|\},$$
$$ComTEM!\{|ComTEM|\} \leftarrow ComTEM!\{|ComTEM|\},$$
$$ComTEM!\{|ComTEM|\} \leftarrow FakeTEM!\{|ComTEM|\}]]$$

F. PROCESS Modeling

Ultimately, we give $PROCESS$ to simulate the internal information processing procedure in a module.

$PROCESS() =_{df}$

$CheckKey?msg_{pro}.E((k_{R1}^{-1}), g^{k_1}, g^{k_2}, R1).k_{R2}$

$$\begin{pmatrix} (CheckKey!msg_{ack}.YES \rightarrow PROCESS()) \\ \lhd k_{R1} == k_{R2} \ \&\& \ g^{k_2} == g^{k_2}\_f \rhd \\ (CheckKey!msg_{ack}.NO \rightarrow PROCESS()) \end{pmatrix}$$

$\square GetData?msg_{pro}.E(k_{SB}, c)$

$$\begin{pmatrix} (GetData!msg_{ack}.YES \rightarrow PROCESS()) \\ \lhd k_{SB} == k_{SA} \ \| \ k_{SB} == k_{S\_f} \rhd \\ (GetData!msg_{ack}.NO \rightarrow PROCESS()) \end{pmatrix}$$

$PROCESS$ is used to deal with whether the negotiated session key is faked and whether the content can be transmitted to the module requiring for it. Then it sends feedback messages to TSM_0 and TEM_0 separately.

G. INTRUDER Modeling

We also regard $INTRUDER$ as a process and it can intercept messages transmitted on $ComTSM$ and $ComTEM$ or fake messages on $FakeTSM$ and $FakeTEM$ at any time.

First, we define set FACT which contains all the facts that can be learned by the intruder.

$$FACT =_{df} TSMs \cup TLMs \cup TEMs \cup MSG_{datE}$$
$$\cup \{K, K^{-1}, K_S, K_{S\_f}\} \cup \{N, Name\}$$
$$\cup \{E(key, content) \mid key \in \{K^{-1}, K_S, K_{S\_f}\},$$
$$content \in \{N, Name, Content\}\}$$

Next, we define the rules to express how the intruder can deduce new facts from what it has known, shown as follows:

$$\{K, E(K^{-1}, c)\} \mapsto c, \qquad \{K_S, E(K_S, c)\} \mapsto c,$$
$$\{K_{S\_f}, E(K_{S\_f}, c)\} \mapsto c, \quad \{K, c\} \mapsto E(K, c),$$
$$\{K_S, c\} \mapsto E(K_S, c), \qquad \{K_{S\_f}, c\} \mapsto E(K_{S\_f}, c),$$
$$F \mapsto f \wedge F \subseteq F' \Rightarrow F' \mapsto f$$

where, set F denotes the facts the intruder has known, and f is the fact deduced from set F. $F \mapsto f$ represents that fact f can be deduced from the set F.

The first three rules describe that the intruder can use the corresponding key to decrypt the encrypted messages and get some contents. In the same way, the next three rules represent encryption. The ?nal rule is a structural rule, explaining that the intruder can deduce fact f from a lager set F', if f can be deduced from set F.

In addition, we define the $Info$ function to represent the facts which a intruder can learn from the intercepted and eavesdropped messages:

$$Info(msg_e.g) =_{df} \{g\} \quad Info(msg_e.c) =_{df} \{c\}$$
$$Info(msg_e.E(k, g, r)) =_{df} \{E(k, g, r)\}$$

$$Info(msg_e.E(k, c)) =_{df} \{E(k, c)\}$$
$$Info(msg_m.m.n.k) =_{df} \{m, n, k\}$$
$$Info(msg_m.m.n.c) =_{df} \{m, n, c\}$$
$$Info(msg_m.m.n.l.c) =_{df} \{m, n, l, c\}$$

where $g \in Nonce$, $m, n \in Module$, $k \in Key$, $r \in ID$, $l \in Label$, $c \in Content$.

Finally, we declare a channel $Deduce$ used for deducing new facts:

$$\text{Channel } Deduce : \ Fact.P(Fact)$$

We allow that the intruder can overhear all the messages transmitted between routers and learn all the facts from the messages, but it cannot intercept the messages transmitted between the modules in the same router. Meanwhile, the intruder can fake a message if it has learned some facts and deduce a new fact from known ones. Beyond that it can also use a key that it knows in fake sessions.

Based on the above, now we give the formalization of $INTRUDER$ as below:

$INTRUDER(F) =_{df}$

$\square_{m \in MSG_E} FakeTSM?m \rightarrow INTRUDER(F \cup Info(m))$

$\square\square_{m \in MSG_E \cap Info(m) \subset F} FakeTSM!m \rightarrow INTRUDER(F)$

$\square\square_{m \in MSG_E} FakeTEM?m \rightarrow INTRUDER(F \cup Info(m))$

$\square\square_{m \in MSG_E \cap Info(m) \subset F} FakeTEM!m \rightarrow INTRUDER(F)$

$\square\square_{f \in Fact, f \notin F, F \mapsto f} Init\{e = false\} \rightarrow Deduce.f.F \rightarrow$

$$\begin{pmatrix} (DaLeakSuc\{e = true\} \rightarrow INTRUDER(F \cup \{f\})) \\ \lhd (f == c \ \&\& \ f == h) \ \| \ (f == c \ \&\& \ f == n) \rhd \\ (DaLeakErr\{e = flase\} \rightarrow INTRUDER(F \cup \{f\})) \end{pmatrix}$$

When the intruder intercepts a message in MSG_E, it can deduce some information from this message, and it may also replace some contents and send a fake message to other honest entities.

IV. VERIFICATION

In this section, we use model checker PAT to implement the formal model which has been formalized in section III. At the same time, we carry out some security properties verification of our system.

A. Security Specification

We want to check whether the intruder can intercept or fake messages successfully in the whole system. Thus, we test if our system is against the following specifications:

$$SPEC_{TSM} =_{df} CHAOS \ (\Sigma - \{|FakeTSM|\})$$
$$SPEC_{TEM} =_{df} CHAOS \ (\Sigma - \{|FakeTEM|\})$$

CHAOS(A) [?] is the most uncertain and divergent process of alphabet A. It can perform any events from the alphabet A, where Σ is the set of all events. For instance, if the process TSM is allowed to perform any events except those occurring on the channel $FakeTSM$, we can say that it satisfies the specification $SPEC_{TSM} =_{df} CHAOS \ (\Sigma - \{|FakeTSM|\})$. If the system with the intruder refines these specifications, it is indeed secure.

B. Properties Verification

In this subsection, we verify five properties: Deadlock Freedom, Key Faking, Level Mechanism, Data Availability and Data Leaking. As formalised above, $System_I()$ is used to denote the model with an intruder and give the results of verification at the end.

Property 1: Deadlock Freedom

$$\#assert \ System_I() \ deadlockfree;$$

The *deadlock-free* property is a primitive in PAT which means a system can avoid the deadlock. This property verifies whether our system can run into the deadlock state.

Property 2: Key Faking

$$\#define \ Key\_Faking\_Success \ k == true;$$
$$\#assert \ System_I() \ reaches \ Key\_Faking\_Success;$$

We define *Key Faking* to denote the situation that the intruder can break the mutual authentication between routers and successfully tamper with the key.

Property 3: Level Mechanism

$$\#define \ Level\_Mechanism\_safe \ m == true;$$
$$\#assert \ System_I() \ reaches \ Level\_Mechanism\_safe;$$

The property *Level Mechanism* means that contents can only be processed and cached by higher-level processes in routers to provide security and privacy.

Property 4: Data Availability

$$\#define \ Data\_Acquisition\_Success \ a == true;$$
$$\#assert \ System_I() \ reaches \ Data\_Acquisition\_Success;$$

The property *Data Availability* represents that the contents can be obtained by the modules which require these contents.

Property 5: Data Leaking

$$\#define \ Data\_Leakaging\_Success \ e == true;$$
$$\#assert \ System_I() \ reaches \ Data\_Leakaging\_Success;$$

We say that a system satisfies this property, if an intruder can intercept and crack the encrypted messages transmitted between the honest entities. This property verifies whether the content is leaked in the process of message transmission.

Fig. 6. Verification Result of the Properties in $System_I$

As shown in Fig. 6, the properties *Deadlock Freedom*, *Level Mechanism* and *Data Availability* are all valid, which means

that the system cannot run into the deadlock state and the modules of routers can only process lower-level contents and obtain what they want respectively. Meanwhile, we also find the properties *Key Faking* and *Data Leaking* are invalid. These two properties ensure the correctness of key authentication and the security of data transmission in our model with the intruder respectively. Therefore, we can get a conclusion that data transmission in MCAC router architecture is safe.

V. CONCLUSION AND FUTURE WORK

This paper focuses on the security and correctness of MCAC router architecture through formal methods. Firstly, we have formalized three modules comprising TEM, TLM and TSM in MCAC router architecture with CSP. Then we verified five properties related to security through the model checker PAT including *Deadlock Freedom, Key Faking, Level Mechanism, Data Availability* and *Data Leaking*. Consequently, we can conclude that the correctness and security of MCAC router architecture are guaranteed from the results of verification.

In the future, we will follow with interest the other properties of MCAC router architecture. Meanwhile we will also explore the way to model and verify other access control solutions of ICN.

REFERENCES

[1] Van Jacobson, Diana K. Smetters, James D. Thornton, Michael F. Plass, Nick Briggs, Rebecca Braynard: Networking named content. Commun. ACM 55(1): 117-124 (2012)

[2] Dinh Nguyen, Kohei Sugiyama, Atsushi Tagami: Cache the Queues: Caching and Forwarding in ICN from a Congestion Control Perspective. ITC 2016: 243-251

[3] Gavin Lowe, A. W. Roscoe: Using CSP to Detect Errors in the TMN Protocol. IEEE Trans. Software Eng. 23(10): 659-669 (1997)

[4] Qi Li, Ravi Sandhu, Xinwen Zhang, Mingwei Xu: Mandatory Content Access Control for Privacy Protection in Information Centric Networks. IEEE Trans. Dependable Sec. Comput. 14(5): 494-506 (2017)

[5] PAT: Process Analysis Toolkit. http://pat.comp.nus.edu.sg/

[6] Anand Seetharam: On Caching and Routing in Information-Centric Networks. IEEE Communications Magazine 56(3): 204-209 (2018)

[7] Towards seamless mobility in ICN : connectivity, security, and reliability. (Vers une mobilit transparente dans le rseau ICN : connectivit, scurit, et fiabilit). Pierre and Marie Curie University, France, 2018

[8] Stephen D. Brookes, C. A. R. Hoare, A. W. Roscoe: A Theory of Communicating Sequential Processes. J. ACM 31(3): 560-599 (1984)

[9] Yuan Fei, Huibiao Zhu: Modeling and Verifying NDN Access Control Using CSP. ICFEM 2018: 143-159

[10] Hoare, C.A.R.: Communicating Sequential Processes. Prentice-Hall, Upper Saddle River (1985)

[11] Anupam Datta, Ante Derek, John C. Mitchell, Dusko Pavlovic: A derivation system and compositional logic for security protocols. Journal of Computer Security 13(3): 423-482 (2005)

Data-sparsity Service Discovery using Enriched Neural Topic Model and Attentional Bi-LSTM

Li Yao, Bing Li, Jian Wang
School of Computer Science,
Wuhan University,
Wuhan, China
e-mail: jianwang@whu.edu.cn

Abstract—In recent years, the amount of Web services has increased dramatically, and service discovery aiming to help users identify appropriate services matching their requirements thus becomes increasingly important. Many studies based on machine learning techniques have been reported to improve the performance of service discovery. A major obstacle in Web service discovery is the data sparsity in service descriptions. Towards this issue, in this paper, we propose a novel approach based on enriched neural topic model (NTM) and attentional Bi-LSTM. To alleviate the data sparsity issue, we enrich the semantics of each word in service descriptions and queries using external knowledge sources like Wikipedia and combine NTM and the attention mechanism to minimize the noise brought in the enrichment process. Experiments conducted on a real-world dataset show that our approach outperforms several state-of-the-art methods.

Keywords-Web service; service discovery; attention mechanism; neural topic model; Bi-directional LSTM

I. INTRODUCTION

Service-oriented architecture (SOA) facilitates a new paradigm for system development and integration, where system functionalities are encapsulated as loosely coupled and interoperable services. A growing number of Web services or cloud services have thus been created and published to meet the interoperability and flexibility requirements of modern software development in the cloud. The proliferation of Web services offers convenience for developers; however, it also brings difficulties in quickly selecting appropriate candidate services from large scale service registries.

In existing service registries, Web services are usually described in WSDL (Web Service Description Language) or simple natural language texts. Since keyword matching adopted in most service search engines may suffer from retrieving irrelevant services or missing relevant ones, many efforts have been made to address the service matching problem [1-5]. These approaches could be approximately classified into two types. The first group of approaches annotates services and queries using domain ontologies and leverage ontology reasoning for service matching. However, building such problem-specific ontologies and annotating services is time-consuming and sometimes impractical. The other group of approaches using machine learning or deep learning techniques for service matching. For example, topic models like Latent Dirichlet Allocation (LDA) [8] are introduced to obtain the topic

distribution vectors of services and queries and calculate their similarities. Topic models are also combined with word embeddings [11] to alleviate the sparsity of semantic information by leveraging the advantages of embedding models in transforming words from discrete representations into high dimensional continuous vector space. Furthermore, many sequential models like LSTM [9] are also adopted as encoders of input texts.

These machine learning or deep learning techniques have made remarkable progress in many NLP (natural language processing) tasks [3, 4, 17]. However, there are still many obstacles to leveraging them in service discovery. On the one hand, the words in service descriptions and queries are limited and extremely sparse, which makes the complex models hard to extract effective feature vectors from input texts. On the other hand, the useful words extracted from service descriptions represented in service description languages like WSDL can hardly form a natural sentence and are lack of context information, which are necessary for the model training of these sequential models. To address this issue, we propose a novel deep learning-based service matching model. In our model, we enrich the semantics of each word in service descriptions and queries using external knowledge sources (e.g., Wikipedia). Because the enrichment process will inevitably bring noise, we leverage a neural topic model to extract topic distributions from enriched service descriptions and employ the attention mechanism in weighing the various words in texts according to their topic distributions. The contributions of this paper are summarized as follows:

- We propose a novel service discovery approach by combining Bi-LSTM and the neural topic model.

- We present a semantics enrichment technique to generate better topic distributions for service descriptions.

The rest of the paper is organized as follows. Section II discusses related work. Section III introduces the details of our approach, and Section IV shows the results of experiments. Finally, Section V concludes the paper and puts forward our future work.

II. RELATED WORK

As a fundamental topic in services computing, Web service discovery or matching has been extensively investigated in the

past decade. Web service discovery aims to identify appropriate services according to user requirements, which could be functional or nonfunctional. In this paper, we focus our attention on matching functional requirements. Existing studies on service discovery can be approximately categorized into two types: ontology-based approaches, and machine learning or deep learning-based approaches.

To overcome the limitation of keyword matching based on the structure of WSDL documents, many ontology-based matching techniques have been proposed. SAWSDL-MX [22] and OWLS-MX [1] are the representatives of this type. These approaches firstly annotate services and queries using domain ontologies and then leverage ontology-based semantical reasoning for service matching, which can obtain ideal discovery performance based on ontological reasoning. However, the major obstacle is the absence of appropriate ontologies for the matching tasks since general-purpose ontologies will not work, and it is time-consuming and sometimes impractical to construct such problem-specific ontologies and annotate services and queries using the ontologies.

With the prevalence of machine learning techniques, some researchers leverage machine learning algorithms in service discovery. At first, many clustering techniques are employed to group similar Web services in advance. For example, Liu et al. [3] and Elgazzar et al. [4] extracted functionality-related elements, including content, type, message, port, and service name from WSDL documents, which are regarded as the input of subsequent clustering and matching processes. Zhang et al. clustered service goals to improve discovery results [2, 5]. Recently, deep learning has become a mainstream tool for NLP tasks, and there are also some attempts to employ techniques such as word embedding in service discovery. For example, Tian et al. [7] combined word embedding and LDA to improve the performance of service discovery. Xiong et al. [17] leveraged the strategy in service recommendation to generate textual features from service descriptions.

To produce better feature vectors of service descriptions and queries, attention mechanisms, together with LSTM, are also widely adopted. For example, Cao et al. [15] applied attention mechanisms and LSTM in Web service classification. Shi et al. [16] leveraged attention-based LSTM in service recommendation. In their work, service descriptions are enriched internally to address the semantic sparsity. Besides, Yang et al. [14] proposed a service classification approach by combining CNN with LSTM.

Inspired by the studies that attempt to enrich service descriptions and queries [6], in this paper, we introduce explanations for each word according to its description on Wikipedia into service descriptions and queries as the external knowledge and enrich the descriptions to alleviate the data sparsity problem. What's more, in our model, a neural topic model based on VAE (variational auto-encoder) [20] is employed to extract topics, and an attention-based Bi-LSTM is used as the encoder of service descriptions, which can help obtain more accurate vector representations.

III. PROBLEM DEFINITION AND SOLUTION

In this section, we first state the problem to be studied. In Section III.B, an overview of our approach is presented. In the rest parts, we introduce modules of the proposed model in detail. In Table I, we list the symbols frequently used in this section and their corresponding meanings.

TABLE I.   SYMBOLS USED IN THIS PAPER

Symbol	Meaning
S	A set of descriptions of services and queries
s_i	A description of a service or a query
X_{BoW}	The bag-of-words vector of s
E_s	The embedding matrix of s
H_s	The output hidden state matrix of s in LSTM
A_s	The weight vector of s
O_s	The feature vector of s
θ_s	The topic distribution of X_{Bow}
L	The number of words in s
dim	The dimension of word embeddings
V	The size of the vocabulary in S
K	The topic number in a topic model
k	The number of top-ranked services in $\widehat{R_S}$
h	The hidden size of LSTM
$\widehat{r_i}$	The matching score of s_i with a query
$\widehat{R_S}$	A list of matching scores of candidate services

A. Problem Definition

Suppose there are some candidate services in a registry. Given a user query, the problem to be addressed in this paper is how to properly match and rank services in the registry according to the user query. During this process, descriptions of services and queries (represented in WSDL or natural language texts) are the only information we can leverage. More formally, let O_q denote the feature vector of a query q and given a service description s_i (consisting of L words) in S, the feature vector O_i of s_i can be extracted. The matching score between s_i and q is $\widehat{r_i} = cosine(O_i, O_q)$. Consequently, the matching scores of all services are $\widehat{R_S} = \{\widehat{r_1}, \widehat{r_2}, ..., \widehat{r_n}\}$, and our proposed approach aims to return a list of services S_k with k highest scores.

B. Overview of Our Approach

Towards this problem, we proposed a Web service discovery approach based on attentional Bi-LSTM and enriched NTM (neural topic model), named as AENTM. As shown in Fig. 1, our model consists of three parts:

1) Neural Topic Model

Each description of a service or a query is represented as a bag-of-word vector X_{BoW}, and then the topic distribution θ_s is generated from X_{BoW} by a multi-layer perceptron. Besides, the module needs to reconstruct X_{BoW} from θ_s.

2) Attention-based Bi-LSTM Encoder

A description of a service or a query, s_i, is converted into an embedding matrix, and then fed into Bi-LSTM. The output is the hidden state matrix H_s, which contains the context feature of each word in s_i.
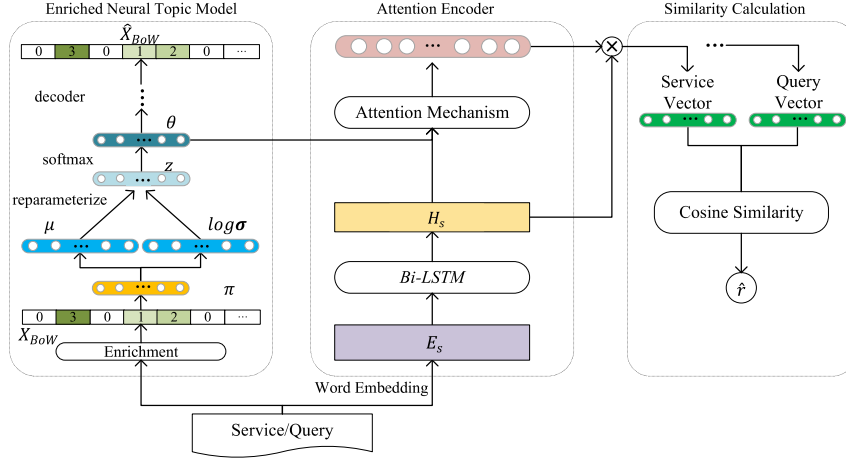
Figure 1. The overall framework of the proposed model.

An attention mechanism takes H_s along with θ_s as input, and produces a weight vector A_s for s_i. Consequently, the feature vector O_s is produced by the multiplication of A_s and O_s.

3) Similarity Calculation:

This module calculates the cosine similarity between feature vectors of service descriptions and queries, and the results are viewed as matching scores between them.

C. Enriched Neural Topic Model

NTM [20] is firstly proposed in a short text classification model to address data sparsity. According to the successful application of the neural variational inference mechanism in topic modeling [18, 19, 21], topic distributions generated by the neural topic model is analogous to Bayesian non-parametric topic models. Inspired by [20], the neural topic model in our approach is based on a variational automatic encoder trained on the overall corpus composed by descriptions of services and queries. Assume that the description of s_i consists of words $\{w_1, w_2, \ldots, w_L\}$, it is represented as a bag-of-word vector X_{BoW} before being fed into the neural topic model. In the neural topic model, the topic vector extracted by the model is represented as $\theta \in \mathbb{R}^K$. Note that in this paper, we only describe the structure and the data flow. More details can be found in [19, 20].

The first layer of NTM extracts hidden vector π, from which the input of reparameterization is generated. Then, the latent vector z is the result of a reparameterization with parameter pair (μ, σ). Both μ and σ are generated from π through a multi-layer perceptron. To normalize z, a softmax function is applied to generate topic vector $\theta \in \mathbb{R}^K$. Given the vector θ, the following parts of the topic model are supposed to reconstruct the input vector X_{BoW}, which will be employed as a decoder.

The calculation process of the neural topic model is described as follows.

$$\pi = relu(\boldsymbol{W}^\pi \cdot X_{BoW} + b_\pi), \qquad (1)$$

$$\mu = relu(\boldsymbol{W}^\mu \cdot \pi + b_\mu), \qquad (2)$$

$$log\boldsymbol{\sigma} = relu(\boldsymbol{W}^\sigma \cdot \pi + b_\sigma), \qquad (3)$$

$$Draw \ \boldsymbol{z} \sim \boldsymbol{\mathcal{N}}(\mu, \sigma^2), \qquad (4)$$

$$\theta = softmax\left(relu(\boldsymbol{W}^\theta \cdot \boldsymbol{z} + b_\theta)\right), \qquad (5)$$

$$\hat{X}_{BoW} = relu(\boldsymbol{W}^\phi \cdot \theta + b_\phi), \qquad (6)$$

where $\boldsymbol{W}^*$ and $\boldsymbol{b}^*$ are parameters to be learned, $relu$ is an activation function, and softmax is the normalized function for outputting topic distributions.

Due to the requirements of backward propagation, it is important to adjust Equation (4) such that gradients of parameters in the model are able to propagate. The reparameterization can be described as:

$$u = Draw \ u \sim \boldsymbol{\mathcal{N}}(0,1), \qquad (7)$$

$$z = u * \sigma + \mu. \qquad (8)$$

Equations (9) and (10) reveal the transformation from input to a latent topic distribution, which is regarded as an encoder in VAE. In addition, a decoder is employed to infer the output in the form of bag-of-words from θ. In our model, a multilayer perceptron is treated as a decoder, and according to basic VAE, the loss function is defined as:

$$\mathcal{L} = D_{KL}(q(z)||p(z|x)) - \mathbb{E}_{q(z)}[p(x|z)], \qquad (9)$$

where $p(z|x)$ represents the distribution of z with the input x, $q(z)$ is the standard normal distribution, and $p(x|z)$ is the probability distribution output by the decoder when taking z as input.

As mentioned in the previous section, the descriptions of services and queries are extremely sparse. To alleviate the issue, we introduce explanations from Wikipedia for each word in the description. In other words, for each word w_i in a description, our approach will search its corresponding explanation page in Wikipedia and extract the first paragraph explaining w_i. Note that there are a few words that have no corresponding explanation pages, and we ignore these cases and do not enrich them.

The enrichment operation can thus bring auxiliary information to the NTM module, which aims to improve the quality of topic distribution as well as the attention module further.

D. Attention-based Bi-LSTM

LSTM is widely applied in extracting features from texts. Our model leverages a bi-directional LSTM (Bi-LSTM) to capture information in the context of each word in the input sequence. Furthermore, an attention mechanism is adopted to weigh each word in the input sequence. More specifically, with the latent topics induced by the NTM described previously and the output of Bi-LSTM, the attention module produces a weight vector in the output of Bi-LSTM. Finally, the weight vector A_s and the hidden states H_s are multiplied to obtain the feature vector of the service description or query s_i.
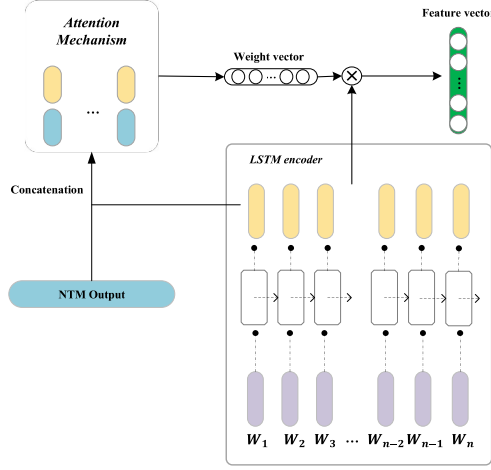
Figure 2.   Attention-based Bi-LSTM Module

As shown in Fig 2, firstly, given a sequence of words $s_i = \{w_1, w_2, w_3, \ldots, w_L\}$, we take the embedding of each word as the input of Bi-LSTM. The output of Bi-LSTM is the hidden state of each word. Next, hidden states of Bi-LSTM are concatenated with topic distribution θ_s, and then processed into an unnormalized weighted vector $a_s \in \mathbb{R}^{L \times 1}$.

$$E_s = embedding(s_i), \tag{12}$$

$$(\overleftarrow{H_s}, \overrightarrow{H_s}) = BiLSTM(E_s), \tag{13}$$

$$H_s = [\overleftarrow{H_s}, \overrightarrow{H_s}], \tag{14}$$

$$a_s = W^a \cdot tanh(W^\theta \cdot \theta_s + W^h \cdot h_i), \tag{15}$$

$$A_s = softmax(a_s), \tag{16}$$

$$O_s = A_s^T \cdot H_s, \tag{17}$$

where $embedding$ denotes the operation that maps words to vectors using a pre-trained embedding model, and H_s is the result of the concatenation of two direction hidden states of Bi-LSTM $(\overleftarrow{H_s}, \overrightarrow{H_s})$. A_s in Equation (16) is the normalized a_s. As the result of Equation (17), $O_s \in \mathbb{R}^{L \times 2h}$ represents the feature vector of s_i, which is fed into the similarity module subsequently.

Note that the encoder part of queries and services share the same attention-based Bi-LSTM module.

E. Similarity Calculation Module

The similarity calculation module produces matching scores from the input feature vectors of services and queries. Since the corpus or words of queries and services can be processed together, services and queries share the same NTM as well as the same attention-based Bi-LSTM. We adopt the widely-used cosine similarity to calculate their matching scores:

$$\hat{r} = R \cdot cosine(O_s, O_q), \tag{18}$$

where O_s and O_q denote the feature vectors of a service and a query, respectively. In particular, the result of the cosine function belongs to the interval $[-1, 1]$. Nevertheless, in some circumstances, different levels that measure the similarity between the query and the service may exist. Here, R is a coefficient that scales the output of the cosine function to match different situations.

IV.   EXPERIMENTS

In this section, we evaluate our approach and explore the factors that influence the performance of our model on a public Web service dataset.

A. Experimental settings

1) Dataset Description: SAWSDL-TC[1] is a WSDL collection for the service retrieval test, which consists of 1080 Web services and 42 queries represented in WSDL documents. These services belong to nine domains. A set of graded relevance (ranging from 1 to 3) for each query is provided, where 3 represents the highest relevance and 1 represents the least relevance. We utilized several preprocessing steps, including spelling correction, tokenization, stopword removal, and lemmatization, to extract words from documents. We employed the GooleNews[2] as the pre-trained embedding. Words absent in the dictionary of pre-trained embeddings were removed in the descriptions. As mentioned before, we leveraged Wikipedia to provide external knowledge for words in descriptions.

2) Evaluation Metrics: We adopted several commonly used evaluation metrics, including Precision, Recall, F1, and Normalized Discounted Cumulative Gain (NDCG), to evaluate the performance of our model. We evaluated the performance of top k services in the ranking list, where k ranges from 5 to 30.

3) Competing Approaches: We compared our proposed model with several state-of-the-art approaches.

- LDA [8]: LDA is a representative topic modeling method. We employed LDA to generate topic distributions for service descriptions and calculated the cosine distances of topic distributions between queries and candidate service descriptions.

- Lucene[3]: Lucene is a popular and high-performance text search method, which is the basis of many search engines. In our experiments, service descriptions were indexed according to Lucene.

- Doc2Vec [13]: Doc2Vec is an unsupervised model based on Word2vec. We trained a doc2vec model on queries and service descriptions, and calculated the cosine distance between their feature vectors.

---

[1] http://projects.semwebcentral.org/projects/sawsdl-tc/
[2] https://code.google.com/archive/p/word2vec/
[3] https://www.elastic.co/
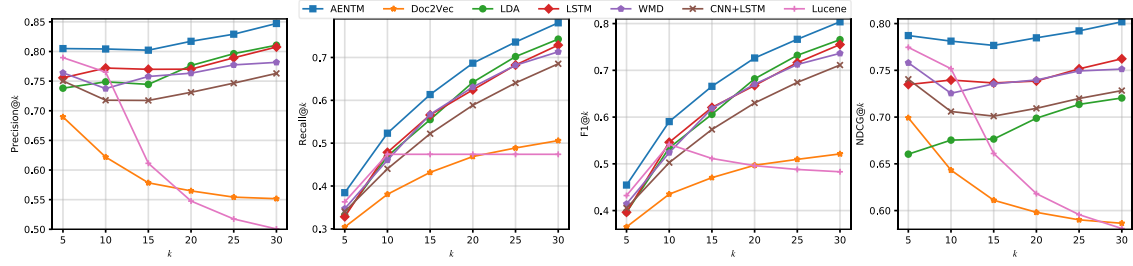
Figure 3. Performance comparison of different models
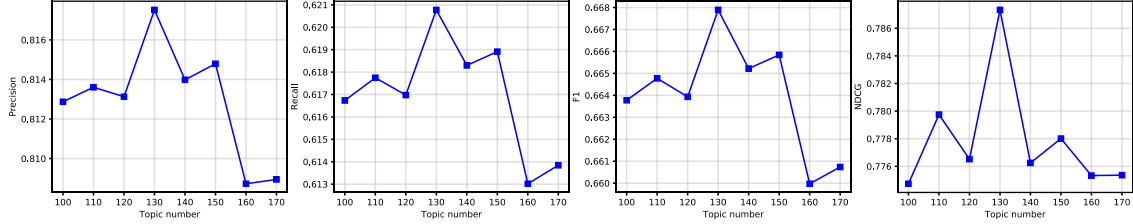
Figure 4. Impact of the topic number

- **WMD [12]:** WMD is a widely used method to measure the similarity of two documents or sentences based on a pre-trained word embedding. We used the pre-trained embeddings trained on the GoogleNews.

- **LSTM [9]:** To explore the effect of the attention mechanism and NTM module, we experimented on a Bi-LSTM with the same configuration in our model.

- **CNN+LSTM [14]:** CNN is a popular encoder in text-similarity calculation tasks [10]. In this experiment, we encoded texts by two CNNs, and then sent the output of the CNN encoder to a Bi-LSTM.

4) Parameter settings: The coefficient R in Equation (18) was set to 3 since the highest relevance level is 3. The learning rate for the encoder part was set to 0.0003 and trained with 30 epochs. The hidden size of Bi-LSTM was 150. To ensure the reliability of our experiment, we conducted a 5-fold validation on the dataset.

B. Performance comparison

According to the result presented in Fig. 3, our approach outperforms all competing methods across all ranking positions, which shows the advantages of our approach in addressing the data sparsity issue of Web services. More specifically, as a representative IR technique, Lucene suffers from a recall problem, and thus the performance decreases when k increases. It is indicated that LDA addresses the issue and achieves a good result in the precision, recall, and F1, but it still suffers from lower NDCG scores. Compared with LDA, deep learning approaches such as LSTM and WMD can achieve similar performance on F1, while they show better results on NDCG. Nevertheless, CNN+LSTM performs even worse than LDA, which indicates that the architecture is not very suitable for this task.

C. Impact of the topic number

We analyze the effects of parameter settings in our approach. The topic number of a topic model is a vital parameter that affects the performance as well as the quality of the topic distribution. In our approach, the topic number is still a hyperparameter that needs to be tuned on different datasets. Fig. 4 shows the change of performance when the topic number changes on the dataset. It is indicated that when the topic number is set to 130, our approach can obtain the best performance on all four metrics.

D. Response Time

Response time is an important consideration in service discovery. The response time of all approaches is shown in Table II. Note that the deep learning methods were performed with the help of a GPU (1080Ti), and other approaches were calculated on a CPU (Intel Xeon E5-2630).

TABLE II. RESPONSE TIME COMPARISON

Method	Response Time (*ms*)
LDA	428.74
LSTM	14.25
CNN+LSTM	23.36
AENTM	41.60
Lucene	2.29
WMD	4259.66
Doc2Vec	169.27

As shown in Table II, as a proven high-performance information retrieval technique, Lucene is the fastest approach. WMD costs the longest response time due to its high time complexity. The response time of LSTM, AENTM, and CNN+LSTM is relatively low. Although our AENTM model is relatively complicated, it does not take a much longer time to match services.

E. Effects of enrichment

As mentioned before, the services in the dataset belong to nine domains. To show the effects of the description enrichment, we reduced the topic distributions of descriptions (with 110-dimension vectors) generated by NTM to 2-dimension vectors, and then clustered them into nine clusters, as shown in Fig. 5. Moreover, the quality of clustering is usually measured by the CH score. We found that after the enrichment, the CH score increases from 2638 to 3180. Both the CH score and the visualization can demonstrate that after the enrichment of service descriptions, topic distributions generated by NTM become more distinguishable and cohesive, which further suggests that the enrichment can strengthen the semantics of descriptions to a certain extent.
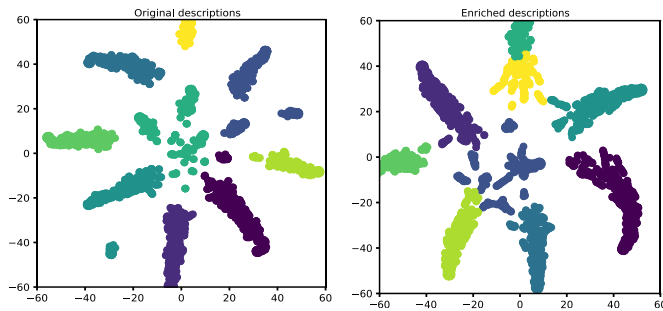
Figure 5.   Change of topic clusters before/after the enrichment

V.   CONCLUSIONS

In this paper, we propose a service discovery model by integrating enriched NTM with attention-based Bi-LSTM. An enrichment method is also adopted to leverage knowledge on Wikipedia to alleviate the lack of sufficient semantics in service descriptions. Experiments conducted on an open dataset show that our approach outperforms several state-of-the-art methods on discovery performance.

In the future, we plan to improve our approach from the following aspects. Firstly, the enrichment in our model employs the data on Wikipedia directly, which also brings lots of noise. The explanation should be selected precisely, and it may be helpful if techniques like attention mechanisms are applied. Secondly, in our experiment, the NTM module is hard to train due to the VAE part it leveraged, which needs to be further investigated.

REFERENCES

[1] M. Klusch, B. Fries, and K. Sycara, "Automated semantic web service discovery with OWLS-MX," in Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems, May 2006, pp. 915-922.

[2] N. Zhang, J. Wang, K. He, and Z. Li, "An approach of service discovery based on service goal clustering," in 2016 IEEE International Conference on Services Computing (SCC), June 2016, pp. 114-121. IEEE.

[3] W. Liu, and W. Wong, "Web service clustering using text mining techniques," IJAOSE, 3(1), 2009, pp. 6-26.

[4] K. Elgazzar, A.E. Hassan, and P. Martin, "Clustering wsdl documents to bootstrap the discovery of web services," in 2010 IEEE International Conference on Web Services, July 2010, pp. 147-154. IEEE.

[5] N. Zhang, J. Wang, Y. Ma, K. He, Z. Li, and X.F. Liu, "Web service discovery based on goal-oriented query expansion," Journal of Systems and Software, 142, 2018, pp. 73-91.

[6] X. Hu, N. Sun, C. Zhang, and T.S. Chua, "Exploiting internal and external semantics for the clustering of short texts using world knowledge," in Proceedings of the 18th ACM conference on Information and knowledge management, November 2009, pp. 919-928.

[7] G. Tian, J. Wang, Z. Zhao, and J. Liu, "Gaussian LDA and word embedding for semantic sparse web service discovery," in International Conference on Collaborative Computing: Networking, Applications and Worksharing, November 2016, pp. 48-59. Springer, Cham.

[8] D.M. Blei, A.Y. Ng, and M.I. Jordan, "Latent dirichlet allocation," Journal of machine Learning research, 3(Jan), 2003, pp. 993-1022.

[9] S. Hochreiter, and J. Schmidhuber, "Long short-term memory," Neural computation, 9(8), 1997, pp. 1735-1780.

[10] N. Kalchbrenner, E. Grefenstette, and P. Blunsom, "A Convolutional Neural Network for Modelling Sentences," in Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics, 2014, pp. 655-665.

[11] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean, "Efficient estimation of word representations in vector space," ICLR Workshop, 2013.

[12] M. Kusner, Y. Sun, N. Kolkin, and K. Weinberger, "From word embeddings to document distances," in International conference on machine learning, 2015, pp. 957-966.

[13] Q. Le, and T. Mikolov, "Distributed representations of sentences and documents," in International conference on machine learning, 2014, pp. 1188-1196.

[14] Y. Yang, W. Ke, W. Wang, and Y. Zhao, "Deep Learning for Web Services Classification," in 2019 IEEE International Conference on Web Services (ICWS), 2019, pp. 440-442. IEEE.

[15] Y. Cao, J. Liu, B. Cao, M. Shi, Y. Wen, and Z. Peng, "Web services classification with topical attention based Bi-LSTM," in International Conference on Collaborative Computing: Networking, Applications and Worksharing, 2019, pp. 394-407. Springer, Cham.

[16] M. Shi, and J. Liu, "Functional and contextual attention-based LSTM for service recommendation in Mashup creation," IEEE Transactions on Parallel and Distributed Systems, 30(5), 2018, pp. 1077-1090.

[17] R. Xiong, J. Wang, N. Zhang, and Y. Ma, "Deep hybrid collaborative filtering for web service recommendation," Expert systems with Applications, 110, 2018, pp. 191-205.

[18] Z. Cao, S. Li, Y. Liu, W. Li, and H. Ji, "A novel neural topic model and its supervised extension," in 29th AAAI Conference on Artificial Intelligence, IAAI, 2015, pp. 2210-2216.

[19] Y. Miao, E. Grefenstette, and P. Blunsom, "Discovering discrete latent topics with neural variational inference," in Proceedings of the 34th International Conference on Machine Learning-Volume 70, 2017, pp. 2410-2419. JMLR. org.

[20] J. Zeng, J. Li, Y. Song, C. Gao, M.R. Lyu, and I. King, "Topic Memory Networks for Short Text Classification," in Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, 2018, pp. 3120-3131.

[21] Y. Xiao, T. Zhao, and W.Y. Wang, "Dirichlet variational autoencoder for text modeling," arXiv preprint arXiv:1811.00135.

[22] M. Klusch, P. Kapahnke, and I. Zinnikus, "Hybrid adaptive web service selection with SAWSDL-MX and WSDL-analyzer," in European Semantic Web Conference, May 2009, pp. 550-564. Springer.

Explainable Deep Convolutional Candlestick Learner

Jun-Hao Chen[1] , **Samuel Yen-Chi Chen**[2] , **Yun-Cheng Tsai**[3] and **Chih-Shiang Shur**

[1]Department of Computer Science and Information Engineering, National Taiwan University
[2]Department of Physics, National Taiwan University
[3]School of Big Data Management, Soochow University
Corresponding Author: Yun-Cheng Tsai, pecutsai@gm.scu.edu.tw

Abstract

Candlesticks are graphical representations of price movements for a given period. The traders can discover the trend of the asset by looking at the candlestick patterns. Although deep convolutional neural networks have achieved great success for recognizing the candlestick patterns, their reasoning hides inside a black box. The traders cannot make sure what the model has learned. In this contribution, we provide a framework which is to explain the reasoning of the learned model determining the specific candlestick patterns of time series. Based on the local search adversarial attacks, we show that the learned model perceives the pattern of the candlesticks in a way similar to the human trader.

Keywords: local search adversarial attacks, explainable artificial intelligence, candlesticks, time series encoding, financial vision.

1 Introduction

The candlestick patterns recognition lies at the heart of trading and the foundation of all technical analysis. Therefore, understanding how to interpret candlestick is a critical step in becoming a trader. Traders need a candlestick patterns recognition tool to help them discover valuable information from candlestick. Although object detection and pattern recognition technologies have been prevailed in the computer vision field, traders generally cannot rely on these tools to gain insights of the candlestick patterns due to the lack of acquiring trading knowledge-based feature representations.

According to Tsai et al., they proposed an extended Convolutional Neural Networks (CNN) approach to recognize the candlestick patterns automatically [4]. Even though the deep learning based model has three significant advantages, including non-linearity, robustness, and adaptive manner, the traders cannot trust what the model recognizes the patterns from these charts precisely without explainability. However, the deep learning based models have several disadvantages, including lack of explanation capability [3] and difficulty in designing models. These difficulties will hinder the widely application of deep learning methodologies in critical

fields. We provide a framework based on adversarial attacks to demonstrate the adaptiveness and robustness of our model.

1.1 Candlestick Pattern of Time Series

The candlestick draws in a coordinate system, with the horizontal axis representing time and the vertical axis representing price. Time is from left to right on the X-axis. The nearest candlestick is the latest corresponding period. Price is from top to bottom on the Y-axis. The higher the position of the candlestick, the higher the price in those markets at the time. Conversely, the lower the position of the candlestick, the lower the market price at that time. Following the chart is drawn from historical prices according to specific rules. These features help traders to see the price trend. The three more common types of charts are histograms, line charts, and the most widely used candlestick. The candlestick is originated from Japan in the 17th century and has been popular in Europe and the United States for more than a century, especially in the foreign exchange market. As the most popular chart in technical analysis, traders should have an understanding of it. It is named after a candle, as shown in Figure 1. Each bar of candlestick draws from open price, high price,
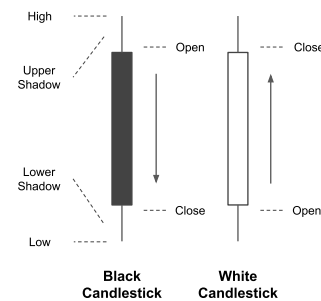
Figure 1: The shape of a candlestick.

lowe price, and close price. Open price is the first price during the period. High price is the highest price during the period. Low price is the lowest price during the period. Close price is the last price during the period. If the close price is higher than the open price, the candlestick follows the top of the candle body is the close price; the bottom is the open price; and

the color is usually green or white. If the close price is lower than the open price, the candlestick follows the open price above the candle body; the close price below; and the color is usually red or black. In some cases, the candlestick has no hatching because the open or close price coincides with the high or low price. If the candle is very short, the open and close prices of the candlestick are very similar.

1.2 The 8 Most Powerful Candlestick Patterns

The trick is in identifying some commonly occurring candlestick patterns and then building a market context around it. We provide the most eight common candlestick patterns to analysis our explainable model as follows:

1. Morning Star is a visual pattern made up of a tall black bar, a smaller black or white bar with a short body and long shadows, and a third tall white bar. The middle bar of the morning star captures a moment of market indecision where the bears begin to give way to bulls. The third bar confirms the reversal and can mark a new uptrend. Figure 2 shows the morning star based on the description.

2. Evening Star is a bearish candlestick pattern consisting of the latest three bars: a large white bar, a small-bodied bar, and a black bar. The pattern will be more visible with a large black bar than with a small black bar. Figure 3 shows the evening star based on the description.

3. Bullish Engulfing forms when a small black bar is followed the next bar by a large white bar, the body of which completely overlaps or engulfs the body of the previous bar.

4. Bearish Engulfing consists of an up white bar followed by a large down black bar that eclipses or "engulfs" the smaller up bar.

5. Shooting Star is a bearish bar with a long upper shadow, little or no lower shadow, and a small real body nears the low of the day. It appears after an uptrend.

6. Inverted Hammer looks like an upside down version of the hammer candlestick pattern, and when it appears in an uptrend is called a shooting star.

7. Bullish Harami is a black long bar followed by a white smaller bar that the later one is completely covered by the former. It indicates the end of a bearish trend.

8. Bearish Harami is composed of a long white bar and a small black bar that the later one is completely covered by the former. It indicates the end of a bullish trend.

1.3 Explain our Model

We use a Gramian Angular Field (GAF) time series encoder to emphasize the time series features for the Convolutional Neural Networks (CNN) model. The Gramian Angular Field (GAF) is a new time series encoder proposed by Wang and Oates [5]. It represents time series data in a polar coordinate system and uses various operations to convert these angles into the symmetry matrix. The GAF-CNN model is a two-step approach including Gramian Angular Field (GAF) time series encoder [5] and Convolutional Neural Networks
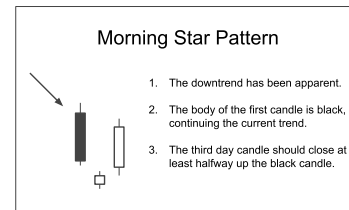
Figure 2: **Illustration of Morning Star Pattern.** The left-hand side shows the appearance of the Morning Star pattern. The right-hand side shows the critical rules of the Morning Star pattern.
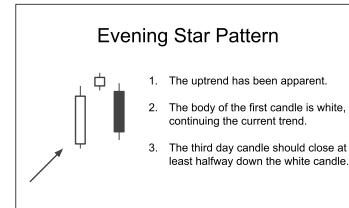
Figure 3: **Illustration of Evening Star Pattern.** The left-hand side shows the appearance of the Evening Star pattern. The right-hand side shows the critical rules of the Evening Star pattern.

(CNN) model. Teh GAF encoder makes time series data based on open, high, low, and close prices to GAF matrices. Our GAF-CNN model can capture the 8 major candlestick patterns with 90.0% accuracy. Then we would like to know if the model learned the features as human seen. We use the Local Search Attack Adversarial model [2] to attack the GAF matrices. The attacked regions are on the main diagonal of the GAF matrices. We defined these 10 bars based on the rules. The last 3 bars form the OHLC patterns, and a trend emerges in the rest of the bars. Figure 4 illustrates the attack region on GAF matrix. 1500 GAF matrices were attacked in each label. If most of the GAF matrices can be attacked successfully, it means the region human seen is similar to what the GAF-CNN model has learned.
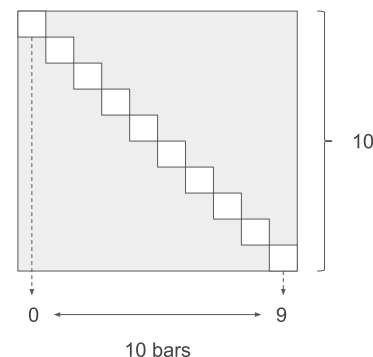
Figure 4: The GAF matrix for the local search attack. In this work, we attack the diagonal elements as these locations

2 Methods

2.1 GAF-CNN

The paper uses the summation version of the GAF. Each element of the GAF matrix is the cosine of the summation of aspects. Our first step is to making a GAF matrix to normalize the given time series data X into values between $[0, 1]$. The following equation shows the simple linear normalization method

$$\widetilde{x}_i = \frac{x_i - \min(X)}{\max(X) - \min(X)},$$

where $\widetilde{x}_i$ represents the normalized data. After normalization, our second step is to represent the normalized time series data in the polar coordinate system. The following two equations show how to get the angles and radius from the rescaled time series data. Finally, we sum the aspects and use the cosine function to make $GAF = \cos(\phi_i + \phi_j)$. Equation 1 and 2 show how to get the angles and radius from the rescaled time series data. Finally, we sum the angles and use the cosine function to make the GAF as equation 3.

$$\phi = \arccos(\widetilde{x}_i), -1 \le \widetilde{x}_i \le 1, \widetilde{x}_i \in \widetilde{X} \quad (1)$$

$$r = \frac{t_i}{N}, t_i \in \mathbb{N} \quad (2)$$

$$\text{GAF} = \widetilde{X}^T \cdot \widetilde{X} - (I - \widetilde{X}^2)^{\frac{T}{2}} \cdot (I - \widetilde{X}^2)^{\frac{1}{2}} \quad (3)$$

The GAF has two essential properties. The first, the mapping function from the normalized time series data to GAF is bijective when $\phi \in [0, \pi]$. In other words, normalize data to $[0, 1]$ can transform the GAF back into normalized time series data by the diagonal elements. The second, in contrast to Cartesian coordinates, the polar coordinates preserve absolute temporal relations. Once the GAF transform completed, the 3-d data can be inputs for the CNN model training. The architecture of our CNN model is similar to LeNet [1], including two convolutional layers with 16 kernels and one fully-connected layer with 128 densest.

2.2 Local Search Attack

There are several *adversarial machine learning* models to study the robustness of trained deep neural networks (DNN) models. The aim of these models are to generate input examples that are very close to the legitimate ones while causing the model to misclassify. There are two types of such models. One of them is *white-box* attack, in which the attacker know the complete model parameters. The other one is the *black-box* attack, in this case, the attacker does not have the model parameters, however, the attacker can try or query this model and read its outputs. In general, it is harder to attack a model if we do not have the information about it. Surprisingly, it has been shown that it is possible to successfully attack a model without the knowledge of its parameters. Further, it is even not necessary to perturb the whole input image. In *local search adversarial attack* method [2], it is possible to attack a handful of points in an image through the local greedy search. With this in mind, we hypothesize that if we can make the classifier to misclassify through perturbing only a small number of pixels on the image, it is highly possible that these pixels are crucial for the model to classify.

Algorithm 1 Local Search Attack

Load a single GAF two-dimensional array A
Set T = length of the time series
Keep a copy of A in memory D
Initialize the counter $t = 0$
for episode $= 1, 2, \ldots, R$ **do**
    if $t = 10$ **then**
        Reinitialize the A to the original value from memory D
        Reset the counter $t = 0$
    end if
    for $l = 1, 2, \ldots, T$ **do**
        Sampling a random perturbation scale r_l from uniform distribution $[0.8, 1.2]$
        Calculate the perturbated result $= r_l \times A[l, l]$
        if $r_l \times A[l, l] \ge 0.5 \vee r_l \times A[l, l] \le -0.5$ **then**
          $A[l, l]$ keeps the original value.
        else
          Set $A[l, l] = r_l \times A[l, l]$
        end if
    end for
    $t = t + 1$
    Recalculate the time series from perturbated A and then encode into a new GAF matrix A'
    if A' is adversarial **then return** A'
    end if
end for

We apply the following scheme to investigate the possible regions that are critical for the classification process. Logically, if a pixel is important in the final classification result, then a perturbation of that pixel should result in a degradation of the confidence score or even a misclassification. To achieve of this, we propose a method which is modified from the *local search attack* [2]. First of all, we define the set of points that can be perturbed. In this work, in order to maintain the consistency of the original time series data and the GAF matrix, we only perturbate the diagonal elements in the GAF matrix. Once we obtain the perturbated diagonal elements, we then calculate the corresponding values of non-diagonal elements and output the perturbated GAF matrix. Secondly, send this perturbated GAF into the CNN model to get the classification results. If the perturbated input is not misclassified, simply repeat the procedure described above. The detail of the algorithm is in Algorithm 1.

The parameters of our local search attack model are $r = \text{uniform}(0.8, 1.2)$; $d = 0$; $t = 10$; $R = 150$; and reset $= 10$.

3 Experimental Results

We use EUR/USD 1-minute open, high, low, and close price data to produce our experimental results. The training data is from January 1, 2010 to January 1, 2016. The testing data is from January 2, 2016 to January 1, 2018. There are eight patterns and each label includes 1500 samples. If the pattern does not belong to any one of the eight patterns, we set the kind of patterns as the label 0 and there are 3000 samples. These data produce the following results. Figure 5 shows the result of attacked morning star pattern and Figure 6 shows the result of attacked evening star pattern. With the modified local search attack model, we can reach 64.36% success attack rate on average. Table 1 presents the full results with at least

40.0%. This result suggests that it's plausible to focus attack region on the diagonal. Our GAF-CNN model actually recognizes the diagonal patterns, where the last 3 bars form the major patterns and the rest represent the trend. According to
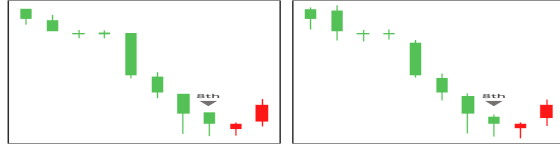
(a) The first attack result example.
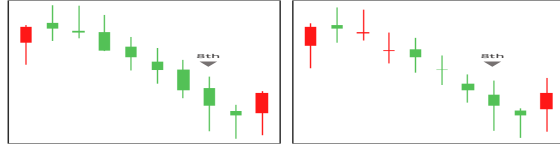
(b) The second attack result example.

Figure 5: The attack result example of morning star pattern. The left hand side shows the original pattern, and the right hand side shows the pattern after attack.
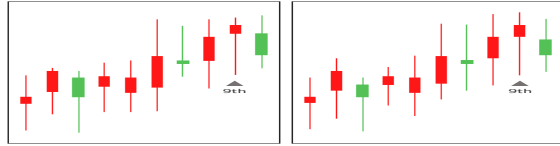
(a) The first attack result example.
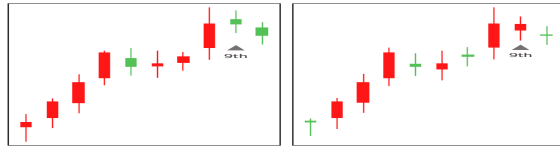
(b) The second attack result example.

Figure 6: The attack result example of evening star pattern. The left hand side shows the original pattern, and the right hand side shows the pattern after attack.

subsection 1.2, the morning star composes a downtrend and three-bar pattern, including a large black bar, a small-bodied bar, and a white bar. Figure 5-(a) shows that most of the bars change insignificantly after the perturbation. The front portion of candlestick remains downtrend, but the 8th bar reduces significantly. The changing of the 8th bar makes the pattern violate the morning star rules and lead to misclassification. In Figure 5-(b), there is some changing in the front portion, but still, obey the downtrend rules. The 8th bar also reduces significantly, causing the misclassification. The evening star pattern composes of the uptrend and three-bar pattern: a large white bar, a small-bodied bar, and a black bar. After the perturbation, the three-bar design violates the rules. Figures 6-(a) and 6-(b) show that the 9th bar changes significantly, and

Label	Success Rate	Percent (%)
1	631 / 1500	42.1
2	972 / 1500	64.8
3	1079 / 1500	71.9
4	1319 / 1500	87.9
5	602 / 1500	40.1
6	932 / 1500	62.1
7	953 / 1500	63.5
8	1238 / 1500	82.5

Table 1: The attack ratio of local search attack for each label.

the last bar becomes smaller, making the whole pattern invisible. The results show that our local search adversarial attack approach can explain the GAF-CNN model learned as human has seen and understand how GAF-CNN model recognize candlestick pattern. Our explainable GAF-CNN model is trustworthy and reliable for traders compared to others without knowing the underlying learning experience.

4 Conclusion

The paper has two contributions. The first is that our GAF-CNN model constructs an innovation field of financial vision research for candlestick recognition. The second is that we propose an approach based on the modified local search adversarial attack to explain the reason for the GAF-CNN model on how to determine the different candlestick patterns. Our GAF-CNN model can identify eight types of the candlestick and understands the feeling as a human has seen. We can confirm that the GAF-CNN model has indeed learned the sense of the candlestick from the trader. The GAF-CNN will be perfect for building a complete explainable trading model. We provide an open-source implementation and training data for the paper in the following URL: https://github.com/pecu/FinancialVision.

References

[1] Y. LeCun et al. Lenet-5, convolutional neural networks. *URL: http://yann. lecun. com/exdb/lenet*, page 20, 2015.

[2] N. Narodytska and S. Kasiviswanathan. Simple black-box adversarial attacks on deep neural networks. In *2017 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 1310–1318, July 2017.

[3] W. Samek, T. Wiegand, and K.-R. Müller. Explainable artificial intelligence: Understanding, visualizing and interpreting deep learning models. *arXiv preprint arXiv:1708.08296*, 2017.

[4] Y.-C. Tsai, J.-H. Chen, and C.-C. Wang. Encoding candlesticks as images for patterns classification using convolutional neural networks. *arXiv preprint arXiv:1901.05237*, 2019.

[5] Z. Wang and T. Oates. Encoding time series as images for visual inspection and classification using tiled convolutional neural networks. In *Proceedings of the 2015 Association for the Advancement of Artificial Intelligence (AAAI) Workshops*, pages 40–46, 2015.

Conditional Normalizing Flow-based Generative Model for Zero-Shot Recognition

Haiping Zhang, Xinwei Zhu, Dongjin Yu
School of Computer Science
Hangzhou Dianzi University, HDU
Hangzhou, China
zhanghp, zhuxinwei, yudj@hdu.edu.cn

Liming Guan, Zhongjin Li
School of Information Engineering
Hangzhou Dianzi University, HDU
Hangzhou, China
glm@hdu.edu.cn, lzjhdu@163.com

Abstract—Most existing studies on zero-shot recognition（ZSR）are typically about learning a shared embedding space to allow comparison of class prototypes by using nearest-neighbor methods, which suffer from hubness and bias problem. Recent studies attempted to directly synthesize samples of unseen classes by using generative model and have not encountered the aforementioned problems. However, their performance is limited by the inherent problems of VAE and GAN, such as reconstruction loss, mode collapse and unstable training procedure. In this paper, we explore and exploit a novel architecture of the generative model for ZSR, referred to as conditional normalizing flow-based generative model (CNFG). The proposed model consists of a cascade of affine couple transformations and can capture the low-distribution modes of real data density by virtue of its stable and exact log-likelihood maximum training procedure. Extensive experiments and result comparisons of 5 benchmarks have indicated that the normalizing flow-based model is superior to other generative models for ZSR in generalized settings.

Keywords—zero shot recgnition, generative model, affine couple transformation, hubness problem, model collapse

I. Introduction

Zero-shot recognition (ZSR) is a learning paradigm that attempts to recognize one object without any (or with zero) annotated data of the object in the training set. Motivated by the learning paradigm of human cognition, ZSR uses auxiliary semantic information of the category to train an effective model, which is required to correctly recognize not only the categories that appear in the training set (seen class) but also those that do not appear in the training set (unseen class). The key point of ZSR is to effectively explore and leverage the semantic knowledge of category that are shared between the seen and unseen class. Early studies on ZSR have mainly focused on the identification of a discriminative semantic representation of categories, such as semantic attributes or word embedding of labels. Leveraging these semantic knowledges, a mapping function from a visual/semantic embedding space to a shared embedding space is learned in the training set and then applied to the testing set. The data distribution of the two domains considerably vary. Thus, several intrinsic problems of the existing mapping-based methods are encountered.

Hubness Problem: [1] has theoretically and empirically demonstrated that hubness curse is an intrinsic characteristic of
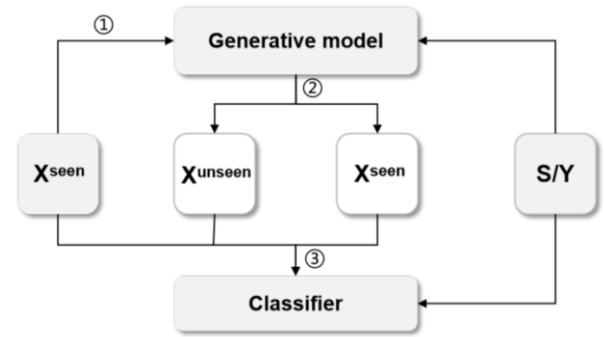
Figure 1. Overall pipeline of the proposed model

data distribution density in a high-dimensional space. That is, some hub vector points, which are not similar to other vector points, may be near many other points in a high dimensional space if measured using the nearest-neighbor search methods. The category label in the paradigm of mapping-based ZSR methods is determined by using the nearest-neighbor classifier to identify the most similar class prototype in the shared embedding space. Thus, [2] argue that the hubness problem also severely pollute the existing zero-shot method.

Domain Shift and Bias Problem: [3] argue that the mapping function, which is learned from the seen class, is often biased when applied directly to the unseen class because of disjoint classes and the inconsistent manifestation of visual attributes between training data and testing data. This occurrence is referred to as the projection domain shift problem. Coincidentally, [4] also empirically show that the learned mapping function does not perform well in the generalized setting because mapping functions are biased either toward the seen class or the unseen class.

Generative models have recently shown great potential for ZSR and have not encountered the aforementioned problems. These generative model-based ZSR methods attempted to synthesize samples of unseen classes conditioned by the semantic information by using variational autoencoder (VAE) or generative adversarial network (GAN) and cast zero-shot problem as a traditional supervised recognition problem. In this present study, we introduce a novel architecture of the generative model for ZSR, referred to as conditional normalizing flow-

based generative model (CNFG), which consists of a cascade of affine couple transformations and can capture the low-distribution modes of real data density by virtue of its stable and exact loglikelihood maximum training procedure.

To summarize, the main contribution of this study is threefold.

1) To the best of our knowledge, we are the first to explore and exploit the normalizing flow-based generative model to synthesize unseen data for ZSR problem.

2) We theoretically analyze and derive the formula of our proposed model and argue that our model can capture the low-distribution mode from real data density.

3) We also conduct extensive experiments and comparisons on 5 benchmarks, which shown that the normalizing flow-based generative model is superior to other generative models for ZSR in generalized settings.

II. RELATED WORK

Most existing studies on ZSR are typically about learning a shared embedding space to allow comparison of class prototypes by using nearest-neighbor methods. The pioneering work [5] is the first to propose embedding each class label into the space of attribute vector and cast earlier attribute-based multi-task learning as a label-embedding problem. ESZSL [6] argue that existing approaches to ZSR are highly sophisticated and propose a simple but effective compatibility function to model the relationship between visual embedding and attribute vector by explicitly regularizing the objective function. Prompted by the encoder-decoder paradigm, SAE [7] present a novel architecture consisting of two components. The encoder is responsible for projecting a visual representation into the semantic space similar to most existing ZSR models, and an additional decoder performs the reconstruction from a semantic representation to the visual space. Instead of embedding into a semantic space, DEM [8] propose to use the visual embedding space as the shared embedding space, which less frequently encounters the hubness problem. [9]innovatively introduce graph convolutional network (GCN) for predicting the class visual prototype by using both semantic embedding and the categorical relationships, with semantic embedding as input and visual embedding space as the shared embedding space.

Moreover, generative models have recently shown great potential for ZSR. GAMM [10] compare four different architecture of conditional data generators and emphasize the importance and efficiency of aligning the distributions of real and fake data by using explicit measure metric of distribution divergence, such as the KL divergence and maximum mean discrepancy. CVAE [11] uses the vanilla conditional variational auto-encoder (cVAE) model to directly generate samples conditioned by the given attribute representation for the class. SE-GZSL [12] further introduce a feedback-driven mechanism for cVAE architecture, which is coupled with a multivariate regressor to learn a projection from the cVAE decoder output to the representation of attributes that help increase the discriminative nature of the generated data. f-CLSWGAN [13] enhances Wasserstein GAN by adding a classification loss to the original generator loss for ZSR and enforces the model to
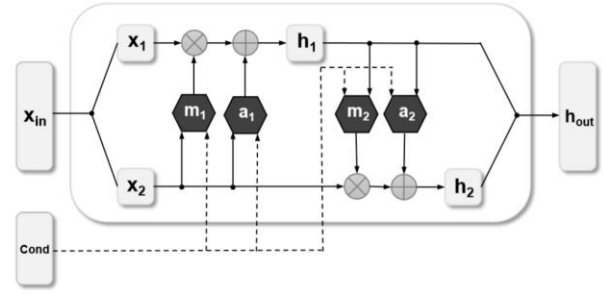
Figure 2. Pipeline of conditional affine coupling block(CACB)

generate sufficient image features that are more suitable for training a final multi-modal classifier. Inspired by cycle consistency loss, [14] introduces a multi-modal cycle consistency loss term that enforces better reconstruction from the generated visual representations back to semantic embedding. Using cyclic consistency loss and dual adversarial loss, [15] also proposed a novel model, referred to as GDAN, which combines visual-to-semantic projection, semantic-to-visual projection and a metric learning module in a unified framework and boosts the performance of ZSR. GMN [16] equip a conditional GAN with the gradient matching loss, which can measure the quality of the gradient signal acquired from the synthesized samples.

III. PRELIMINARY

Estimating the underlying distribution density $\tilde{p}(x)$ of the dataset X is a classical challenging task in machine learning. To learn the most representative generative model, the KL divergence between real distribution $\tilde{p}(x)$ and estimation distribution $q(x)$ has to be minimized

$$
\begin{aligned}
\mathrm{KL}\big(\tilde{p}(x) \parallel q(x)\big) &= \int \tilde{p}(x) \log \frac{\tilde{p}(x)}{q(x)} dx \\
&= E_{x\sim\tilde{p}(x)}\left[\log\frac{\tilde{p}(x)}{q(x)}\right] \\
&= c - E_{x\sim\tilde{p}(x)} \log q(x)
\end{aligned}
\tag{1}
$$

or equally, maximizing the likelihood function $E_{x\sim\tilde{p}(x)} \log q(x)$. The basic idea of the modern generative model is to introducing a latent variable z and then convert $q(x)$ into an integral formula of the following distribution

$$
\mathrm{q}(x) = \int q(x,z)dz = \int q(z)q(x|z)dz
\tag{2}
$$

where the prior distribution $q(z)$ of the latent variable z can be set as a common distribution density, such as the standard Gaussian density. The conditional distribution $q(x|z)$ presents a generative procedure, which can be conditional Gaussian density or Fermi-Dirac density.

However, the integral Formula (2) is intractable to optimization. Instead of minimizing $\mathrm{KL}\big(\tilde{p}(x) \parallel q(x)\big)$, VAE introduce a posterior distribution $\mathrm{p}(z|x)$, referred to as the encoder procedure, and descend to minimize the KL divergence of the joint distribution density

$$
\begin{aligned}
&\mathrm{KL}\big(p(x,z) \parallel q(x,z)\big) \\
&= \mathrm{KL}\big(\tilde{p}(x)p(z|x) \parallel q(z)q(x|z)\big)
\end{aligned}
$$

$$= \iint \tilde{p}(x)p(z|x) \log \frac{\tilde{p}(x)p(z|x)}{q(x|z)q(x)} dzdx$$

$$= E_{x \sim \tilde{p}(x)} \left[\int p(z|x) \log \frac{p(z|x)}{q(x|z)q(z)} dz \right]$$

$$= E_{x \sim \tilde{p}(x)} \left[E_{x \sim p(z|x)} [-\log q(x|z)] + KL\big(p(z|x) \| q(z)\big) \right] \quad (3)$$

which is an upper bound of $KL\big(\tilde{p}(x) \| q(x)\big)$ and is usually easy to calculate. The first item of Formula (3) is the reconstruction loss, and the second item is the KL loss of VAE. One of the problems of VAE is that the generated images are usually blurry. Owing to the Gaussian assumption of $p(z|x)$ and the upper bound of optimization, the representation ability of VAE is restricted. Moreover, [11] indicate that the generated image features of VAE are unimodal, which means that VAE cannot capture the low-distribution modes of the real probability distribution density.

The normalizing flow-based invertible generative models take a different way, which supposes the conditional distribution $q(x|z)$ as a Fermi-Dirac density

$$q(x|z) = \delta\big(x - G^{-1}(z)\big) \quad (4)$$

and tackle the aforementioned integral Formula (1) directly by a well-designed $G(z)$, which needs to ensure not only the invertibility

$$x = G^{-1}(z) \Leftrightarrow z = G(x) \quad (5)$$

but also, the tractable computability of Jacobian determinant

$$\frac{\partial G(x)}{\partial x} \quad (6)$$

If we set the prior distribution $q(z)$ as a standard multivariate Gaussian density

$$q(z) = \frac{1}{(2\pi)^{D/2}} exp\left(-\frac{1}{2} \|z\|^2\right) \quad (7)$$

estimation distribution $q(x)$ can be inferred by the integral transformation under the assumption of what $G(z)$ is invertible

$$q(x) = q\big(G(x)\big) = \frac{1}{(2\pi)^{D/2}} exp\left(-\frac{1}{2} \|G(x)\|^2\right) \frac{\partial G(x)}{\partial x} \quad (8)$$

whose logarithmic form is

$$\log q(x) = -\frac{D}{2} \log(2\pi) - \frac{1}{2} \|G(x)\|^2 + \log \frac{\partial G(x)}{\partial x} \quad (9)$$

which is the objective function of the normalizing flow-based invertible generative model. The invertibility is to satisfy the generative procedure, and the tractable computability of the Jacobian determinant is to facilitate the calculation of the loss function. To meet the requirements, the strategy of the normalizing flow-based invertible generative model is to use affine coupling blocks to construct $G(x)$. The method is presented in detail in the following section.

IV. APPROACHES

Our goal is to directly synthesize samples of unseen classes by explicitly modeling the underlying distribution of training data by using a powerful CNFG model. We can then train an ordinary supervised learning classifier by using synthesized

TABLE I. STATISTICS OF FIVE BENCHMARKS

Dataset	Total class	Seen class	Unseen class	Total instance	Train instance	Test instance (unseen/seen)	Attributes
AwA1	50	40	10	30475	19832	5685/4958	85
AwA2	50	40	10	37332	23527	7913/5882	85
CUB	200	150	50	11788	7057	2679/1764	312
SUN	717	645	72	14340	10320	1440/2580	102
aPY	32	20	12	15339	5932	7924/1483	64

unseen data and real seen data. The classifier can be any off-the-peg model, such as support vector machine (SVM) and SoftMax classifier. The overall pipeline of our model is illustrated in Figure 1.

A. Affine coupling block

The affine coupling block is the basic module of CNFG model, which was proposed by NICE [17] and popularized by Glow [18]. It is a combination of additive coupling block and multiplicative coupling block. An affine coupling block first splits the input x_{in} into x_1 and x_2, and then transforms $[x_1, x_2]$ into $[h_1, h_2]$ by applying the affine coupling transformation

$$\begin{aligned} h_1 &= x_1 \\ h_2 &= x_2 \otimes exp\big(m_2(x_1)\big) + a_2(x_1) \end{aligned} \quad (10)$$

whose inverse is

$$\begin{aligned} x_1 &= h_1 \\ x_2 &= \big(h_2 - a_2(h_1)\big) \oslash exp\big(m_2(x_1)\big) \end{aligned} \quad (11)$$

and the lower triangular Jacobians matrix is

$$\frac{\partial f}{\partial x} = \begin{pmatrix} \mathbb{I}, & \mathbb{O} \\ em \otimes \dfrac{\partial m_2}{\partial x_1} \otimes x_2 + \dfrac{\partial a_2}{\partial x_1}, & em \end{pmatrix} \quad (12)$$

where $em = exp\big(m_2(x_1)\big)$.

To improve the nonlinearity of transformation, [19] extends the affine coupling block by introducing a more complex affine transformation

$$\begin{aligned} h_1 &= x_1 \otimes exp\big(m_1(x_2)\big) + a_1(x_2) \\ h_2 &= x_2 \otimes exp\big(m_2(h_1)\big) + a_2(h_1) \end{aligned} \quad (13)$$

The conditional variant of the affine coupling block was first proposed by cINN [20]. Since the sub-transformation (m_i and a_i) of each affine coupling block is not inverted, cINN concatenate the conditional information c to the input of the sub-transformation and does not violate the assumption of invertibility. We can obtain the conditional affine coupling transformation by simply replacing $m_i(x)$ and $a_i(x)$ with $m_i(x,c)$ and $a_i(x,c)$ in Formula (13), respectively. The pipeline of the conditional affine coupling block is presented in Figure 2.

B. Architecture of our model

Our model is built on the principle of the conditional affine coupling transformation. The overall pipeline of the proposed model is presented in detail in Figure 1. Specifically, several

TABLE II.   RESULTS IN THE GENERALIZED ZSR SETTING

Method	SUN			CUB			AWA1			AWA2			aPY		
	acc^s_{avg}	acc^{us}_{avg}	H	acc^s_{avg}	acc^{us}_{avg}	H	acc^s_{avg}	acc^{us}_{avg}	H	acc^s_{avg}	acc^{us}_{avg}	H	acc^s_{avg}	acc^{us}_{avg}	H
CONSE	6.8	39.9	11.6	1.6	**72.2**	3.1	0.4	88.6	0.8	0.5	**90.6**	1.0	0.0	**91.2**	0.0
CMT	8.1	21.8	11.8	7.2	49.8	12.6	0.9	87.6	1.8	0.5	90.0	1.0	1.4	85.2	2.8
DAP	4.2	25.1	7.2	1.7	67.9	3.3	0.0	**88.7**	0.0	0.0	84.7	0.0	4.8	78.3	9.0
IAP	1.0	37.8	1.8	0.2	72.8	0.4	2.1	78.2	4.1	0.9	87.6	1.8	5.7	65.6	10.4
SSE	2.1	36.4	4.0	8.5	46.9	14.4	7.0	80.5	12.9	8.1	82.5	14.8	0.2	78.9	0.4
DEVISE	16.9	27.4	20.9	23.8	53.0	32.8	13.4	68.7	22.4	17.1	74.7	27.8	4.9	76.9	9.2
SJE	14.7	30.5	19.8	23.5	59.2	33.6	11.3	74.6	19.6	8.0	73.9	14.4	3.7	55.7	6.9
LATEM	14.7	28.8	19.5	15.2	57.3	24.0	7.3	71.7	13.3	11.5	77.3	20.0	0.1	73.0	0.2
ALE	21.8	33.1	26.3	23.7	62.8	34.4	16.8	76.1	27.5	14.0	81.8	23.9	4.6	73.7	8.7
SAE	8.8	18.0	11.8	7.8	54.0	13.6	1.8	77.1	3.5	1.1	82.2	2.2	0.4	80.9	0.9
SYNC	7.9	43.3	13.4	11.5	70.9	19.8	8.9	87.3	16.2	10.0	90.5	18.0	7.4	66.3	13.3
ESZSL	11.0	27.9	15.8	12.6	63.8	21.0	6.6	75.6	12.1	5.9	77.8	11.0	2.4	70.1	4.6
GFZSL	0.0	39.6	0.0	0.0	45.7	0.0	1.8	80.3	3.5	2.5	80.1	4.8	0.0	83.3	0.0
CVAE	-	-	26.7	-	-	34.5	-	-	47.2	-	-	51.2	-	-	-
SE-GZSL	30.5	40.9	34.9	53.3	41.5	46.7	67.8	56.3	61.5	68.1	58.3	62.8	-	-	-
GMMN	37.7	39.7	38.7	55.9	49.1	52.3	**70.1**	51.5	59.3	77.3	46.3	57.9	64.4	28.5	39.5
GDAN	**89.9**	38.1	**53.4**	**66.7**	39.3	49.5	-	-	-	67.5	32.1	43.5	75.0	30.4	**43.4**
CNFG (ours)	41.2	**43.6**	42.3	62.3	47.1	**53.6**	69.5	57.4	**62.8**	69.3	58.1	**63.2**	66.8	31.0	42.3

conditional affine coupling blocks can be cascaded and constructed into a more complex and powerful generative model, referred to as conditional normalizing flow-based generative model (CNFG). Suppose each affine coupling transformation denoted as f_i, where $i = 1, 2, \dots, n$, then we can obtain a composite function

$$
\begin{aligned}
z &= f_n(h^{(n)}, c) \\
&= f_n\left(f_{n-1}(h^{(n-1)}, c)\right) \\
&= \cdots \\
&= f_n(f_{n-1}(\dots f_0(x, c) \dots)) \\
&= G(x, c)
\end{aligned}
\tag{14}
$$

which is the conditional variant of the well-desired $G(x, c)$ in Formula (4). We generally denote the encoding procedure of the CNFG model as $G(x, c; \theta)$. The inverse or decoding procedure of the network is denoted as $G^{-1}(z, c; \theta)$, representing the generative procedure. Our goal is to optimize the network parameters θ by maximizing the logarithmic form of q(x) in Formula (9). The Jacobian matrix of affine coupling transformation strictly adheres to the lower or upper triangular form, as seen in Formula (12) and we can view it as a constant

$$
\frac{\partial f}{\partial x} = c
\tag{15}
$$

The partial derivative with respect to x of $G(x, c)$ is also a constant C.

$$
\frac{\partial G(x, c)}{\partial x} = \frac{\partial f_n}{\partial f_{n-1}} \cdot \frac{\partial f_{n-1}}{\partial f_{n-2}} \dots = \frac{\partial f_0}{\partial x} = C
\tag{16}
$$

Thus, the objective function log $q(x, c)$ in Formula (9) can be simplified as

$$
\log q(x, c) = -\frac{1}{2} \|G(x, c)\|^2 + C
\tag{17}
$$

The maximum log likelihood $E_{x \sim \tilde{p}(x)} \log q(x, c)$ is equal to the minimum of $1/2 \|G(x, c)\|^2$. Finally, the maximum likelihood training procedure can be implemented by simply minimizing the mean square value of $z = G(x, c)$.

Encoder procedure: We first extract all image features from real image instances by using the pre-trained ResNet101, then designate the 2048-dim image features x of the seen class and the corresponding attributes vector *c* as the inputs of the CNFG model. The input image feature x is transformed into the latent variable z via 12 conditional affine coupling blocks. The optimization objective is to minimize the mean square of z.

Decoder procedure: Once the generative model is learned, we can sample the latent variable z from the multivariate Gaussian density and then combine z with the attribute vector c of the unseen/seen class as the inputs of the generative model. The output is the synthesized instance of the corresponding class. We can generate any number of instances for the unseen/seen class because the data distribution of the latent variable z and the attribute vector *A* of the seen/unseen class is already known.

Classification procedure: With the "pseudo" annotated data, an off-the-peg supervised classification model can be trained. In the proposed model, we use the SVM as the final classifier. In the generalized setting, the classifier is biased toward the seen class to a certain extent if we only use the original annotated instance of the seen class and the synthesized instance of the unseen class. Thus, we augment the original annotated instance of the seen class with the synthesized instance of seen class as well.

V. EXPERIMENT

We present experiments on the five publicly released benchmarks: Animals with Attributes 1 (**AWA1**) [21] , Animals with Attributes 2 (**AWA2**) [22], Caltech-UCSD Birds 200 (**CUB**) [23], SUN attribute database (**SUN**) [24] and aPascal & aAyahoo (**aPY**) [25]. The statistical data for these benchmarks is listed in Table I. In accordance with [22], we use the harmonic mean average accuracy of the seen and unseen class in the testing set as the evaluation metric, which is defined as

$$
H = 2 * \frac{acc^{seen}_{avg} * acc^{unseen}_{avg}}{acc^{seen}_{avg} + acc^{unseen}_{avg}}
\tag{18}
$$

where acc^{seen}_{avg} and acc^{unseen}_{avg} represent the average accuracy of the seen and unseen class, respectively. We realize our model
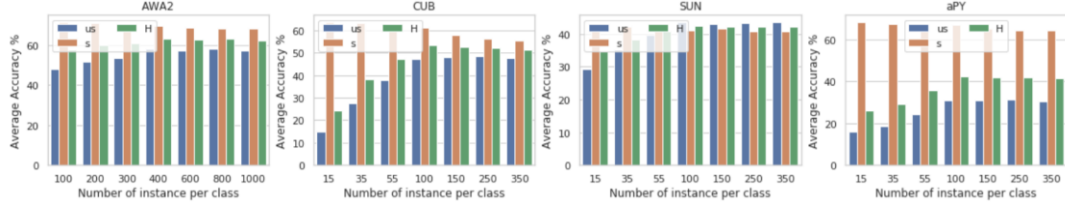
Figure 3. Analysis of the effects of the number of synthesized instances on acc_{avg}^{unseen}(**us**), acc_{avg}^{seen}(**s**) and **H** scores.
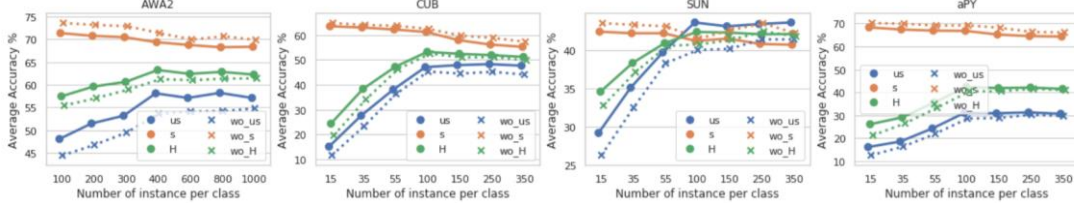
Figure 4. Analysis of the effects of the data augmentation on acc_{avg}^{unseen}(**us**), acc_{avg}^{seen}(**s**), **H** scores, where **wo_*** represent without data augmentation

with the deep learning framework Keras. Specifically, we construct CNFG model with 12 affine couple blocks. Each affine couple block consists of the *Shuffle*, *Split*, *Concat*, *AffineCouple* and *Subnetworks* modules. The *Shuffle* module first disrupts the order of the input vector for fully mixing the information and increasing the nonlinearity of transformation. The *Split* and *Concat* modules are responsible for dividing the input x into two parts before affine transformation and reassemble it back after affine transformation. The *AffineCouple* module is implemented with the corresponding subnetworks a_i and m_i by using a multilayer perceptron with 3 or 5 hidden layers and receiving the attribute vector directly as conditional information. The hidden layers have 1,024 units that are half the dimension of the image features. All multilayer perceptron subnetworks use ReLU activation function and appropriate dropout layers to avoid over-fitting. In the training procedure, we use Adam as our optimizer with the hyper-parameters learning rate = 0.001 and momentum = (0.9, 0.999).

A. Results Analysis

In accordance with [22], we randomly divide all seen class instances into 80% and 20% parts in the class level for the generalized setting. The two parts are denoted as X_{seen}^{train} seen and X_{seen}^{test}, respectively. We train our generative model on the training set D_{seen}^{train}, which consists of X_{seen}^{train} and the corresponding attributes representation A_{seen}^{train}, then synthesize the pseudo instance of the seen and unseen classes by using our trained generative model and denoting them as X_{seen}^{pseudo} and X_{unseen}^{pseudo}, respectively. We finally combine these pseudo instances with the original seen data X_{seen}^{train} to fit a multi-class linear SVM as the final classifier. Once the final classifier is fitted, we evaluate the performance of the fitted classifier on X_{seen}^{test} and X_{unseen}^{test} using average accuracy metric and denoting them as acc_{avg}^{seen} and acc_{avg}^{unseen}. Naturally, we calculate the harmonic mean value by using Formula (18) and present all acc_{avg}^{seen}, acc_{avg}^{unseen} and H scores on each dataset, as seen in Tables II.

Table II shows that the family of mapping-based methods have pervasive higher acc_{avg}^{unseen} scores and lower acc_{avg}^{seen}, H scores. These results demonstrate that the bias problem prevails in mapping-based ZSR methods, and those methods are not suitable for the generalized ZSR setting. Meanwhile, the family of ZSR methods based on the generative model made a good tradeoff between acc_{avg}^{seen} and acc_{avg}^{unseen}. The proposed CNFG model improves over the mapping-based method by 25% on AWA1/AWA2 benchmark and achieves the significant performance on the other benchmarks. We attribute this improvement to the efficiency of the generative model at capturing the underlying distributions. We also compare the proposed model with recent state-of-the-art methods based on the generative models. As shown in the bottom area of Table II, the proposed method outperforms most of the ZSR methods that are based on VAE or GAN. This difference in performance is attributed to the following: 1) the VAE and GAN have their own inner limitations, which are stated in Section III. 2) the proposed models can capture some low-distribution modes of real data density by virtue of its stable and exact log-likelihood maximum training procedure.

B. Number of Synthesized Instances

Although we can synthesize any number of instances for each class by using the generative model, it is inadvisable to arbitrarily generate large amounts of synthesized instances. In this section, we conduct several control experiments to evaluate the effects of NUM on the final classifier, which denotes the number of synthesized instances per class. We generate 7 different numbers of synthesized instances for each class by using the trained CNFG model. Specifically, we generate [15, 35, 55, 100, 150, 250, 350] instances per class for the CUB, SUN, and aPY datasets, as well as [100, 200, 300, 400, 600, 800, 1000] instances per class for the AWA1/AWA2 dataset, which is the large-scale dataset on the basis of the number of instances per class. The result is shown in Figure 3. Observation based on Figure 3 include the following: 1) With an increase in the

number of synthesized instances, the average accuracy of the unseen class acc_{avg}^{unseen} improves significantly for all datasets. This increasement is expected because there are no instances of the unseen class exist in the beginning. By contrast, this evident improvement demonstrates that the synthesized unseen data are very close to the real testing data of the unseen class. Thus, this increasement also indirectly proves the generative ability of the proposed model. 2) As the number of synthesized instances increases, the average accuracy of the seen class acc_{avg}^{seen} mildly decreases in the testing set. This result is expected because the final classifier is trained on an increasing number of synthesized unseen data. The higher acc_{avg}^{seen} may be irregular in the beginning, given that the final classifier is unintentionally biased toward the seen class. 3) The harmonic mean score H first increases rapidly but does not improve substantially upon reaching certain level because the acc_{avg}^{seen} score and acc_{avg}^{unseen} scores change in opposite directions with an increase in the number of synthesized instances. Thus, synthesizing numerous instances is unnecessary.

C. Data Augmentation

In the generalized setting, we can fit the final classifier by merely using synthesized unseen data and original seen data or augment original seen data with synthesized seen data. To evaluate the effectiveness of data augmentation, we trained two different models for each benchmark with or without augmented seen data in the generalized setting. As shown in Figure 4, the accuracy of the seen class acc_{avg}^{seen} is apparently higher than that of the unseen class acc_{avg}^{unseen} if we train the final classifier only on original seen data and synthesized unseen data. This mean that the final classifier is biased toward the seen class to a certain extent. The gap between the acc_{avg}^{seen} and acc_{avg}^{unseen} scores has been alleviated in the case of data augmentation. The improvement is explained by the fact that the synthesized seen data enlarging the decision space of SVM not only for the seen class but also for the unseen class.

REFERENCES

[1] Miloš Radovanovic, Alexandros Nanopoulos, and Mirjana Ivanovi ́c. Hubs in space: Popular nearest neighbors in high-dimensional data. Journal of Machine Learning Research, 11(Sep):2487–2531, 2010.

[2] Angeliki Lazaridou, Georgiana Dinu, and Marco Baroni. Hubness and pollution: Delving into cross-space mapping for zero-shot learning. In Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers), pages 270–280, 2015.

[3] Yanwei Fu, Timothy M Hospedales, Tao Xiang, and Shaogang Gong. Transductive multi-view zero-shot learning. IEEE transactions on pattern analysis and machine intelligence, 37(11):2332–2345, 2015.

[4] Wei-Lun Chao, Soravit Changpinyo, Boqing Gong, and Fei Sha. An empirical study and analysis of generalized zero-shot learning for object recognition in the wild. In European Conference on Computer Vision, pages 52–68. Springer, 2016.

[5] Zeynep Akata, Florent Perronnin, Zaid Harchaoui, and Cordelia Schmid. Label-embedding for attribute-based classification. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 819–826, 2013.

[6] Bernardino Romera-Paredes and Philip Torr. An embarrassingly simple approach to zero-shot learning. In International Conference on Machine Learning, pages 2152–2161, 2015.

[7] Elyor Kodirov, Tao Xiang, and Shaogang Gong. Semantic autoencoder for zero-shot learning. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 3174–3183, 2017.

[8] Li Zhang, Tao Xiang, and Shaogang Gong. Learning a deep embedding model for zero-shot learning. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 2021–2030, 2017.

[9] Xiaolong Wang, Yufei Ye, and Abhinav Gupta. Zero-shot recognition via semantic embeddings and knowledge graphs. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 6857–6866, 2018.

[10] Maxime Bucher, Stéphane Herbin, and Frédéric Jurie. Generating visual representations for zero-shot classification. In Proceedings of the IEEE International Conference on Computer Vision, pages 2666–2673, 2017.

[11] Ashish Mishra, Shiva Krishna Reddy, Anurag Mittal, and Hema A Murthy. A generative model for zero shot learning using conditional variational autoencoders. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops, pages 2188–2196, 2018.

[12] Vinay Kumar Verma, Gundeep Arora, Ashish Mishra, and Piyush Rai. Generalized zero-shot learning via synthesized examples. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 4281–4289, 2018.

[13] Yongqin Xian, Tobias Lorenz, Bernt Schiele, and Zeynep Akata. Feature generating networks for zero-shot learning. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 5542–5551, 2018.

[14] Rafael Felix, Vijay BG Kumar, Ian Reid, and Gustavo Carneiro. Multimodal cycle-consistent generalized zero-shot learning. In Proceedings of the European Conference on Computer Vision (ECCV), pages 21–37, 2018.

[15] He Huang, Changhu Wang, Philip S Yu, and Chang-Dong Wang. Generative dual adversarial network for generalized zero-shot learning. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 801–810, 2019.

[16] Mert Bulent Sariyildiz and Ramazan Gokberk Cinbis. Gradient matching generative networks for zero-shot learning. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 2168–2178, 2019.

[17] Laurent Dinh, David Krueger, and Yoshua Bengio. Nice: Non-linear independent components estimation. arXiv preprint arXiv:1410.8516, 2014.

[18] Durk P Kingma and Prafulla Dhariwal. Glow: Generative flow with invertible 1x1 convolutions. In Advances in Neural Information Processing Systems, pages 10215–10224, 2018.

[19] Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. Density estimation using real nvp. arXiv preprint arXiv:1605.08803, 2016.

[20] Lynton Ardizzone, Carsten Lüth, Jakob Kruse, Carsten Rother, and Ullrich Köthe. Guided image generation with conditional invertible neural networks. arXiv preprint arXiv:1907.02392, 2019.

[21] Christoph H Lampert, Hannes Nickisch, and Stefan Harmeling. Learning to detect unseen object classes by between-class attribute transfer. In 2009 IEEE Conference on Computer Vision and Pattern Recognition, pages 951–958. IEEE, 2009.

[22] Yongqin Xian, Bernt Schiele, and Zeynep Akata. Zero-shot learning-the good, the bad and the ugly. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 4582–4591, 2017.

[23] Peter Welinder, Steve Branson, Takeshi Mita, Catherine Wah, Florian Schroff, Serge Belongie, and Pietro Perona. Caltech-ucsd birds 200. 2010.

[24] Genevieve Patterson and James Hays. Sun attribute database: Discovering, annotating, and recognizing scene attributes. In 2012 IEEE Conference on Computer Vision and Pattern Recognition, pages 2751–2758. IEEE, 2012.

[25] Ali Farhadi, Ian Endres, Derek Hoiem, and David Forsyth. Describing objects by their attributes. In 2009 IEEE Conference on Computer Vision and Pattern Recognition, pages 1778–1785. IEEE, 2009.

An SNN Construction Method Based on CNN Conversion and Threshold Setting

Ying Shang, Yongli Li, Feng You

Department of Computer Science, Beijing University of Chemical Technology, Beijing, China
shangy@mail.buct.edu.cn, Liaibm@inbox.worktile.com, youf@mail.buct.edu.cn

Abstract—**We present a method to converse the Convolutional Neural Network (CNN) to the Spiking Neural Network (SNN), and we post a threshold adjustment algorithm for SNN. First, the adjustment strategy for CNN is introduced. Then after training, the weight parameters in the model are extracted, which is the corresponding synaptic weight in the layer of the SNN. Finally, a new threshold-setting algorithm based on feedback is proposed to solve the critical problem of the threshold setting of neurons in the SNN. We evaluate our method on the Cifar10 data sets released by Hinton's team. The experimental results show that the image classification accuracy of the SNN is more than 98% of that of CNN, and the theoretical value of power consumption per second is 3.9 *mW*.**

Keywords—Convolutional Neural Network; Spiking Neural Network; Threshold Setting Algorithm

I. INTRODUCTION

CNN is an artificial neural network with human visual processing as the model. It is the most successful network for image classification, and it has been widely used in the fields of target recognition, target detection, target positioning, and so on. However, due to the limitations of CNN, it requires a lot of computational power and training data. [1-3]

SNN is the neural network model closest to biological neural network at present, with strong computing power, which makes the application of SNN gradually increase, such as robot control, pattern recognition, speech recognition [4-7], image recognition [8-10], and target detection [11] and so on. SNN has similar supervised learning strategies with the traditional artificial neural network [12]. Still, the network model transmits the spike-timing signal, and the error function is not continuously differentiable, so it cannot use a monitoring algorithm for network training like a traditional neural network. [13,14]

Therefore, using the mature algorithm and framework of CNN to get the network model to solve related problems, and then converse CNN to SNN, this has become a direction that can be explored. In this way, both CNN and SNN can be considered, and the training can be carried out with the help of the mature CNN algorithm method, which also provides a solution for the application of SNN.

The main contributions of this paper are summarized below. First, we address a method to transfer the weight parameters in the CNN model to the synaptic weight in the SNN and realize the conversion from CNN to SNN. Second, a threshold-setting algorithm based on feedback adjustment is proposed to set the threshold of the neurons in SNN. Finally, experiment results based on benchmarks show that the proposed method can converse CNN to SNN, therefore to reduce the power consumption and the threshold adjustment algorithm is effective, which can improve the SNN classification accuracy.

The rest of this paper is organized as follows. In Section II, related work is discussed. Section III introduces our proposed method. In Section IV, we perform a comprehensive set of experiments on image classification. Finally, we conclude our work and suggest directions for future work.

II. RELATED WORK

SNN is a third-generation neural network, which can more truly simulate the information transmission mechanism between biological neurons. SNN is modeled using the synaptic transmission mechanism of biological neurons. This mechanism can describe various neuronal action potential triggers and transitions, as well as electroweak sequence coding operations. Based on this unique information transmission mechanism, we can imitate the mechanism of neurons in the brain to process information. Researchers proved that the SNN model has a stronger sense of calculating think than the first generation and second generation of the neural network. [15,16] Common SNN models include HH model [17], IF model, SRM model [18], Izhikevich model [19], etc.

$$\begin{cases} V(t) = V(t\text{-}1) + L + X(t) \\ IF V(t) \geq \theta \text{ , produce pulse and } V(t) = 0 \\ IF V(t) \prec \quad \text{set } V(t) = V_{\min} \end{cases} \quad (1)$$

In the above equation, L is a constant parameter, and X(t) is the sum of the inputs of all the synapses connected to the neuron in time. Once V(t) exceeds the threshold, this neuron is activated and generates a spike, and its membrane potential V(t) is reset to 0. The membrane potential of the neuron is not allowed to be set lower than Vmin, and Vmin is usually set to 0, which conforms to the biological characteristics of the neuron. Setting the threshold is an important issue.

For a neuron in the convolutional layer, X(t) can be defined as follows:

$$X(t) = \sum A(t)\, K \quad (2)$$

A(t) is the input spike from the previous layer, and it has a value of 1 or 0; 1 represents spike input, 0 represents no spike input; K is the weight of the convolution kernel connected to the neuron.

In the SNN model, the spike-timing signal is transmitted, and the error function is not continuously differentiable, which makes it impossible to train the network with a supervisory algorithm like the traditional neural network. However, CNN's mature algorithm and framework are used to get the network model to solve related problems, and then CNN is mapped to SNN, which becomes a direction to be explored.

Some researchers have studied the conversion of CNN to SNN. Cao and Grossberg [20] studied how the frequency-based perceptual neural network model could be transformed into an SNN model without losing performance, but they did not give a specific implementation of this method. The pulse-like HMAX method is the basic implementation of the original HMAX algorithm [21], but it does not explain how to transform the general HMAX structure into the pulse and CNN structure. Masquelier and Thorpe used unsupervised STDP method to learn unclassified features [22] and then used non-impulse classifier to classify these features, which resulted in the fact that the recognition results could not reach the accuracy of CNN similar to them. Perez-Carrasco [23] USES kernel projection to calculate convolution, but this method requires a lot of hardware resources. Xing F's method can be used to large-scale employment of spiking networks [24].

To analyzing the neurons, network structure, and network input of the two neural networks, the simulation of visual processing can be realized on SNN. This paper proposes a method to converse CNN to SNN. After CNN conversed to SNN, the threshold value of the neuron group in SNN needs to be set so that SNN can achieve the classification effect of CNN. In this paper, cifar10's CNN recognition is taken as an example, and a threshold setting method is proposed. After experimental verification, the SNN achieves the classification result close to that of CNN after applying the threshold setting method.

III. SNN CONSTRUCTION METHOD

In this section, we introduce our method to construct an SNN model. First, build a CNN model, and obtain the number of convolution kernels in the model, the combination of pooling layers, the number of iterations, and the learning rate. To skipping the difficult training problem of SNN, we map the CNN model weight parameters to SNN to achieve CNN to SNN conversion. To improving the accuracy of SNN classification, the threshold adjustment algorithm based on feedback adjustment is designed by using the idea of feedback adjustment algorithm (BP) in SNN training to set the threshold of neuron groups in impulsive neural networks. Create SNNs with high computational power and high image classification accuracy.

A. Converting to a Spiking Neural Network

The differences between SNN and CNN are reflected in the following three aspects:

1. Neuron: CNN's model is designed with biological neurons as the model. The main operation on CNN is convolution operation. The SNN model is the simulation of biological characteristics of biological neurons and the real simulation of biological neurons. The neurons participate in the complex integration operation and can realize the simple operation of convolution neurons.

2. Network structure: the structure of the CNN model is a typical multilayer feedforward network structure, which is similar to the feedforward network structure on the SNN model. Therefore, a similar network structure with CNN can be realized in the SNN model.

3. Input: CNN takes the pixel value of the image as the input of the network, while SNN is the spike converted from the image. A large number of studies have shown that the spike train following the Poisson distribution can well simulate the discharge of neurons, and the resulting spike train can be used as the input of the feedforward neural network.
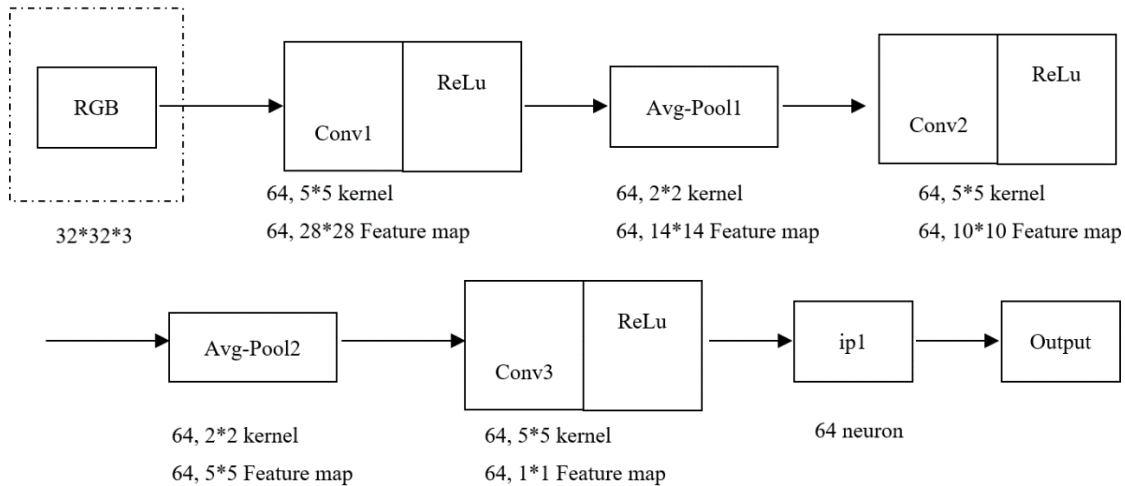
Figure 1. *detailed structure of the adjusted CNN*

Therefore, converse CNN to SNN, the parameters of the CNN model must be applied to SNN. Since SNN processes spike information, if the spike signal transmitted from the previous layer is received, the synaptic weight coefficient of the neuron

is 1; otherwise, it is 0, so no negative value will appear in the output layer of SNN. Therefore, the LRelu activation function used by the convolution layer is replaced by the Relu function of non-negative conversion. Then, no bias term is used in each layer of CNN, and the value of bias is set to 0. Finally, to reduce the errors caused by the increase in the number and complex structure of neurons, the maximum pooling that requires the two-layer group of neurons is realized through average pooling. The adjustment of CNN is summarized as follows. Figure 1 is the structure of the adjusted CNN.

B. Spike generation layer

In SNN, images need to be converted into spike signals in order to be used as network input. Therefore, before the input layer of SNN, there is the spike generation layer, whose function is to convert images into spike signals.

The response function of the spiking neuron is defined as follows:

$$\rho(t) = \sum_{i=1}^{k} \delta(t - t_i) \tag{3}$$

K is the number of spikes in the spike train, and t is the arrival time of the spike. According to the properties of the spike function, the number of received spikes in a minor interval can be calculated by the formula (4):

$$n = \int_{t1}^{t2} \rho(t)dt \tag{4}$$

The instantaneous discharge frequency of the spiking neuron is expressed by the expectation of the neuron response function.

$$\gamma(t) = \frac{dn(t)}{dt} = E\big(\rho(t)\big) \tag{5}$$

The mean value of the neuron response function within a short time interval can be approximated as the firing frequency of the neuron, as shown in formula (6):

$$\gamma_M(t) = \frac{1}{M}\sum_{j=1}^{M} \rho_j(t) \tag{6}$$

If the spike generation is assumed to be independent of each other, and the instantaneous discharge frequency is a constant. Set the number of spikes in the period as k, and then the probability of containing n spikes in the period can be expressed by formula (7):

$$P(n, t_1, t_2) = \frac{k!}{(k-n)!n!}p^n q^{k-n} \tag{7}$$

$p = \frac{(t_1 - t_2)}{T}$, $q = 1 - p$, while $k \to +\infty$ and the average discharge frequency is a constant, which can be substituted into equation 4-5 to get formula 4-6:

$$P(n, t_1, t_2) = e^{-\gamma\Delta t}\frac{\gamma\Delta t^n}{n!} \tag{8}$$

The spike generation layer converts the input image into the Poisson spike-timing signal, which is described as follows: I_{ijk} (k=1,2,3) is the image map of the input to the spike production layer. At time t, if the spike production function *spike( )<c* I_{ijk}(k=1,2,3) Where, the function of *spike(j,j)* is to generate a number following the distribution *(0,1),* and c is a constant, which is used to scale the spike frequency generated. Each simulation time can be set to milliseconds. After passing through the spike generation layer, the image is transformed into the spike signal, and the gray images of R, G, and B can be obtained after processing.

The adjusted CNN was trained several times to obtain a better detection model, which was used as a model of the SNN. The convolution kernel weight was mapped to the neuron synapse value of the SNN. Figure 2 is the detailed network structure diagram of the SNN.
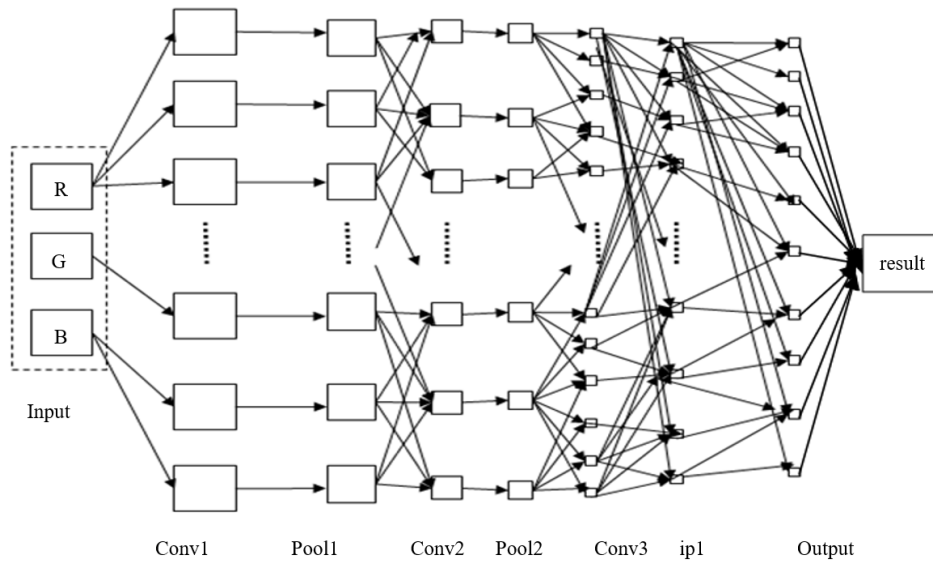
Figure 2. *detailed block diagram of the SNN*
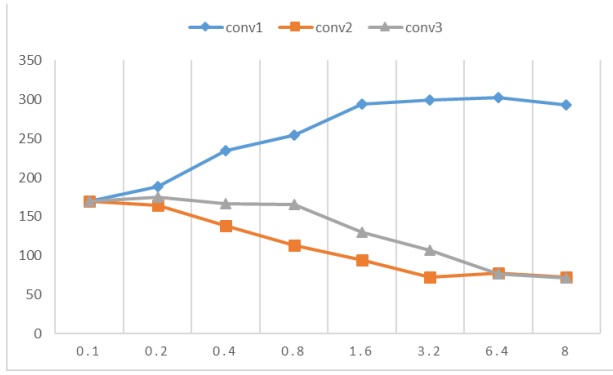
Figure 3. *Eeffect of threshold values on neuron groups conv1, conv2 and conv3 on output*

C. Threshold Setting Algorithm

In SNN, the threshold determines the spike generation of neurons. Only when the value of the neuron is greater than the threshold value, the neuron will generate a spike and transmit it to the lower neuron group. Therefore, the setting of the threshold will also have an impact on the characteristics extracted from SNN.

In Figure3, the horizontal axis is the threshold value, and the vertical axis represents the number of spikes received by the output neurons corresponding to the input. The higher the value is, the more accurate the classification is. It shows that, for the neuron group in SNN, when the threshold is too small, many features will be transferred to the next layer, which will include many features that have not been extracted, resulting in low accuracy of the final classification results. However, when the threshold is too large, the threshold, resulting in insufficient feature information and low accuracy of classification results, will filter out the features that can be used as classification.

Therefore, it is necessary to select an appropriate threshold for the neuron group so that the classification can be completed to the maximum extent, and the classification accuracy of the network can be improved.

We apply the idea of the back-propagation algorithm to the SNN threshold setting and propose a threshold adjustment algorithm based on feedback. This algorithm combines the threshold with the final classification effect. First, each layer is initialized with a threshold, and the initial value is set to 0, then set the maximum value for the neuron group to be adjusted, and then use the binary search method to set the threshold. The method to determine whether a threshold is optimal is:

Through the output feature map and convolution of the current neuron group The neural network corresponds to the output feature map, and observes whether the features of the feature map in the CNN and the SNN are consistent, that is, whether the brighter parts of the feature map are consistent;

Calculate the number of pulses of output neurons corresponding to the image in the output result, and determine whether the value is the maximum value;

Algorithm Threshold Setting Algorithm Based on Feedback

Input: a category picture in the classification, the threshold pi of the neuron group

Output: The ratio of the number of spikes of the corresponding neuron to the total number of output spikes
1: function threshold setting ($p1, p2, p3$)
2: Initialize the three-layer threshold in the SNN to *0.1, 0.1, 0.1*
3: Store the output of the network at max and n is the step size of the threshold change
4: repeat
5:     for all   ($p1, p2, p3$)   do
6:         $i = 1,2,3$; // represents the number of neurons
7:         Calculate the output result of the network *max1* when it is the *conv* layer *conv[i]* and the threshold $p [i] = p [i] + n$
8:         if *max1> max*
9:             Update threshold $p[i] = p[i] + n$
10:        else $p[i] = p[i] - n$
11:    end for
12: until the stop condition is reached
13: Output the threshold of the neuron group
14: end function

When the above two conditions are met, the threshold value is determined to be the threshold value of the threshold group, and then the next group of threshold group debugging until the final debugging is completed. The algorithm is as fs.

IV. EXPERIMENT AND ANALYSIS

A. Experimental Design

We carried out experiments on the Cifar10 data set, which is provided by the Hinton team. It is a common data set for classification problems. It consists of 10 categories: aircraft, cars, birds, cats, deer, dogs, frogs, horses, boats, and trucks. There are 60,000 images in the data set, which are divided into 50,000 train image samples, 5,000 for each category. Test 10,000 image samples, 1000 for each category. The size of the images in the data set is 32*32, and they are all color images. Table list the data set and the CNN classified result.

The experimental hardware environment was CPU Intel(R) 3.6GHz, RAM 32.00GB, the operating system was Ubuntu16.04, and the program compilation environment was Python2.7. The framework of CNN is Caffe.

B. Experimental Results and Analysis

This paper conducts experiments and analysis of the following two research questions (RQ):

RQ1:In this paper, CNN conversed to SNN. Whether the SNN consistent with CNN?

RQ2:What is the influence of the threshold-setting algorithm based on feedback on the classification accuracy of SNN?

1) RQ1

The consistency of the network model refers to the statistical analysis of the output results of the network model under the same data set and the analysis of the number of correctly classified and wrongly classified pictures on the data set. The higher the number is, the higher the consistency of the model will be. This section verifies the consistency of the unadjusted CNN model, the adjusted CNN model, and the SNN model on the test in the cifar10 data set.

Input a 32*32 RGB color image, when changing the image into a spike signal, the RGB separately coded into three ones. Figure 4 shows the feature map in the SNN. For the limitation of length, this paper only lists the feature graphs and output results of each layer in SNN. The characteristics of each layer feature graph corresponding to CNN are consistent, and the prediction trend of SNN results is consistent with that of CNN.
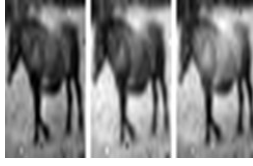
(a)input

(b)Conv1 feature map   (c)Pool1 feature map
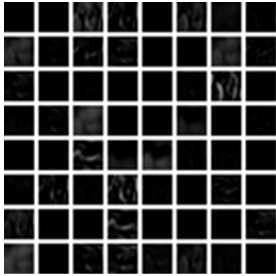
(d)Conv2 feature map   (e)Pool2 feature map

(f) Conv3 feature map   (g)output

Figure 4. *(a) ~ (g) are the output characteristics and results of each layer of the SNN*

2) RQ2

Through the threshold-setting algorithm, the threshold value of the network is obtained, as shown in Table 1 below.

In order to verify the effectiveness of the threshold-setting algorithm proposed in this paper, we randomly selected 100 pictures of each category in the test data set for the experiment. The classification accuracy is shown in Figure 5, where the threshold-setting algorithm is identified as C_SNN.

We found that after applying the threshold adjustment algorithm, the classification accuracy of ship and truck was slightly lower than that of no threshold setting, and the classification accuracy of other categories was improved.

TABLE I. THRESHOLDS FOR EACH NEURON GROUP OF SNN

Neuron	Threshold
Conv1	6.8
Pool1	0.75
Conv2	0.1
Pool2	0.75
Conv3	0.1
ip1	0.7
ip2	0.7
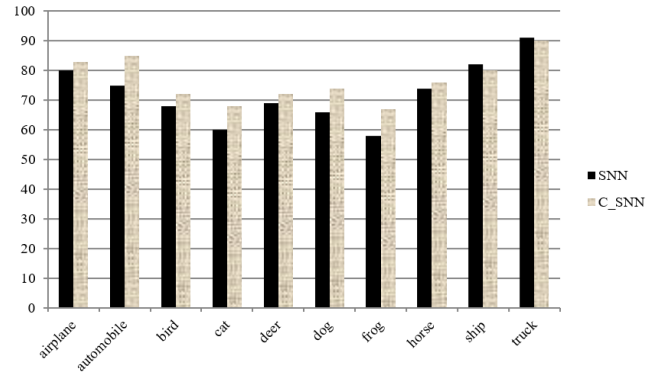
Figure 5. *Accurate Classification of SNN before and after Threshold Setting*

TABLE II. CLASSIFICATION ACCURACY OF ALL THE MODELS

	CNN (%)	Adjust CNN (%)	C_SNN (%)	C_SNN/CNN (%)
airplane	83.5	82.3	82	98.2
automobile	85.8	85.2	84.7	98.7
bird	71.7	71.5	70.6	98.5
cat	67.5	66.9	65.8	97.5
deer	84.2	83.6	82.8	98.3
dog	73.2	71.8	71.5	97.7
frog	77.1	76.9	75.4	97.8
horse	79.5	79.3	78.9	99.2
ship	86.7	85.7	84.9	97.9
truck	94.7	92.8	91.5	96.6
Ave	**80.39**	**79.6**	**78.8**	**98.04**

Table2 shows the statistical accuracy of each network model. The accuracy of SNN can reach 98% of that of CNN. The SNN obtained by CNN conversion can achieve the target classification problem and achieve good results.

Compared with CNN, the power consumption of SNN is lower. The theoretical analysis of SNN energy consumption shows that there are about 14200,000 synapses in the SNN model in this paper, and each synapse consumes a small amount of focus. Assuming that this model can process 742 pictures per second, the power consumed per second of this model can be calculated by the following formula:

$$P_{SNN} = 1.42 \times 10^7 \times 742 \times \alpha \ W \tag{9}$$

According to the data provided by Cruz-Albrecht, the energy consumption of each neuromorphic circuit is α = 0.37 mJ, so

$$P_{SNN} \approx 3.9 \ mW \tag{10}$$

Under the same hardware system, compared with the CNN, the SNN consumes less energy and has higher efficiency.

V. CONCLUSION AND FUTURE WORK

This paper studies the problem of image classification using CNN and SNN. A method of conversion CNN to SNN is proposed, and an adjustment strategy is designed for CNN to reduce the errors caused by the conversion. Through these adjustments, the spike generation layer, convolution layer, pooling layer, and output layer are constructed by using the feedforward network structure. The weight parameters in the adjusted CNN model are transferred to the synaptic weight in the SNN to complete the construction of the network model and realize the simulation of image classification. Finally, according to the relationship between the threshold and the classification results, a threshold-setting algorithm based on feedback is proposed, which can improve the classification accuracy of the pulse neural network.

In future work, the relationship between the threshold value of neurons and the characteristic graph of the network in SNN is deeply analyzed. The threshold value determines whether the neuron is activated to achieve feature selection. Therefore, the research on the relationship between the threshold value and feature value can provide more guidance that is effective for the threshold setting.

REFERENCES

[1] Cireşan D C, Meier U, Gambardella L M, et al. Deep, big, simple neural nets for handwritten digit recognition[J]. Neural computation, 2010, 22(12): 3207-3220.

[2] Farabet C, Couprie C, Najman L, et al. Learning hierarchical features for scene labeling[J]. IEEE transactions on pattern analysis and machine intelligence, 2013, 35(8): 1915-1929.

[3] Krizhevsky A, Sutskever I, Hinton G E. Imagenet classification with deep convolutional neural networks[C]. Advances in neural information processing systems. 2012: 1097-1105.

[4] Morri A L, Gomez F, Jimenez F Z. A spiking neural network for real-time spanish vowel phonemes recognition [J]. Neurocomputing, 2017, 226: 249-261.

[5] Silva M, Vellasco M, Cataldo E. Evolving spiking neural networks for recogition of aged voices [J]. J of Voice,2017, 31(1): 24-33.

[6] Dominguez M J, Jimenez A, Rois N A, et al. Multilayer spiking neural network for audio samples classification using SpiNNaker[C]. The 25th Int Conf on Artificial Neural Networks. Barcelona: IEEE, 2016: 45-53.

[7] Cerezuela E, Jimenez A, Paz R, et al. Sound recognition systems using spiking and MLP neural networks[C]. The 25th Int Conf on Artificial Neural Networks. Barcelona:IEEE, 2016: 363-371.

[8] Stromatias E, Soto M, Serrano G. An event-driven classifier for spiking neural networks fed with synthetic or dynamic vision sensor data[J]. Frontiers in Neuroscience,2017, 11(28): 350.

[9] Sun Q, Wu Q, Wang X, et al. A spiking neural network for extraction of features in colour opponent visual pathways and FPGA implementation [J]. Neucocomputing, 2017,228: 119-132.

[10] Ltaief M, Bezine H, Alimi A M. Training a spiking neural network to generate online handwriting movements[C].The 16th Int Conf on Intelligent Systems Design and Applications. Porto: ISEP, 2016: 289-298.

[11] Matsubara T, Torikai H. An asynchronous recurrent network of cellular automation-based neurons and its production of spiking neural network activities [J]. IEEE Trans on Neural Networks and Learning Systems, 2016,27(4): 836-852.

[12] Knudsen E I. Supervised learning in the brain[J]. J of Neuroscience, 1994, 14(7): 3985-3997.

[13] Bohte S M. The evidence for neural information processing with precise spike-times: A survey[J]. Natural Computing, 2004, 3(2): 195-206.

[14] Filp Ponulak and Andrzej. Introduction to spiking neural networks: Information processing, learning and applications [J]. Polish Neuroscience Society, 2011, 71: 409-433.

[15] Maass W. Networks of spiking neurons: The third generation of neural network models [J]. Neural Networks, 1997, 10(9):1659 -1671.

[16] Hodgkin A L, Huxley A F. A quantitative description of membrane current and its application to conduction and excitation in nerve [J]. Bulletin of Mathematical Biology, 1989, 52(1 - 2):25-71.

[17] Hodgkin A L, Huxley A F. A quantitative description of membrane current and its application to conduction and excitation in nerve[J]. J Physiol，1952，117(4) : 500-544.

[18] Gersterner w, Kistler W. Spiking neuron models [M]. Cambridge University Press，2002.

[19] Izhikevich E M.Which model to use for cortical spiking neurons?" Neural Networks [J]. IEEE Trans Neural Netw，2004，15(5) : 1063-1070.

[20] Cao Y , Grossberg S , Markowitz J . How does the brain rapidly learn and reorganize view-invariant and position-invariant object representations in the inferotemporal cortex?[J]. neural networks the official journal of the international neural network society, 2011, 24(10):1050-1061.

[21] Folowosele, F., Vogelstein,R.J.,&Etienne-Cummings, R.Towards a cortical prosthesis: implementing a spike-based HMAX model of visual object recognition in silico. IEEE Journal on Emerging and Selected Topics in Circuits and Systems.516-525.

[22] Masquelier,T.,&Thorpe,S.J. Unsupervised learning of visual features through spike timing dependent plasticity. PLoS Computational Biology,3,0247-0257.

[23] Perez-Carrasco,J.A.,Serrano,C.,Acha,B.,Serrano-Gotaeerdona,T.,& Linares-Barranco,B.Spike –based convolutional network for real-time processing. In 2010 International Conference on Pattern Recognition (pp:3085-3088).

[24] Xing F , Yuan Y , Huo H , et al. Homeostasis-Based CNN-to-SNN Conversion of Inception and Residual Architectures[M]. Neural Information Processing, 2019, 173-184.

Multi Classification of Alzheimer's Disease using Linear Fusion with TOP-MRI Images and Clinical Indicators

Qiao Pan, Golddy Indra Kumara, Jiahuan Chu
School of Computer Science
Donghua University, Shanghai, China
panqiao@dhu.edu.cn, gikumara@outlook.com, 1944362@qq.com

Abstract—**With the development of artificial intelligence, computer-aided diagnosis plays an increasingly important role in Alzheimer's disease (AD). In this paper, a new multi-classification diagnostic algorithm based on TOP-MRI images and clinical indicators is proposed. The features of TOP-MRI images and clinical indicators are fully exploited for multi-classification diagnosis of AD. First, we design TOP-CNN-NN model based on three VGGNet-16 convolutional neural networks and a single hidden layer neural network to extract the image feature vector of brain three orthogonal planes (TOP) MRI images. Then we screen clinical data using CfsSubsetEval evaluator to compose clinical feature vector. Then, the image feature vector and indicator feature vector are fused by using linear fusion method of multi-source data based on canonical correlation analysis (CCA). Finally, the fusion vector becomes the input for multi-classification classifier to distinguish three stages of AD: control normal (CN), mild cognitive impairment (MCI) and Alzheimer's disease (AD). The proposed algorithm is validated using the Alzheimer's Disease Neuroimaging Initiative (ADNI) dataset. Experiments show a good performance since the accuracy of the proposed algorithm in the multi-classification of AD can reach 86.7%.**

Keywords— Alzheimer's disease; linear fusion; multi-source data; multi-classification; deep learning

I. INTRODUCTION

Alzheimer's Disease (AD) is a neurodegenerative disease characterized by progressive cognitive decline that irreversibly affects all cognitive functions of human brain, and ultimately leads to severe impairment or premature death of the individual's daily activities [1]. According to [2], 50 million people are suffering from AD in 2018. About 8% of the elderly aged 65 to 85, and 35% aged 85 and over are affected by AD disease [3]. Clinically, AD is divided into three stages: control normal (CN), mild cognitive impairment (MCI), and Alzheimer's disease (AD). MCI is the early manifestation of AD, the transition state from CN to AD [4].

Magnetic resonance imaging (MRI) is often used as a basis for diagnosing AD due to its spatial resolution, high accessibility, and good contrast [6]. Common methods for computer-aided diagnosis of AD using MRI images are extracting features based on 3D medical images, using region of interest (ROI) to diagnose AD, and using image segmentation to measure the morphology of hippocampus, entorhinal cortex and amygdala to diagnose AD [8].

However, there are still problems in these methods including difficulty in designing algorithms based on 3D medical images due to their high dimensionality, noise and sparsity.

This paper proposes a new multi-classification diagnostic algorithm based on TOP-MRI images and clinical indicators. The three orthogonal planes (TOP) is a tangent plane in three directions centered on the spatial geometric center of the brain, clinical indicators include demographic information, neuropsychological assessment and biological detection. The main contributions of this paper are as follows:

- There is no need to label and divide ROI in extracting image feature vector by using TOP-CNN-NN model, which reduces the difficulty of prior knowledge.

- Only three different planar MRI images are needed to mine brain feature information. It avoids the difficulty of representation and modeling, which also improves the efficiency of model training and classification.

- CCA is used to fuse TOP-MRI image feature vector and clinical indicator feature vector. Considering various types of data, it conforms to the clinical reality. The validity of the proposed classification model is verified by using ADNI data sets.

II. RELATED WORK

Nowadays most methods of AD classification diagnostic study is to pre-process MRI image, extract features from the image and input the features into the classifier to predict diseases. AD multi-classification is proposed by Hiroki Karasawa using deep convolutional 3D neural network made of 36 Convolutional Layers 1 Dropout Layer, 1 Average Pooling Layer and 1 FC Layer in general [9]. Carlos Platero et al. raised a Hippocampal segmentation based on patch method and non-rigid registration label fusion method. ROI was marked initially by means of non-rigid registration label fusion method first, and then marked by means of patch method [10]. Devvi Sarwinda used oriented gradients of three orthogonal planes histogram to extract dynamic texture features, then they took advantage of probability principal component analysis (PPCA) for dimensionality reduction, and sort AD, MC, NC using random forest classifier [7]. Tooba Altaf et al. present a classification method of combined features, adopting MRI image textural feature

method with clinical data. In image textural feature extraction, segmenting the image into three areas: gray matter, white matter and cerebrospinal fluid, extracting features by means of GLCM, SIFT, HOG and LBP techniques, gaining clinical data by adopting characteristic indexes of FAQ, NPI, GDS [11]. Tong T. et al. present a nonlinear graph fusion (NGF) method to gain the complementary information across modalities [12].

III. METHODS

The multi-classification diagnosis algorithm proposed includes four modules: image feature extraction, indicator feature selection, feature vector fusion and disease classification diagnosis. Fig. 1 shows the framework of the algorithm.

In the image feature extraction module, three MRI images from the three orthogonal planes of the brain are selected. The images are pre-processed and then input them into the TOP-CNN-NN model to extract image feature vector. In the indicator feature selection module, the clinical indicators are selected by CfsSubsetEval evaluator and combined to form indicator feature vector. In the feature vector fusion module, image feature vector and indicator feature vector are linearly fused by Canonical Correlation Analysis (CCA). In the disease classification diagnosis module, input the fusion feature vectors to the multi-classifier to distinguish the three stages of AD: control normal (CN), mild cognitive impairment (MCI), and Alzheimer's disease (AD).

A. Image feature extraction

In this module, TOP-CNN-NN model is constructed to extract the feature vector from the MRI images. The model consists of three VGGNet-16 convolutional neural networks (CNN) and a single hidden layer network (NN). The framework is shown in Fig. 2.

First, The MRI image is pre-processed, then it is extracted to the VGGNet-16 convolutional neural network. Second, the three preliminary feature vectors from the VGGNet-16 are weighted by voting. Lastly, input the weight vector into the single hidden layer network to generate the fusion feature vector.

1) Image pre-processing

In this module, three orthogonal planes (TOP) MRI images are selected as input of image feature extraction model. TOP MRI images contains important information for the diagnosis of AD, such as the hippocampus, entorhinal cortex and amygdala.

Different planar images show that patients with AD have atrophy, ventricular enlargement and other pathological features compared with normal (CN) and mild cognitive impairment (MCI) patients, as shown in Fig. 3.

Many problems on MRI images occurs from the detection equipment and techniques such as irregularity, high noise, different shade, etc. To solve these problems, this module uses three steps to pre-process the image. The steps are as follows:

a) Geometry transformation

The main purpose of geometry transformation is to improve the spatial position of brain imaging area in MRI images. It is used to correct the systematic errors of magnetic resonance imaging (MRI) instruments and the random errors of imaging position (e.g. imaging angle, perspective relationship). In this module, translation and rotation are used to solve the problems of image position offset and angle deflection caused by imaging angle in brain MRI images. Zoom is used to unify different size of brain images caused by the difference of perspective relationship.
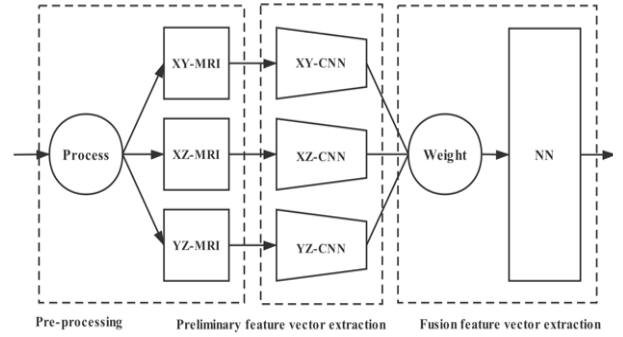
Fig. 2. TOP-CNN-NN image feature extraction model framework
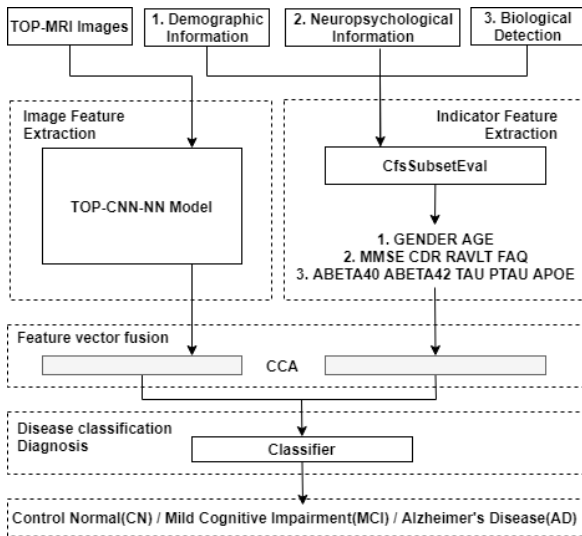
Fig. 1. Framework of Multi-Classification Diagnosis Algorithm
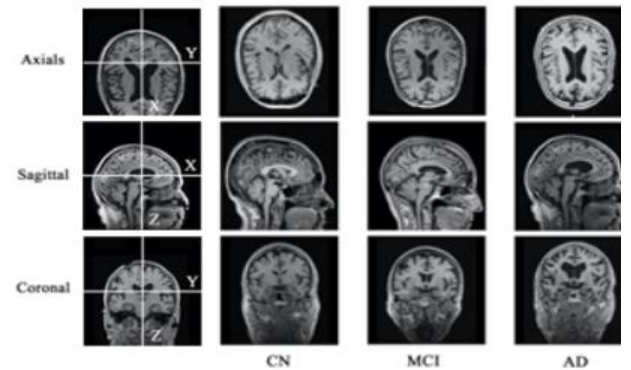
Fig. 3. Different brain planes of patients' MRI images with CN, MCI, AD

b) Image noise reduction

In this module, salt-and-pepper noise is processed by median filter, and Gaussian noise is processed by Gaussian filter. The median filtering technology makes the image smooth by sorting the pixels in the field according to the gray level, and then choosing the median value in the group as the output pixel value. Gaussian filtering is a weighted averaging process for the whole image. The value of each pixel is obtained by weighted averaging of its own and other pixel values in its neighborhood. The coordinates with domain size of $(2n+1) \times (2n+1)$ are brought into (1) to calculate the pixel values of the corresponding coordinates:

$$G(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{(x-n-1)^2 + (y-n-1)^2}{2\sigma^2}} \qquad (1)$$

where x, y is the coordinates of the pixels, σ are the values that need to be set.

c) Standardization

The purpose of pixel value standardization is to scale the original image pixel value and limit the pixel value to a certain range. The standardization process is defined as:

$$MRI_S = \frac{MRI - \mu}{max\left(\sigma, \frac{1.0}{\sqrt{N}}\right)} \qquad (2)$$

where MRI is the image matrix, μ is the image mean, σ is the standard variance, and N is the number of MRI image pixels. Consistent input data can be obtained by standardization, which will avoid different shades and contrast problems. It can improve the convergence speed of the model and the accuracy of the classification diagnosis model.

2) Preliminary feature extraction

Convolutional Neural Networks (CNN) is a class of feedforward neural networks with convolutional computation and deep structure, which has been widely used in image fields [13]. In this module, we use VGGNet-16 to obtain the tangential images of the three orthogonal planes of the brain: Axials, Sagittal, and Coronal. Then these three tangential images are trained for XY-CNN, XZ-CNN and YZ-CNN. These three CNN models are used to extract the preliminary feature vector of the respective planes.

VGGNet is a deep convolutional neural network that explores the relationship between the depth of the convolutional neural network and its performance. By repeatedly stacking 3*3 small convolution kernels and 2*2 maximum pooling layers, a convolutional neural network has been constructed with 16-19 layers. In this module, the model is based on VGGNet-16, which is shown in Fig. 4.

The input of VGGNet-16 is RGB image of 224 x 224 size. In the process of image convolution, the feature map MRI_i representing layer i of VGGNet-16 is used. Assuming that
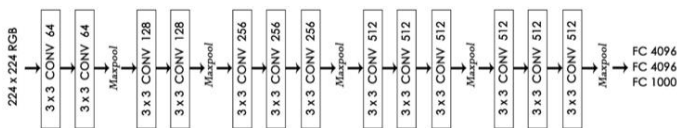
MRI_i is a characteristic graph of convolution layer, its generating process can be described as:

$$MRI_i = f(MRI_{i-1} \times W_i + b_i) \qquad (3)$$

where W_i is the weight vector of the ith layer convolution kernel. The operation symbol "$\times$" is the convolution operation of the convolution kernel and the $(i-1)^{th}$ layer image or feature map, and the output of the convolution and the offset vector of the ith layer b_i are added. The feature map MRI_i of the ith layer is obtained by the nonlinear excitation function f(x). The VGGNet-16 model uses a 23-layer convolution layer. The low-level convolutional layer of the model extracts some low-level features such as edges and lines. The high-level convolutional layer of the model iteratively extracts more complex features from low-level features. After each convolution layer, a max pooling layer is added for more complete and important features. Assuming that MRI_i is the Max pooling layer, its generating process can be defined as:

$$MRI_i = Maxpooling(MRI_{i-1}) \qquad (4)$$

For several image feature values extracted by filter, max pooling only retains the largest pooling layer feature. This operation can reduce the number of model parameters and reduce the over-fitting problem. Finally, VGGNet-16 uses the 3 fully connected layer to combine the extracted features. Assuming that full connection layer has p parameters, x_n is the input or n feature graphs, and its generating process can be defined as:

$$FC_p = W_{p1} * x_1 + W_{p2} * x_2 + ... + W_{pj} * x_n + b_p \qquad (5)$$

where p is the number of the full connection layer and w is the weight matrix. Each neuron in the connective layer is fully connected to all the neurons in the preceding layer. Full connection layer can integrate local information with category discrimination in convolution layer. The initial eigenvector mentioned in this chapter is the output of the last full connection layer in VGGNet-16. The dimension of preliminary feature is 1000.

3) Fusion feature vector extraction

Since each planar image in three orthogonal planes (TOP) of the brain has its own characteristics, there are differences in the regions and expressions of interest in the VGGNet-16 feature extraction process. A voting weighted vector fusion method as shown in Fig. 5 is adopted, which can highlight the respective features and reduce the vector fusion problem caused by feature differences.

Fig. 4. Network parameters of VGGNet-16 model
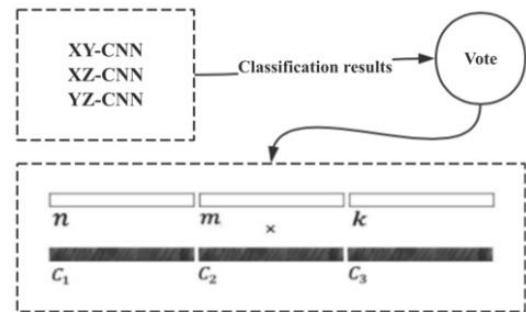
Fig. 5. Voting weighting process of image preliminary feature vector

Firstly, by counting the classification results corresponding to the TOP. If the classification results are the same, use it as the voting result, otherwise the higher classification accuracy will be taken as voting result. Then, each voting result corresponds to a weight vector. Assume that the preliminary feature vectors of XY-CNN, XZ-CNN, and YZ-CNN are $C_1=\{\alpha_1,\ldots,\alpha_{1000}\}$, $C_2=\{\beta_1,\ldots,\beta_{1000}\}$ and $C_3=\{\gamma_1,\ldots,\gamma_{1000}\}$, respectively. Then the voting weighting operation can be defined as:

$$C = (n\alpha_1, \ldots, n\alpha_{1000}, m\beta_1, \ldots, m\beta_{1000}, k\gamma_1, \ldots, k\gamma_{1000}) \quad (6)$$

where n, m and k are weighting factors. After changing the values of the weighting factors n, m and k, the proportions of the three preliminary feature vectors in the fused feature vector are no longer balanced. In the process of setting the weighting factor, assuming that the classification result of one of the planes is the same as the voting result, the preliminary feature vector weight extracted by the plane is increased.

Finally, input the voting weighted feature vector C into the single hidden layer neural network. The single hidden layer neural network will fuse the feature vector of three orthogonal planes and output the fusion feature vector with lower dimension, which is beneficial for the linear fusion on the next step with the clinical indicator feature vector, and avoid the over-fitting problem. The image feature vector output by the TOP-CNN-NN model is the hidden layer output of the single hidden layer network, and the dimension of the fusion feature vector is 50.

B. Indicator feature selection

The ADNI dataset contains clinical information for each subject, such as gene detection, demographic information, neuropsychological assessment, biological detection, etc.

In this module, we use the CfsSubsetEval evaluator to evaluate the classification capabilities and redundancy of each indicator. During the evaluator's selection, the indicators with high correlations with disease classification results but low correlations with each other are selected. As long as the subset does not contain indicators that are more relevant to the current indicator, the indicators that are most relevant to the disease classification results are continuously added. The evaluator will use the missing value as a separate value, or it can distribute the missing value count along with the other values according to the frequency of occurrence. Selecting a subset of indicators can also help eliminating irrelevant and duplicate indicators. The relationship between the two indicators I_1 and I_2 can be measured by symmetric uncertainty, defined as:

$$U(I_1, I_2) = 2\frac{H(I_1) + H(I_2) - H(I_1, I_2)}{H(I_1) + H(I_2)} \quad (7)$$

where the basis of the entropy function H is the probability of each indicator. H（I_1,I_2) is the joint entropy of I_1 and I_2, it is calculated from the probability of all combinations of I_1 and I_2. The range of uncertainty is 0-1.

Feature selection based on correlation determines the superiority of an indicator set, defined as:

$$R = \frac{\sum_j U(I_j, C)}{\sqrt{\sum_i \sum_j U(I_i, I_j)}} \quad (8)$$

where C is the category of AD, and I_i and I_j are all indicators in the indicator set.

Through the CFS evaluator, this module selects 11 indicators as clinical indicators. There are 3 types of clinical indicators: demographic information, neuropsychological assessment, and biological detection. Demographic information consists of patients' gender and age.

The indicators for neuropsychological assessment are Mini-Mental State Examination (MMSE), Clinical Dementia Rating (CDR), Rey Auditory Verbal Learning Test (RAVLT), and Functional Activity Questionnaire (FAQ). MMSE is the most common scale for clinically examining intelligence. It can comprehensively, accurately and quickly respond to the mental state and the degree of cognitive impairment of the subject. The patient's condition is reflected by the total score of the scale.

The CDR is obtained by retrieving information from patients and their families to complete the cognitive impairment degree assessment, which will quickly assess the severity of the patient's condition. Areas of assessment include memory, orientation, judgment and problem-solving skills, work and social skills, family life and personal hobbies, and the ability to live independently.

The RAVLT immediate and delayed test evaluates the patient's speech memory. The patients will listen to a certain amount of content in this test and then performs immediate and delayed recall to judge their condition. Studies have shown that RAVLT distinguishes AD from other neuropsychological assessments [14]. Lastly, the details regarding daily chores is measured in FAQ. Activities that require higher cognitive abilities can prove to be quite useful in assessing the condition of dementia subjects.

The indicators for biological detection are amyloidβ-peptide (Aβ), Tau protein (Tau), water-soluble phosphorylated Tau protein (P-Tau), and Apolipoprotein E (ApoE 4). Amyloid β is the main component of senile plaque, which is a characteristic of neuropathology of AD. The most important Aβ are Aβ40 and Aβ42 [14]. The study found that the amyloid beta protein in brain tissue of AD patients increased significantly [17], therefore the detection of plasma Aβ levels is helpful for detecting AD.

Tau protein is a microtubule-related protein with low molecular mass, and is prone to form paired helical filaments (PHFs) after abnormal phosphorylation and glycosylation, and further constitute neurofibrillary tangles, which are the characteristic pathological manifestations of AD [14]. The level of tau protein in the cerebrospinal fluid of patients with moderate to severe AD was significantly higher than the normal CN subjects, and the increase of this index was earlier than the occurrence of clinical dementia symptoms, which can be used for the prediction of AD [15].

ApoE is one of the most important apolipoproteins in the central nervous system. It is involved in the mobilization and redistribution of cholesterol. It is also necessary to maintain
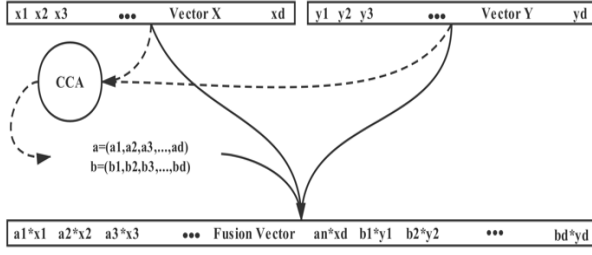
Fig. 6. Feature vector fusion process based on CCA

the integrity of myelin sheath and neuron cell membrane after the development and injury of the nervous system [14], and its protein level in plasma is affected by ApoE genotype. Related studies have shown that patients with ApoE genotype have a higher risk of progression from MCI to AD, thus ApoE has a certain reference for the diagnosis of AD [16].

C. Feature vector fusion

Multi-source data fusion integrates data from different data types through some data fusion rules, absorbs the characteristics of different types of data, and extracts standards from them. It is better and has more rich information than single data.

Multi-source data can be fused at three levels: vector level, feature level, and decision level. Decision-level fusion is achieved by synthesizing the classification results of multiple classifiers. However, decision-level integration is not suitable for the fusion of images and indicators; Feature-level fusion is a more common way, and descriptive that can express the correlation between features better. However, extracting the descriptive characteristics of medical images requires a certain prior knowledge, which is a challenging task. Therefore, this paper chooses to fuse MRI images and clinical indicators at the vector level.

A common vector-level fusion method aim is to connect two feature vectors end-to-end to generate a new feature vector. This method does not consider the relationship between two feature vectors. Canonical correlation analysis (CCA) is used in this chapter to analyze the correlation between image and index feature vectors, which will generate new fusion feature vectors. CCA is a statistical method for dealing with the interdependence between two random vectors. CCA plays a very important role in multi-source statistical analysis and also a valuable multi-source data processing method [18]. It is not only suitable for information fusion, but also suitable for removing redundant information.

In Fig. 6, assume that the MRI image feature vector is x and the clinical indicator feature vector is y. To extract typical related features, it is recorded as $\alpha^T x$ and $\beta^T y$ (one pair of typical variables), where $\alpha = (a_1, a_2, a_3, \ldots, a_d)$ and $\beta = (b_1, b_2, b_3, \ldots, b_d)$. The projection directions α and β can be obtained by maximizing the following criterion functions:

$$\rho = \frac{E[\alpha^T xy^T \beta]}{\sqrt{E[\alpha^T xx^T \alpha] \times E[\beta^T xy^T \beta]}} = \frac{\alpha^T E[xy^T]\beta}{\sqrt{\alpha^T E[xx^T]\alpha \times \beta^T E[xy^T]\beta}} = \frac{\alpha^T S_{xy}\beta}{\sqrt{\alpha^T S_{xx}\alpha \times \beta^T S_{yy}\beta}} \quad (9)$$

where S_{xx} and S_{yy} is covariance matrix, S_{xy} is the covariance matrix between x and y. The fusion feature vector can be obtained by multiplying the image feature vector and the indicator feature vector with their corresponding typical variable α and β, as defined as follows:

$$C = (\alpha, \beta)^T (x, y) \quad (10)$$

where C is the fusion feature vector of image and indicator.

D. Disease classification diagnosis

This paper diagnoses Alzheimer's disease in three stages: control normal (CN), mild cognitive impairment (MCI) and Alzheimer's disease (AD), by inputting fusion feature vectors of images and indicators into the classifiers.

The multi-classifier selected in this paper is decision tree. Each internal node represents a test on an attribute, each branch represents a test output, and each leaf node represents a disease type. It represents a mapping relationship between object attributes and object values, using algorithms ID3, C4.5 and C5.0 spanning tree algorithm to use informatics theory of entropy.

IV. EXPERIMENT

The experimental dataset used in this study was obtained from Alzheimer's Disease Neuroimaging Initiative (ADNI). The dataset contains MRI images and clinical indicators. There are 302 patients in 3 categories: 91 in control normal (CN), 141 in mild cognitive impairment (MCI), and 70 in Alzheimer's disease (AD).

The MRI image uses three orthogonal planes section images obtained by T1 weighted and three-dimensional magnetization preparatory gradient echo sequences. It has high spatial and time resolution, high signal-to-noise ratio, small artifacts, and good contrast to the internal structure of the brain which is conducive to showing small brain changes.

There are 11 clinical indicators which come from demographic information, neuropsychological assessment and biological detection. The demographic information has two indicators: gender and age. Table 1 shows that there is no significant difference in the distribution of age, regardless of gender, in the three stages of AD.

Neuropsychological assessment consists of four scales: MMSE, CDR, RAVLT, and FAQ. Table 2 shows the Neuropsychological statistical assessment of patients with CN, MCI, and AD. The MMSE scores of patients from CN to AD decreased significantly, while the values of CDR, RAVLT and FAQ increased.

Biological detection consists of two amyloid proteins, two Tau proteins and apolipoprotein E. The changes of biomarkers were shown in Fig. 7. The levels of ABETA40 and ABETA42 in CN patients were lower than AD patients, APOE volume, and the levels of tau and P-tau were higher.

The proposed algorithm is validated in terms of accuracy. The accuracy is calculated as:

$$\text{Accuracy} = T \div C \times 100\% \quad (11)$$

where T is the correct number of samples and C is the total number of samples participating in the classification. The accuracy of the experiment results is obtained by calculating the average value of the five experiments.

TABLE I. DEMOGRAPHIC INFORMATION STATISTICS

	Gender		Age
CN	Male = 49		75.7±5.0 [65.8-85.5]
	Female = 42		
MCI	Male = 95		74.1±7.6 [59.2-89.1]
	Female = 46		
AD	Male = 37		74.7±7.8 [59.2-90.2]
	Male = 33		

Note: Gender Unit - Subjects, Age Unit - Year (Quantile Statistics of T Distribution)

TABLE II. NEUROPSYCHOLOGICAL ASSESSMENT STATISTICS

	MMSE	CDR	RAVLT	FAQ
CN	29.1±7.6 [27.1-31.1]	0.02±0.10 [-0.18-0.23]	3.35±3.0 [-2.60-9.30]	0.20±0.73 [-1.25-1.65]
MCI	27.1±1.8 [23.5-30.6]	1.61±0.93 [-0.23-3.45]	4.92±2.2 [0.54-9.30]	6.32±3.1 [-1.21-14.1]
AD	23.7±1.9 [19.8-27.5]	4.16±1.47 [1.24-7.09]	4.49±2.2 [0.46-8.51]	12.5±6.7 [-1.05-26.0]

Note: Scale Unit – Score (Quantile Statistics of T Distribution)

TABLE III. COMPARISON AMONG MULTI-CLASSIFICATION PERFORMANCE USING FOUR DIFFERENT BRAIN PLANAR IMAGES DATA

Data	Method	Accuracy
AXIALS-MRI	XY-CNN	48.3%
SAGITTAL-MRI	XZ-CNN	58.3%
CORONAL-MRI	YZ-CNN	51.7%
TOP-MRI	**TOP-CNN-NN**	**75.0%**

TABLE IV. COMPARISON BETWEEN THE PERFORMANCES OF MULTI-CLASSIFICATION MODELS BASED ON MEDICAL IMAGES

Author	Data	Method	Accuracy
TONG T [12]	MRI	NGF	56.3%
	MRI + PET	NGF	56.5%
LIU M [19]	MRI + PET	ISML	53.8%
Our Method	**MRI**	**TOP-CNN-NN**	**75.0%**
ZHE X [20]	MRI	SVM-RFE	85.6%

A. Evaluation of image feature extraction module

In this module, a TOP-CNN-NN model is constructed to extract the feature vector of MRI images. Table 3 shows the comparison among multi-classification performance based on different brain planar MRI images. The results show that the classification performance based on TOP-MRI images is better than that based on single plane MRI images. The experimental results verify the rationality of using three orthogonal planes MRI images in feature extraction module.

Table 4 shows performance comparison among multi-classification models based on medical images proposed by other papers and this paper. The experimental results show that the performance of this model is better than Tong T's NGF method [12] and Liu M's ISML method [19]. Although the performance of the model established in this paper is not better than the method based proposed by Zhe X [20], this proposed model does not need prior knowledge and clinicians' participation in the process of modeling.

ABETA40 ABETA42 APOE TAU PTAU
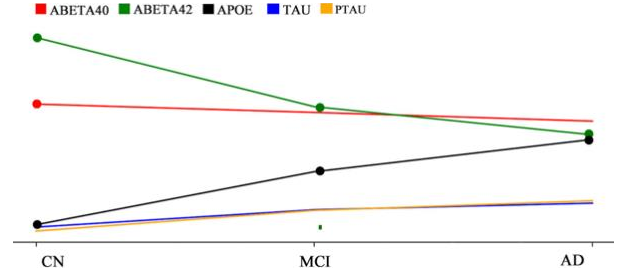
Note: values of each index in the figure have been normalized, and the difference in vertical coordinates has no practical reference value. It mainly refers to the change of a single index.

Fig. 7. Breakdown Chart of Biological Detection Indicators Change

TABLE V. COMPARISON AMONG THE MULTI-CLASSIFICATION OF PERFORMANCE USING DIFFERENT TYPES OF CLINICAL INDICATOR

Data	Method	Accuracy
DEMOGRAPHIC-2	D-TREE	38.3%
	KNN	31.7%
NEUROPSYCHOLOGY-4	D-TREE	83.3%
	KNN	81.7%
BIOLOGY-5	D-TREE	43.3%
	KNN	45.0%
MERGE-28	D-TREE	78.3%
	KNN	75.0%
MERGE-11	**D-TREE**	**85.0%**
	KNN	**78.3%**

TABLE VI. MULTI-CLASSIFICATION PERFORMANCE COMPARISON WITH DIFFERENT COMBINATIONS OF IMAGES AND INDICATORS

Data	Accuracy
TOP-MRI	75.0%
CLINICAL	84.4%
MRI + CLINICAL	76.2%
TOP-MRI + CLINICAL + CCA	**86.7%**

B. Evaluation of indicator feature selection module

In this module, we use the CfsSubsetEval evaluator to evaluate the classification capabilities and the redundancy of each clinical indicator. This selected indicator is needed to compose clinical feature vector.

Table 5 shows the comparison among multi-classification performance of model using different types of clinical indicators. The data of 11 indicators includes 3 types (demographic, neuropsychology, biology). Performance of classification by only using neuropsychological assessment has almost similar performance compared to 11 merged indicators and slightly better in KNN method. However, using neuropsychological assessment alone is subjective.

C. Evaluation of feature vector fusion module

In this module, image feature vector and clinical feature vector are fused by using linear fusion method of multi-source data based on canonical correlation analysis (CCA). Table 6 shows the comparison of multi-classification performance among different combinations based on TOP-MRI images and clinical indicators. The experimental results show that performing linear fusion on TOP-MRI images and clinical indicators based on CCA gives the best performance.

TABLE VII.   COMPARING MULTI-CLASSIFICATION PERFORMANCE
OF VARIOUS MULTI-CLASSIFICATION DIAGNOSTIC ALGORITHMS

Author	Data	Method	Accuracy
TONG T[12]	MRI + CLINICAL	NGF	53.8%
ZHU X [21]	MRI + PET	SDFS	61.1%
ALTAF T [11]	WHOLE MRI + CLINICAL	FF	75.0%
THIS PAPER	**TOP-MRI + CLINCIAL**	**OUR METHOD**	**86.7%**

D. Evaluation of multi-classification algorithm

Table 7 shows the performance comparison between multi-classification diagnosis model proposed by other papers and multi-classification diagnosis model of fused images and indicators proposed in this paper. The accuracy of Tong T proposed graph-based non-linear fusion method (NGF) using CSF and genotype fusion of MRI, PET and clinical data was 53.8% [12]. The accuracy of Zhu X proposed method of sparse discriminant feature selection (SDFS) using MRI and PET images as experimental data was 61.1% [21]. In Altaf T proposed method by fusing the features of MRI images and other clinical data (FF) [11], the classification accuracy of fusion of whole brain images and clinical data was 75%. Compared with the multi-classification diagnosis model of Alzheimer's disease proposed in other papers, the accuracy of the model proposed in this paper can reach 86.7%. The experimental results verify that the multi-classification diagnosis model of Alzheimer's disease proposed in this paper is effective.

V. CONCLUSION

In this paper, we have presented a new multi-classification algorithm using linear fusion with TOP-MRI images and clinical indicators to diagnose AD. Experiments show that our method has high accuracy and effective in performing multi-classification diagnosis for AD.

Influence of extraction position of three orthogonal planes on classification performance can be explored more in the future. The sensitivity and specificity of indicators for differentiating types of diseases and improving the classification accuracy of AD can also be studied.

REFERENCES

[1] Wang Yutong, Xuan Zhidong. Progress in Epidemiology of Alzheimer [J]. Chinese Journal of Practical Neurological Diseases,2015,18(20):118-119.

[2] Brookmeyer R, Johnson E Ziegler-Graham K et al. Forecasting the global burden of Alzheimer's disease[J]. Alzheimer's and Dementia,2007,3(3):186-191

[3] Dallas P S, Cara L R,Naveed S A review of epidemiological evidence for general anesthesia as a risk factor for Alzheimer's disease[J].Progress in Neuro –Psychopharmacology & Biological Psychiatry,2013(47):122-127

[4] Wee C Y, Yap P T, Zhang D, et al. Identification of MCI individuals using structural and functional connectivity networks[J]. Neuroimage, 2012, 59(3):2045-2056.

[5] S. Wang, Y. Zhang, G. Liu, P. Phillips, T.-F. Yuan, Detection of Alzheimer's disease by three-dimensional displacement field estimation in structural magnetic resonance imaging, J. Alzheimer's Dis. 50 (1) (2016) 233–248.

[6] T. Altaf, S.M. Anwar, N. Gul, N. Majeed, M. Majid, Multi-class Alzheimer disease classification using hybrid features, Future Technologies Conference, IEEE (2017) 264–267.

[7] Sarwinda D, Bustamam A. 3D-HOG Features –Based Classification using MRI Images to Early Diagnosis of Alzheimer's Disease[C]// 2018 IEEE/ACIS 17th International Conference on Computer and Information Science (ICIS). IEEE Computer Society, 2018.

[8] Yuanyuan Chen, Haozhe Jia, Zhaowei Huang, Yong Xia: Early Identification of Alzheimer's Disease Using an Ensemble of 3D Convolutional Neural Networks and Magnetic Resonance Imaging. BICS 2018: 303-311

[9] Karasawa H, Liu C L, Ohwada H. Deep 3D Convolutional Neural Network Architectures for Alzheimer's Disease Diagnosis[J]. 2018.

[10] Platero C, Tobar M C. Combining a Patch-based Approach with a Non-rigid Registration-based Label Fusion Method for the Hippocampal Segmentation in Alzheimer's Disease[J]. Neuroinformatics, 2017, 15(2):165-183.

[11] Altaf T, Anwar S M, Gul N, et al. Multi-class Alzheimer's disease classification using image and clinical features[J]. Biomedical Signal Processing and Control, 2018, 43:64-74.

[12] Tong T, Gray K, Gao Q, et al. Multi-Modal Classification of Alzheimer's Disease Using Nonlinear Graph Fusion[J]. Pattern Recognition, 2016, 63:171-181.

[13] Gu, J., Wang, Z., Kuen, J., Ma, L., Shahroudy, A., Shuai, B., Liu, T., Wang, X., Wang, L., Wang, G. and Cai, J., 2015. Recent advances in convolutional neural networks. arXiv preprint arXiv:1512.07108.

[14] Chen Yi, Zhang Baorong. Advances in core biomarkers related to Alzheimer's disease [J]. life sciences,2014,26(01):2-8.

[15] Johannes, Schröder,Elmar, Kaiser, Peter, Schönknecht,Aoife, Hunt,Philipp A, Thomann,Johannes, Pantel,Johannes, Schröder.[CSF levels of total tau protein in patients with mild cognitive impairment and Alzheimer's disease].[J].Zeitschrift fur Gerontologie und Geriatrie,2008,41(6):497-501.

[16] Hsiung G Y R, Sadovnick A D, Feldman H. Apolipoprotein E epsilon4 genotype as a risk factor for cognitive decline and dementia: data from the Canadian Study of Health and Aging [J]. Cmaj, 2004, 171(8):863-867.

[17] Glenner GG, Wong CW. Alzheimer's disease: initial report of the purification and characterization of a novel cerebrovascular amyloid protein [J]. Biochemical and Biophysical Research Communications, 1984, 120(3):885-890.

[18] Sun Quansen, Zeng Sheng-gen, Wang Ping-an, et al. Canonical correlation analysis theory and its application in feature fusion [J]. Journal of Computer Science, 2005, 28(9).

[19] Liu M, Zhang D, Adeli E, et al. Inherent Structure-Based Multiview Learning with Multitemplate Feature Representation for Alzheimer's Disease Diagnosis[J]. IEEE Transactions on Biomedical Engineering, 2016, 63(7):1473-1482.

[20] Zhe X, Yi D, Tian L, et al. Brain MR Image Classification for Alzheimer's Disease Diagnosis Based on Multifeature Fusion[J]. Computational and Mathematical Methods in Medicine, 2017, 2017:1-13.

[21] Zhu X, Suk H I, Shen D. Sparse Discriminative Feature Selection for Multi-class Alzheimer's Disease Classification[M]/ Machine Learning in Medical Imaging. 2014.

Deep Hashing with Large Batch Training for Cross-modal Retrieval

Xuewang Zhang[1,2] and Yin Zhou[1*]

[1]School of Software Engineering, Chongqing University of Posts and Telecommunications, Chongqing, China
[2]School of Microelectronics and Communication Engineering, Chongqing University, Chongqing, China
zhangxw@cqupt.edu.cn , 646031898@qq.com

Abstract—**Cross-modal hashing has attracted considerable attention as it can implement rapid cross-modal retrieval through mapping data of different modalities into a common Hamming space. With the development of deep learning, more and more cross-modal hashing methods based on deep learning are proposed. However, most of these methods use a small batch to train a model. Large batch training can get better gradients and can improve training efficiency. In this paper, we propose a deep hashing with large batch training (DHLBT), which uses large batch training and introduces orthogonal regularization to improve the generalization ability of our model. Moreover, we consider the discreteness of hash codes, therefore, we add the distance between hash codes and features to the objective function. Extensive experiments on three benchmarks show that our method achieves better performance than several existing hashing methods.**

Keywords: cross-modal hashing; large batch training; orthogonal regularization; the distance between hash codes and features

I. INTRODUCTION

With the rapid growth of multimedia data with different modalities and the increasing demands of users, cross-modal retrieval is becoming increasingly attractive. Modeling the relationship between different modalities is the key of cross-modal retrieval. The key challenge is a "heterogeneous gap" between different modalities, where the similarity among them cannot be measured directly [1]. However, cross-modal hashing methods can effectively bridge the gap [2, 3]. The hashing methods convert the high-dimensional features of data into a fixed-length hash code. Semantically similar data has similar hash codes. By XOR bitwise operation of hash codes, the similarity of data can be quickly obtained. Moreover, the storage space can be effectively reduced by only storing the hash codes of the data, instead of storing the high-dimensional features.

In recent years, deep learning has received good results in image processing and natural language processing. Therefore, more and more scholars have begun to apply deep learning technology to cross-modal hashing methods [2-7]. However, most of these methods use a small batch size to train the model. For example, in [2, 3, 5], the batch size is 64, and the maximum batch size is 128 [6]. However, when training a model in small batch size, the loss function cannot get a good gradient because of the limited number of samples in each batch, which makes the parameter update not good enough and affects the retrieval performance of the final trained model.

Large batch training which means using large batch size to train, e.g. 2048, 4096, or 8192, which is much larger than 64 or 128, can cover more samples each time when update parameters, resulting in better gradients and shorter training time per epoch. Therefore, more and more scholars in different fields are studying large batch training to get better performance [8-12], while no scholar has explored large batch training in the field of cross-modal hashing. So, it makes sense to study large batch training in the field of cross-modal hashing. However, increasing the batch size will cause the training extremely unstable [9], and then will easily lead to a "generalization gap" problem [13]. Orthogonal regularization will keep the norm of a matrix unchanged and lead the gradients to faithful propagation which will prevent the gradient from vanishing [10, 14]. In the field of image generation, Brock et al. [10, 14] introduced orthogonal regularization, which proves that orthogonal regularization achieves better performance. In multimodal retrieval, Wang et al. [15] also introduced orthogonal regularization, which reduces the redundancy of hash codes and improves performance. Moreover, hash codes are discrete. Relaxing the discrete learning problem of hash codes into continuous learning problem is the common practice of most cross-modal hashing methods. However, when continuous real value features of data are converted into hash codes, information loss will occur, which affects the performance so that the hash codes cannot represent the data well [5]. [5] adds the hash code to the objective function and learns the discrete hash code without relaxing. Inspired by these, we propose a method called deep hashing with large batch training (DHLBT). This method includes three major features, which are 1) Large batch size is used to train the model; 2) Orthogonal regularization is used to improve the generalization ability of the model; 3) Distances between hash codes and features are added to the objective function.

The rest of this paper is presented as follows. Section 2 introduces the proposed DHLBT approach. Section 3 shows the experiments. Finally, conclusions are made in Section 4.

II. DEEP HASHING WITH LARGE BATCH TRAINING

In this section, we will describe the details of our proposed method.

A. Notations

In this paper, we only consider image and text modal data. Therefore, there are two kinds of retrieval tasks in this paper: 1) text query image task and 2) image query text task. Assume that we have k training data, image modality is denoted as $I = \{I_i\}_{i=1}^k$, text modality is denoted as $T = \{T_i\}_{i=1}^k$. Then, we use $F = \{F_{I_i}, F_{T_i}\}_{i=1}^k$ to denote the low dimensional features of data, $q = \{q_{I_i}, q_{T_i}\}_{i=1}^k$ to denote the query data, $H = \{H_{I_i}, H_{T_i}\}_{i=1}^k$ to denote the hash codes of data, and $\|\cdot\|_{Frobenius}$ to denote the Frobenius norm of a matrix, respectively.

B. Network structure

Many cross-modal hashing methods based on deep learning, e.g. SCH-GAN [2], use convolutional neural networks (CNN) to extract the features of images as input values for training. In this paper, we use VGG-19 [16] to extract the features of the images and encode them as hash codes through two fully-connected layers. While for texts, the texts are represented by the bag-of-words (BoW) features and are also encoded into hash codes through two fully-connected layers. The whole DHLBT model is shown in Fig. 1.
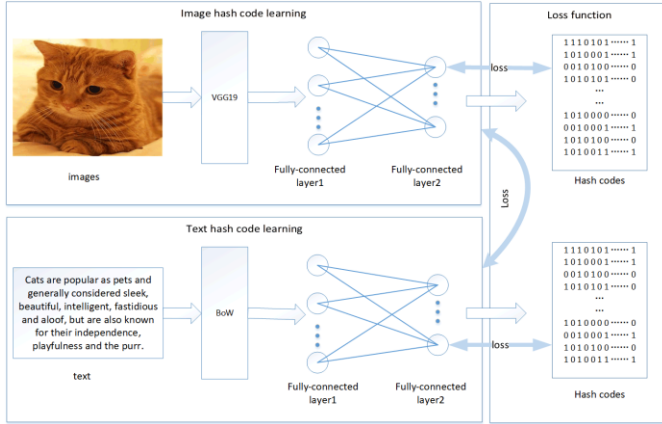
Figure 1. The framework of our DHLBT model.

C. Feature Learning Part

We firstly map the extracted image or text features to a common space through the fully-connected layer1 in the Fig. 1, then obtain the low-dimensional features through the fully-connected layer2 in the Fig. 1. The activation functions for the fully-connected layer1 and the fully-connected layer2 are tanh function and sigmoid function, respectively. The process can be represented as:

$$F = sigmoid(W_{c_2}(\tanh(W_{c_1}f + B_{c_1}) + B_{c_2}))$$ (1)

where W are the weights, B is the bias, c_1 denotes the fully-connected layer1, c_2 denotes the fully-connected

layer2. f denote the input value of VGG-19 [16] features of images or BoW features of texts. The low-dimensional features of images F_I and the low-dimensional features of texts F_T have the same shape, which allows us to measure the similarity between them. The hash code length is also the same as the dimension of the low-dimensional features so that the low-dimensional features F can be directly mapped to the hash codes H by the threshold function:

$$H = \begin{cases} 1, & if \ F \geq 0.5 \\ 0, & if \ F < 0.5 \end{cases}$$ (2)

D. Hashing Objectives

Our objective function is mainly divided into three parts, which are: 1) the distance between the features of the images F_I and the features of the texts F_T, 2) the distance between the features F and the hash code H, and 3) the regularization items of W and B. The image query text task and the text query image task are symmetric. Therefore, we take the text query image task as an example to show the objective function in the following parts.

The distance between F_I and F_T:

$$D_{q_{Ti}I^+_i} = \left\| F_{q_{Ti}} - F_{I^+_i} \right\|_2^2$$ (3)

$$D_{q_{Ti}I^-_i} = \left\| F_{q_{Ti}} - F_{I^+_i} \right\|_2^2$$ (4)

where D denotes distance, I_i^+ denotes semantically similar image and I_i^- denotes semantically dissimilar image with text query q_{Ti}. $D_{q_{Ti}I^+_i}$ are the distance between I_i^+ and q_{Ti}. $D_{q_{Ti}I^-_i}$ are the distance between I_i^- and q_{Ti}. We use a margin-based hinge loss function to measure the loss, which is shown below:

$$L_1 = \frac{1}{n}\sum_i^n \max(0, \beta + D_{q_{Ti}I^+_i} - D_{q_{Ti}I^-_i})$$ (5)

where β is a margin parameter between $D_{q_{Ti}I^+_i}$ and $D_{q_{Ti}I^-_i}$, and β is an adjustable hyper-parameter. n is the number of triplet (q_{Ti}, I_i^+, I_i^-). While reducing the loss L_1, $D_{q_{Ti}I^+_i}$ will be reduced and $D_{q_{Ti}I^-_i}$ will be increased simultaneously. This also conforms to the principle that small distance between semantically similar data and the large distance between semantically dissimilar data. In the process of training optimization, we intend to decrease the value of $D_{q_{Ti}I^+_i}$ and increase the value of $D_{q_{Ti}I^-_i}$ simultaneously. Therefore, the optimization process can be transformed into a binary classification problem, and then, we apply sigmoid cross-entropy as the loss function on it. The sigmoid cross-entropy formula for binary classification problem is shown below:

$$loss = -[z\ln(sigmoid(x)) + (1-z)\ln(1-sigmoid(x)))]$$
$$s.t. \quad z \in \{0,1\} \tag{6}$$

where x represent a input value, and it can be assigned by either $D_{q_{Ti}I_i^+}$ or $D_{q_{Ti}I_i^-}$. z denotes a target value. For $D_{q_{Ti}I_i^+}$, we want $D_{q_{Ti}I_i^+}$ as small as possible, that is, let $z = 0$, bring it into (6), as shown in equation (7):

$$loss_1 = -\ln(1 - sigmoid(D_{q_{Ti}I_i^+})) = \ln(1 + e^{D_{q_{Ti}I_i^+}}) \tag{7}$$

For $D_{q_{Ti}I_i^-}$, we want $D_{q_{Ti}I_i^-}$ as large as possible, that is, let $z = 1$, bring it into (6), as shown in equation (8):

$$loss_2 = -\ln(sigmoid(D_{q_{Ti}I_i^-})) = \ln(1 + e^{-D_{q_{Ti}I_i^-}}) \tag{8}$$

By combining equation (7) and (8), we have our second loss item:

$$L_2 = \frac{1}{n}\sum_i^n (loss_1 + loss_2) \tag{9}$$

The distance between F and H :

Hash codes are discrete, and information loss will occur in the process while converting real value features F to hash codes H :

$$D_{H_{q_{Ti}}F_{q_{Ti}}} = \left\| H_{q_{Ti}} - F_{q_{Ti}} \right\| \tag{10}$$

$$D_{H_{I_i}F_{I_i}} = \left\| H_{I_i^+} - F_{I_i^+} \right\| + \left\| H_{I_i^-} - F_{I_i^-} \right\| \tag{11}$$

where $D_{H_{q_{Ti}}F_{q_{Ti}}}$ denotes the distance between the low-dimensional features $F_{q_{Ti}}$ of text query q_{Ti} and hash codes $H_{q_{Ti}}$ of text query q_{Ti}. $D_{H_{I_i}F_{I_i}}$ denotes the distance between the low-dimensional features F_I of images I and hash codes H_I of images I. The following loss function can be obtained:

$$L_3 = \frac{1}{n}\sum_i^n (D_{H_{q_{Ti}}F_{q_{Ti}}} + D_{H_{I_i}F_{I_i}}) \tag{12}$$

In the optimization process, the loss function will make hash codes more and more close to the features and will reduce the information loss caused by the conversion process from the features to hash codes.

The regularization items of W and B :

Large batch training has low stability while training. To minimize the negative effect of the problem, we introduce the orthogonal regularization as the penalty term of W. For B, we still use L2 regularization as a penalty term. The loss item is as follows:

$$L_4 = \theta \left\| W^{transpose}W - I_{identity} \right\|_{Frobenius}^2 + \omega \left\| B \right\|_{Frobenius}^2 \tag{13}$$

where $W^{transpose}$ is the transpose of the weight matrix W and $I_{identity}$ is an identity matrix. B denotes the bias. θ and ω denotes the hyper-parameters.

By combining L_1, L_2, L_3 and L_4 together, we can get the full objective:

$$\min L = L_1 + \lambda L_2 + \gamma L_3 + L_4 \tag{14}$$

where λ and γ denote adjustable hyper-parameters.

We also take the text query image task as an example to show the training process of our method in Algorithm 1.

Algorithm 1 Training Process of DHLBT

Input: training data I, T
Output: weights W and bias B
1: **initialize**: Randomly initialize W and B, the batch size is b and the number of training epochs is e;
2: **for** $epoch = 0, 1, 2, ..., e-1$ **do**
3:     **if** $epoch\%30 == 0$ **then**
4:         **for** $q_T = T_1, T_2, T_3, ..., T_k$ **do**
5:             Randomly sample m points from I^+ and m points from I^- to make up a triplet set (q_T, I^+, I^-) as training data.
6:         **end for**
7:     **end if**
8:     **for** $step = 1, 2, ..., \lceil k*m/b \rceil$ **do**
9:         Train network and update parameters W and B by equation (14);
10:     **end for**
11: **end for**

III. EXPERIMENTS

In this section, we evaluate the performance of DHLBT on two datasets, and compare the result with several current state-of-the-art methods.

A. DATASETS

We use 2 datasets for experiments: Wikipedia [17] and MIRFlickr [18], which are widely used public datasets in cross-modal hashing. And to evaluate this method more fully, we added a larger data set NUS-WIDE [19] for experiments.

Wikipedia dataset [17] is a popular dataset which consists of 2866 text/image pairs divided into 10 categories. Following [2], Wikipedia dataset is separated into two parts: 1) a training data of 2173 pairs which are also used as the retrieval database and 2) a query set of 693 pairs. Each image is represented by 4096 deep features extracted by the fc2 layer of 19-layer VGGNet [16] from Keras applications, and each text is represented as a 1000-dimensional BoW vector.

MIRFlickr dataset [18] contains 25000 images that are collected from the Flickr website and they are annotated with some of 24 provided labels. Each image is described with some textual tags. Therefore, each instance is a text-image pair. Following [2, 20], firstly, we preprocess raw tags of these images by removing punctuations and stop words. Then,

we count the number of times for each word appeared in these tags. We only keep words that appeared at least 20 times and add them to the vocabulary of BoW. Furthermore, we remove instances that do not contain the word of the vocabulary and that do not have textual tags or labels. We take 5% of instances in each category as the query set and the rest of the instances as the retrieval database. In addition, we sample 5000 data pairs from the retrieval database as the training data. Each image is represented by 4096 deep features extracted by the fc2 layer of 19-layer VGGNet from Keras applications, and each text is represented as a 1386-dimensional BoW vector.

NUS-WIDE dataset [19] contains 269648 images that are collected from the Flickr website and they are annotated with some of 81 provided labels. Each image is described with some textual tags. Therefore, each instance is a text-image pair. We select the 10 most common labels and the corresponding 186577 text-image pairs. We take 2000 of pairs in each category as the query set and the rest of the pairs as the retrieval database. In addition, we sample 5000 data pairs from the retrieval database as the training data. Each image is represented by 4096 deep features extracted by the fc2 layer of 19-layer VGGNet from Keras applications, and each text is represented as a 1386-dimensional BoW vector. Table 1 shows the number of samples in each set intuitively.

TABLE I.　　STATISTICS OF TWO BENCHMARK DATASETS

	Wikipedia	MIRFlickr	NUS-WIDE
Dataset Size	2866	20819	186577
Training Set	2173	5000	5000
Query Set	693	1041	2000
Retrieval Set	2173	19778	186577
labels	10	24	10

B. EVALUATION PROTOCOL

We perform two kinds of retrieval tasks for each dataset: 1) retrieving text by image query, termed image→text; and 2) retrieving image by text query, termed text → image. Following [2], we utilize Hamming ranking to evaluate DHLBT and compared the result with the other state-of-the-art methods. Specifically, we first obtain the hash codes of images and texts, and then compute the Hamming distance between query with all the retrieval database. After ranking the Hamming distance list, we use 2 widely used assessment standards to evaluate the retrieval performance, which are shown below:

1) Mean Average Precision (MAP): The mean of all queries' average precisions (AP) called MAP. $AP = \frac{1}{R}\sum_{k=1}^{n}\frac{R_k}{k}\times rel_k$ is the definition of AP where R is the amount of the related data in the retrieval database, n is the amount of retrieval database, R_k is the amount of the related data in the top k ranks of the Hamming distance ranking list, and rel_k is an indicator of relevance of the Hamming distance ranking list which is set to 1 if the data at k-th position is related and 0 elsewise.

2) Precision Recall curve (PR-curve): The precision at the certain recall of the Hamming distance ranking list, that often evaluates the performance of retrieval.

C. BASELINES AND IMPLEMENT DETAILS

We compare two non-deep learning methods: SePH [20] and GSPH [21], which are both supervised methods. For SePH and GSPH, they are kernel-based methods and both of them achieved best results by using KLR which respectively created in two ways: 1) k-means algorithm and 2) random sampling. So, for these two hashing methods, we use KLR to learn hash function and create kernel by using k-means algorithm (klr+k) and random sampling (klr+r). In addition, we also compare our methods to three state-of-the-art deep learning-based methods, including SCH-GAN [2], UGACH [3] and DCMH [5]. SCH-GAN is a semi-supervised method. UGACH is an unsupervised method and DCMH is a supervised method. In all experiments, two modal data of image and text are used. When the data of one modal is used as the query set, the data of the other modal is used as the retrieval set. Source codes of all methods are kindly provided by the corresponding authors. For the parameters mentioned in all methods, we directly adopt the original parameter settings used in their codes. For an objective comparison between different methods, we use the same image and text features as input data features for all compared methods. Specifically, for Wikipedia and NUS-WIDE datasets, we use the 4096 deep features extracted by the fc2 layer of 19-layer VGGNet from Keras applications for images, and 1000-dimensional bag-of-words features for texts; For MIRFlickr dataset, we use the 4096 deep features extracted by the fc2 layer of 19-layer VGGNet from Keras applications for images, and 1386-dimensional bag-of-words features for texts. For DCMH, which is an end-to-end method, we add an experiment which directly uses original image features as the input value of the image network. $DCMH_{vgg19}$ and $DCMH_{original}$ denote these two versions of DCMH, respectively. For our method, we set $\lambda = 0.01$, $\gamma = 0.01$, $\theta = 0.0001$ and $\omega = 0.01$. Similar to [2, 3], for each data, the corresponding data is selected to form 4 triplets for training, that is, m in Algorithm 1 is set to 4. So, there are $2173*4 = 8692$ triplets for Wikipedia dataset, $5000*4 = 20000$ triplets for MIRFlickr and NUS-WIDE datasets. The batch size is set to 8192. And the learning rate for Wikipedia dataset is 0.08, the learning rate for MIRFlickr dataset is 0.016 and the learning rate for NUS-WIDE dataset is 0.016. The hash code bits are 16, 32, and 64, and β is 6, 8, and 10, respectively. We implement the proposed DHLBT by Tensorflow applications. All the experiments conducted on a server with hardware of NVIDIA GTX 1080Ti graphic card, Intel(R) Xeon(R) E5-2620 v4 2.10GHz CPU, 128 GB memory. The model was built by Python3.5.2 and Tensorflow 1.11.0.

D. EXPERIMENTAL RESULTS

We demonstrate the MAP scores of all methods on MIRFlickr, Wikipedia and NUS-WIDE datasets in Table 2, Table 3 and Table 4. From the result, it can be observed that our method achieves the best retrieval accuracy at 32-bit and 64-bit hash code length over all datasets. In general, compared with the second-best method SCH-GAN, in the task of image query text, our method is about 1.8%, 8.2% and 1.1% higher on MIRFlickr, Wikipedia and NUS-WIDE datasets, respectively. And in the task of text query image,

our method is about 1.3%, 2% and 0.2% higher on MIRFlickr, Wikipedia and NUS-WIDE datasets, respectively. This is mainly because we use large batch size to train the model which can get better gradients and use orthogonal regularization to improve the generalization ability of our model. And it is also because the distance between the hash codes and the features of data is added to the loss function which makes the hash codes are more realistic to represent the features of data. From the results, we can see that the hash code length has a remarkable impact on the MAP scores. For 16-bit hash code, the length is not enough to get sufficient information. Although our method has achieved the best results on Wikipedia dataset, it is only the second-best on MIRFlickr and NUS-WIDE datasets, indicating that the hash code length has a certain impact on MAP scores.

TABLE II.   THE MAP SCORES ON MIRFLICKR DATASET

Methods	image→text			text→image		
	16	32	64	16	32	64
SePH$_{klr+r}$ [20]	0.7364	0.7367	0.7451	0.7486	0.7514	0.7573
SePH$_{klr+k}$ [20]	0.7377	0.7459	0.7467	0.7522	0.7595	0.7599
GSPH$_{klr+r}$ [21]	0.7279	0.7425	0.7541	0.7579	0.7693	0.7760
GSPH$_{klr+k}$ [21]	0.7374	0.7485	0.7584	0.7614	0.7729	0.7798
UGACH [3]	0.6100	0.6045	0.5848	0.6278	0.6029	0.6101
DCMH$_{original}$ [5]	0.7296	0.7363	0.7386	0.7639	0.7650	0.7703
DCMH$_{vgg19}$ [5]	**0.7433**	0.7527	0.7592	0.7669	0.7792	0.7837
SCH-GAN [2]	0.7203	0.7481	0.7609	0.7661	0.7851	0.7884
Ours	0.7410	**0.7571**	**0.7718**	**0.7822**	**0.7915**	**0.7953**

TABLE III.   THE MAP SCORES ON WIKIPEDIA DATASET

Methods	image→text			text→image		
	16	32	64	16	32	64
SePH$_{klr+r}$ [20]	0.5009	0.5287	0.5393	0.5508	0.5955	0.6190
SePH$_{klr+k}$ [20]	0.4997	0.5252	0.5413	0.5584	0.6009	0.6122
GSPH$_{klr+r}$ [21]	0.5064	0.5289	0.5320	0.5701	0.6001	0.6237
GSPH$_{klr+k}$ [21]	0.5117	0.5318	0.5390	0.5801	0.6036	0.6207
UGACH [3]	0.3332	0.3605	0.3688	0.3222	0.3323	0.3471
DCMH$_{original}$ [5]	0.4503	0.4506	0.4120	0.7419	0.7238	0.6940
DCMH$_{vgg19}$ [5]	0.4387	0.4698	0.4809	0.8279	0.8457	0.7927
SCH-GAN [2]	0.5207	0.5370	0.5076	0.8352	0.8351	0.8288
Ours	**0.5528**	**0.5712**	**0.5688**	**0.8426**	**0.8502**	**0.8572**

TABLE IV.   THE MAP SCORES ON NUS-WIDE DATASET

Methods	image→text			text→image		
	16	32	64	16	32	64
SePH$_{klr+r}$ [20]	0.6537	0.676	0.6792	0.6769	0.6857	0.6836
SePH$_{klr+k}$ [20]	0.6625	0.685	0.6848	0.6681	0.6875	0.6886
GSPH$_{klr+r}$ [21]	0.6703	0.6858	0.6961	0.676	0.6838	0.7028
GSPH$_{klr+k}$ [21]	**0.6746**	0.696	0.7049	0.6756	0.6986	0.7052
UGACH [3]	0.6231	0.6296	0.6293	0.6152	0.6116	0.6152
DCMH$_{original}$ [5]	0.6008	0.6419	0.6383	0.6439	0.6739	0.6709
DCMH$_{vgg19}$ [5]	0.6341	0.6552	0.6631	0.6803	0.699	0.7058
SCH-GAN [2]	0.6647	0.6909	0.7027	**0.6862**	0.7086	0.7128
Ours	0.6625	**0.7007**	**0.7179**	0.6848	**0.7091**	**0.7189**
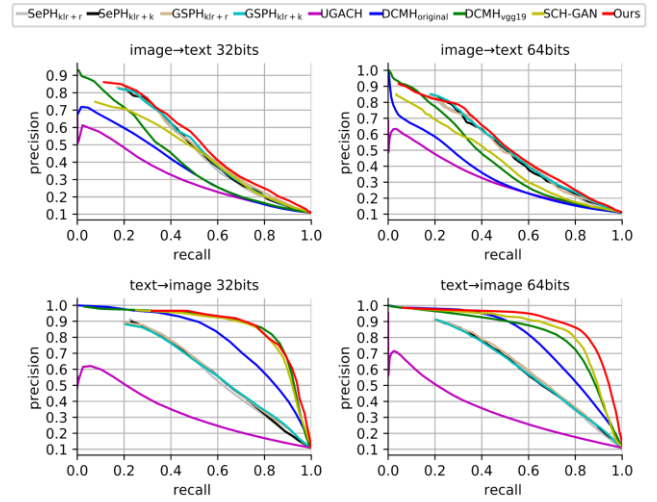
Figure 2.   The PR-curves on Wikipedia dataset.
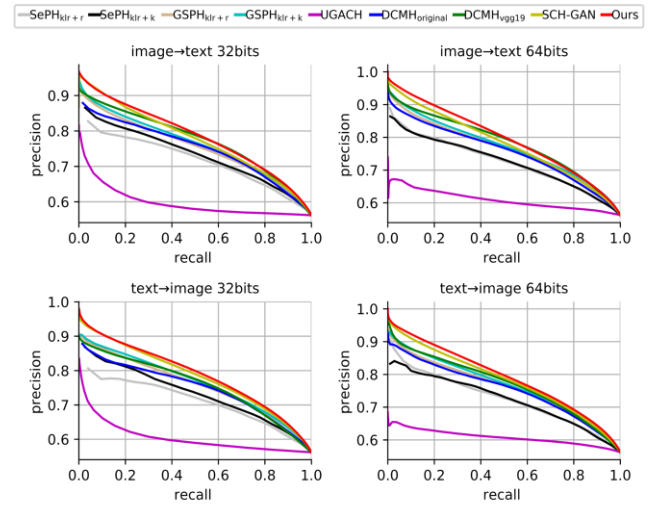
Figure 3.   The PR-curves on MIRFlickr dataset.
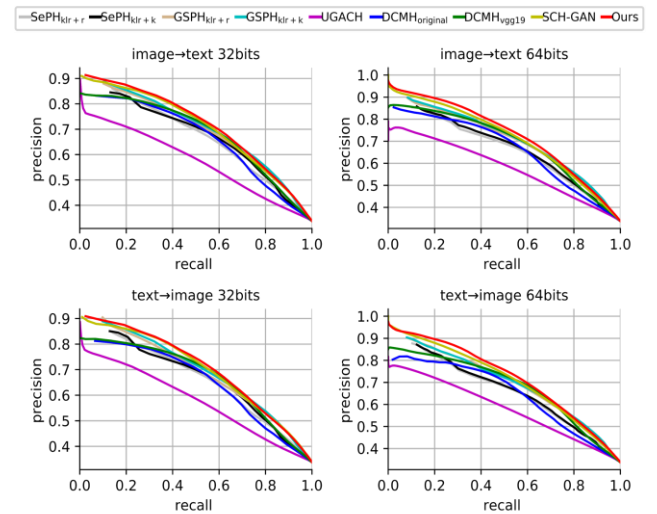
Figure 4.   The PR-curves on NUS-WIDE dataset.

Batch Size	Learning rate	Ortho.	MAP (image → text)			MAP (text → image)			The sum of MAP (image→text and text → image)			Time of each epoch		
			16	32	64	16	32	64	16	32	64	16	32	64
512	0.02	N	0.5406	0.5506	0.5612	0.8280	0.8524	0.8432	1.3686	1.4030	1.4044	9.3s	9.6s	10.2s
512	0.02	Y	**0.5588**	0.5629	0.5691	0.8312	**0.8576**	0.8479	1.3900	1.4205	1.4170	16.0s	16.3s	16.8s
2048	0.04	N	0.5509	0.5594	0.5699	0.8231	0.8462	0.8326	1.3740	1.4056	1.4025	8.1s	8.2s	8.5s
2048	0.04	Y	0.5559	0.5668	**0.5702**	0.8380	0.8486	0.8545	1.3939	1.4154	1.4247	10.9s	11.0s	11.3s
8192	0.08	N	0.5487	0.5563	0.5649	0.8217	0.8431	0.8327	1.3704	1.3994	1.3976	7.5s	7.5s	7.9s
8192	0.08	Y	0.5528	**0.5712**	0.5688	**0.8426**	0.8502	**0.8572**	**1.3954**	**1.4214**	**1.4260**	8.2s	8.3s	8.6s
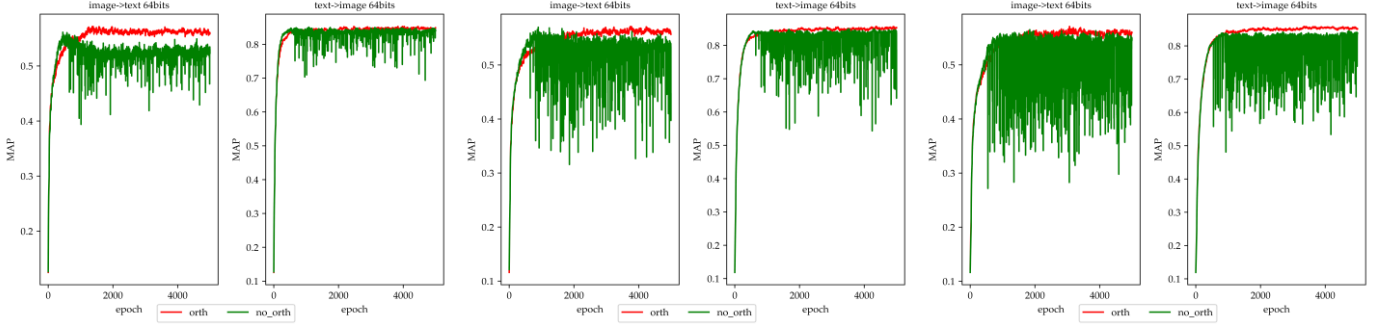
Figure 5.   The changes of MAP during the training process with the batch size of 512, 2048 and 8192, respectively.
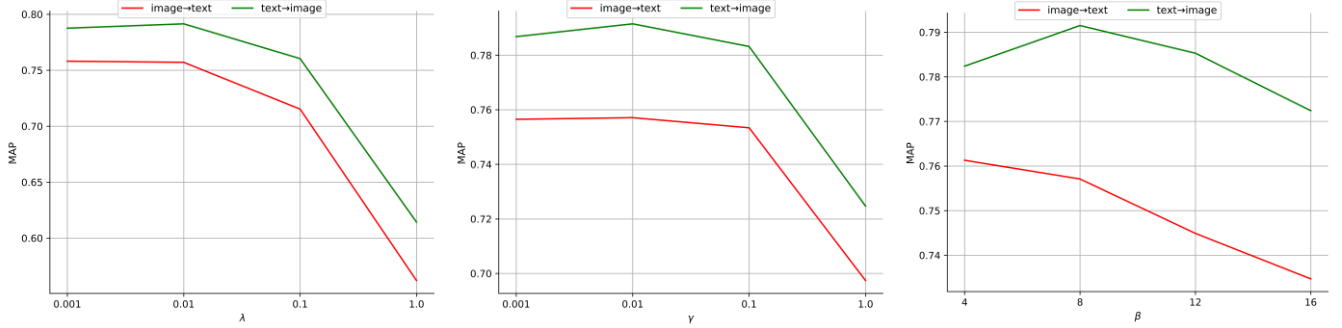
Figure 6.   The changes of MAP in different value of the hyperparameter λ , γ and β , respectively.

The PR-curves at 32- and 64-bit hash code length on Wikipedia, MIRFlickr and NUS-WIDE datasets are shown in Fig. 2, Fig. 3 and Fig. 4, respectively. The result shows that our method performs better than other state-of-the-art methods.

In addition, we also compare the effects of different batch sizes and orthogonal regularization on our model training on the Wikipedia dataset. The batch size is set to 512, 2048, and 8192, respectively. When increasing the batch size, it is necessary to increase the learning rate to ensure the convergence speed, so the learning rates are 0.02, 0.04, and 0.08, respectively. N means that orthogonal regularization is not used, and Y means orthogonal regularization is used. When orthogonal regularization is not used, we replace it with L2 regularization, that is, equation (13) is replaced by:

$$L_4 = \omega(\|w\|^2_{Frobenius} + \|B\|^2_{Frobenius}) \qquad (15)$$

We keep configuration for all parameter except the learning rate. The result of the comparison is shown in Table 5. When evaluating the performance, we need to consider not only the map of image → text, but also the map of text → image. Therefore, the map sum of text→image and image→text is given in Table 5. It can be seen that using orthogonal regularization can obviously achieve better performance when using same batch size, but the training time of each epoch will be increased. Extend the batch size will significantly increase training speed. Simply increasing the batch size without using orthogonal regularization does not achieve good performance. When the batch size is 8192 and the orthogonal regularization is used, we achieve the best performance.

We also analyze the training stability of our model. The changes of MAP at 64-bit hash code length during the training process with the batch size of 512, 2048 and 8192 on Wikipedia dataset are shown in Fig. 5. "orth" means that orthogonal regularization is used, and "no_orth" means

orthogonal regularization is not used. The result shows that the changes of MAP are more volatile with the increase of batch size. And the changes of MAP become stable and the model can get better performance when used the orthogonal regularization.

At last, we conduct sensitivity experiments for hyperparameter λ, γ and β. λ is set to 0.001, 0.01, 0.1 and 1.0, respectively. γ is set to 0.001, 0.01, 0.1 and 1.0, respectively. β is set to 4, 8, 12 and 16, respectively. When performing sensitivity experiments on one hyperparameter, other hyperparameters remain fixed. That is, we set λ=0.01 when performing sensitivity experiments on γ or β. We set γ=0.01 when performing sensitivity experiments on λ or β. We set β=8 when performing sensitivity experiments on λ or γ. Fig. 6 shows the changes of MAP at 32-bit hash code length in different value of the hyperparameter λ, γ and β on MIRFlickr dataset. The result shows that the changes of MAP are not volatile when the value of λ or γ is set to between 0.001 and 0.01. The changes of MAP are volatile in different value of the hyperparameter β. When the value of β is set to 8, we get the best MAP scores. For hyperparameter $\theta = 0.0001$ and $\omega = 0.01$, we follow [10] and [2], respectively, which are the best parameters selected by the corresponding authors through sufficient experiments. In order to reduce hyperparameters, we set λ and γ to the same value of ω, which is 0.01.

IV. CONCLUSIONS

In this paper, we propose a deep hashing with large batch training (DHLBT) for the cross-modal hashing retrieval. DHLBT is the cross-modal hashing which uses large batch training and uses orthogonal regularization to improve the generalization ability of our model. Moreover, the distance between hash codes and features is added to the objective function which makes hash codes to represent data more realistically. The effectiveness of DHLBT is demonstrated through the experiments on three widely-used datasets: Wikipedia, MIRFlickr and NUS-WIDE.

V. ACKNOWLEDGMENT

REFERENCES

[1] Y. Peng, X. Huang, and Y. Zhao, "An overview of cross-media retrieval: Concepts, methodologies, benchmarks, and challenges," IEEE Trans. Circuits and Syst. Video Technol., vol. 28, no. 9, pp. 2372-2385, Sep. 2018.

[2] J. Zhang, Y. Peng, and M. Yuan, "SCH-GAN: Semi-supervised cross-modal hashing by generative adversarial network," IEEE Trans. Cybern., vol. 50, no. 2, pp. 489-502. 2020.

[3] J. Zhang, Y. Peng, and M. Yuan, "Unsupervised generative adversarial cross-modal hashing," in Proc. 32nd AAAI Conf. Artif. Intell., New Orleans, Lousiana, USA, 2-7 February 2018, pp. 539-546.

[4] L. Wu, Y. Wang, and L. Shao, "Cycle-consistent deep generative hashing for cross-modal retrieval," IEEE Trans. Image Process., vol. 28, no. 4, pp. 1602 - 1612, Apr. 2019.

[5] Q. Y. Jiang, and W. J. Li, "Deep cross-modal hashing," in Proc. 30th IEEE Conf. Comput. Vis. Pattern Recognit., Honolulu, HI, USA, 21-26 July 2017, pp. 3232–3240.

[6] Z. Ji, W. Yao, W. Wei, H. Song, and H. Pi, "Deep multi-level semantic hashing for cross-modal retrieval," IEEE Access, vol. 7, pp. 23667-23674. 2019.

[7] G. Wu, Z. Lin, J. Han, L. Liu, G. Ding, B. Zhang, and J. Shen, "Unsupervised deep hashing via binary latent factor models for large-scale cross-modal retrieval," in Proc. 27th Int. Joint Conf. Artif. Intell., Stockholm, Sweden, 13-19 July 2018, pp. 2854-2860.

[8] P. Goyal, P. Dollar, R. Girshick, P. Noordhuis, L. Wesolowski, A. Kyrola, A. Tulloch, Y. Jia, and K. He (2017). "Accurate large minibatch SGD: Training imagenet in 1 hour." [Online]. Available: http://arxiv.org/abs/1706.02677

[9] Y. You, I. Gitman, and B. Ginsburg (2017). "Large batch training of convolutional networks." [Online]. Available: https://arxiv.org/abs/1708.03888

[10] A. Brock, J. Donahue, and K. Simonyan (2018). "Large scale GAN training for high fidelity natural image synthesis." [Online]. Available: https://arxiv.org/abs/1809.11096

[11] X. Jia, S. Song, W. He, Y. Wang, H. Rong, F. Zhou, L. Xie, Z. Guo, Y. Yang, L. Yu, T. Chen, G. Hu, S. Shi, and X. Chu (2018). "Highly scalable deep learning training system with mixed-precision: Training imagenet in four minutes." [Online]. Available: https://arxiv.org/abs/1807.11205

[12] Y. You, J. Hseu, C. Ying, J. Demmel, and C. J. Hsieh (2019). "Large-batch training for LSTM and beyond." [Online]. Available: https://arxiv.org/abs/1901.08256

[13] N. S. Keskar, D. Mudigere, J. Nocedal, M. Smelyanskiy, and T. P. T. Ping (2016). "On large-batch training for deep learning: Generalization gap and sharp minima." [Online]. Available: https://arxiv.org/abs/1609.04836

[14] A. Brock, T. Lim, J. M. Ritchie, and N. Weston (2016). "Neural photo editing with introspective adversarial networks." [Online]. Available: https://arxiv.org/abs/1609.07093

[15] D. Wang, C. Peng, M. Ou, and W. Zhu, "Deep multimodal hashing with orthogonal regularization," in Proc. 24th AAAI Conf. Artif. Intell., Buenos Aires, Argentina, 26-27 July 2015, pp. 2291-2297.

[16] K. Simonyan, and A. Zisserman (2014). "Very deep convolutional networks for large-scale image recognition." [Online]. Available: http://arxiv.org/abs/1409.1556

[17] J. C. Pereira, E. Coviello, G. Doyle, N. Rasiwasia, G. R. G. Lanckriet, R. Levy, and N. Vasconcelos, "On the role of correlation and abstraction in cross-modal multimedia retrieval," IEEE Trans. Pattern Anal. Mach. Intell., vol. 36, no. 3, pp. 521-35, Mar. 2014.

[18] M. J. Huiskes, and M. S. Lew, "The MIR flickr retrieval evaluation," in Proc. 1st ACM Int. Conf. Multimedia Inf. Retr., Vancouver, British Columbia, Canada, 30-31 October 2008, pp. 39-43.

[19] T.-S. Chua, J. Tang, R. Hong, H. Li, Z. Luo, and Y.-T. Zheng. "NUS-WIDE: A real-world web image database from national university of singapore" in Proc. of the 2009 ACM Int. Conf. on Image and Video Retr., Santorini, Fira, Greece, 8-10 July 2009, pp. 48-56.

[20] Z. Lin, G. Ding, M. Hu, and J. Wang, "Semantics-Preserving Hashing for Cross-View Retrieval," in Proc. 28th IEEE Conf. Comput. Vis. Pattern Recognit., Boston, MA, USA, 7-12 June 2015, pp. 3864-3872.

[21] D. Mandal, K. N. Chaudhury, and S. Biswas, "Generalized semantic preserving hashing for cross-modal retrieval," IEEE Trans. Image Process., vol. 28, no. 1, pp. 102-112, Jan. 2019.

Algebraic Higher-Abstraction for Software Refactoring Automation

Iaakov Exman and Alexey Nechaev
Software Engineering Department
The Jerusalem College of Engineering – JCE - Azrieli
Jerusalem, Israel
iaakov@jce.ac.il, alosha82@gmail.com

Abstract— **Software systems losing modularity along their life cycle require refactoring to restore their understandability. However, refactoring is very complex and expensive when done at the lower abstraction source program level. This paper's message is: refactoring is both simpler and mathematically rigorous when done at a higher-abstraction level. This work proposes a novel algebraic approach to refactoring, done at a higher-abstraction level, which is then back-translated to a refactored source program level. The algebraic approach, combining the Modularity Matrix and Laplacian Matrix representations of software, has two advantages over conventional source program refactoring. First, software higher-abstractions refactoring by a spectral approach is amenable to automation. Second, the algebraic representation is a reliable source of refactoring rules, with the potential of reaching a rule set finally leading to full automation. The refactoring technique is concisely analyzed and illustrated by case studies.**

Keywords: Software Modularity; Spectral Refactoring; Algebraic Higher-Abstraction level; Modularity Matrix; Laplacian Matrix; Refactoring Rule Set; Refactoring Automation.

I. INTRODUCTION

Refactoring of legacy program code is necessary to recover desirable qualities of software modularity: viz. to enable software understanding and maintainability by software engineers. Refactoring has often been done at source code level, less frequently at a higher model level.

This work proposes that refactoring done at a higher abstraction level of software is much more productive and rigorous than the usual lower level approaches.

The higher abstraction level consists of the Linear Software Models, an algebraic theory of modular software composition, in which software systems are represented by matrices, such as a Modularity Matrix and/or a Laplacian Matrix. Spectral refactoring applied to these matrices is mathematically rigorous and amenable to automation

Beyond the advantages of the higher model level for refactoring in practice, the algebraic approach is a reliable source of refactoring rules that may potentially lead to a complete automation of refactoring.

A. Overall Algebraic Higher-Abstraction Approach

The Algebraic Higher-Abstraction Refactoring is depicted in the right-hand-side of Fig. 1.
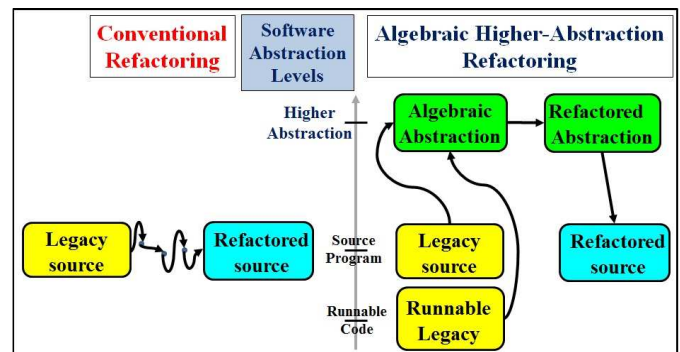
Figure 1. Algebraic Higher-Abstraction Refactoring – Conventional refactoring (left-hand-side of figure) works with complex source code. Algebraic Higher-abstraction Refactoring (right-hand-side of figure) climbs to a Higher-Abstraction level, where rigorous spectral tools resolve modules coupling in a simpler way. Its 3 stages are: 1st: source code translation into Algebraic Abstraction; 2nd: legacy abstraction modularized into a Refactored abstraction; 3rd: back-translation into Refactored source code. (All figures in color online).

Algebraic Higher-Abstraction Refactoring is simpler because higher-abstraction focuses on software architecture, ignoring source code clutter, irrelevant to modularity.

B. Algebraic Software Modularity

Within Linear Software Models, Modularity Matrix columns stand for *structors* – generalizing object-oriented programming classes – and rows stand for *functionals* – generalizing class methods. A 1-valued matrix element structor *provides* its functional, e.g. a class containing the declaration/definition of a method, usable by other classes. Otherwise, an element is zero-valued.

A schematic Modularity Matrix is seen in Fig. 2. It features:

- *Linear independent Matrix vectors* – true for structors among themselves and functionals among themselves;
- *Block-Diagonal Modules* – structors/functionals vectors of each module are orthogonal to other modules' vectors.

The matrix in Fig. 2 is a standard Modularity Matrix, i.e. square and without *outliers*. A Functional like F2 provided by two Structors (S2 and S3) is due to inheritance.

	S1	S2	S3	S4
F1	1	1	1	
F2	0	1	1	
F3	0	0	1	
F4				1

Figure 2. Schematic Modularity Matrix – It has 4 *structor* columns (S1 to S4), 4 *functional* rows (F1 to F4) and 2 block-diagonal modules (light blue background): a 3*3 upper-left block and a 1*1 lower-right block. Zero-valued elements outside modules are omitted for clarity.

A Laplacian Matrix is obtained from the Modularity Matrix through a bipartite graph [25]. Graph edges fit to Modularity Matrix 1-valued elements: for instance, the 1-valued (S2,F1) matrix element (Fig. 2) fits the graph edge from vertex S2 to vertex F1 (Fig. 3). Bipartite graphs have two vertex sets with edges linking only vertices in different sets.
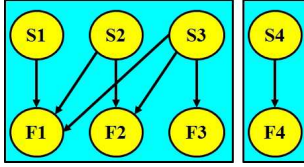
Figure 3. Bipartite Graph from Modularity Matrix in Fig. 2 – It has two vertex sets: an upper structors set (S1 to S4), and a lower functionals set (F1 to F4). Arrows pointing down mean that structors provide functionals. Rectangles (light blue) contain vertices belonging to a given module, a connected component.

The Laplacian Matrix [26] (in Fig. 4) is generated from the bipartite graph (in Fig. 3), according to equation (1):

$$L = D - A \qquad (1)$$

where L is the Laplacian matrix, D is the Degree matrix of the graph vertices and A is the Adjacency matrix of vertex pairs.

	F1	F2	F3	F4	S1	S2	S3	S4
F1	3				-1	-1	-1	
F2		2			0	-1	-1	
F3			1		0	0	-1	
F4				1				-1
S1	-1	0	0		1			
S2	-1	-1	0			2		
S3	-1	-1	-1				3	
S4				-1				1

Figure 4. Schematic Laplacian Matrix – This Laplacian is generated from the bipartite graph in Fig. 3 by equation (1). Its diagonal is the Degree matrix D (in green) displaying vertex' degrees of the Bipartite graph. The upper-right quadrant (identical to the Modularity Matrix with a minus sign) together with the lower-left quadrant is the negative of the graph Adjacency matrix A.

C. A Running Example

We introduce a running example, to clarify the notion of outlier and its relationship to refactoring.

Outliers are 1-valued matrix elements outside of any block-diagonal modules. Outliers cause coupling of pairs of modules, which require refactoring to decouple these modules.

Fig. 5 shows a Modularity Matrix of an actual software sub-system of Intellij IDEA [15]. This matrix has two block-diagonal modules, and one outlier element coupling these two modules. Coupling means that the outlier Structor S5 belongs to the lower-right module, while its Functional F1 belongs to the upper-left module. This is an intermediate matrix: one knows the outlier location, but it must still be refactored (see section V).

Structor → Functional ↓		Javac2	Javac2 #1	uiDesigner. Ant. Javac2	Nested Form Loader	Ant.Nested Form Loader
		S1	S2	S4	S3	S5
Compile( )	F1	1	1	1		1
Accept( )	F2	0	1	0		
uiDesigner.ant. Javac2<init>( )	F4	0	0	1		
getClassToBind Name( )	F3				1	1
AntNestedForm Loader<init>( )	F5				0	1

Figure 5. Intellij IDEA Modularity Matrix with single outlier – It has 5 *structor* columns (S1 to S5), 5 *functional* rows (F1 to F5) and 2 block-diagonal modules (light blue background): a 3*3 upper-left block and a 2*2 lower-right block. There is one outlier element (F1,S5) (hatched red background) coupling the two modules. Zero-valued elements outside modules are omitted for clarity.

D. Paper Organization

The remaining of the paper is organized as follows. Section II mentions related work. Section III describes the Algebraic Higher-Abstraction Refactoring. The refactoring software architecture of the Re-Factory System is detailed in Section IV. Section V illustrates and analyzes refactoring by means of a few case studies. Section VI concludes the paper with a discussion.

II. RELATED WORK

This is a very concise review of the Modularity literature, due to strict space limitations.

A. Linear Software Models

Linear Software Models, a rigorous linear algebra theory, were developed by Exman et al. (e.g. [7], [8]), to solve the problem of software system composition from sub-systems. Software modularization by spectral methods highlights outliers coupling modules. A procedure to improve software system design is described in [9]. The Perron-Frobenius theorem (e.g. Gantmacher [14]) is central for the Modularity Matrix theory.

Exman and Sakhnini [11] generate from the Modularity Matrix a Laplacian Matrix, which obtains the same modules as the Modularity Matrix, by similar spectral methods. The Fiedler theorem [2], [13] is central to the Laplacian theory. The Fiedler

eigenvector, fitting the lowest non-zero Laplacian eigenvalue, can be used to split too sparse modules and locate outliers.

B. Alternative Modularity Analysis

There are various less formal matrix techniques for modularity. Baldwin and Clark describe a Design Structure Matrix (DSM) in their "Design Rules" book [3]. DSM has been applied to many fields including software engineering (see e.g. Browning [5], Cai and Sullivan [6]). For alternative clustering techniques of software modules see Shtern and Tzerpos [22].

C. Automated Modularity Refactoring

Surveys of the multitude of papers dealing with software refactoring are found in [19], [20]. Fewer works strictly focus on automated refactoring (e.g. [23]). The work of Bavota [4] refactors software by rearranging Java packages, combining two "machine learning" methods. The paper by Zanetti [27] uses "networks theory", with probabilistic class relocation, depending on numbers of adjacent neighbors. An article by Abdeen [1] automatically reduces packages coupling and cyclic connectivity, using a "Genetic Algorithm", minimally modifying existing packages.

III. ALGEBRAIC HIGHER-ABSTRACTION REFACTORING

Our refactoring proposal, instead of dealing with a complex source program, or using specialized algorithms, climbs the software abstraction levels, and solves the problem in the Higher-Abstraction level with rigorous and general linear algebra. It returns to the source level the already refactored software system.

In the algebraic representation of software systems, the refactoring problem consists of recognizing each outlier which couples a pair of modules, and by relocating each outlier to a block-diagonal module, to decouple the pair of modules.

The approach essence is:

- **_Preserve overall functionality_** without change, while **_changing/creating structors_**.

A. 1st Stage: Generate Laplacian and its modules

The Algebraic Higher-Abstraction Refactoring starts from a Modularity Matrix obtained from classes/methods of a program source – the **SUD (S**oftware **U**nder **D**esign**)** – and/or its compiled code (see right-hand side of Fig. 1). The Modularity Matrix generates a Laplacian Matrix by the following steps:

- _extract a bipartite graph_ from the Modularity Matrix;
- _generate the Laplacian_ from bipartite graph, by eq. (1);
- _obtain module sizes and locations_ from the Laplacian eigenvectors, fitting zero-valued eigenvalues;
- _split sparse modules_ by the Laplacian Fiedler vector.

For the next stages (especially Back-Translation), the Algebraic Higher-Abstraction Refactoring saves the **SUD** source in a dedicated data structure: a three-columns table, whose columns are 1- functional declaration; 2- name of structor providing the functional; 3- functional implementation. The table length is the number of functionals in the program.

B. 2nd Stage: Matrices Modularization

Next, Modularity Matrix outliers are found and decoupled.

Algorithm 1 – Find-&-Decouple outliers

Input: Laplacian Matrix and its modules (from 1st Stage)

Preparation:
  Module Info Vector – saves modules size and location;
  Modularity Matrix – insert Laplacian module boundaries into the Modularity Matrix;

Find Outliers: – by comparing Modularity Matrix with Module Info Vector;
  Create **Matrix Outlier Vector** – with names of structor and functional containing outliers;

Decouple Outliers: – Create _new Columns/Rows for each outlier (group)_ **–** between coupled modules;
  Single Outlier Relocation – to new column/row element;
  Outliers Group Relocation – to new columns/rows group.

Output: Refactored Modularity Matrix & outliers.

C. 3rd stage: Back-Translation to Refactored Source

Back-Translation demands challenging actions:

a) Software matrices translation – attempting to foresee every translation problem from a refactored matrix (e.g. insertion of attribute values) into source code;

b) Gradual Collection of Generic Refactoring Rules – instead of ad-hoc decisions, obtain a refactoring rule set, the basis of a future potentially complete Algebraic Higher-Abstraction automation refactoring.

The Back-Translation pseudo-code is shown next.

Algorithm 2 – Back-Translation to Refactored Source Program

Input: Refactored Modularity Matrix (from Algorithm 1)

Preparation: Create **new source files** – e.g. new .java files;
  Insert **untouched Structor Functionals** into new files – structors from which no functionals were decoupled;
  Insert **Decoupled Functionals (DF)** into new files – by "Module Info Vector" and "Matrix Outlier Vector";

Loop: (on all DFs)   Search for **DF Calls** –
  Get **resources used by DF** – to be assembled;
  If (**trivial resources**) – attribute assignments as x=5, copy them from the original DF file, to the new file;
  Else if (non-trivial resources) – as another Function call, copy the "_Calling-line_" from the original DF file to the new file and adjust the relevant path, if needed;
  In any case (including no resources) – write new method call for the DF, in any class from which the DF function is called;
  Possibly use consumer matrices [12]–to find DF calls;

Output: Refactored Source Program & its Modularity Matrix (for the software engineer convenience).

Some relevant issues are:

– *Why create new source files instead of saving old files?*

Since the structors having decoupled functionals ought to be necessarily composed anew, it is desirable to have a single uniform way of saving the original source, i.e. by the three-columns table that was created in the 1st Stage (see sub-section A "*1st Stage: Generate Laplacian and its modules*").

D. Conjecture: Finite Refactoring Rule Set

A cardinal issue for the Algebraic Higher-Abstraction refactoring potential automation is whether the refactoring rule set is finite. Therefore we state the following conjecture.

Conjecture 1 – Algebraic Higher-Abstraction Finite Refactoring Rule Set

The number of refactoring rules in the Algebraic Higher-Abstraction Software Refactoring is finite and small.

The plausibility arguments for this conjecture are
1) The *number of refactoring types* is finite; these include:

- Single outlier – just a single unit to be relocated;
- Outliers Group – a finite small group of units to be relocated, of the order of the sub-matrix size;
- Outliers Array (sequential data) and its access functions – finite group of the order of the array size.

2) The *number of refactoring checking cases for each type referring to a group* is finite; these include:

- Direct matrix check – of the order of a sub-matrix size;
- Saved source Check – existence of different specific implementations, e.g. in an inheritance case with overridden function, of the order of the group size;
- Conceptual semantic check – where algebraic check alone is not sufficient, of the order of group size;
- Specific problems after decoupling – e.g. appearance of empty classes, of the order of the matrix dimension.

Some of these refactoring types and cases will be illustrated in section V of this paper.

IV. RE-FACTORY SYSTEM: SOFTWARE ARCHITECTURE AND IMPLEMENTATION

Re-Factory is a prototype software system designed and implemented to test case studies and the results of this work.

A. Re-Factory System: Software Architecture

The software architecture of the **Re-Factory** System, schematically shown in fig. 6, is composed of four sub-systems:

a- **Modulaser** – based upon an up-to-date version of this previously existing software tool [10], written in Java; inputs .class or .jar files and outputs their Modularity Matrix. In principle this tool may be adapted to deal with programs in other Object Oriented languages;
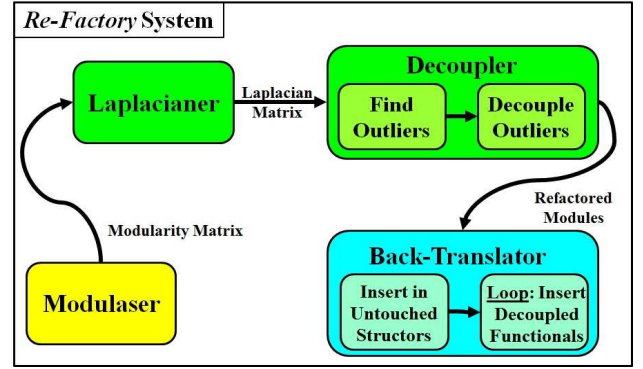
Figure 6. **Re-Factory** System Software Architecture – It has four sub-systems: a- an up-to-date version of the *Modulaser* tool outputs a Modularity Matrix; b- the *Laplacianer* outputs the corresponding Laplacian Matrix; c- the *Decoupler* finds and decouples outliers, then outputs **SUD** module sizes and locations; d- the *Back-Translator* outputs the refactored **SUD** modules back-translated to the source level, done in two steps: 1- inserting functionals in untouched structors; 2- Loop inserting decoupled functionals in new files.

b- **Laplacianer** – new sub-system, added to the Modulaser, generates the Laplacian and its eigenvalues/eigenvectors from the Modularity Matrix. It was extended and tested by functions of various linear algebra API libraries.

c- **Decoupler** – this sub-system has two components. One finds outliers, by direct use of Fiedler eigenvectors [13]. The other one decouples outliers using the current set of Modularity Matrix refactoring rules. More design details will be provided in an extended version of this paper.

d- **Back-Translator** – this last sub-system is also designed with two components. One of them reconstitutes the untouched structors. The other one performs a loop inserting decoupled outliers and necessary resources composing new source files.

B. Re-Factory System: Implementation

The *Re-Factory* implementation throughout the system, adopted the Modulaser Java language for compatibility. This included some frequently used API linear algebra libraries, also in Java, to calculate Laplacian eigenvalues/eigenvectors:

- JAMA (A Java Matrix Package) [21];
- LA4J (Linear Algebra for Java) [17];
- JBLAS (Linear algebra for Java, based upon BLAS and LAPACK) [16].

V. CASE STUDIES: SINGLE AND GROUPS OF OUTLIERS

The Case Studies section illustrates and analyzes two refactoring case studies with diverse characteristics.

A. Single Outlier Refactoring

The 1st case study is a Javac2 compiler sub-system of the Intellij IDEA system [15]. This is an interesting case since the initial Modularity Matrix (Fig. 7) is puzzling: it is difficult to decide which the modules are and how many outliers are in this system. Only the Laplacian splitting resolves the puzzle.

Figure 7. Intellij IDEA Javac2 Modularity Matrix with outliers – It has two *potential* modules: one upper-left, another lower right (light blue background), whose actual sizes are not known yet. The potential modules are coupled by one or two outliers (F1,S4) and (F1,S5) (dark blue background). Coupling issues are resolved in this work by calculating the eigenvectors of the fitting Laplacian.

From the Modularity Matrix in Fig. 7 a Laplacian was generated. This Laplacian has a single zero-valued eigenvalue, thus a single whole matrix module. The Laplacian eigenvectors are shown in Fig. 8: the single module eigenvector and the Fiedler eigenvector.

Figure 8. Intellij IDEA Javac2 Laplacian eigenvectors – The upper row has ten vertex indices – functionals and structors – of the bipartite graph. The mid-row contains a single whole matrix module equal elements' eigenvector. The lower row shows the Fiedler eigenvector elements. It splits the single module into two smaller modules by the elements signs: negative (blue) and positive (green).

The Fiedler vector element signs split the Modularity matrix into two modules: upper-left of 3*3 size (F1, F2, F4, S1, S2, S4) and lower-right of 2*2 size (F3, F5, S3, S5). The unique outlier (F1,S5) is revealed outside both modules, as seen in the intermediate matrix (Fig. 5 in *Running Example* in section I).

Relocating the outlier (F1,S5) is now shown in Fig. 9. It has been moved to the newly created row/column (F6,S6) diagonal element. Please compare the neater refactored fig. 9 with fig. 7.

Figure 9. Refactored Intellij IDEA Javac2 Modularity Matrix – It has two modules: a 3*3 upper-left and a 2*2 lower right (light blue). The outlier (dark blue) has been relocated to a diagonal position (F6,S6) in the intersection of newly created column and row, in between the previously coupled modules. The previous (F1, S5) position is marked (hatched red).

Figure 10. Horizontal row case study – This matrix contains a single 8*8 big module, since the horizontal row F7 filled with 1-valued matrix elements couples all the smaller potential modules. The latter are five 1*1 diagonal modules and one 3*3 lower-right block-diagonal module (light blue background). The five horizontal matrix elements (dark blue hatched background) in row F7 are the source of the coupling problems to be solved.

B. Outlier Group Refactoring: Horizontal Row

The 2nd case study, the **horizontal row** illustrates an outlier group, with multiple usage of the same function. It is actually found in several software systems, e.g. the "Modulaser" [10] itself, and the "Tagger" software program [24]. This **horizontal row** example, seen in Fig. 10, may cause 3 potential problems:

a) **Non-implemented inheritance** – the five 1-valued row F7 matrix elements, from the left, are an outlier group to be relocated to the main diagonal. Yet, the 1-valued row of elements e.g. due to inheritance, may not be present in the original source code, except the parent class. For back-translation, an inherited but not overridden outlier function should be referred to the parent class.

b) **Need to check source code for overridden function** – one cannot distinguish which of the five 1-valued matrix elements were overridden by just checking the matrix.

c) **Inability to know if coupled related tasks should not be decoupled** – for example, the whole module in Fig.10 illustrates a Laplacianer task, computing eigenvalues and eigenvectors by differing linear algebra APIs, (see sub-section B of section IV); the five outliers perform the same task in different ways and it is not clear whether they should be decoupled. This is an example of a conceptual problem.

VI. DISCUSSION

A. Comparison with other Refactoring Approaches

The case studies in section V illustrate important features of the Algebraic Higher-Abstraction Refactoring approach:

- **Neat Representation** – the algebraic representation of software systems by matrices clearly eliminates irrelevant source code clutter;
- **Generic Rigorous Procedure** – the usage of Laplacian eigenvectors for modularization is a generic rigorous mathematical procedure, avoiding ad-hoc trial and error and specialized refactoring algorithms;

- **Exact Number of Relocations** – no need to guess how many outlier relocations should be performed; the Fiedler vector reveals the exact number of outliers, as illustrated in the Intellij IDEA case study.
- **Refactoring Amenable to Automation** – the rigorous mathematical procedure, together with a finite and small refactoring rule set, is amenable to automation.

On the other hand, there still are specific problems to be solved on the way to complete automation.

B. Collecting an Algebraic Rule Set

The refactoring Rule Set collection can be seen under two perspectives: 1- **rule classification** into groups, as was tentatively done in sub-section D of section III; 2- **rule conceptualization** possibly leading to a more formal (eventually algebraic) comprehensive and self-consistent rule set. Conjecture 1 on a plausible Finite Rule Set for Higher-Abstraction supports the second perspective.

C. Algebraic and Conceptual Refactoring Separability

This research has been performed under the assumption that one can refactor software systems exclusively based upon algebraic considerations, without conceptual semantic considerations. Some case studies investigated in this work hint that the assumption is not universal. But it could still be the case that the assumption is valid in a significant majority of cases.

D. Future Work

The paper's results, in particular the conjecture of the finite refactoring rule set, deserve formal proofs and extensive verification for a variety of software systems. These will be presented in an extended version of this paper.

E. Main Contribution

This paper's main contribution is an Algebraic Higher-Abstraction refactoring, replacing conventional and less formal approaches, and amenable to software refactoring automation.

REFERENCES

[1] H. Abdeen, H. Sahraoui, O. Shata, N. Anquetil and S. Ducasse, "Towards Automatically Improving Package Structure While Respecting Original Design Decisions", in Proc. 20th WCRE Working Conf. on Reverse Engineering, pp. 212-221, (October 2013). DOI: https://doi.org/10.1109/WCRE.2013.6671296

[2] N.M.M. de Abreu, "Old and new results on algebraic connectivity of graphs", Linear Algebra and its Applications, 423, pp. 53-73, 2007. DOI: https://doi.org/10.1016/j.laa.2006.08.017.

[3] C.Y. Baldwin and K.B. Clark, *Design Rules*, Vol. I. The Power of Modularity, MIT Press, MA, USA, 2000.

[4] G. Bavota, M. Gethers, R. Oliveto, D. Poshyvanyk and A. De Lucia, "Improving Software Modularization via Automated Analysis of Latent Topics and Dependencies", ACM Transactions on Software Engineering and Methodology (TOSEM), Vol. 23, pp. 4, (February 2014). DOI: https://doi.org/10.1145/2559935

[5] T.Y. Browning, "Applying the Design Structure Matrix to System Decomposition and Integration Problems: A Review and New Directions", IEEE Trans. Eng. Management, Vol. 48, pp. 292-306, 2001.

[6] Y. Cai and K.J. Sullivan, "Modularity Analysis of Logical Design Models", in *Proc. 21st IEEE/ACM Int. Conf. Automated Software Eng. ASE'06*, pp. 91-102, Tokyo, Japan, 2006.

[7] I. Exman, "Linear Software Models", Extended Abstract, in I. Jacobson, M. Goedicke and P. Johnson (eds.), *GTSE 2012, SEMAT Workshop on General Theory of Software Engineering*, pp. 23-24, KTH Royal Institute of Technology, Stockholm, Sweden, 2012. Video: http://www.youtube.com/watch?v=EJfzArH8-ls

[8] I. Exman, "Linear Software Models: Standard Modularity Highlights Residual Coupling", Int. Journal on Software Engineering and Knowledge Engineering, vol. 24, pp. 183-210, March 2014. DOI: 10.1142/S0218194014500089

[9] I. Exman, "Linear Software Models: Decoupled Modules from Modularity Eigenvectors", Int. Journal on Software Engineering and Knowledge Engineering, vol. 25, pp. 1395-1426, October 2015. DOI: 10.1142/S0218194015500308

[10] I. Exman and P. Katz, "Modulaser: A Tool for Conceptual Analysis of Software Systems", in Proc. SKY 2016, 7th Int. Workshop on Software Knowledge, pp. 19-26, ScitePress, Portugal, 2016.

[11] I. Exman and R. Sakhnini, "Linear Software Models: Bipartite Isomorphism between Laplacian Eigenvectors and Modularity Matrix Eigenvectors", Int. Journal of Software Engineering and Knowledge Engineering, Vol. 28, No 7, pp. 897-935, 2018. DOI: http://dx.doi.org/10.1142/S0218194018400107

[12] I. Exman and H. Wallach, "A Software System is Greater than its Modules' Sum: Providers & Consumers' Modularity Matrix", in SEKE'2019 31st Int. Conf. on Software Engineering and Knowledge Engineering, Lisbon, Portugal, pp. 75-81, July 2019. DOI: https://doi.org/10.18293/SEKE2019-003

[13] M. Fiedler, "Algebraic Connectivity of Graphs", *Czech. Math. J.*, Vol. 23, (2) 298-305, 1973.

[14] F.R. Gantmacher, The Theory of Matrices, Volume Two, Chelsea Publishing Co., New York, NY, USA, 1959. Chapter XIII, page 53, Available in the Web (out of copyright): https://archive.org/details/theoryofmatrices00gant.

[15] Intellij IDEA – IDE for Java Virtual Machine (2020). https://www.jetbrains.com/idea/

[16] JBLAS - fast linear algebra library for Java based on BLAS and LAPACK (2010) - http://jblas.org/

[17] LA4J – Linear Algebra for Java library (updated 2015) - http://la4j.org/

[18] B. S. Mitchell and S. Mancoridis, On the automatic modularization of software systems using the Bunch tool, IEEE Trans. Softw. Eng. 32 (2006) 193−208.

[19] T. Mens and T. Tourwe, "A Survey of Software Refactoring", IEEE Trans. Software Eng., Vol. 30, pp. 126-139, (2004). DOI: 10.1109/TSE.2004.1265817

[20] M. Mohan and D. Greer, "A Survey of Search-based Refactoring for Software Maintenance", J. Soft. Eng. Res. & Dev., (2018) 6:3. DOI: https://doi.org/10.1186/s40411-018-0046-4

[21] NIST, JAMA: A Java matrix package (2012), http://math.nist.gov/javanumerics/jama/

[22] M. Shtern and V. Tzerpos, "Clustering Methodologies for Software Engineering", in *Advances in Software Engineering*, vol. 2012, Article ID 792024, 2012. DOI: 10.1155/2012/792024

[23] G. Szoke, C. Nagy, R. Ferenc amd T. Gyimothy, "Designing and Developing Automated Refactoring Transformations: An Experience Report", 23rd IEEE Int. SANER Conf., Vol. 5, pp. 693-697, (2016). DOI: https://doi.org/10.1109/SANER.2016.17

[24] Tagger – software preprocessor from simple markup language to Adobe InDesign input – provided by Daniel Jackson, CSAIL, MIT – personal communication, August 2018.

[25] E. W. Weisstein, Bipartite graph (2020), http://mathworld.wolfram.com/BipartiteGraph.html

[26] E. W. Weisstein, Laplacian matrix (2020), http://mathworld.wolfram.com/LaplacianMatrix.html

[27] M.S. Zanetti, C.J. Tessone, I. Scholtes and F. Schweitzer, "Automated Software Remodularization Based on Move Refactoring", in Proc. MODULARITY '14, pp. 73-83, (April 2014). DOI: http://dx.doi.org/10.1145/2577080.2577097

Reliable Compilation Optimization Selection Based on Gate Graph Neural Network

Jiang Wu, Jianjun Xu, Xiankai Meng
College of Computer
National University of Defense Technology
wujiang_nudt@163.com, jjxu@nudt.edu.cn, mengxiankai12@nudt.edu.cn

Abstract—**For different programs or applications, it is necessary to select the appropriate compilation optimization pass or subsequence for the program. To solve this problem, machine learning is widely used as an efficient technology. However, the most important problem in using machine learning is the extraction of program features. How to ensure the integrity and effectiveness of program information is the key to the problem. In addition, when compiling and optimizing the selection problem, the measurement indicators are often program performance, code size, etc. There is not much research on program reliability which needs the longest measurement time and the most complicated measurement methods. This paper proposes a GGNN-based compilation optimization pass selection model. We extend the deep neural network based on GGNN, and build a learning model which learns heuristics for program reliability. The experiment was performed under the clang compilation framework. The alternative compilation optimization pass adopts the C language standard compilation optimization passes. Compared with the traditional machine learning method, our model improves the average accuracy by 5% ~ 11% in the optimization pass selection for program reliability. At the same time, experiments show that our model has strong scalability.**

Keywords- compilation optimization selection; AST; GGNN; reliability; clang

I. INTRODUCTION

In the past few decades, compiler developers have designed and implemented a large number of compilation optimization options in response to compilation optimization needs in various complex situations. In actual development, the standard compilation optimization pass provided by the compiler is difficult to adapt the requirements for the program to be compiled in complex scenarios. On the one hand, the program to be compiled has different semantics and compilation goals. It is difficult to obtain the optimal optimization effect by using the standard compilation optimization pass directly. If an inappropriate optimization pass is used, it may even bring negative effects about program performance, etc. On the other hand, with the continuous development of the hardware architecture, the compilation environment becomes increasingly complex, and the compilation optimization pass should be adjusted accordingly. Therefore, how to choose the best compilation optimization pass for the program to be compiled among the intricate optimization options. Become a challenging scientific problem. The algorithms used in this field mainly include heuristic search algorithms and machine learning algorithms. The heuristic search algorithm uses a heuristic method to search the optimal compilation optimization pass in the compilation optimization option combination space. For example, the VISTA interactive compilation system [1] uses a combination of genetic algorithms and human-assisted guidance to search for optimal compilation optimization passes; the open source framework " OpenTuner " [2] uses a variety of evolutionary algorithms, including genetic algorithms, to get a speedup of up to 2.8 times; Jantz et al. [3] use genetic algorithms to select the optimal compilation optimization pass for the JIT compiler. And some other selection schemes based on some multi-objective optimization algorithms, for example, Lok et al. [4] [5] use SPEA2, NSGA-II and IBEA to select the compilation optimization pass for the program to be compiled that meets the target code execution speed, scale and other goals.

However, the heuristic search algorithm can generate efficient compilation optimization sequences, but it takes a lot of time to run the entire iterative process. Gradually researchers began to use machine learning algorithms to select compilation optimization sequences. A large number of algorithms based on SVM and LR are widely used. The work [6] used code runtime characteristics to characterize the program to be compiled to train the logistic regression model; Ashouri et al. [7] analyzed the dependencies between optimization options in the compiler's LLVM, using program dynamic characteristics to train the Bayes network, then use this model to predict the optimization options that should appear in the next stage until the prediction is completed; the open source compiler "Milepost GCC" [8], which is a modularized, modified form of GCC4.4 scalable compiler that supports static feature extraction of the program to be compiled, trains machine learning models, and predicts the compilation effect of the compiled optimization sequence. A large number of machine learning algorithms perform feature extraction on programs, both dynamic and static features., it is difficult to extract program information completely and efficiently. Most work has tried to transfer natural language methods and does not capitalize on the unique opportunities offered by code's known semantics. For example, long-range dependencies induced by using the same variable or function in distant locations are often not considered. Such models miss out on the opportunity to capitalize on the rich and well-defined semantics of source code. Therefore, constructing a graph to represent complete program information and training in conjunction with a graph neural network is a more effective way to ensure the integrity of the program information as much as possible.

In addition, from the perspective of compiling optimization goals, most researches focus on the execution speed of the target

machine code [9] [10]. Statistics shows that the target code is used in machine learning algorithms. The acceleration ratio as the optimization target accounted for the vast majority of the research, accounting for more than 80% of this part of the research. Another optimization goal that researchers are concerned about is the size of the target code [11] [12]. It is a very important optimization goal, especially in the case of the current widespread application of embedded programs, reducing the storage space as much as possible can bring significant benefits. However, there is not much research on the use of machine learning for compilation optimization orienting program reliability research.

In order to extract the program information as completely as possible, while taking advantage of the advantages of machine learning, we combine GGNN, program reliability analysis and compilation optimization selection problems. We abstract the C raw code into a graph with data flow and type hierarchies, and then build a program optimization oriented graph neural network for program reliability. Our work replaces the need for compile-time or static code features, merging feature and heuristic construction into a graph and send it to a graph neural network. Then learning to get which clang standard compilation optimization can bring the highest reliability gain for a specific C code. By using the PIN [13] tool for verification, our model has an average accuracy improvement of 5% ~ 11% compared to traditional machine learning algorithms without our extended GGNN. At the same time, our model is also highly scalable and can adjust the size of the output layers to solve different problems.

II. PROGRAM AS GRAPH

A. Abstract Syntax Tree

As an intermediate representation of the source code for parsing and semantic analysis, AST [14] is a tree-structured data describing the syntax rules and execution order of the code, which is obtained after the code is parsed using irrelevant context rules. In the AST, leaf nodes represent identifiers in the source code, while non-leaf nodes represent syntactic structures. As the parse tree of the source code, the AST basically covers the following syntax structures: *Selecting structure* (IF, SWITCH, etc.); *Loop structure* (WHILE, FOR, etc.); *Sequence structure* (expressions, assignment statements, etc.). Therefore, AST, as an intermediate representation of the source code, can effectively retain the syntactic context information related to the programming language.

B. Function Call Graph

FCG [15] is used to characterize information related to control flow in source code. Each node in a function call graph represents a function, and the edges in it represent the calling relationship between functions. Understanding the calling relationship between functions is of great help to understand the hierarchical structure of the program, and clarifying the function calling relationship is a key part of program analysis.

C. Data Flow Graph

DFG [16] explicitly contains the data logic of the two aspects of data transfer and data processing in the source code. Nodes in DFG represent entities, such as variable declarations, operands,

operators, structures, etc., and the edges in them represent the data relationships that exist between these entities. DFG can describe the data logic and program functions of the source code and is used to analyze the dynamic runtime data flow information of the program. From the perspective of data transmission, DFG describes the movement and transformation of data streams from input to output. Because it can clearly reflect the logic that the program must complete, it has become one of the most commonly used methods of program analysis.

(a)

(b)
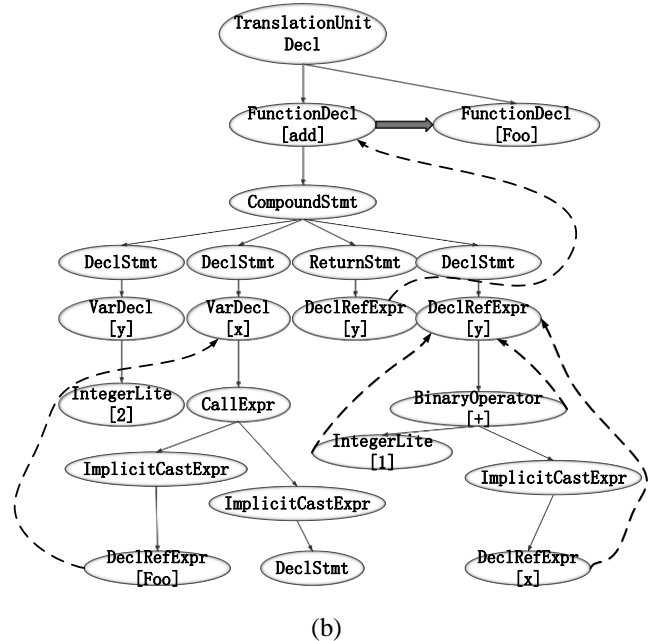
Figure 1. Example of the co-AST.

By comparing the characteristics of AST, FCG, and DFG, it is clear that each different form of the intermediate expression form only describes the program source code from a certain angle. The AST only contains static information related to the grammatical structure, and the latter two are used to describe the runtime dynamic information related to the control flow and the data flow. In particular, the angles of the latter two references are different. FCG starts with a coarse-grained function, while DFG starts with fine-grained variables, operators, and operands. The source code is special executable text, and both static syntax information and dynamic runtime information are important. Therefore, the fusion of the code information carried in the AST,

FCG, and DFG helps reduce the information loss caused by the transformation of source code to intermediate expressions.

We have established a joint program analysis graph co-AST, which combines the characteristics of AST, FCG, and DFG. As shown in Figure 1, based on the source code of Fig. 1 (a), we constructed the co-AST graph as Fig. 1 (b). The complete co-AST graph has a total of seven types of edges. We introduce the edges of FCG and DFG into the original AST structure. As shown in Figure 1 (b), the solid arrow is the function call identifier, and the dashed curve arrow is the identification of the data stream This combination greatly enriches the information in the program graph, thereby speeding up the spread of information in GGNN and improving the effect of model training.

III. CONSTRUCTION OF EXTENDED GGNN

The graph model used in this paper involves a directed graph $G = (V, E)$, where V represents a set of nodes of size $|V|$ and E represents a set of size $|E|$. The nodes in V are represented by node number i, the directed edges in E are represented by e_{ij}, and e_{ij} represents the edge pointed by node i to node j. For different types of edges in the graph, use the edge type set $L_K = \{l_1, l_2, ..., l_k\}$ to represent. The connection relationship between the nodes in the graph is represented by the connection matrix A. There are two design schemes for the dimension of A. The first design is $A \in R^{|V| \times 2|V|}$, directed edge e_{ij} in the figure is seen as two different types of access edges, one is the outgoing edge of node i and the other is the incoming edge of node j. The second design is $A \in R^{|V| \times |V|}$, it only considers the directed edge e_{ij} as the incoming edge of node j. The connection matrix in this paper uses the second scheme.

The element A_{ij} in the i row and the j column is a $d \times d$ matrix (d represents the node state vector dimension). A_{ij} is also called the propagation matrix on edge e_{ij}, which represents the information propagation rules from node i to node j. For example, Fig. 2 (c) shows the connection matrix corresponding to the data flow graph shown in Fig. 2 (b), where the two rectangular-framed matrices are the propagation matrices corresponding to e_{3y} and e_{yz} respectively.

In the assignment statement shown in Fig. 2 (a), we are concerned about whether the number 3 can be passed to the variable z, as shown in Fig. 2 (b). To this end, nodes 3 and z can be regarded as the source node and the target node respectively, and their feature vectors are initialized as $h_3^0 = [1,0]$ and $h_z^0 = [0,1]$ (the first dimension of the two-dimensional vector is 1, which means that 3 can reach the node), and the feature vector table of node y is initialized as $h_y^0 = [0,0]$. The propagation matrix A_{ij} determines how the information of each dimension of node i is propagated to the various dimensions of node j. "0" represents no propagation and "1" represents complete propagation. For example, A_{3y} in Fig. 2 (c) indicates that node 3 only passes the information of its first dimension to the first dimension of node y. In this way, the result of multiplying vector h_3 and A_{3y} is still $h_y = [1, 0]$, indicating that the data has not been passed to the target node z. However, A_{yz} in Fig. 2 (c) indicates that the

information of the first dimension of node y is to be transferred to the first dimension of node z. Therefore, the result of multiplying vector h_y and A_{yz} is $h_z = [1,0]$, indicating that data can reach the target node z.

$$y = 3$$
$$z = y$$

(a) Sample code

(b) DFG

(c) Connection matrix
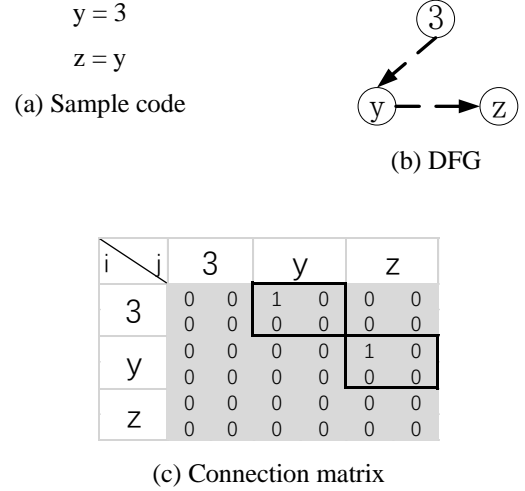
Figure 2.   Example of connection matrix and propagation matrix.

In GGNN [17], the information propagation of nodes on different types of edges is achieved through different multilayer perceptron, and the propagation matrix on the edges is represented by the trainable multilayer perceptron weights $W_e \in R^{d \times d}$. It should be noted that the connection matrix shown in Fig. 2 (c) only represents the weight of the multi-layer perceptron after the graph model has converged on the reachability task. The GGNN model has an iterative t-round of node state information propagation process, as follows: the state information of node i is initialized to a vector $h_i^{(1)} \in R^d$. During the t-th round of iteration, each central node i gathers all neighbor node information to get the node interaction context $m_i^{(t)} \in R^d$, as shown in (1) (where N_i represents the set of neighbor nodes of i). In response to the current interaction context, the node i updates its own state information $h_i^{(t)}$ after t round. The GRU unit is used in GGNN different from GNN. The GRU unit considers the relationship between node status information in different update rounds. That is, when the node updates during the round t-th, the node hidden layer vector expression $h_i^{(t)}$ and the state information $h_i^{(t-1)}$ of the previous round have a time series relationship, as shown in (2). GNN only uses edges as a means of propagation, but does not distinguish the functions of different edges. And GNN does not set independent learnable parameters for edges, which means that some characteristics of edges cannot be learned through the model. This is also the main reason we use GGNN as shown in Fig. 3.

$$m_i^{(t)} = \Sigma_{j \in N_i} A_{ij} \cdot h_j^{(t-1)}. \tag{1}$$

$$h_i^{(t)} = GRU(h_i^{(t-1)}, m_i^{(t)}). \tag{2}$$

Figure 3.   Extended GGNN architecture

Embedding

Message Propagation
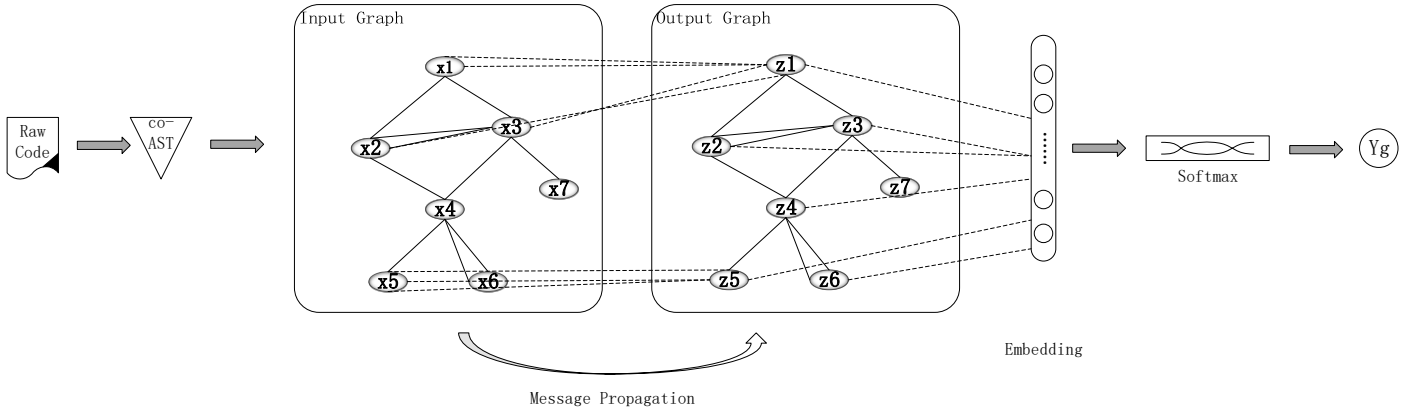
During the information propagation of the graph model, $m_i^{(t)}$ is the interaction context of node i in the whole graph. Whether it is GNN or GGNN, $m_i^{(t)}$ is obtained by directly accumulating the product of feature information $h_j^{(t-1)}$ of the neighbor node j and the propagation matrix A_{ij} on the edge e_{ij}. In the topology of the graph, different nodes have different properties in the topology. In the GNN and GGNN models, the topological properties of the nodes are directly expressed as hidden nodes. Based on this, in the substructure composed of the central node i and its neighbor nodes N_i, our model abandons the way of directly accumulating the product of $h_j^{(t-1)}$ and A_{ij} to calculate $m_i^{(t)}$. We hope that the model automatically learns how to calculate $m_i^{(t)}$ and that the central node could pay more attention to those neighbor nodes whose topology information is important, because these neighbor nodes determine the interaction context of the node i on the graph to a greater extent.

Therefore, we have extended GGNN. We assign different weights to each neighbor node to characterize its importance to the central node α_{ij}, it gets function mapping through neural network $a : R^d \times R^d \rightarrow R$, a calculates the correlation coefficient between the central node i and its neighbor j, and uses the *softmax* function to normalize the correlation coefficients of all neighboring nodes, such as (3) shown:

$$\alpha_{ij} = softmax\ (a\ (h_i^{(t-1)}, h_j^{(t-1)})).\qquad(3)$$

The weight parameter of the neural network a is only related to the round of information propagation. The same round of information propagation, a is shared by all nodes. Different propagation rounds have different parameters for a. The uncertainty of the number of neighbor nodes j of a node i will result in a variable number of a_{ij}, and it is not possible to directly implement the *softmax* function provided by the framework Tennsorflow. This paper implements *softmax* to adapt to the changing number of neighbor nodes:

$$a_{ij} = a_{ij} - max(a_{i1}, a_{i2}, ..., a_{ij}).\qquad(4)$$

$$\alpha_{ij} = \frac{e^{a_{ij}}}{\Sigma_{k \in N_i} e^{a_{ik}}}.\qquad(5)$$

So the interaction context $m_i^{(t)}$ of node in of our expanded GGNN is shown in (7):

$$m_i^{(t)} = \sum_{j \in N_i} \alpha_{ij} \cdot A_{ij} \cdot h_j.\qquad(6)$$

Because the selection of program compilation optimization pass is to analyze the program according to the embedded expression of the program algorithm graph co-AST, after obtaining the final graph node embedding vector expression $h_i^{(T)}$, the embedding vector h_G of the entire co-AST graph needs to be calculated. This paper proposes a node vector probability fusion method, which generates a graph embedding vector from the node embedding vector. As shown in (7), the $f(h_i^{(T)}, h_i^{(1)})$ is a fully-connected neural network, which learns the probability that node i will be fused based on node attributes $h_i^{(1)}$ and topology information $h_i^{(T)}$. The activation function in f uses *sigmoid*, whose final output is a value of [0, 1]. The g is also implemented by using a fully connected layer neural network, which uses the *tanh* function to activate the output. The calculation of h_G is similar to (6). In the end, the program compilation optimization pass selection l_G is derived from the function *sofmax*:

$$h_G = \sum_{i \in V} f\left(h_i^{(T)}, h_i^{(1)}\right) \cdot g(h_i^{(T)}).\qquad(7)$$

$$l_G = softmax\ (h_G).\qquad(8)$$

In our extended GGNN, the computation of α_{ij} of central node and its neighbor nodes can be parallelized and computationally efficient. Moreover, it implements the calculation method of the model automatically learning the interaction context, without having to consider the number of neighbor nodes that changes. If using neural networks to learn to calculate the interaction context, it will definitely need to face the problem

that the neural network weight dimensions cannot be unified due to the inconsistent number of neighbors in each node.

IV. EXPERIMENT AND ANALYSIS

In order to verify the effectiveness of the model proposed in this paper in the selection of reliability-oriented program compilation optimization pass, we not only evaluate the model from the perspective of pass selection accuracy, but also analyze the ability of the model to learn topology from the perspective of co-AST graph node embedding expressions.

A. Configuration

In order to cover more program categories in our training set, we have expanded on the standard C test suite MiBench [18]. We still adopt the program classification method of MiBench, but we have made a lot of expansions in the number of programs. We use the open source compilation tool *clang* to parse the raw code to get the program's AST data set. We further add edges representing data flow information and edges representing function call graphs to the AST tree for each program, and finally obtain the final co-AST data set. In our experiments, the co-AST data set is divided into a training set, a validation set, and a test set according to a ratio of 8: 1: 1. We use the PIN tool to evaluate the reliability of the program and generate the training set. Our program reliability evaluation indicators refer to Sridhran [19].

The optimization algorithm used for model training is the SGD of the ADAM optimizer [20]. The loss function uses cross entropy. The weight parameter initialization in the model uses Glorot [21] initialization method. In the experiment, the information propagation layer (information iteration round) is set to 4 layers, and the number of neurons in each propagation layer, that is the propagation matrix vector dimension d, is a hyperparameter. The choice of this hyper-parameter mainly considers the speed of model convergence and the model's loss value. For this reason, we determined after experiments that when the hidden layer vector dimension d is 270, the model's convergence loss value is relatively small, and the model training speed is also relatively fast. Therefore, we set the hidden layer vector dimension d to 270 to complete the subsequent experiments.

B. Result and analysis

In the experiment, we construct the classification task of 4. The main content of this task is to judge, for a specific C raw code, when using the clang standard compilation optimization passes -O1, -O2, -O3, and -OS, which one is more reliable for the program. This is different from many others that focus on the impact of compilation optimization passes on program speed and code size. The benchmark comparison experiment selected in this experiment is TreeBased Convolution Neural Network (TBCNN). To our knowledge, TBCNN is by far the best performing work on source code classification tasks. In addition, LSTM [22] is widely used in text classification tasks and our model is aimed at the improvement of the GGNN model. We also test the LSTM and GGNN models. The experimental results are shown in TABLE I, where exGGNN is our extended GGNN. Therefore, there are four models in the controlled trial, LSTM, TBCNN, GGNN and exGGNN.

TABLE I. ACCURACY OF OPTIMIZATION PASS SELECTION

Different Model	Accuracy		
	Minimum	*Maximum*	*Average*
LSTM	82.2%	84.3%	83.9%
TBCNN	84.5%	88.6%	86.7%
GGNN	87.3%	93.5%	89.2%
exGGNN	**87.9%**	**98.1%**	**94.1%**

The experimental results in Table 1 show that the accuracy of exGGNN in the code optimization pass selection problem has improved significantly, indicating that our model has achieved the expected results for this problem. Better than LSTM and TBCNN shows that choosing GGNN to deal with such problems is a better choice, and better than GGNN shows that our extension has played a important role. Then, in order to evaluate whether the data flow edge and function call graph edge are useful, we remove one of the 7 types of edges and use the exGGNN model to learn the co-AST graph after deleting a certain edge to implement the optimization pass selection. Observe the effect of each edge on the selection accuracy of the program. The experimental results are shown in TABLE II. The double underline indicates the co-AST graph with this type of edge removed.

TABLE II. TABLE TYPE STYLES

Different Edge	Program Category (MiBench)					
	auto-mo-tive	con-sumer	net-work	of-fice	secu-rity	tele-comm
AST	**0.99**	0.02	0.09	**0.95**	0.06	0.09
Operand	**0.92**	0.07	0.07	**0.92**	0.02	0.05
LastUse	**0.92**	0.07	0.07	**0.12**	0.02	0.05
Compute	**0.92**	0.07	0.07	**0.92**	0.02	0.05
Return	**0.94**	0.07	0.07	**0.92**	0.02	0.09
Formal	**0.99**	0.08	0.08	**0.05**	0.02	0.05
Call	**0.92**	0.07	0.07	**0.92**	0.03	0.05

The experimental results show that for most program tasks in MiBench, deleting a certain type of edge has little effect on the accuracy of program optimization pass selection accyracy. There may be information redundancy in the seven types of edges, so any type of edge deletion will not have a significant impact on the accuracy of program classification. But for some programs, deleting these types of edges can significantly reduce or improve the accuracy of program classification. Therefore, the construction of co-EAST is effective and can further improve the extracted program information.

We also compare the convergence trend of exGGNN / GGNN / TBCNN loss values in the co-AST / AST intermediate expression form. As shown in Fig.4, the graph model not only has a smaller final convergence loss value, but also has a faster convergence rate than TBCNN. The reason why the graph network model converges faster is that the graph model has

stronger constraints on AST nodes than TBCNN. More specifi-cally, in the TBCNN model, the convolution operation forces one-way propagation of information from child nodes to the parent node. While the graph model involves, for each node, it is two-way information propagation between all neighboring nodes. This information dissemination can gradually spread to the entire graph structure.
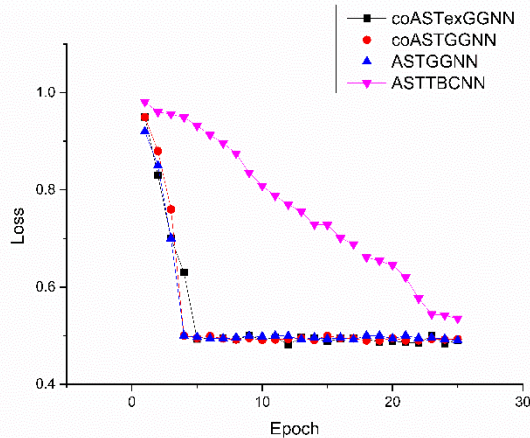
Figure 4. Variation of loss values for the four models.

In order to select a highly reliable compilation and optimization pass, we propose a learning strategy through GGNN. The experimental results show that our model achieves a higher accuracy on the pass selection problem and is better than similar neural networks models. As the first attempt to combine graph neural networks with program reliability, we obviously achieved our experimental goals. Although the program's running time and code size are important indicators of program evaluation, the reliability of the program can not be ignored, especially in the booming aerospace field, the reliability of the program is always the first consideration. We are working on combining optimization sequence generation and graph neural networks, hoping to find a better solution to the phase ordering problem.

REFERENCES

[1] Kulkarni P, Zhao W, Moon H, et al. Finding effective optimization phase sequences[J]. ACM SIGPLAN Notices, 2003, 38(7): 12-23.

[2] Ansel J, Kamil S, Veeramachaneni K, et al. Opentuner: An extensible framework for program autotuning[C]//Proceedings of the 23rd international conference on Parallel architectures and compilation. 2014: 303-316.

[3] Jantz M R, Kulkarni P A. Performance potential of optimization phase selection during dynamic JIT compilation[C]//Proceedings of the 9th ACM SIGPLAN/SIGOPS international conference on Virtual execution environments. 2013: 131-142.

[4] Lokuciejewski P, Plazar S, Falk H, et al. Multi-objective exploration of compiler optimizations for real-time systems[C]//2010 13th IEEE

International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing. IEEE, 2010: 115-122.

[5] Lokuciejewski P, Plazar S, Falk H, et al. Approximating Pareto optimal compiler optimization sequences—a trade‐off between WCET, ACET and code size[J]. Software: Practice and Experience, 2011, 41(12): 1437-1458.

[6] Cavazos J, Fursin G, Agakov F, et al. Rapidly selecting good compiler optimizations using performance counters[C]//International Symposium on Code Generation and Optimization (CGO'07). IEEE, 2007: 185-197.

[7] Ashouri A H, Bignoli A, Palermo G, et al. Micomp: Mitigating the compiler phase-ordering problem using optimization sub-sequences and machine learning[J]. ACM Transactions on Architecture and Code Optimization (TACO), 2017, 14(3): 1-28.

[8] Fursin G, Kashnikov Y, Memon A W, et al. Milepost gcc: Machine learning enabled self-tuning compiler[J]. International journal of parallel programming, 2011, 39(3): 296-327.

[9] Martins L G A, Nobre R, Cardoso J M P, et al. Clustering-based selection for the exploration of compiler optimization sequences[J]. ACM Transactions on Architecture and Code Optimization (TACO), 2016, 13(1): 1-28.

[10] Ashouri A H, Mariani G, Palermo G, et al. Cobayn: Compiler autotuning framework using bayesian networks[J]. ACM Transactions on Architecture and Code Optimization (TACO), 2016, 13(2): 1-25.

[11] Foleiss J H, da Silva A F, Ruiz L B. The effect of combining compiler optimizations on code size[C]//2011 30th International Conference of the Chilean Computer Science Society. IEEE, 2011: 187-194.

[12] Plotnikov D, Melnik D, Vardanyan M, et al. An Automatic tool for tuning compiler optimizations[C]//Ninth International Conference on Computer Science and Information Technologies Revised Selected Papers. IEEE, 2013: 1-7.

[13] Luk C K, Cohn R, Muth R, et al. Pin: building customized program analysis tools with dynamic instrumentation[J]. Acm sigplan notices, 2005, 40(6): 190-200.

[14] Shen V R L. Novel Code Plagiarism Detection Based on Abstract Syntax Tree and Fuzzy Petri Nets[J]. International Journal of Engineering Education, 2019, 1(1).

[15] Hassen M, Chan P K. Scalable function call graph-based malware classification[C]//Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy. 2017: 239-248.

[16] Weyerhaeuser C, Mindnich T, Baeumges D, et al. Augmented query optimization by data flow graph model optimizer: U.S. Patent 10,241,961[P]. 2019-3-26.

[17] Li, Yujia, Tarlow, Daniel, Brockschmidt, Marc, et al. Gated Graph Sequence Neural Networks[J]. Computer Science, 2015.

[18] Guthaus M R , Ringenberg J S , Ernst D , et al. MiBench: A free, commercially representative embedded benchmark suite[C]// Workload Characterization, 2001. WWC-4. 2001 IEEE International Workshop on. IEEE, 2002.

[19] Sridharan V, Kaeli D R. Eliminating microarchitectural dependency from architectural vulnerability[C]//2009 IEEE 15th International Symposium on High Performance Computer Architecture. IEEE, 2009: 117-128.

[20] Kingma D P, Ba J. Adam: A method for stochastic optimization[J]. arXiv preprint arXiv:1412.6980, 2014.

[21] Glorot X, Bengio Y. Understanding the difficulty of training deep feedforward neural networks[C]//Proceedings of the thirteenth international conference on artificial intelligence and statistics. 2010: 249-256.

[22] Dyer C, Ballesteros M, Ling W, et al. Transition-based dependency parsing with stack long short-term memory[J]. arXiv preprint arXiv:1505.08075, 2015.

Trends in Software Reverse Engineering

Dr. Rehman Arshad

Research Associate, The Hut Group

M90 3DP, Manchester, United Kingdom

rehman.khan@thehutgroup.com

Abstract

The polymorphic domain of software reverse engineering varies since 90s due to multiple reasons. Some of the primary reasons include the acceptance of new programming languages, underlying technique of reverse engineering and the desired output notation of the reverse engineering that varies with evolution of software. The purpose of this paper is to provide a trend-based taxonomy of reverse engineering that can classify the differences and similarities in the reverse engineering throughout the years.

Key Words —Reverse Engineering, Softwrae Comprehension[1]

1. Introduction

"Reverse engineering can be viewed as a process of analysing a system to identify the system's components and their interrelationships" [14]. This process is used to comprehend, or analyse a system in order to extract an architecture/notation for various purposes e.g., re-structuring of legacy systems [2, 6, 31], understanding the code traces [23], find out the feature locations [37, 51] and developing understanding of the systems with poor documentation [19].

The purpose of this paper is to analyse the pros and cons and to reason about the changes in the nature of reverse engineering throughout the years. This paper can be used as a reference document to understand the factors and reasons that change the nature of software reverse engineering since 90s.

There are few factors that are responsible for the change in nature of reverse engineering throughout the years i.e., targeted programming language, underlying technique to reverse engineer software and the output notation of reverse engineering. Most reverse engineering approaches try to tackle the programming language that is considered *legacy* for a specific domain OR better upcoming options are on the horizon for that particular domain. Therefore, most of the approaches from 90s to early 2000s were applied on ***COBOL*** and ***C*** code bases [18, 21] whereas, most of the current ones are being applied on ***Java*** [6, 44]. Secondly, the notation of retrieved output after reverse engineering started as system documentation/comprehension [12] and went through the concept-lattices/graphs and code traces [9, 13] that can help in understanding the composition of a system. Nowadays, reverse engineering is moving towards architectural retrieval that can enable reusability of resources [2, 29].

This paper only covers those approaches that deal with transformation of source code from one notation to another. The succeeding sections will discuss the framework of classification and discuss the reverse engineering approaches with respect to the timeline since 90s.

2. Framework of Classification

The proposed framework and the approaches that we cover are presented in Table. 1[2]. The stated table has following parameters of classification.

- ***Timeline:*** Timeline has been classified into five intervals. Starting from an interval of ten years[3] followed by the four intervals of five years each.
- ***Approaches:*** This column shows the total number of reverse engineering approaches that we have covered during each phase.
- ***Approach Name and Reference:*** As the name suggests, these columns will show all the appraoches along with their references.

---

[1]DOI reference number: 10.18293/SEKE2020-061

[2]**Table 1 Legend:**
F: Feature Model, **C**: Re-structured Code, **NEC**: Non-Explicit Components
EC: Explicit Components, **VB** : View Based **G**: Concept-Lattices/Graphs, **CT**: Rank-Based Mapping/Code Trace
CO: COBOL, **LI**: Language Independent, **J**: Java, **OO**: Object-Oriented

[3]First interval consists of a decade rather than five years due to a smaller number of approaches proposed in that specific timeframe.

Timeline	Approaches	Approach Name	Reference	Parsing Based	Plan Based	Transformational	Translational	F	C	NEC	EC	VB G	VB CT	CO	C	C++	LI	J	OO
90s-2000	5	RECAST	[18]	✓								✓		✓					
		Sub-System Identification	[36]	✓							✓					✓			
		Ward and Bennet's Approach	[48]	✓					✓									✓	
		Burd and Munro's Approach	[12]			✓					✓			✓					
		Design Components	[29]	✓							✓					✓			
2001-2005	9	Concern Graphs	[40]				✓				✓							✓	
		Favre et al.	[25]	✓						✓									✓
		Systematic Method Approach	[31]		✓					✓								✓	
		Dynamic Feature Traces	[23]				✓						✓					✓	
		Concept Analysis	[21]				✓						✓		✓				
		Trace Dependency Analysis	[19]		✓								✓			✓			
		Software Evolution Analysis	[27]				✓				✓								✓
		Locating Features in Source Code	[22]				✓						✓						✓
		Automatic Generation	[39]	✓									✓					✓	
2006-2010	21	Dependence Graph	[40]				✓				✓				✓			✓	
		Javacompext	[6]	✓								✓						✓	
		Chouambe at al.	[15]	✓								✓						✓	
		Antoun et al.	[2]		✓							✓						✓	
		L2CBD	[30]									✓					✓		
		CORE	[32]		✓					✓									✓
		Bunch Tool	[35]	✓						✓							✓		
		Natural Language Parsing	[1]		✓								✓					✓	
		Source Code Retrieval	[34]		✓								✓			✓		✓	
		Combining FCA with IR	[38]				✓				✓							✓	
		STRADA	[20]										✓					✓	
		Call-Graph	[9]		✓								✓		✓	✓			
		Focused-view on Execution	[10]				✓						✓		✓	✓			
		Scenario-Driven Dynamic Analysis	[42]				✓						✓			✓			
		Featureous	[37]				✓						✓					✓	
		Static and Dynamic Analysis	[41]	✓									✓					✓	
		Heuristics Based Approach	[8]		✓								✓					✓	
		Landmark and Barriers	[47]		✓						✓							✓	
		SNIAFL	[52]				✓						✓		✓			✓	
		Cerberus	[17]		✓								✓					✓	
		Concern Identification	[45]	✓							✓							✓	
2011-2015	13	RecoVar	[51]	✓							✓						✓		
		Semi-Automatic Approach	[46]			✓							✓					✓	
		Archimetrix	[16]	✓							✓						✓		
		Quality-centric Approach	[28]		✓						✓								✓
		Memory-constrained Environment	[49]		✓						✓								✓
		Erdemir et al.	[24]	✓						✓									✓
		Product Variants	[50]			✓			✓							✓			
		Evolutionary Algorithms	[33]			✓		✓										✓	
		Software Configurations using FCA	[3]				✓	✓											✓
		Reverse Engineering Feature Models	[44]		✓			✓								✓			
		Component Oriented Architecture	[4]	✓							✓							✓	
		MoDisco	[11]	✓							✓								
		Language Independent Approach	[53]	✓						✓								✓	
2016-Current	3	Shatnawi et al.	[43]		✓					✓									✓
		Alshara et al.	[5]		✓						✓							✓	
		RX-MAN	[7]	✓							✓							✓	

Table 1: Trends in Software Reverse Engineering

- *Technique:* The parameter *Technique* is further classified based on the Gannod's and Cheng's framework as follows [26]:
    - *Parsing Based:* approaches use parsers like AST (Abstract Syntax Tree) to capture a code base for reverse engineering without losing any detail.
    - *Plan Based:* approaches use heuristics and define abstraction models to capture the source code.
    - *Transformational:* techniques transform one notation of semantics into another by specifying a formal context.
    - *Translational:* ones translate a program into an equivalent formal specification e.g., creation of a directed graph from source code.
- *Retrieved Notation:* shows us the output that each approach offers. This parameter has been classified into feature models, restructured code, non-explicit components (no defined composition), explicit components (components that follow a component model) and view based output (further classified into code traces and concept-lattices/graphs).
- *Programming Language:* represents the targeted language of a reverse engineering approach. The languages are classified into *COBOL*, *C*, *C++*, *Java*, *OO* (work on any object-oriented code) and *LI* (language independent approaches).

3. Trends in Software Reverse Engineering

Based on the proposed framework and 51 approaches that we have covered in this paper (Table. 1), there is a clear pattern that shows the variation of software reverse engineering since 90s.

Almost all the covered approaches from 90-2000s are parsing-based i.e., heuristics and plan-based reverse engineering was not developed enough to conduct the process of reverse engineering. All the approaches were applied and designed for *COBOL* and *C/C++* code bases i.e., *Java* was becoming popular in late 90s and was not considered because its code bases did not reflect legacy code in that era. The popular notations of the output of reverse engineering were components or graphical representations that can help in code comprehension although, the components' notations were not specified by some component model. Components in that era were usually defined as loosely coupled code chunks without proper definition of composition.

From 2001-2005, translation-based reverse engineering was the preferred way instead of parsing-based reverse engineering. Most of the approaches targeted object-oriented legacy code[4]. An important change in this phase is the preferred output of reverse engineering i.e., code trace that can help in finding the feature locations in a code base. Most

approaches (e.g., [23] [27]) conducted dynamic reverse engineering to map legacy code bases in terms of the features of software.

From 2006-2010, the domain of reverse engineering was all about heuristics i.e., reverse engineering was based on plan-based or translation-based (translation via heuristics) methodologies. This was the era of visualisation-based reverse engineering i.e., all the approaches produced either code traces or concept lattices to visualise the dependencies e.g., [9] [40]. Many established domains like *IR* (information retrieval) and *NLP* (natural language parsing) were involved in reverse engineering to produce better results from heuristics (e.g., [1] [38]). In this phase, 61% of the covered approaches targeted Java i.e., the trend moved specifically towards Java rather than general *OO*. It was justified due to the fact that by the end of this phase, many enterprise java code-bases were started to be considered legacy code.

From 2011-2015, the parsing-based reverse engineering was again on the rise i.e., 42% approaches from this phase were based on parsing. It was due to the fact that most approaches extracted ADL-based components/architectural-notations from the legacy code-bases. Such architectural notations demand preservation of the functionality of an original code base and heuristics cannot guarantee lossless extraction of architecture. 21% of the covered approaches were plan-based and a few of the approaches extracted features/feature models (e.g., [3] [44]). 53% approaches considered object-oriented code-bases.

The current phase (2016-current) of software reverse engineering still revolves around architectural re-usability. Software reverse engineering is moving away from graphs and code traces towards components. *OO* code in general and *Java* in particular is the favourite choice of current approaches. Table. 1[5] shows the overall statistics of the trends in software reverse engineering.

4. Conclusion

In this paper, we have covered more than 50 approaches to determine the trends and variations in software reverse engineering since 90s. Our framework shows that reverse engineering is moving from code comprehension and graphs towards components and architectural notations.

The adaptation in the techniques of reverse engineering went through phases of parsing-based, translational and plan-based reverse engineering whereas, most of the recent reverse engineering approaches are again in favour of parsing with an aim of architectural retrieval that requires the preservation of syntactic code structure.

References

[1] Surafel Lemma Abebe and Paolo Tonella. Natural language parsing of program element names for concept extraction. In *18th International Conference on Program Comprehension (ICPC), 2010 IEEE*, pages 156–159. IEEE, 2010.

---

[4]Such approaches can be applied on any object-oriented code though most of them chose *Java* as the targeted language.

[5]MoDisco is a framework and L2CBD is a methodology rather than concrete approaches therefore, no programming language/Technique is specified for them respectively.

[2] Marwan Abi-Antoun, Jonathan Aldrich, and Wesley Coelho. A case study in re-engineering to enforce architectural control flow and data sharing. *Journal of Systems and Software*, 80(2):240–264, 2007.

[3] R Al-Msie'Deen, Marianne Huchard, A-D Seriai, Christelle Urtado, and Sylvain Vauttier. Reverse engineering feature models from software configurations using formal concept analysis. In *CLA 2014: Eleventh International Conference on Concept Lattices and Their Applications*, volume 1252, pages 95–106, 2014.

[4] S. Allier, S. Sadou, H. Sahraoui, and R. Fleurquin. From object-oriented applications to component-oriented applications via component-oriented architecture. In *2011 Ninth Working IEEE/IFIP Conference on Software Architecture*, pages 214–223, June 2011.

[5] Zakarea Alshara, Abdelhak-Djamel Seriai, Chouki Tibermacine, Hinde Lilia Bouziane, Christophe Dony, and Anas Shatnawi. Migrating large object-oriented applications into component-based ones: Instantiation and inheritance transformation. *SIGPLAN Notices*, 51(3):55–64, October 2015.

[6] Nicolas Anquetil, Jean-Claude Royer, Pascal Andre, Gilles Ardourel, Petr Hnetynka, Tomas Poch, Dragos Petrascu, and Vladiela Petrascu. Javacompext: Extracting architectural elements from java source code. In *16th Working Conference on Reverse Engineering, 2009. WCRE'09.*, pages 317–318. IEEE, 2009.

[7] Rehman Arshad and Kung-Kiu Lau. Reverse engineering encapsulated components from object-oriented legacy code. In *Proceedings of The 30th International Conference on Software Engineering and Knowledge Engineering, 2018*. KSI Research Inc., 2018.

[8] Fatemeh Asadi, Massimiliano Di Penta, Giuliano Antoniol, and Yann-Gaël Guéhéneuc. A heuristic-based approach to identify concepts in execution traces. In *14th European Conference on Software Maintenance and Reengineering (CSMR), 2010*, pages 31–40. IEEE, 2010.

[9] Johannes Bohnet and Jürgen Döllner. Analyzing feature implementation by visual exploration of architecturally-embedded call-graphs. In *Proceedings of the 2006 international workshop on Dynamic systems analysis*, pages 41–48. ACM, 2006.

[10] Johannes Bohnet, Stefan Voigt, and Jurgen Dollner. Locating and understanding features of complex software systems by synchronizing time-, collaboration-and code-focused views on execution traces. In *The 16th IEEE International Conference on Program Comprehension, 2008. ICPC 2008.*, pages 268–271. IEEE, 2008.

[11] Hugo Bruneliere, Jordi Cabot, Grégoire Dupé, and Frédéric Madiot. Modisco: A model driven reverse engineering framework. *Information and Software Technology*, 56(8):1012–1032, 2014.

[12] Elizabeth Burd and Malcolm Munro. Investigating component-based maintenance and the effect of software evolution: a reengineering approach using data clustering. In *Proceedings of International Conference on Software Maintenance, 1998.*, pages 199–207. IEEE, 1998.

[13] Kunrong Chen and Václav Rajlich. Case study of feature location using dependence graph, after 10 years. In *18th International Conference on Program Comprehension*. IEEE, 2010.

[14] Elliot J. Chikofsky and James H Cross. Reverse engineering and design recovery: A taxonomy. *IEEE software*, 7(1):13–17, 1990.

[15] Landry Chouambe, Benjamin Klatt, and Klaus Krogmann. Reverse engineering software-models of component-based systems. In *12th European Conference on Software Maintenance and Reengineering, 2008. CSMR 2008.*, pages 93–102. IEEE, 2008.

[16] Markus Detten, Marie Christin Platenius, and Steffen Becker. Reengineering component-based software systems with archimetrix. *Software Systems Model.*, 13(4):1239–1268, October 2014.

[17] Marc Eaddy, Alfred V Aho, Giuliano Antoniol, and Yann-Gaël Guéhéneuc. Cerberus: Tracing requirements to source code using information retrieval, dynamic analysis, and program analysis. In *The 16th IEEE International Conference on Program Comprehension, 2008. ICPC 2008.*, pages 53–62. IEEE, 2008.

[18] Helen M. Edwards and Malcolm Munro. RECAST: Reverse engineering from COBOL to SSADM specification. In *Proceedings of 15th International Conference on Software Engineering, 1993*, pages 499–508, May 1993.

[19] Alexander Egyed. A scenario-driven approach to trace dependency analysis. *IEEE Transactions on Software Engineering*, 29(2):116–132, 2003.

[20] Alexander Egyed, Gernot Binder, and Paul Grunbacher. Strada: A tool for scenario-based feature-to-code trace detection and analysis. In *Companion to the proceedings of the 29th International Conference on Software Engineering*, pages 41–42. IEEE Computer Society, 2007.

[21] Thomas Eisenbarth, Rainer Koschke, and Daniel Simon. Derivation of feature component maps by means of concept analysis. In *Fifth European Conference on Software Maintenance and Reengineering, 2001.*, pages 176–179. IEEE, 2001.

[22] Thomas Eisenbarth, Rainer Koschke, and Daniel Simon. Locating features in source code. *IEEE Transactions on Software Engineering*, 29(3):210–224, 2003.

[23] Andrew David Eisenberg and Kris De Volder. Dynamic feature traces: Finding features in unfamiliar code. In *Proceedings of the 21st IEEE International Conference on Software Maintenance, 2005. ICSM 2005.*, pages 337–346. IEEE, 2005.

[24] Ural Erdemir, Umut Tekin, and Feza Buzluca. Object oriented software clustering based on community structure. In *2011 18th Asia-Pacific Software Engineering Conference*, pages 315–321, Dec 2011.

[25] Jean Marie Favre, Frederic Duclos, Jacky Estublier, Remy Sanlaville, and Jean Jacques Auffret. Reverse engineering a large component-based software product. In *Proceedings Fifth European Conference on Software Maintenance and Reengineering*, pages 95–104, 2001.

[26] Gerald C Gannod and Betty HC Cheng. A framework for classifying and comparing software reverse engineering and design recovery techniques. In *Proceedings of Sixth Working Conference on Reverse Engineering, 1999.*, pages 77–88.

IEEE, 1999.

[27] Orla Greevy, Stéphane Ducasse, and Tudor Girba. Analyzing feature traces to incorporate the semantics of change in software evolution analysis. In *Proceedings of the 21st IEEE International Conference on Software Maintenance, 2005. ICSM 2005.*, pages 347–356. IEEE, 2005.

[28] S. Kebir, A. D. Seriai, S. Chardigny, and A. Chaoui. Quality-centric approach for software component identification from object-oriented code. In *2012 Joint Working IEEE/IFIP Conference on Software Architecture and European Conference on Software Architecture*, pages 181–190, Aug 2012.

[29] Rudolf K Keller, Reinhard Schauer, Sébastien Robitaille, and Patrick Pagé. Pattern-based reverse-engineering of design components. In *Proceedings of the 21st international conference on Software engineering*, pages 226–235. ACM, 1999.

[30] Haeng-Kon Kim and Youn-Ky Chung. Transforming a legacy system into components. In Marina Gavrilova, Osvaldo Gervasi, Vipin Kumar, C. J. Kenneth Tan, David Taniar, Antonio Laganá, Youngsong Mun, and Hyunseung Choo, editors, *Computational Science and Its Applications - ICCSA 2006*, pages 198–205, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[31] Soo Dong Kim and Soo Ho Chang. A systematic method to identify software components. In *11th Asia-Pacific Software Engineering Conference*, pages 538–545, Nov 2004.

[32] Sameer Kumar and Promma Phrommathed. *Research methodology*. Springer, 2005.

[33] Roberto Erick Lopez-Herrejon, José A Galindo, David Benavides, Sergio Segura, and Alexander Egyed. Reverse engineering feature models with evolutionary algorithms: An exploratory study. In *Search Based Software Engineering*, pages 168–182. Springer, 2012.

[34] Stacy K Lukins, Nicholas A Kraft, and Letha H Etzkorn. Source code retrieval for bug localization using Latent Dirichlet Allocation. In *Reverse Engineering, 2008. WCRE'08. 15th Working Conference on*, pages 155–164. IEEE, 2008.

[35] Brian S. Mitchell and Mancoridis Spiros. On the automatic modularization of software systems using the bunch tool. *IEEE Transactions on Software Engineering*, 32(3):193–208, March 2006.

[36] Hausi A Müller, Mehmet A Orgun, Scott R Tilley, and James S Uhl. A reverse-engineering approach to subsystem structure identification. *Journal of Software: Evolution and Process*, 5(4):181–204, 1993.

[37] Andrzej Olszak and Bo Nørregaard Jørgensen. Featureous: a tool for feature-centric analysis of java software. In *18th International Conference on Program Comprehension (ICPC), 2010*, pages 44–45. IEEE, 2010.

[38] Denys Poshyvanyk and Andrian Marcus. Combining formal concept analysis with information retrieval for concept location in source code. In *15th IEEE International Conference on Program Comprehension, ICPC 2007.*, pages 37–48. IEEE, 2007.

[39] Martin P Robillard. Automatic generation of suggestions for program investigation. In *ACM SIGSOFT Software Engineering Notes*, volume 30, pages 11–20. ACM, 2005.

[40] Martin P Robillard and Gail C Murphy. Concern graphs: finding and describing concerns using structural program dependencies. In *Proceedings of the 24th international conference on Software engineering*, pages 406–416. ACM, 2002.

[41] Abhishek Rohatgi, Abdelwahab Hamou-Lhadj, and Juergen Rilling. An approach for mapping features to code based on static and dynamic analysis. In *The 16th IEEE International Conference on Program Comprehension, ICPC 2008.*, pages 236–241. IEEE, 2008.

[42] Maher Salah, Spiros Mancoridis, Giuliano Antoniol, and Massimiliano Di Penta. Scenario-driven dynamic analysis for comprehending large software systems. In *Software Maintenance and Reengineering, 2006. CSMR 2006. Proceedings of the 10th European Conference on*, pages 10–pp. IEEE, 2006.

[43] Anas Shatnawi, Abdelhak-Djamel Seriai, Houari Sahraoui, and Zakarea Alshara. Reverse engineering reusable software components from object-oriented apis. *Journal of Systems and Software*, 131:442–460, 2017.

[44] Steven She, Rafael Lotufo, Thorsten Berger, Andrzej Wasowski, and Krzysztof Czarnecki. Reverse engineering feature models. In *2011 33rd International Conference on Software Engineering (ICSE)*, pages 461–470. IEEE, 2011.

[45] Mircea Trifu. Improving the dataflow-based concern identification approach. In *13th European Conference on Software Maintenance and Reengineering, CSMR 2009.*, pages 109–118. IEEE, 2009.

[46] Marco Tulio Valente, Virgilio Borges, and Leonardo Passos. A semi-automatic approach for extracting software product lines. *IEEE Transactions on Software Engineering*, 38(4):737–754, 2012.

[47] Neil Walkinshaw, Marc Roper, and Murray Wood. Feature location and extraction using landmarks and barriers. In *Software Maintenance, 2007. ICSM 2007. IEEE International Conference on*, pages 54–63. IEEE, 2007.

[48] MP Ward and KH Bennett. A practical program transformation system for reverse engineering. In *Reverse Engineering, 1993., Proceedings of Working Conference on*, pages 212–221. IEEE, 1993.

[49] Hironori Washizaki and Yoshiaki Fukazawa. Extracting components from object-oriented programs for reuse in memory-constrained environments. 2014.

[50] Yinxing Xue, Zhenchang Xing, and Stan Jarzabek. Feature location in a collection of product variants. In *19th Working Conference on Reverse Engineering, (WCRE). 2012*, pages 145–154. IEEE, 2012.

[51] Bo Zhang and Martin Becker. Recovar: A solution framework towards reverse engineering variability. In *4th International Workshop on Product Line Approaches in Software Engineering, (PLEASE), 2013*, pages 45–48. IEEE, 2013.

[52] Wei Zhao, Lu Zhang, Yin Liu, Jiasu Sun, and Fuqing Yang. Sniafl: Towards a static noninteractive approach to feature location. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 15(2):195–226, 2006.

[53] Tewfik Ziadi, Christopher Henard, Mike Papadakis, Mikal Ziane, and Yves Le Traon. Towards a language-independent approach for reverse-engineering of software product lines. In *Proceedings of the 29th Annual ACM Symposium on Applied Computing*, pages 1064–1071. ACM, 2014.

Do Experienced Programmers put too Much Confidence in Comments?

Elia Eiroa-Lledo, Abby Bechtel, Emily Daskas, Lily Foster,
Raha Pirzadeh, Katie Rodeghiero, Erik Linstead

Fowler School of Engineering, Chapman University
E-mail: mlatlab@chapman.edu

Abstract

We present the results of a small eye-tracking study of novice and experienced programmers asked to deduce the output of Python and Java code snippets. We observe experienced programmers paying more visual attention to code comments, and when the comments provided are purposefully misleading, the experienced programmers are more likely to incorrectly describe the code output than their novice counterparts. While preliminary, these results suggest that experienced programmers, who are well-trained in the importance of documentation as part of the software development process, may have an initial tendency to put too much confidence in code comments when faced with program comprehension tasks.

Keywords- program comprehension, eye tracking, code comments

1. Introduction

As part of their formal training, software engineers learn that well-commented, easily comprehensible code is equally as important as code that runs efficiently and correctly. This is reinforced in professional practice, where software deliverables are accompanied by substantial amounts of documentation, both inside the source code as comments, and outside the source code as API specifications and reference manuals. Such documentation provides valuable insight into the function and usage of code, and becomes a critical resource for programmers who must thoroughly understand pieces of existing code as part of software development activities. This leads to two natural questions.

- How much attention do programmers pay to source code comments when tasked with comprehending a piece of code they did not write?

- Does experience level impact the level of reliance or confidence a programmer has in comments when faced with a program comprehension task?

We conducted a small sample (N=14) eye-tracking study to gain further insight into how developers read Python and Java code and, in particular, what captures their visual attention as part of program comprehension tasks. We observed that when experienced programmers are presented with code and asked to determine its output, they often focus on comments before moving on to code to formulate their response. In contrast, novice programmers often focus on the source code itself. When subsequently asked to verbally describe the purpose of a code snippet they had just read, experienced programmers in our study were more likely to base their answer on the comments accompanying the code, even if the comments were wrong! In this paper, we present some visual attention patterns of programmers of different skill levels from our study and explore whether experience with programming and comfort reading code can lead to too much faith in source code comments. Our results, while preliminary, indicate that, on average, experienced programmers spend a significantly longer amount of time focusing on documentation during a program comprehension task relative to novices and are more likely to be initially mislead by comments that are technically incorrect.

2. Experimental Design and Data Collection

We recruited a convenience sample of 14 programmers from a computer science department at a University. Hence, all of our participants have had formal training in programming and software development at the collegiate level. We segmented our participants into two groups: novice and experienced programmers. We defined a novice coder as someone who has been coding for less than four years and has not had industry experience as a software engineer. We defined an experienced coder as someone who has had more than four continuous years of programming usage and has had industry experience as a software engineer. Our sample

was evenly split between novice and experienced programmers but, there was only one female programmer in each of the categories.

All of the participants were comfortable with coding in Java and Python. Our experts averaged 29 years in age with an average coding experience of 12.3 years (median = 7 years) and reported spending around 10 hours of their free time coding per week. The novice coders averaged 21 years in age with an average coding experience of 1.9 years (median = 1 year) and reported spending around 2 hours of their free time coding per week.

To begin, each participant of the study took a survey which included questions pertaining to their demographics, educational background, and programming experience. After answering the survey, each participant was set up at a computer monitor where the test would take place and their eyes were calibrated on the EyeLink 1000 Plus eye-tracker. After this, a small set of instructions was read to the participant and the test began. There were four questions of interest in our study, all of which contained a block of code with an in-line comment. Three of the four questions had comments which were intentionally misleading. Other types of coding questions were randomly dispersed throughout the test as well. Before viewing each block of code, the participants read instructions which prepared them for what they were about to see and how they were expected to answer. The participants moved at their own pace, clicked through the questions, and they answered verbally when they were ready to move on. Their responses were recorded via a microphone. This allowed them to read the code at their own speed and with their natural eye-gaze patterns, without any time pressure or the need to write anything down.

The EyeLink 1000 Plus is a fast, high-precision, video-based eye-tracking device which follows a person's pupils and tracks their eye-gaze. This data is then available for data visualization via the EyeLink DataViewer. This data allowed us to gather information beyond the verbal answers given to us by the participants. We were able to observe, for instance, if they ignored the comments and only looked at the code or if they read both but, based on their verbal answers, decided to put their trust in one over the other. Ultimately, the data we collected was a compilation of the eye-tracker data and the recorded responses from the participants.

3. Analysis and Discussion

We asked our participants to describe the output of some code snippets; what they did not know is that some of the code had misleading comments. When inspecting the code, novice coders tended to read both the code and the comments before answering the prompt. Although some based their answer on the misleading comments at first, most of them either quickly backtracked their answer correcting it or, after the first question, caught on to the fact that the comments were not correctly describing the function of the code. Some novice coders did fall for the comments every time. This, we deduce, is because they either were not making a concerted effort to correctly answer the questions, or that they were unable to deduce the output of the code regardless of the comments.

The experienced coders, on the other hand, were extremely confident in their answers and moved on as soon as they answered for the first time. Furthermore, none of them called out the misleading comments at any point in the experiment. When discussing the experiment after it was over, some expert coders claimed that they never read comments, even though they had just responded to the questions in accordance with the comments.

Table 1

Question	Proportion of Coders that were Mislead	
	Experts	**Novices**
1	100.000%	57.143%
2	71.429%	28.571%
3[a]	N/A	N/A
4	85.714%	28.571%

a. this question was not misleading

Table 1 shows the percentage of participants who answered each question according to the comments. The majority of experts followed the cues given by the comments every time, whereas most of the novices did not. It is important to note that this reflects the percentage of coders who based their answer on the comments, not whether they got the question wrong. Some participants answered incorrectly but without using the comments.

When deciding whether a programmer based their answer on the comments or not, we considered their verbal responses. Our misleading comments were either the opposite of what the program was doing (figure 1) or contained unrelated keywords such as "checking for prime numbers" that were in fact not pertinent to the function of the code (figure 2). In the first case, if the participant said the program prints a list of even numbers, we determined that they were misled by the comment. In the second case, if the programmer used these spurious keywords in their answer, it suggested that they were basing their response on the comments. This is because the keywords were chosen to be orthogonal to the true function of the code being presented. Therefore, there was no reason that the program should have triggered these

```
public class Class2 {
    public static void main(String[] args) {
        //define the limit
        int limit = 50;
        //prints all even numbers from 1 to the limit
        for(int i=1; i <= limit; i++) {
            //if the number is divisible by 2 then it is even
            if(i % 2 != 0) {
                System.out.print(i + " ");
            }
        }
    }
}
```

Figure 1: Example of a misleading comment being the opposite of the truth

```
public class Class1 {
    public static void main(String[] args) {
        int i = 1, n = 10, t1 = 0, t2 = 1;
        System.out.print(n);
        //condition for sum to be the sum of two prime numbers
        while (i <= n) {
            System.out.print(t1 + " + ");
            int sum = t1 + t2;
            t1 = t2;
            t2 = sum;
            i++;
        }
    }
}
```

Figure 2: Example of a misleading comment being completely different than the actual code

responses.

Our observations are further supported by looking at a series of statistics from our participants. When looking at the average proportion of time spent on comments versus the "meat" of the code for each of the two groups, we can see a big difference (Table 2). In all of the questions, experts spent as much as 13.9% more time on comments than novices. This is a big difference, especially when considering that experts spent less time, on average, on every question. When looking at the average over all questions, experts spent over 7% more time reading comments than novices. It is notable to point out that although most novices did not fall for the comments after the first question, they still read the comments in every question. This could be because they were curious to see if they were correct about them being wrong.

The emphasis put on comments by our experienced participants is further visualized by the aggregate heat maps produced. Using EyeLink DataViewer, we produced heat maps for the images inspected by each of the subjects. To create these heat maps, we applied a two-dimensional Gaussian distribution to each of the fixations. The center is the location of the fixation. The width is regulated by a sigma value of degrees of visual angle, making it so that the area affected by the fixation increases as the sigma value increases. The duration of the individual fixations then

Table 2

Question	Relative Time Spent on Comments		
	Experts	Novices	Difference
1	63.519%	49.541%	13.978%
2	27.057%	26.936%	0.121%
3[a]	26.090%	15.242%	10.848%
4	17.461%	13.687%	3.774%
Average	33.532%	26.352%	7.180%

a. this question was not misleading

weighs the height of the Gaussian. This 2D Gaussian is added to an internal map by adding weight to that area of the map. This process is then applied to all fixation points and is normalized.
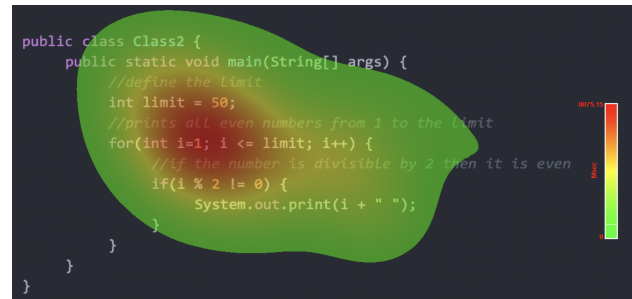
Experts
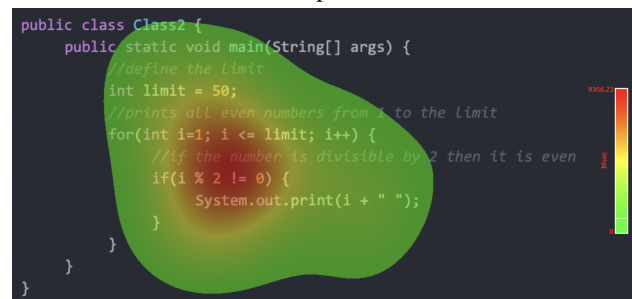
Novices

Figure 3: Heat map showing where experts (top) and novices (bottom) spent the most time looking for question 1

Figure 3 is the aggregate heat map for experienced and novice coders for question 1. In this graph, it is quite apparent that the experts paid significant attention to the comment while paying little to no attention to the core of the program. This bias toward the comment becomes more apparent when we compare this map to the aggregate map for

the novice coders. In the novices map, we can appreciate the attention that the subjects put on the source code as opposed to the comments. The experts clearly read all three comments and came up with their answer, whereas the novices read the whole snippet but paid more attention to the functioning parts of the code. Since these heat maps are for question 1, the novices had not discovered that the comments were misleading. If we look at the aggregate maps for question 2, we see an even bigger difference in the visual pattern, although the proportion of time spent does not differ much between the two groups.
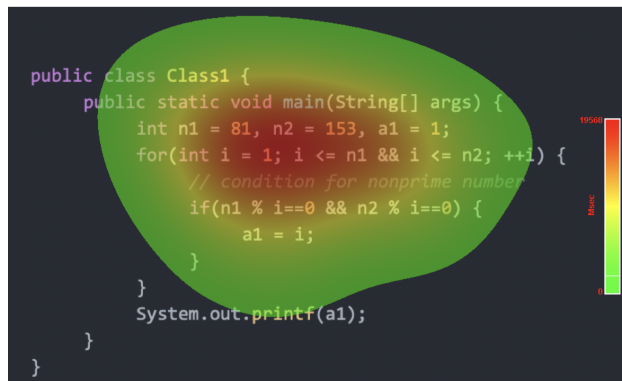
Experts
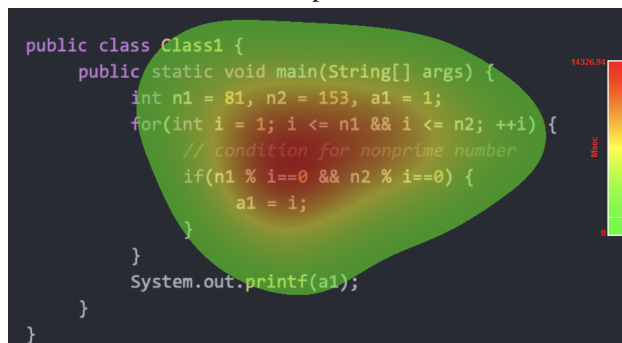
Novices

Figure 4: Heat map showing where experts (top) and novices (bottom) spent the most time looking for question 2

Figure 4 is the aggregate heat map for the experts and novices for question 2. Here, it is clear that the experts read the comment and scanned the code whereas the novices spent most of their time on the actual code. Further, in this specific question, it appears the experts read the first few lines of code and the comment, and then they felt confident enough to answer without analyzing the rest of the code. The most common answer given by expert coders for this question was "this program gives you every non-prime number between n1 = 81 and n2= 153". Taking a look at the code it is obvious that this program is not checking whether

a number is prime or not. It is calculating the greatest common denominator of n1 = 81 and n2= 153. We determined if a coder was misled by the comment if their answer contained either of the key words "prime" or "non-prime".

These results, while based on a small sample, are both important and surprising. Before starting the experiment, we hypothesized that the experts would look at the short code snippets and would be able to easily determine their functionality. We also thought that novices would be more inclined to look at the comments because they are less comfortable with coding in general and would want the help and insights that comments typically provide. Our observations indicated the contrary. These results are evidence of the emphasis put on comments in a work environment. Although all students are taught that comments are important in their computer science classes, this importance does not necessarily materialize in school-based activities centered on small-scale programming tasks. When in school, students usually only look at and work on their own code. In this style of working, code comments are not imperative to integration activities. Comments become critical in industry, however, where developers are expected to work on code that they did not write, or code that they themselves wrote a long time before. Therefore, it makes sense that professional developers would be more prone to look at comments to guide their answer. This inclination to look at comments is a good thing, so long as the comments are accurate and up to date. Our results emphasize the importance of updating comments when editing code, and to write accurate comments when programming, as developers are quick to trust comments that are technically unsound.

4. Related Works

Eye-tracking has been extensively used in modeling how programmers visually process source code as part of software development activities. In a survey paper exploring 63 studies published between 1990 and 2017 regarding eye-tracking in programming, researchers found that the majority of eye-tracking studies could be categorized into five general topics: code/program comprehension, debugging, non-code comprehension, collaboration, and traceability [9]. Additionally, researchers uncovered a pattern showing that many of the published studies using eye-tracking are based on the same concepts being retested with new data. Our research incorporates each of these five themes, while also introducing a new topic of inline comments.

A similar paper analyzed the various approaches to conducting and using eye movement data in the context of source code [5, 6]. When designing studies using eye movement data, researchers concluded that methods yielding qualitative data were just as important as methods yielding quantitative data. Our research supports similar method-

ologies discussed in the paper by the utilization of eye-tracking data as well as vocal recordings of each participant's thought process. Another example in the research compared expert and novice programmers and how linearly they read source code [3]. The study found that while natural language is read linearly, novice programmers read the source code less linearly, and expert programmers even less linearly. Similarly, our research compared expert and novice programmers, mainly focusing on how susceptible programmers are to misleading comments.

Previously, researchers found that programmers reviewing code skim the file until a snippet of code prompted them to slow down and review a section more thoroughly. These triggers included identifiers, inconsistent code changes, and other confusing or incorrect code [2]. They also found a correlation between the time of the initial scan of a program and the efficiency of identifying errors within code reviews [13]. The researchers concluded that programmers are often not thorough when reviewing code. Their findings show that the eye tends to follow the source code left to right, scan for methods, and jump around to keywords [11]. Additionally, many programmers read about 73% of the code within the first 30% of the review time. Those who took more time scanning were able to identify the errors more efficiently [13].

Another study found that the way a programmer read code appeared to be done in a sequence of patterns such as scanning, jumping ahead and back to look and verify details [8]. This pattern is similar to how individuals read natural text, with the exception that some programmers parse code bottom to top opposed from top to bottom [11]. A workshop analyzing how novices comprehend code also concluded that the participants would approach the code as natural text at first but then utilize more sophisticated patterns as they become more familiar with Java [7].

Moreover, a study comparing differences in reading code found that programmers were more likely to read from top to bottom when they had formed a hypothesis about the code and its function and from bottom to top when they had no understanding of the code and needed to sift through it. The expert programmers tended to focus more on a broader block of code while the novice programmers read line by line [1]. This is corroborated by our study where novices read the whole code snippet, including comments, even when they knew they were misleading. Opposed to our experts that seemed to answer after reading a few lines and the comments. In another study focused on analyzing eye movement when reading Java code, researchers found that the lines with a method call, an if statement, or a variable were fixated on the most [10]. Moreover, attention was focused on the understanding of operators, keywords, and literals, while minimal time was spent on separators [4].

Researchers have been looking for a way to analyze the cognitive processes of developers while they interact with software artifacts. Eye-tracking allows for the visualization of gaze pathways as programmers review code, but it lacks a way to map physiological data to the source code. The developers of VITALISE created a Javascript program that allows researchers the ability to record biometric data from EEG, fMRI, fNIRS, and other measurement devices, mapping it against data recorded in the form of heatmaps from an eye-tracking device. This application of neuroimaging opens doors to understanding the cognitive load on developers as they program [12].

5. Future Work

Due to the nature of the study, some limitations should be noted. First, we used a convenience sample size of both experienced and novice coders from the same institution. It is possible that this sample is not representative of the entirety of the target population and cannot be generalized to other subject groups. Second, while the experiment was conducted in a uniform manner across all 14 subjects, the reality is that the small sample size is a restriction on the certainty with which we are able to say that experienced coders visually pay more attention to comments while reading code snippets. Although we saw a statistically significant difference in the percentage of experienced coders and novice coders who were mislead by the comments, a larger sample size will be necessary to further validate the preliminary results reported here. These limitations are motivation for continuing work in this area.

The results we obtained from this experiment have left the door open for future adaptations and extensions of this study. Using short, simple functions with single in-line comments, we found that experienced coders have a tendency to place a large amount of trust in comments, while novice coders rely more on code. Would these results, however, remain the same if we changed the scope of the code or the commenting style?

It could be interesting to investigate if the programmers would utilize the comments in the same way if the scope of the code changed from the short functions we used in this experiment to longer, more complex programs. For instance, it is possible that we would see that the same novice coders who relied on the code in this experiment might begin to rely more and more on comments as the code shifts from straight-forward to increasingly more complex. How much time and effort needs to be put into reading code before a coder who usually relies on code alone decides to put their trust into comments? A future study revealing whether there is a point in which the length or complexity of a program changes the way a programmer utilizes comments may produce informative results.

The scope of a program, however, may not be the only

factor worth investigating. It could also be noteworthy to see if commenting style plays a role in the trust programmers place in comments. Would our experiment have produced the same results if we used descriptive block comments at the top of each function instead of single, in-line comments? Future works could lead us to exploring if commenting style is simply a preference and that comments are ultimately utilized in the same way, by the same people, or if it affects who reads them and the extent of which they are trusted.

Finally, it may also be interesting to look into how aware coders are of how they use comments. If at the conclusion of this study, for example, we asked each participant about how important comments are to them and how heavily they rely on them, would their responses align with how they performed? Perhaps experienced coders are so familiar with reading code and comments that, especially when the code is fairly straight forward, it becomes somewhat of a mindless activity and they are not fully aware of the extent to which they are using them. This would be yet another interesting adaptation of this study which could provide more information into why we obtained the results that we did.

6. Conclusion

We conducted a small study to gain insight into how programmers read code. In particular, we aimed to find how much a coder relies on and, ultimately, trusts comments, and whether this behavior is related to experience. Using a research-grade, we found that experienced coders in our sample were quick to base their knowledge of a program according to the accompanying comments, even when the comments are wrong. In some cases, they appeared to neglect looking at source code completely. Novice coders, on the other hand, were more likely to accurately describe a program, as they prioritised scanning through and reading the source code, even if they had previously read the comments. Our experiment, while small, has shown that experienced coders feel more confident in relying on comments when working through a program comprehension task, while novice coders tend to be more cautious and use comments in careful conjuction with the source code itself.

References

[1] N. J. Abid, J. I. Maletic, and B. Sharif. Using developer eye movements to externalize the mental model used in code summarization tasks. In *Proceedings of the 11th ACM Symposium on Eye Tracking Research & Applications*, pages 1–9, 2019.

[2] A. Begel and H. Vrzakova. Eye movements in code review. In *Proceedings of the Workshop on Eye Movements in Programming*, pages 1–5, 2018.

[3] T. Busjahn, R. Bednarik, A. Begel, M. Crosby, J. H. Paterson, C. Schulte, B. Sharif, and S. Tamm. Eye movements in code reading: Relaxing the linear order. In *2015 IEEE 23rd International Conference on Program Comprehension*, pages 255–265. IEEE, 2015.

[4] T. Busjahn, R. Bednarik, and C. Schulte. What influences dwell time during source code reading? analysis of element type and frequency as factors. In *Proceedings of the Symposium on Eye Tracking Research and Applications*, pages 335–338, 2014.

[5] T. Busjahn, C. Schulte, and E. Kropp. Developing coding schemes for program comprehension using eye movements. In *PPIG*, page 15, 2014.

[6] T. Busjahn, C. Schulte, B. Sharif, A. Begel, M. Hansen, R. Bednarik, P. Orlov, P. Ihantola, G. Shchekotova, and M. Antropova. Eye tracking in computing education. In *Proceedings of the tenth annual conference on International computing education research*, pages 3–10, 2014.

[7] T. Busjahn, C. Schulte, S. Tamm, and R. Bednarik. Eye movements in programming education ii: Analyzing the novice's gaze. 2015.

[8] A. Jbara and D. G. Feitelson. How programmers read regular code: a controlled experiment using eye tracking. *Empirical software engineering*, 22(3):1440–1477, 2017.

[9] U. Obaidellah, M. Al Haek, and P. C.-H. Cheng. A survey on the usage of eye-tracking in computer programming. *ACM Comput. Surv.*, 51(1), Jan. 2018.

[10] C. S. Peterson, N. J. Abid, C. A. Bryant, J. I. Maletic, and B. Sharif. Factors influencing dwell time during source code reading: a large-scale replication experiment. In *Proceedings of the 11th ACM Symposium on Eye Tracking Research & Applications*, pages 1–4, 2019.

[11] P. Rodeghero and C. McMillan. An empirical study on the patterns of eye movement during summarization tasks. In *2015 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, pages 1–10. IEEE, 2015.

[12] D. Roy, S. Fakhoury, and V. Arnaoudova. Vitalse: Visualizing eye tracking and biometric data.

[13] H. Uwano, M. Nakamura, A. Monden, and K.-i. Matsumoto. Analyzing individual performance of source code review using reviewers' eye movement. In *Proceedings of the 2006 symposium on Eye tracking research & applications*, pages 133–140, 2006.

Formal verification of an abstract version of Anderson protocol with CafeOBJ, CiMPA and CiMPG

Duong Dinh Tran, Kazuhiro Ogata
School of Information Science
Japan Advanced Institute of Science and Technology (JAIST)
1-1 Asahidai, Nomi, Ishikawa 923-1292, Japan
Email: {duongtd,ogata}@jaist.ac.jp

Abstract— **Anderson protocol is a mutual exclusion protocol. It uses a finite Boolean array shared by all processes and the modulo (or reminder) operation of natural numbers. This is why it is challenging to formally verify that the protocol enjoys the mutual exclusion property in a sense of theorem proving. Then, we make an abstract version of the protocol called A-Anderson protocol that uses an infinite Boolean array instead. We describe how to formally specify A-Anderson protocol in CafeOBJ, an algebraic specification language and how to formally verify that the protocol enjoys the mutual exclusion property in three ways: (1) by writing proof scores in CafeOBJ, (2) with a proof assistant CiMPA for CafeOBJ and (3) with a proof generator CiMPG for CafeOBJ. We mention how to formally verify that Anderson protocol enjoys the property by showing that A-Anderson protocol simulates Anderson protocol.**

Keywords-**algebraic specification language; mutual exclusion protocol; proof assistant; proof generator; proof score**

I. Introduction

Anderson protocol [1] is a mutual exclusion protocol. The protocol uses a finite Boolean array whose size is the same as the number of processes participating in the protocol. It also uses the modulo operation of natural numbers and an atomic operation fetch&incmod. fetch&incmod takes a natural number variable x and a non-zero natural number constant N and atomically does the following: setting x to $(x+1) \% N$, where $\%$ is the modulo operation, and returning the old value of x.

It is challenging to formally verify that Anderson protocol satisfies desired properties, such as the mutual exclusion property, in a sense of theorem proving. This is because the protocol uses a finite array and the modulo operation of natural numbers. Then, we make an abstract version of the protocol by using an infinite Boolean array instead of a finite Boolean array, using fetch&inc instead of fetch&incmod and stopping use of the modulo operation, where fetch&inc is an atomic operation that atomically does the following: setting x to $x + 1$ and returning the old value of x. The abstract version is called A-Anderson protocol or

A-Anderson. A-Anderson is formalized as an observation transition system (OTS) [2], [3], the OTS is specified in CafeOBJ [4] and it is formally verified in three ways that the OTS enjoys the mutual exclusion property with CafeOBJ, CiMPA [5] and CiMPG [5]. CafeOBJ is an algebraic specification language. Its processor is called CafeOBJ. The first implementation of CafeOBJ was done in Common Lisp, while the second implementation was done in Maude [6], a sibling language of CafeOBJ. The second implementation is called CafeInMaude [7]. CafeInMaude Proof Assistant (CiMPA) is a proof assistant for CafeOBJ and CafeInMaude Proof Generator (CiMPG) is a proof generator that takes annotated proof scores in CafeOBJ and generates proof scripts for CiMPA.

Proof scores can be written in a similar way to write programs in a similar sense of Larch Prover (LP) [8]. The proof score approach to formal verification is flexible in this sense. This is one advantage of the approach. The approach, however, has a disadvantage. Proof scores are subject to human errors. What CafeOBJ essentially does for proof scores is reduction. If human users overlook some cases, CafeOBJ does not point them out. To get rid of the disadvantage, CiMPA has been developed. Although CiMPA is not subject to human errors, it is not flexible enough. To make each advantage of proof score and CiMPA available, CiMPG has been developed. Given proof scores that should be annotated a little bit, CiMPG generates proof scripts that are fed into CiMPA. If CiMPA can successfully discharge all goals with the generated proof scripts, the proof scores are correct for the goals.

The rest of the paper is organized as follows: Sect. II mentions Anderson protocol and its abstract version. Sect. III describes the formal specification of the abstract version in CafeOBJ. Sect. IV describes the formal verification that the abstract version enjoys the mutual exclusion property by writing proof scores in CafeOBJ. Sect. V describes the formal verification with CiMPA. Sect. VI describes the formal verification with CiMPG. Sect. VII mentions simulation-based verification between A-Anderson and Anderson protocols. Sect. VIII mentions related work. Sect. IX concludes the paper.

II. ANDERSON PROTOCOL AND ITS ABSTRACT VERSION

We suppose that there are N processes participating in Anderson protocol. The pseudo-code of Anderson protocol for each process i can be written as follows:

Loop "Remainder Section"
    rs : $place[i] := \text{fetch\&incmod}(next, N)$;
    ws : **repeat until** $array[place[i]]$;
        "Critical Section"
    cs : $array[place[i]],$
            $array[(place[i] + 1) \% \text{N}] := \text{false, true}$;

We suppose that each process is located at rs, ws or cs and initially located at rs. $place$ is an array whose size is N and each of whose elements stores one from $\{0, 1, \ldots, N-1\}$. Initially, each element of $place$ can be any from $\{0, 1, \ldots, N-1\}$ but is 0 in this paper. Although $place$ is an array, each process i only uses $place[i]$ and then we can regard $place[i]$ as a local variable to each process i. $array$ is a Boolean array whose size is N. Initially, $array[0]$ is true and $array[j]$ is false for any $j \in \{1, \ldots, N-1\}$. $next$ is a natural number variable and initially set to 0. $\text{fetch\&incmod}(next, N)$ atomically does the following: setting $next$ to $(next + 1) \% N$ and returning the old value of $next$. $x, y := e_1, e_2$ is a concurrent assignment that is processed as follows: calculating e_1 and e_2 independently and setting x and y to their values, respectively.

We also suppose that there are N processes participating in an abstract version of Anderson protocol. The abstract version is called A-Anderson protocol. The pseudo-code of A-Anderson protocol for each process i can be written as follows:

Loop "Remainder Section"
    rs : $place[i] := \text{fetch\&inc}(next)$;
    ws : **repeat until** $array[place[i]]$;
        "Critical Section"
    cs : $array[place[i] + 1] := \text{true}$;

We use an infinite Boolean array $array$ instead of a finite one and do not use %. fetch&inc is used instead of fetch&incmod. $\text{fetch\&inc}(next)$ atomically does the following: setting $next$ to $next + 1$ and returning the old value of $next$. We also suppose that each process is located at rs, ws or cs and initially located at rs. Initially, each element of $place$ can be any natural number but is 0 in this paper, $array[0]$ is true, $array[j]$ is false for any non-zero natural number j and $next$ is 0.

III. SPECIFICATION OF A-ANDERSON PROTOCOL

Each state of A-Anderson protocol can be characterized by the following pieces of information: the location of each process, the value stored in $next$, the value stored in each element of $place$ and the value stored in each element of $array$. Therefore, we use the following observation functions:

```
op pc : Sys Pid -> Label .
op next : Sys -> SNat .
op place : Sys Pid -> SNat .
op array : Sys SNat -> Bool .
```

where `Sys` is the sort of states, `Pid` is the sort of process IDs, `Label` is the sort of rs, ws and cs, `SNat` is the sort of natural numbers and `Bool` is the sort of Boolean values. We do not use any infinite arrays in the specification. Instead, we use the observation function `array` to observe the value stored in each element that is given to `array` as its second argument.

We use one constructor that represents an arbitrary initial state:

```
op init : -> Sys {constr} .
```

`init` is defined in terms of equations, specifying the values observed by the four observation functions in an arbitrary initial state as follows:

```
eq pc(init,P) = rs .
eq next(init) = 0 .
eq place(init,P) = 0 .
eq array(init,I)
= (if I = 0 then true else false fi) .
```

where `P` is a CafeOBJ variable of `Pid` and `I` is a CafeOBJ variable of `SNat`.

We use three transition functions that are also constructors:

```
op want : Sys Pid -> Sys {constr}
op try  : Sys Pid -> Sys {constr}
op exit : Sys Pid -> Sys {constr}
```

The three transition functions capture the actions that each process moves to ws from rs, tries to move to cs from ws and moves back to rs from cs, respectively. The reachable states are composed of the four constructors.

Each of the three transition functions is defined in terms of equations, specifying how the values observed by the four observation functions change. Let `S` be a CafeOBJ variable of `Sys`, `P` & `Q` be CafeOBJ variables of `Pid` and `I` & `J` be CafeOBJ variables of `SNat`.

`want` is defined as follows:

```
ceq pc(want(S,P),Q)
= (if P = Q then ws else pc(S,Q) fi)
if c-want(S,P) .
ceq place(want(S,P),Q)
= (if P = Q then next(S) else place(S,Q) fi)
if c-want(S,P) .
ceq next(want(S,P))
= s(next(S)) if c-want(S,P) .
eq array(want(S,P),I) = array(S,I) .
ceq want(S,P) = S if c-want(S,P) = false .
```

where `c-want(S,P)` is `pc(S,P) = rs`. `s` of `s(next(S))` is the successor function of natural numbers. The equations say that if `c-want(S,P)` is true, the location of `P` changes to ws, the location of each other

process Q does not change, the P's *place* changes to *next*, each other process Q's *place* does not change, *next* is incremented and *array* does not change in the state denoted `want(S,P)`; if `c-want(S,P)` is false, nothing changes.

   `try` is defined as follows:

```
ceq pc(try(S,P),Q)
= (if P = Q then cs else pc(S,Q) fi)
if c-try(S,P) .
eq place(try(S,P),Q) = place(S,Q) .
eq array(try(S,P)) = array(S) .
eq next(try(S,P),I) = next(S) .
ceq try(S,P) = S if c-try(S,P) = false .
```

where `c-try(S,P)` is

```
pc(S,P) = ws and array(S,place(S,P)) = true
```

The equations say that if `c-try(S,P)` is true, the location of P changes to `ws`, the location of each other process Q does not change, *place* does not change, *array* does not change and *next* does not change in the state denoted `try(S,P)`; if `c-try(S,P)` is false, nothing changes.

   `exit` is defined as follows:

```
ceq pc(exit(S,P),Q)
= (if P = Q then rs else pc(S,Q) fi)
if c-exit(S,P) .
eq place(exit(S,P),Q) = place(S,Q) .
eq next(exit(S,P)) = next(S) .
ceq array(exit(S,P),I) =
(if I = s(place(S,P)) then true
 else array(S,I) fi) if c-exit(S,P) .
ceq exit(S,P) = S if c-exit(S,P) = false .
```

where `c-exit(S,P)` is `pc(S,P) = cs`. The equations say that if `c-exit(S,P)` is true, the location of P changes to `rs`, the location of each other process Q does not change, *place* does not change, *next* does not change, the Ith element of *array* is set true if I equals `s(place(S,P))` and each other element of *array* does not change in the state denoted `exit(S,P)`; if `c-exit(S,P)` is false, nothing changes.

IV. FORMAL VERIFICATION WITH PROOF SCORES

Let S be a CafeOBJ variable of `Sys`, P & Q be CafeOBJ variables of `Pid` and I & J be CafeOBJ variables of `SNat`. One desired property A-Anderson protocol should satisfy is the mutual exclusion property that is expressed as follows:

```
eq mutex(S,P,Q)
= ((pc(S,P) = cs and pc(S,Q) = cs)
   implies (P = Q)) .
```

The expression (or the term) says that if there are processes in the critical section, there is one, namely that exists at most one process in the critical section at any given moment.

   To prove that A-Anderson protocol enjoys the property, we need to use the following lemmas:

```
eq inv1(S,P,Q)
= ((pc(S,P) = ws and array(S,place(S,P))
```

```
    = true and (P = Q) = false)
  implies
  (pc(S,Q) = cs or (pc(S,Q) = ws and
   array(S,place(S,Q)) = true)) = false) .
eq inv2(S,P)
= ((pc(S,P) = cs)
   implies (array(S,place(S,P)) = true)) .
eq inv3(S,P,Q)
= ((place(S,P) = place(S,Q) and (P = Q)
    = false)
   implies (place(S,P) = 0)) .
eq inv4(S,P)
= (place(S,P) = next(S)
   implies (next(S) = 0)) .
eq inv5(S,P)
= (place(S,P) < s(next(S))) = true .
eq inv6(S,P)
= (pc(S,P) = cs or (pc(S,P) = ws and
   array(S,place(S,P)) = true))
   implies array(S,next(S)) = false .
eq inv7(S) = array(S,s(next(S))) = false .
eq inv8(S,I,J)
= (array(S,J) = true and I < s(J))
   implies array(S,I) = true .
```

where `s` used in `s(next(S))` and `s(J)` is the successor function of natural numbers.

   We prove `mutex(S,P,Q)` for all reachable states S and all process IDs P & Q by structural induction on S. There are four cases to tackle: (1) `init`, (2) `want`, (3) `try` and (4) `exit`. Let us consider case (3). What to prove is `mutex(try(s, r), p, q)`, where s is a fresh constant of `Sys` representing an arbitrary state and p, q and r are fresh constant of `Pid` representing arbitrary Process IDs. The induction hypothesis is `mutex(s,P,Q)` for all process IDs P & Q. Let us note that s is shared by `mutex(try(s, r), p, q)` and `mutex(s,P,Q)`, while the variables P and Q can be replaced with any terms of `Pid`, such as p and q.

   Case (3) is first split into two sub-cases: (3.1) `pc(s, r) = ws` and (3.2) `(pc(s, r) = ws) = false`. Case (3.2) can be discharged, while it is necessary to split case (3.1) into two sub-cases: (3.1.1) `q = r` and (3.1.2) `(q = r) = false`. It is also necessary to split case (3.1.1) into two sub-cases: (3.1.1.1) `p = r` and (3.1.1.2) `(p = r) = false`. Case (3.1.1.1) can be discharged, while it is still necessary to split (3.1.1.2) into two sub-cases: (3.1.1.2.1) `array(s,place(s,r)) = true` and (3.1.1.2.2) `array(s,place(s,r)) = false`. Case (3.1.1.2.2) can be discharged, but we need to split case (3.1.1.2.1) into two sub-cases again: (3.1.1.2.1.1) `pc(s,p) = cs` and (3.1.1.2.1.2) `(pc(s,p) = cs) = false`. Feeding the proof scores of case (3.1.1.2.1.1) and case (3.1.1.2.1.2) into CafeOBJ, CafeOBJ returns `false` and `true`, respectively. Case (3.1.1.2.1.1) says that process p is located at `cs`, process r (or q since q = r) is located at `ws` and `array(s,place(s,r)) = true`. In case

(3.1.1.2.1.1), process `r` can move to `cs`, breaking the property concerned because there are two processes `p` and `r` located at `cs`. Therefore, we need to conjecture a lemma to discharge case (3.1.1.2.1.1). Such a lemma can be conjectured from the assumptions made in case (3.1.1.2.1.1). We have conjectured `inv1` as such a lemma. The proof score of case (3.1.1.2.1.1) is as follows:

```
open INV .
  op s : -> Sys . ops p q r : -> Pid .
  eq pc(s, r) = ws . eq q = r .
  eq (p = r) = false .
  eq array(s,place(s,r)) = true .
  eq pc(s,p) = cs .
  red inv1(s,r,p)
      implies mutex(s, p, q)
      implies mutex(try(s, r), p, q) .
close
```

In order to discharge case (3.1.2), we need to split it into two sub-cases: (3.1.2.1) `p = r` and (3.1.2.2) `(p = r) = false`. If `p` and `q` are swapped, case (3.1.2.1) becomes exactly the same as case (3.1.1.2). Hence, case (3.1.2.1) can be discharged in the same way as case (3.1.1.2). We also need to use `inv1` as a lemma but should use `inv1(s,r,q)` instead of `inv1(s,r,p)`. The proof score of a sub-case derived from case (3.1.2.1) that corresponds to case (3.1.1.2.1.1) is as follows:

```
open INV .
  op s : -> Sys . ops p q r : -> Pid .
  eq pc(s, r) = ws .
  eq (q = r) = false . eq p = r .
  eq array(s,place(s,r)) = true .
  eq pc(s,q) = cs .
  red inv1(s,r,q)
      implies mutex(s, p, q)
      implies mutex(try(s, r), p, q) .
close
```

(3.1.2.2) is the only unresolved sub-case of case (3). Once again, this case is split into two sub-cases: (3.1.2.2.1) `p = q` and (3.1.2.2.2) `(p = q) = false`. The former can be discharged, while we need to split the latter into two sub-cases: (3.1.2.2.2.1) `array(s,place(s,r)) = true` and (3.1.2.2.2.2) `array(s,place(s,r)) = false`. Both cases can be discharged. Then, case (3) has been discharged.

Case (4) can be discharged in a similar way as case (3) is discharged. We can discharge case (2) without using any lemmas. It is straightforward to discharge case (1). We need to prove `inv1` to complete the formal verification. The proof of `inv1` uses `inv2`, `inv3`, `mutex` and `inv6` as lemmas. `inv2` and `inv5` can be proved independently without use of any other lemmas. The proof of `inv3` uses `inv4` as a lemma. The proof of `inv4` uses `inv5` as a lemma. The proof of `inv6` uses `inv1`, `inv4`, `mutex` and `inv7` as lemmas. The proof of `inv7` uses `inv2`, `inv6` and `inv8` as lemmas. The proof of `inv8` uses `inv2` as a lemma. Let

us note that although the proof of `mutex` uses `inv1` as a lemma and the proof of `inv1` uses `mutex` as a lemma, our argument is not circular. We use simultaneous induction to conduct our proof.

To prove each invariant for an OTS by writing proof scores in CafeOBJ, we first use simultaneous induction on states and do the following: for the base case, it is usually straightforward to discharge the case, and for each induction case, we conduct case splittings and use instances of induction hypotheses (or lemmas) as premises of implications.

It took much less than 1s to run all proof scores with CafeOBJ so as to formally verify that A-Anderson protocol enjoys the mutual exclusion property. The experiment used a computer that carried 3.4GHz microprocessor and 32GB main memory. The same computer was used to conduct the other experiments mentioned in the present paper.

V. FORMAL VERIFICATION WITH CIMPA

The proof score approach to formal verification does not require to explicitly construct proof trees. The outcomes of the approach are open-close fragments written in CafeOBJ that correspond to leaf parts of proof trees. Conducing formal verification by writing proof scores in CafeOBJ, however, we implicitly construct proof trees. Once we have completed formal verification by writing proof scores in CafeOBJ, we must be able to conduct the formal verification with CiMPA. We partially describe formal verification with CiMPA that A-Anderson enjoys the mutual exclusion property.

We first introduce the goals to prove for CiMPA with the command `:goal` as follows:

```
open INV .
:goal{
  eq [inv1 :nonexec]
    : inv1(S:Sys,P:Pid,Q:Pid) = true .
  eq [inv2 :nonexec]
    : inv2(S:Sys,P:Pid) = true .
  ...
  eq [mutex :nonexec]
    : mutex(S:Sys,P:Pid,Q:Pid) = true .
}
```

where the six more lemmas should be written in the place `...`, `inv1`, `inv2` and `mutex` written in square brackets are the names referring to the goals, respectively, and `:nonexec` instructs CafeOBJ not to use the equations as rewrite rules.

Then, we select `S` with the command `:ind on` as the variable on which we start proving the goals by simultaneous induction:

```
:ind on (S:Sys)
:apply(si)
```

The command `:apply(si)` starts the proof by simultaneous induction on `S`, generating four sub-goals for `exit`, `init`, `try` and `want`, where `si` stands for simultaneous

induction. Each sub-goals consists of nine equations to prove. We skip the sequence of commands that discharge the first two sub-goals for `exit` and `init`. We partially describe how to discharge the third sub-goal for `try`. To this end, the first command used is as follows:

```
:apply(tc)
```

where `tc` stands for theorem of constants. The command generates nine sub-goals, one of which is as follows:

```
3-9. TC  eq [mutex :nonexec]:
mutex(try(S#Sys,P#Pid),P@Pid,Q@Pid) = true .
```

The command `:apply(tc)` replaces CafeOBJ variables with fresh constants in goals. `S#Sys` and `P#Pid` are fresh constants introduced by `:apply(si)`, while `P@Pid` and `Q@SNat` are fresh constants introduced by `:apply(tc)`.

To discharge goal 3-9, the following commands are first introduced:

```
:def csb3_9_1 =
 :ctf {eq pc(S#Sys,P#Pid) = ws .}
:apply(csb3_9_1)
:def csb3_9_2 = :ctf {eq Q@Pid = P#Pid .}
:apply(csb3_9_2)
:def csb3_9_3 = :ctf {eq P@Pid = P#Pid .}
:apply(csb3_9_3)
```

Case splittings are carried out based on these three equations. For one generated sub-goal in which we assume that the three equations hold, we use the following commands:

```
:imp [mutex] by
 {P:Pid <- P@Pid ; Q:Pid <- Q@Pid ;}
:apply (rd)
```

The induction hypothesis is instantiated by replacing the variables `P:Pid` and `Q:Pid` with the fresh constants `P@Pid` and `Q@Pid` and the instance is used as the premise of the implication. Then, `:apply(rd)` is used to check if the current goal can be discharged. The goal is discharged in this case. The goal corresponds to case (3.1.1.1) in the last section.

After that, the following commands are written:

```
:def csb3_9_4 =
 :ctf [ array(S#Sys,place(S#Sys,P#Pid)) .]
:apply(csb3_9_4)
:def csb3_9_5 =
 :ctf {eq pc(S#Sys,P#Pid) = cs .}
:apply(csb3_9_5)
```

Case splittings are carried out based on one Boolean term and one equation. For one generated sub-goal in which we assume that the Boolean term is true and the equation holds, we use the following commands:

```
:imp [inv1] by
 {P:Pid <- P#Pid ; Q:Pid <- P@Pid ;}
:imp [mutex] by
 {P:Pid <- P@Pid ; Q:Pid <- Q@Pid ;}
:apply (rd)
```

The lemma `inv1` is instantiated by replacing the variables `P:Pid` and `Q:Pid` with the fresh constants `P#Pid` and `Q#Pid` and the instance is used as the premise of the implication. Next, the induction hypothesis is instantiated by replacing the variables `P:Pid` and `Q:Pid` with the fresh constants `P@Pid` and `Q@Pid` and the instance is used as the premise of the implication. Then, `:apply(rd)` is used to check if the current goal can be discharged. The goal is discharged in this case. The goal corresponds to case (3.1.1.2.1.1) in the last section.

When CiMPA is used to formally verify invariant properties for an OTS, what to do is essentially the same as we do formal verification by writing proof scores in CafeOBJ. The difference is as follows: it is necessary to use the commands given by CiMPA when CiMPA is used.

It took about 22s to run the proof scripts with CiMPA so as to formally verify that A-Anderson protocol enjoys the mutual exclusion property.

VI. FORMAL VERIFICATION WITH CiMPG

After writing proof scores that A-Anderson protocol enjoys the mutual exclusion property, we can confirm that the proof scores are correct by doing the formal verification with CiMPA as described in the last section. Although we are able to conduct the formal verification with CiMPA once we have completed formal verification by writing proof scores in CafeOBJ, it would be preferable to automatically confirm the correctness of proof scores. CiMPG makes it possible to automatically confirm the correctness of proof scores by generating proof scripts for CiMPA from the proof scores.

To use CiMPG, we need to add one open-close fragment to the proof scores. The open-close fragment is as follows:

```
open INV .
  :proof(ander)
close
```

Moreover, we need to write `:id(ander)` in each open-close fragment. For example, the first open-close fragment used in Sect. IV becomes as follows:

```
open INV .
  :id(ander)
  op s : -> Sys . ops p q r : -> Pid .
  eq pc(s, r) = ws . eq q = r .
  eq (p = r) = false .
  eq array(s,place(s,r)) = true .
  eq pc(s,p) = cs .
  red inv1(s,r,p)
      implies mutex(s, p, q)
      implies mutex(try(s, r), p, q) .
close
```

Feeding the annotated proof scores into CiMPG, CiMPG generates the proof script for CiMPA. The generated proof script is quite similar to the one written manually. Feeding the generated proof script into CiMPA, CiMPA discharges all goals, confirming that the proof scores are correct. It took about 626s to generate the proof script with CiMPG.

VII. A-Anderson Protocol Simulates Anderson Protocol

We can use "simulation-based verification for invariant properties [9]" so as to formally verify that Anderson protocol enjoys the mutual exclusion property. To this end, we first need to prove that the OTS formalizing A-Anderson protocol simulates the OTS formalizing Anderson protocol by showing that there exists a simulation relation from the latter OTS to the former OTS. We next need to prove that the simulation relation preserves the mutual exclusion property. Then, since we have formally verified that A-Anderson protocol enjoys the property, we can conclude that Anderson protocol also enjoys the property. We will describe this part in a longer version of the present paper.

VIII. Related Work

Anderson protocol has been formally specified in CafeOBJ and semi-formally verified with CafeOBJ [10]. Proof scores have been partially written and then all necessary lemmas have not been conjectured and used. They have used a simulation relation between Ticket protocol and Anderson protocol, where the former is abstract, while the latter is concrete. But, they have not used any precise definitions of simulation relations.

In the paper [9] that proposes simulation-based verification for invariant properties in the OTS/CafeOBJ method, Alternating Bit Protocol (ABP), a communication protocol, is used as an example. Two more abstract protocols are used. The paper concludes that it is not very beneficial to use the simulation-based verification technique in order to formally verify that ABP enjoys desired invariant properties. It is useful to use the technique so as to formally verify that Anderson protocol enjoys the mutual exclusion property, however, although the present paper does not describe the part in detail.

Farn Wang [11] proves that it is impossible to automatically formally verify that concurrent software systems as processes running algorithms on data-structures with pointers enjoy desired properties if there are an arbitrary number of processes. Then, he proposes a new automatic approximation method to tackle it. He uses the proposed method to formally verify that a revised version of the MCS mutual exclusion protocol [12] enjoys desired properties. It is one piece of future work to formally verify with the Farn Wang's method that Anderson protocol enjoys the mutual exclusion property and to compare his method with the technique used in the present paper. It is another piece of future work to formally verify that the MCS mutual exclusion protocol enjoys the mutual exclusion property with the technique used in the present paper.

IX. Conclusion

We summarize some lessons learned from the case study. (1) Abstraction makes it possible to tackle the formal verification task. Although we were not able to formally verify that Anderson protocol enjoys the mutual exclusion property by writing proof scores in CafeOBJ, we were able to conduct the formal verification for A-Anderson protocol, an abstract version of Anderson protocol. (2) Our experience says that once we have written all proof scores to prove that A-Anderson protocol enjoys the property, it is rather straightforward to write the proof scripts for CiMPA. (3) Although CiMPG can automatically generate the proof script for CiMPA from proof scores in CafeOBJ, it takes time to do so. One piece of our future work for (2) is to prepare a gentle guide for non-experts to writing proof scripts for CiMPA from their experiences of writing proof scores in CafeOBJ. Another piece of our future work for (2) and (3) is to come up with better annotations to proof scores for CiMPG to more efficiently generate the proof scripts from annotated proof scores.

References

[1] T. E. Anderson, "The performance of spin lock alternatives for shared-memory multiprocessors," *IEEE Trans. Parallel Distrib. Syst.*, vol. 1, no. 1, pp. 6–16, 1990.

[2] K. Ogata and K. Futatsugi, "Proof scores in the OTS/CafeOBJ method," in *FMOODS 2003*, 2003, pp. 170–184.

[3] K. Ogata and K. Futatsugi, "Some tips on writing proof scores in the OTS/CafeOBJ method," in *Algebra, Meaning, and Computation*, 2006, pp. 596–615.

[4] R. Diaconescu and K. Futatsugi, *Cafeobj Report*, ser. AMAST Series in Computing. World Scientific, 1998, vol. 6.

[5] A. Riesco and K. Ogata, "Prove it! inferring formal proof scripts from CafeOBJ proof scores," *ACM Trans. Softw. Eng. Methodol.*, vol. 27, no. 2, pp. 6:1–6:32, 2018.

[6] M. Clavel, et al., Ed., *All About Maude*, ser. Lecture Notes in Computer Science. Springer, 2007, vol. 4350.

[7] A. Riesco, K. Ogata, and K. Futatsugi, "A Maude environment for CafeOBJ," *Formal Asp. Comput.*, vol. 29, no. 2, pp. 309–334, 2017.

[8] S. J. Garland and J. V. Guttag, "An overview of LP, the larch power," in *RTA-89*, 1989, pp. 137–151.

[9] K. Ogata and K. Futatsugi, "Simulation-based verification for invariant properties in the OTS/CafeOBJ method," *Electron. Notes Theor. Comput. Sci.*, vol. 201, pp. 127–154, 2008.

[10] K. Ogata and K. Futatsugi, "Specification and verification of some classical mutual exclusion algorithms with CafeOBJ," in *OBJ/CafeOBJ/Maude Workshop at Formal Methods 1999*, 1999, pp. 159–177.

[11] F. Wang, "Automatic verification of pointer data-structure systems for all numbers of processes," in *World Congress on Formal Methods 1999*, 1999, pp. 328–347.

[12] J. M. Mellor-Crummey and M. L. Scott, "Algorithms for scalable synchronization on shared-memory multiprocessors," *ACM Trans. Comput. Syst.*, vol. 9, no. 1, pp. 21–65, 1991.

Plagiarism Detection of Multi-threaded Programs using Frequent Behavioral Pattern Mining

Qing Wang[1,2], Zhenzhou Tian[1,2*], Cong Gao[1,2], Lingwei Chen[3]

[1]School of Computer Science and Technology, Xi'an University of Posts and Telecommunications, Xi'an, China
[2]Shaanxi Key Laboratory of Network Data Analysis and Intelligent Processing, Xi'an, China
[3]College of Information Sciences and Technology, Pennsylvania State University, PA, USA
*Corresponding: tianzhenzhou@xupt.edu.cn

Abstract—**Software dynamic birthmark techniques construct birthmarks using the captured execution traces from running the programs, which serve as one of the most promising methods for obfuscation-resilient software plagiarism detection. However, due to the perturbation caused by non-deterministic thread scheduling in multi-threaded programs, such dynamic approaches optimized for sequential programs may suffer from the randomness in multi-threaded program plagiarism detection. In this paper, we propose a new dynamic thread-aware birthmark FPBirth to facilitate multi-threaded program plagiarism detection. We first explore dynamic monitoring to capture multiple execution traces with respect to system calls for each multi-threaded program under a specified input, and then leverage Apriori algorithm to mine frequent patterns to formulate our dynamic birthmark, which can not only depict the program's behavioral semantics, but also resist the changes and perturbations over execution traces caused by the thread scheduling in multi-threaded programs. Using FPBirth, we design a multi-threaded program plagiarism detection system. The experimental results based on a public software plagiarism sample set demonstrate that the developed system integrating our proposed birthmark FPBirth cope better with multi-threaded plagiarism detection than alternative approaches.**

Index Terms—**Software plagiarism, Dynamic birthmark, Multi-threaded program, Frequent pattern**

I. INTRODUCTION

As modern social coding platforms, such as GitHub and CodeShare, have been emerging as one of the most vibrant and important information sources to software programming ecosystem, the incentive for the developers to copy or abuse the ready-to-use codes from others to expedite their own software developments increases as well. For example, as revealed in 2018, Redcore, a Chinese startup's "self-made" web browser, was found to plagiarize substantial code from Google Chrome. Due to the openness of Android, application (app) plagiarism has become even more prevalent through repackaging [2] such that about 13% of apps hosted in third-party marketplaces are repackaged [19], which poses serious threats to the healthy development of software industry.

In order to detect the evolving software plagiarism, different birthmarking techniques [10], [7], [12], [15], [4] have been developed. In these methods, software birthmark, which is a set of features, is first extracted from a program to uniquely identify the programs, and then birthmark similarities

are measured to determine the potential plagiarism between the programs. Compared to the static birthmark analysis on programs' lexical, grammatical or structural characteristics, dynamic birthmarking techniques [12], [15], [4] construct birthmarks using the captured execution traces from running the programs, which can depict the behaviors and semantics of the programs more accurately and thus enjoy better anti-obfuscation ability. However, due to the perturbation caused by non-deterministic thread scheduling in multi-threaded programs, existing dynamic approaches optimized for sequential programs may suffer from the randomness in plagiarism analysis for multi-threaded programs [13]. For instance, given an input, birthmarks extracted from multiple runs of the same multi-threaded program can be very different; in the extreme cases, such constructed birthmarks may even fail to detect plagiarism between a multi-threaded program and itself [11]. Two dynamic birthmarking methods (i.e., thread-related system call birthmark (TreSB) [11] and thread-oblivious birthmark (TOB) [13]) have been proposed, yet they still suffer from either weak universality or limitation of overall behavior understanding in multiple threads.

To address the aforementioned challenge, we run a number of multi-threaded programs, and analyze their behaviors, from which we observe that the same input may generally enforce the same program function execution, while not all parts of the program get involved in thread interleaving, so that its multiple execution traces under the same input may be similar, but not identical. This calls for a sophisticated method to characterize the behavioral patterns from multiple execution traces. Inspired by the success of motif recognition in DNA sequence analysis where difference-tolerant motifs are extracted to identify common patterns of DNA sequence variations. In this paper, we would like to shift such a paradigm that generalizes motif formulation to abstract the behaviors of the multi-threaded programs through their execution traces. More specifically, we first explore dynamic monitoring to capture multiple execution traces for each multi-threaded program under the same input, and then elaborate Apriori to extract significant frequent patterns over execution traces, based on which, we construct a thread-aware birthmark, called FPBirth, to model the behavior of the multi-threaded program and reduce the impact of interleaving threads. The contributions of this paper are summarized as follows:
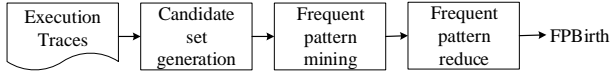
Figure 1: Basic flow of FPBirth extraction.

Figure 2: Basic process of pattern candidate set generation.

- A new and dynamic behavioral representation learning method for multi-threaded programs is proposed over their multiple execution traces through candidate set generation and frequent pattern mining. This allows a refined representation to preserve semantics of execution traces while tolerating differences among them as well.
- Based on extracted frequent patterns, a new thread-aware birthmark FPBirth is constructed, which is leveraged to design a multi-threaded program plagiarism detection system.
- Comprehensive experimental studies on a public software plagiarism sample set are conducted to demonstrate that FPBirth is a reliable thread-aware birthmark, and plagiarism detection system over it can achieve the state-of-the-art results, which also outperforms TreSB and TOB.

II. PROBLEM STATEMENT

In this section, we first define the software plagiarism detection problem. Given two multi-threaded programs p and q, an input I and a thread schedule s to p and q, a thread-aware dynamic software birthmark can be defined as a set of characteristics $f(p, I, s)$ extracted from program p when executing p with the input I and schedule s if and only if both of the following conditions are satisfied [14]:

- $f(p, I, s)$ is obtained only from p itself when executing p with input I and thread schedule s.
- Program q is a copy of $p \Rightarrow f(p, I, s) = f(q, I, s)$.

Obviously, this is an abstract guideline without considering any implementation feasibility. In practice, even if there is a plagiarism correlation between two programs, the constructed birthmarks may not be exactly the same. Therefore, instead of enforcing exact birthmark matching, we measure the similarity between the original program p's birthmark and the suspect program q's birthmark $\mathrm{sim}(f(p, I, s), f(q, I, s))$ to determine the plagiarism. The higher the similarity, the more possible the suspect program q copies code from the original program p. We further set up a threshold ε to obtain the final results:

$$\mathrm{sim}(p_f, q_f) = \begin{cases} \geq 1 - \varepsilon & q \text{ is a copy of } p \\ < \varepsilon & q \text{ is not a copy of } p \quad (1) \\ \text{Otherwise} & \text{Inconclusive} \end{cases}$$

III. PROPOSED METHOD

In this section, we present the detailed method of how we construct thread-aware birthmarks for multi-threaded programs over their execution traces, which is illustrated in Figure 1.

A. Candidate Set Generation

The thread interleaving in multi-threaded programs leads to changes in the program execution traces. To capture such unique behaviors so that the constructed birthmarks are difference-tolerant to the changes among execution traces, we take as input multiple execution traces from a multi-threaded program under the same input, and extract frequent behavioral patterns over execution traces to formulate birthmark. To improve the effectiveness of frequent pattern mining, pattern candidate set is first generated through pre-processor, gram-based slice, and slice merging, which is displayed in Figure 2.
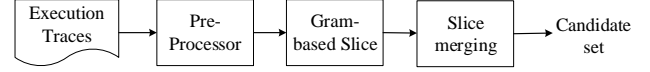
1) Pre-Processor: The pre-processor is to prune the captured execution traces, consisting of system calls related to program and thread operations, where each record in the system call sequence is specified as system call number, name, and return value. However, the raw execution traces are not applicable for direct FPBirth extraction. First, those system calls that fail cannot correctly reflect the program's behaviors [5], which should be considered noises to be filtered out using their return values. Second, those system calls that are invoked randomly may perturb the execution traces, which should be also removed. For example, *futex*, providing a way to keep the thread blocked until certain conditions are met, can be only called when the expected blocking time is long enough; another kind of system calls that are responsible for memory management, such as *mmap* and *brk*, may be invoked only when a particular chunk of memory is involved.

2) Gram-based Slice: Due to its simplicity and scalability, k-gram model [8] in natural language processing is then used to slice up the pre-processed execution traces to form different subsequences of k continuous system calls. Given a pre-processed execution trace $s = (e_1, e_2, \cdots, e_n)$, a series of subsequences split by k-gram can be defined as $\mathrm{grams}(s, k) = \{g_i | g_i = (e_i, e_{i+1}, \cdots, e_{i+k})\}$ $(1 \leq i \leq n - k + 1)$. In this respect, execution traces can be transformed into a set of short sequences to facilitate fast pattern mining while not significantly compromising their important semantic information, which thus greatly ensures the integrity of trace contents.

3) Slice Merging: To generate the candidate set for frequent pattern mining, we further merge all the short sequences sliced by k-gram over multiply execution traces of each multi-threaded program under the same input. In other words, one multi-threaded program with one input will specify one pattern candidate set. As such, given a multi-threaded program p and an input I, a pattern candidate set can be defined as $\mathrm{CanSet}_p^I = \bigcup_{i=1}^{m} \mathrm{grams}(s_i, k)$ where s_i is p's ith execution trace under input I and m is the number of execution traces.

B. Frequent Pattern Mining

Frequent pattern mining is an important research topic in data mining [3], which searches for recurring relationships in a given data set with frequency not less than minimum support threshold, and thus leads to discovery of associations among itemsets. Therefore, based on the generated candidate sets, we

explore a frequent pattern mining method Apriori [1] to dig out the most representative behavioral patterns to birthmark each multi-thread program, which not only preserve semantics of execution traces, but also have strong ability to resist variations caused by thread interleaving.

The key of Apriori is the *apriori* knowledge that all non-empty subsets of a frequent itemset must also be frequent. Therefore, Apriori algorithm follows the iterative steps that frequent t-itemsets (i.e., itemsets that contain t items and have frequency not less than minimum support σ) are generated by joining frequent $(t-1)$-itemsets with itself until no new frequent itemsets are identified. In this way, given a candidate set CanSet_p^I, the generated frequent pattern set over it can be defined as $\mathrm{FreSet}_p^I = \{f_i | \mathrm{count}(f_i) \geq \sigma, 1 \leq i \leq l)\}$ where f_i is ith frequent pattern in CanSet_p^I, and l is the number of frequent patterns in FreSet_p^I.

To perform frequent pattern mining, the length of the input sequences k, which is decided by k-gram slices, must be appropriately considered: (1) excessive length will lead to an explosion in the number of iterations and itemset candidates, and the burden of program running, while (2) the length being too short may enforce short frequent itemset generation; since we utilize frequent itemsets as patterns to construct the birthmark, frequent itemsets being too short will not be able to depict any specific patterns and thus degrade their expressiveness and representativeness to execution traces and the corresponding birthmark's semantics and accuracy to the multi-threaded programs. That is to say, given the input sequences of length k, the length of frequent itemsets t may directly impact on the validity of the constructed birthmark. As such, the length of the input sequences k, and the length range of the frequent itemsets t will be empirically evaluated in the experiments on the sample data to find the best trade-off between the effectiveness and efficiency for multi-threaded program plagiarism detection.

C. Frequent Pattern Reduction

Using frequent pattern mining over CanSet_p^I, we may generate the frequent pattern set FreSet_p^I with a large number of frequent patterns, where according to the implementation of Apriori algorithm, the resulting patterns with shorter length are obviously more than the ones with longer length. On the one hand, shorter patterns are weaker than longer ones in representing program-specific semantic behaviors for less context; on the other hand, shorter patterns themselves may be embedded in longer patterns, which has a major drawback to cause the redundancy, and thus mislead the effect of the constructed birthmark over frequent patterns. Therefore, the removal of such short frequent patterns is indispensable.

More specifically, we here propose a pattern removing method before constructing the birthmark, named insignificant pattern removing, where all the frequent patterns that are included in others as continuous subsequences are insignificant and should be removed. For example, given the pattern "ABCDE", the following pattern "ABC" becomes insignificant

because it is a complete substring and gives no extra information, while the pattern "ADE" will be retained due to its variation on "ABCDE".

Finally, the refined frequent pattern set is used to construct the thread-aware dynamic software birthmark for the program. Note that, for dynamic birthmarks, the number of pattern occurrences is related to the execution behavior of the program to some extent; that is, birthmark similarity should be measured over pattern frequency instead of pattern existence. To facilitate such a similarity calculation, we further transform the frequent pattern set into key-value pair set where the keys represent the frequent patterns and the values refer to their corresponding frequencies. This key-value pair set acts as the program's dynamic birthmark under a specified input, named FPBirth. Accordingly, given a frequent pattern set FreSet_p^I, FPBirth can be defined as $\mathrm{FPBirth}_p^I = \{\langle f_i, \sup(f_i)\rangle | f_i \in \mathrm{FreSet}_p^I\}$ where f_i is ith frequent pattern in FreSet_p^I, and $\sup(f_i)$ is the frequency of pattern f_i (i.e., support count).

IV. FPBirth-based Software Plagiarism Detection

Using FPBirth, we can effectively and dynamically birthmark a multi-threaded program under a specified input. However, a FPBirth birthmark merely abstracts part of the semantics and behaviors of the program under a single input, based on which, the plagiarism detection decision is clearly biased and not reliable. For instance, two different programs may adopt the same standard exception handling mechanism, while any inputs that invoke the exception handling will enforce the same behavioral patterns for both programs. To address this issue, we formulate different inputs and perform multiple executions for each multi-threaded program under each of these inputs to cover as many functional blocks as possible, so that we can construct a series of FPBirth birthmarks to thoroughly represent the semantics and behaviors of the program. Given an original program p, a suspect program q, and a set of inputs $\{I_1, I_2, \cdots, I_d\}$, we accordingly generate a set of FPBirth birthmark pairs for p and q, which can be denoted as $\{(\mathrm{FPBirth}_p^{I_1}, \mathrm{FPBirth}_q^{I_1}), \cdots, (\mathrm{FPBirth}_p^{I_d}, \mathrm{FPBirth}_q^{I_d})\}$. Instead of evaluating the similarity between a single pair of birthmarks, we calculate the similarities for all pairs of birthmarks and take their mean value as the measure of software similarity between p and q, which can be denoted as follows:

$$\mathrm{sim}\,(p_f, q_f) = \sum_{i=1}^{d} \mathrm{sim}\left(\mathrm{FPBirth}_p^{I_i}, \mathrm{FPBirth}_q^{I_i}\right) \Big/ d \quad (2)$$

Based on $\mathrm{sim}\,(p_f, q_f)$ and Eq. (1), we can obtain the final plagiarism detection results, where the threshold ε is adjustable for different sample data set. Note that we aim to outline a general paradigm to explore the similarity between p and q, where the measure models can be instantiated in different ways. In this paper, we employ cosine similarity for measurement, since it is commonly used in high-dimensional positive spaces with the outcome being neatly bounded in $[0, 1]$.

Table I: Benchmark multi-threaded programs

Name	Size(kb)	Version	#Ver	Name	Size(kb)	Version	#Ver	Name	Size(kb)	Version	#Ver
pigz	294	2.3	21	chromium	80,588	28.0.1500.71	1	SOR	593.3	JavaG1.0	44
lbzip	113.3	2.1	1	dillo	610.9	3.0.2	1	blackschole	12.5	Parsec3.0	2
lrzip	219.2	0.608	1	Dooble	364.4	0.07	1	bodytrack	647.5	Parsec3.0	2
pbzip2	67.4	1.1.6	1	epiphany	810.9	3.4.1	1	fludanimate	46.4	Parsec3.0	2
plzip	51	0.7	1	firefox	59,904	24.0	1	canneal	414.7	Parsec3.0	2
rar	511.8	5.0	1	konqueror	920.1	4.8.5	1	dedup	127.2	Parsec3.0	2
cmus	271.6	2.4.3	1	luakit	153.4	d83cc7e	1	ferret	2,150	Parsec3.0	2
mocp	384	2.5.0	1	midori	347.6	0.4.3	1	freqmine	227.6	Parsec3.0	2
mp3blaster	265.8	3.2.5	1	seaMonkey	760.9	2.21	1	streamcluster	102.7	Parsec3.0	2
mplayer	4,300	r34540	1	Crypt	518.1	JavaG1.0	43	swaption	94	Parsec3.0	2
sox	55.2	14.3.2	1	Series	593.3	JavaG1.0	43	x264	896.3	Parsec3.0	2
arora	1,331	0.11	1	SparseMat	593.3	JavaG1.0	43				

V. EXPERIMENTAL RESULTS AND ANALYSIS

A. Experimental Setup

We evaluate the effectiveness of our proposed detection system over FPBirth on a public software plagiarism sample set [11], including 234 multi-threaded programs of different versions, derived from a series of obfuscations (e.g., SandMax, Zelix, UPX) over 35 benchmark multi-threaded programs, which are shown in Table I. The parameter settings to implement our model for evaluation are specified as: $k = 6$ for k-gram slice, which is also the length of the input sequences for frequent pattern mining, minimum support $\sigma = 4$, the length of frequent patterns ranging in $t \in [3, 6]$; for each input, $m = 4$ for the number of execution traces captured. As for the baselines, we compare our approach with two multi-threaded program plagiarism detection methods TreSB and TOB.

B. FPBirth Evaluation

With these settings, we mainly evaluate the resilience and credibility of the thread-aware birthmark FPBirth [12], which can be described as follows [7]:

- *Resilience.* Let p be a program and q be a copy of p generated by applying semantics-preserving code transformations τ. A birthmark is resilient to τ if $\text{sim}(p_f, q_f) \geq 1 - \varepsilon$.
- *Credibility.* Let p and q be independently developed programs. A birthmark is credible if it can differentiate the two programs, that is $\text{sim}(p_f, q_f) < \varepsilon$.

In other words, resilience reflects the ability of birthmark to be resistant to all kinds of semantic-retention code obfuscations, while credibility characterizes the ability of birthmark to distinguish independently developed software.

1) Resilience Evaluation: In this experiment, the benchmark program is taken as the original program while the obfuscated program is taken as the suspect program so that a series of original-suspect comparison pairs are formulated to evaluate the resilience of FPBirth. The experimental results with respect to the similarity distribution under three different obfuscations (H1, H2 and H3) are illustrated in Figure 3(a), where H1 uses different compilers and optimizations (e.g., llvm, gcc, o0 - oS) for weak obfuscation, H2 applies professional obfuscation tools (e.g., SandMark, Zelix, ProGuard) for strong obfuscation, and H3 uses UPX for packing. From the results, we can observe that most of the comparison pairs

(a) Resilience evaluation

(b) Credibility evaluation
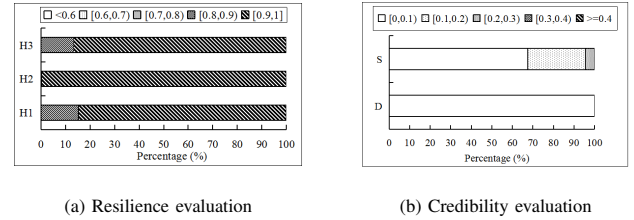
Figure 3: FPBirth Evaluation.

enforce a similarity higher than 0.9; this indicates that FPBirth birthmark enjoys an excellent resistance to the obfuscation strategies involved in this public data set.

2) Credibility Evaluation: In this experiment, the programs independently developed are selected from the data set to evaluate the credibility of FPBirth. More specifically, the selected experiment instances include 6 multi-threaded compression/decompression software, 7 web browsers, and 5 audio player software. We use FPBirth to birthmark the software and then calculate the similarity between them. Figure 3(b) shows the distribution of similarity over similar software and different software, where S stands for software included in the same category and D represents software distributed in different categories. From the results, we can see that the similarity between software belonging to different categories is very low, with the mean similarity below 0.1. This indicates that FPBirth birthmark can effectively distinguish different kinds of software. Due to their remarkable consistency in functions, the similarity between software in the same category is slightly higher, but most of them still fall into a very low similarity range. There are few comparison pairs with a similarity between 0.2 and 0.3 as their designs adopt the same algorithm or both rely on some functional modules. For example, the average similarity between browser Dooble and Epiphany is 0.28, since both browsers use WebKit layout engines. Overall, FPBirth performs well in differentiating independently developed software.

C. Comparisons with Traditional Birthmark Techniques

1) Comparative Analysis on Detection Effect: In this section, we compare FPBirth with TreSB [11] and TOB [13], two traditional thread-aware birthmark techniques, and SCSSB [16], a dynamic birthmark technique also based on system
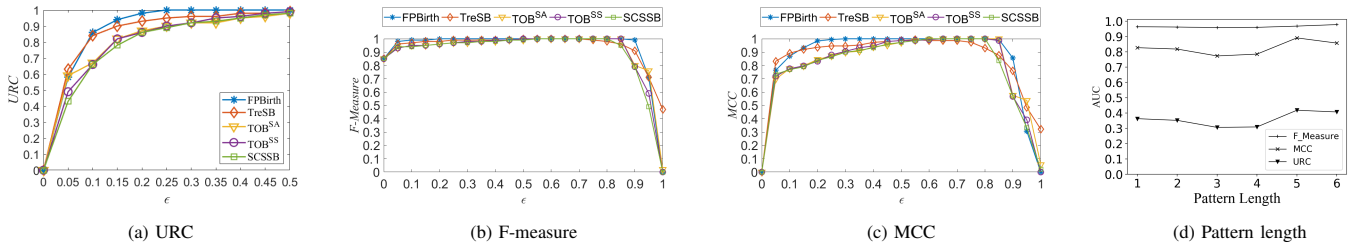
Figure 4: Comparative analysis on detection performance and pattern length.

calls. To quantitatively validate the effectiveness of different methods, we use URC (union of resilience and credibility) [17], F-Measure, MCC (matthews correlation coefficient) [6], and AUC (area under the curve) as the performance measures.

(i) URC. URC is an indicator designed for comprehensively measuring the birthmarks in terms of resilience and credibility:

$$\text{URC} = 2 \times \frac{R \times C}{R + C} \quad (3)$$

where R represents the ratio of plagiarism pairs correctly classified to all comparison pairs with plagiarism, and C represents the ratio of independently developed pairs correctly classified to all comparison pairs with independence (i.e., without plagiarism). The value of URC is between 0 and 1, and the higher the URC, the better the performance of the birthmark. According to the criteria given in Eq. (1), the plagiarism detection result is decided by the threshold ε. We set the effective value range of the threshold as 0-0.5, that is, $1-\varepsilon \geq \varepsilon$. Figure 4(a) shows the comparison between FPBirth and other birthmark techniques under different thresholds. As the blue line shows, FPBirth performs better than the other three birthmarking methods.

(ii) F-Measure and MCC. F-Measure and MCC are commonly used in the field of information retrieval and data mining. In this regard, the "uncertain" part of the criteria given in Eq (1) is removed here, and plagiarism detection is described as a binary classification problem:

$$\text{sim}(p_f, q_f) = \begin{cases} \geq \varepsilon & q \text{ is a copy of } p \\ < \varepsilon & q \text{ is not a copy of } p \end{cases} \quad (4)$$

For F-Measure measurement, the harmonic average of precision and recall is used here, which is described as:

$$\text{F-Measure} = \frac{2 \times Precision \times Recall}{Precision + Recall} \quad (5)$$

MCC is an evaluation metric considering true positives (TP), true negatives (TN), false positives (FP) and false negatives (FN), and can be used to make a reasonable assessment of test effectiveness in the case of unbalanced positive and negative samples, which is denoted as:

$$\text{MCC} =$$
$$\frac{TP \times TN - FP \times FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}} \quad (6)$$

Figure 4(b) and Figure 4(c) respectively show the comparison results between FPBirth and other birthmark techniques under

different thresholds, where FPBirth outperforms TreSB, TOB, and SCSSB in most measurements.

(iii) AUC. With the help of AUC, we can further perform the quantitative analysis of the technical performance of each birthmark with respect to URC, F-Measure, and MCC. Table II summarizes the specific AUC values of different measure metrics for each birthmark technique. It can be observed that all three AUC values of FPBirth are higher than those of traditional birthmark methods, which indicates that FPBirth can cope better with multi-threaded program plagiarism detection.

Table II: Comparison of birthmark techniques over AUC

	SCSSB	TOBSA	TOBSS	TreSB	FPBirth
URC	0.394	0.404	0.402	0.431	0.443
F-Measure	0.916	0.933	0.925	0.952	0.954
MCC	0.820	0.839	0.834	0.875	0.885

2) Comparative Analysis on Time Cost: FPBirth and other three birthmark based detections mainly include trace capture, birthmark generation, and similarity calculation. Considering that the experiments are conducted on the same set of execution traces, in this section, we focus on comparing the time cost of FPBirth with others in terms of birthmark generation (Phase II) and similarity calculation (Phase III). Table III gives the average time cost of each birthmark. From the results, we can observe that the average time of FPBirth to generate birthmark is higher than other methods. The reason behind this is that other methods use k-gram directly to construct birthmarks, while FPBirth takes extra time to mine the frequent patterns that improves the birthmark's thread-aware ability. Since FPBirth constructs more representative frequent patterns, it takes a little more time (9.9 ms on average) for similarity calculation as well, which is still less than TOBSS using maximum weighted dichotomy matching. Though it is more time-consuming, FPBirth is still significant for multi-threaded program plagiarism detection for its better detection effectiveness. Our follow-up plan is to optimize the frequent pattern mining process to improve FPBirth's construction efficiency.

Table III: Comparison of of birthmarks over time cost (ms)

	SCSSB	TOBSA	TOBSS	TreSB	FPBirth
Phase II	103	103	103	102	1556
Phase III	0.1	0.1	20	0.02	9.9

297

D. Evaluation on Pattern Length

As described in Section III-B, the length of frequent patterns directly affects the validity of the constructed birthmark. Therefore, this section specifically analyzes the impact of pattern length on the detection performance. Figure 4(d) displays the AUC values of URC, F-Measure and MCC for plagiarism detection using FPBirth with respect to different pattern lengths. We can see that F-Measure slightly increases as the length increases, while URC and MCC suffer from a drop at length 3, but keep going up afterwards and reach to the best at length 5. Considering all three detection metrics, length 6 gives the best balance. This is the reason why we choose $k = 6$ for k-gram slice and the length of the input sequences for frequent pattern mining. In addition, given the length of the input sequences, frequent patterns after mining and reduction may still enjoy different pattern lengths ranging from 1 to 6. As discussed, patterns being too short may exist in different programs as common behaviors, which may not be able to differentiate a program from others and should be removed. From our experimental results, pattern lengths ranging from 3 to 6 provide the best detection performance, which is what we've set up for our experiments.

VI. RELATED WORK

The existing software plagiarism detection work mainly falls into two categories: static birthmark and dynamic birthmark. For static birthmark, Xie et al. [17] introduced the weighted short sequence birthmark, DroidMoss [19] took hash value of bytecode fragments as birthmark, and ViewDroid [18] presented a functional view graph birthmark; For dynamic bithmark, Wang et al. [16] designed SCSSB (System Call Short Sequence Birthmark) and IDSCSB, and SUPB [9] constructed call sequence diagram of the program. These traditional dynamic birthmarks cannot well address the uncertainty caused by multi-threaded programs. Tian et al. [14] introduced the concept of thread-aware birthmark for the first time. Accordingly, two dynamic birthmarking methods TreSB [11] and TOB [13] were proposed to detect multi-threaded program plagiarism. Differently, our proposed FPBirth takes the frequent patterns from execution traces of multi-threaded programs as birthmark, which preserves the behavioral semantics and also improves the difference-tolerant ability.

VII. CONCLUSION

This paper proposes a new dynamic thread-aware birthmark FPBirth to detect the multi-threaded program plagiarism. More specifically, we first explore dynamic monitoring to capture multiple execution traces with respect to system calls for each multi-threaded program under a specified input, and then leverage Apriori algorithm to mine frequent patterns to formulate our dynamic birthmark, which can not only depict the program's behavioral semantics, but also resist the changes and perturbations over execution traces caused by the thread scheduling in multi-threaded programs. Using FPBirth, we design a multi-threaded program plagiarism detection system. The experimental results based on a public software plagiarism sample set demonstrate that the developed system integrating our proposed birthmark FPBirth outperforms alternative approaches in multi-threaded plagiarism detection.

REFERENCES

[1] R. Agrawal, R. Srikant *et al.*, "Mining sequential patterns", in *icde*, vol. 95, 1995, pp. 3–14.

[2] K. Chen, P. Liu, and Y. Zhang, "Achieving accuracy and scalability simultaneously in detecting application clones on android markets", in *ICSE*, 2014.

[3] J. Han, H. Cheng, D. Xin, and X. Yan, "Frequent pattern mining: current status and future directions", *DMKD*, vol. 15, no. 1, pp. 55–86, 2007.

[4] Y.-C. Jhi, X. Jia, X. Wang, S. Zhu, P. Liu, and D. Wu, "Program characterization using runtime values and its application to software plagiarism detection", *IEEE TSE*, vol. 41, no. 9, pp. 925–943, 2015.

[5] L. Luo, J. Ming, D. Wu, P. Liu, and S. Zhu, "Semantics-based obfuscation-resilient binary code similarity comparison with applications to software plagiarism detection", in *FSE*, 2014, pp. 389–400.

[6] B. W. Matthews, "Comparison of the predicted and observed secondary structure of t4 phage lysozyme", *BBA-Protein Structure*, 1975.

[7] G. Myles and C. Collberg, "Detecting software theft via whole program path birthmarks", in *ISC*, 2004, pp. 404–415.

[8] G. Mylos and C. Collberg, "K-gram based software birthmarks", in *ACM symposium on Applied computing*, 2005, pp. 314–318.

[9] J. Park, D. Son, D. Kang, J. Choi, and G. Jeon, "Software similarity analysis based on dynamic stack usage patterns", in *RACS*, 2015, pp. 285–290.

[10] H. Tamada, M. Nakamura, A. Monden, and K.-i. Matsumoto, "Design and evaluation of birthmarks for detecting theft of java programs", in *IASTED Conf. on Software Engineering*, 2004, pp. 569–574.

[11] Z. Tian, T. Liu, Q. Zheng, M. Fan, E. Zhuang, and Z. Yang, "Exploiting thread-related system calls for plagiarism detection of multithreaded programs", *JSS*, vol. 119, pp. 136–148, 2016.

[12] Z. Tian, T. Liu, Q. Zheng, F. Tong, D. Wu, S. Zhu, and K. Chen, "Software plagiarism detection: a survey", *Journal of Cyber Security*, vol. 1, no. 3, pp. 52–76, 2016.

[13] Z. Tian, T. Liu, Q. Zheng, E. Zhuang, M. Fan, and Z. Yang, "Reviving sequential program birthmarking for multithreaded software plagiarism detection", *IEEE TSE*, vol. 44, no. 5, pp. 491–511, 2017.

[14] Z. Tian, Q. Zheng, T. Liu, M. Fan, X. Zhang, and Z. Yang, "Plagiarism detection for multithreaded software based on thread-aware software birthmarks", in *ICPC*, 2014, pp. 304–313.

[15] Z. Tian, Q. Zheng, T. Liu, M. Fan, E. Zhuang, and Z. Yang, "Software plagiarism detection with birthmarks based on dynamic key instruction sequences", *IEEE Transactions on Software Engineering*, vol. 41, no. 12, pp. 1217–1235, 2015.

[16] X. Wang, Y.-C. Jhi, S. Zhu, and P. Liu, "Detecting software theft via system call based birthmarks", in *2009 Annual Computer Security Applications Conference*. IEEE, 2009, pp. 149–158.

[17] X. Xie, F. Liu, B. Lu, and L. Chen, "A software birthmark based on weighted k-gram", in *2010 IEEE International Conference on Intelligent Computing and Intelligent Systems*, vol. 1. IEEE, 2010, pp. 400–405.

[18] F. Zhang, H. Huang, S. Zhu, D. Wu, and P. Liu, "Viewdroid: Towards obfuscation-resilient mobile application repackaging detection", in *Proceedings of the 2014 ACM conference on Security and privacy in wireless & mobile networks*. ACM, 2014, pp. 25–36.

[19] W. Zhou, Y. Zhou, X. Jiang, and P. Ning, "Detecting repackaged smartphone applications in third-party android marketplaces", in *Proceedings of the second ACM conference on Data and Application Security and Privacy*. ACM, 2012, pp. 317–326.

The Impact of Auto-Refactoring Code Smells on the Resource Utilization of Cloud Software

Asif Imran* and Tevfik Kosar†

Department of Computer Science and Engineering, University at Buffalo
Amherst, New York 14260-1660, USA
Email: *asifimra@buffalo.edu, †tkosar@buffalo.edu

Abstract—Cloud-based software-as-a-service (SaaS) have gained popularity due to their low cost and elasticity. However, like other software, SaaS applications suffer from code smells, which can drastically affect functionality and resource usage. Code smell is any design in the source code that indicates a deeper problem. The software community deploys automated refactoring to eliminate smells which can improve performance and also decrease the usage of critical resources. However, studies that analyze the impact of automatic refactoring smells in SaaS on resources such as CPU and memory have been conducted to a limited extent. Here, we aim to fill that gap and study the impact on resource usage of SaaS applications due to automatic refactoring of seven classic code smells: god class, feature envy, type checking, cyclic dependency, shotgun surgery, god method, and spaghetti code. We specified six real-life SaaS applications from Github called Zimbra, OneDataShare, GraphHopper, Hadoop, JENA, and JAMES which ran on Openstack cloud. Results show that refactoring smells by tools like JDeodrant and JSparrow have widely varying impacts on the CPU and memory consumption of the tested applications based on the type of smell refactored. We present the resource utilization impact of each smell and also discuss the potential reasons leading to that effect.

Index Terms—Code smells, automated refactoring, cloud resource utilization

I. Introduction

Software as a Service (SaaS) running in cloud platforms has become increasingly popular, mainly due to their lower cost, high availability, and quality of service for the users. SaaS applications are becoming mainstream in the everyday lives of users who use them to conduct day-to-day business and personal software needs [1]. SaaS applications are rapidly capturing the market, and more service providers are migrating their software to cloud everyday [2]. The critical advantage of SaaS is its capability to serve millions of users all around the world, harnessing the elasticity, reliability, and scalability of the underlying cloud platform.

Developing SaaS applications differ from desktop applications since those are required to perform on the cloud backbone where multiple software are competing for the compute cloud resources, mainly CPU and memory [3]. Furthermore, the shorter time-to-market affects the development of those open-source software, which are designed to serve multiple users. Those applications are designed by many contributors who work together to solve common issues and regularly add new features. When open source SaaS is incorrectly coded, they can drain cloud resources like CPU and memory, thereby

resulting in wastage of critical resources that are billed by cloud service providers [3]. This results in technical debt and increases the cost of hosting the software in the cloud. We call the erroneous designs as code smells of SaaS.

Resource consuming code smells can degrade the performance of the SaaS application and increase the cost. Users will be demotivated to pay the increased cost and can move to less expensive services provided by the competitors. As a result, fixing those smells can improve the cost-efficiency of critical computing resources without sacrificing the quality of service. Hence, selectively refactoring these code smells would benefit the SaaS service providers as well as the customers of those services. Previous research has focused on exploring the impact of code smells on energy consumption, such as battery usage in smartphones [4]. Also, the importance of detecting and eliminating smells that affect speedup during migration has been addressed [5]. However, there is a lack of empirical study which focuses on analyzing the impact of automatically refactoring smells in SaaS on CPU and memory consumption.

This paper aims to fill the void by analyzing the effect of automatic refactoring of smells on resource utilization of SaaS applications running in the cloud. We conducted an initial search that included 84 repositories and filtered those to match the required pre-specific criteria of this research. We selected seven code smells, which we detected in the selected software using two popular tools, JDeodrant [6] and JSparrow [7], which have capabilities of detecting classic code smells and applying automatic refactoring on those. Out of the seven smells, JDeodrant detected and refactored *god class, feature envy,* and *type checking* whereas JSparrow could detect and refactor four smells namely *cyclic dependency, shotgun surgery, god method,* and *spaghetti code*. We strongly believe our research efforts will help to identify the critical importance of refactoring specific code smells in cloud-based software and their impact on the utilization of precious cloud resources.

The rest of the paper is organized as follows: Section II describes the methodology of our study, Section III presents the results of the experiments and summary of our findings, Section IV discusses the related work in this area, and Section V concludes the paper.

II. Experimental Process

This section describes the goals of this study and the research questions answered, the selection of code smells, and

Smell	Primary Causes	Impact on Software	Refactoring technique
cyclic dependency	-Violation of acyclic modularization [8] -Misplaced elements -Two packages are dependent on each other -Lack of encapsulation	-Difficult to maintain -Single method called for multiple tasks -Has ripple effect on other abstractions [8]	-Encapsulate all packages in a cycle and assign to single team
god method	-Many activities in a single method [9] -Tangled code -Long methods with multiple responsibilities	-Memory leak -Multiple calls to same function [10] -Difficult co-ordination of packages	-Extract method -Replace method with method object
type checking	-Using complex variation of an algorithm requiring execution based on the value of an attribute [11] -Objects in class come from different workers	-Abuse of type casting -Redundant code in a method or class -Less flexible code	-Replace "instanceof" from code -Strategy pattern
spaghetti code	-Convoluted code -Continuous addition of new code and no removal of obsolete ones [12] -Procedural code design	-Difficult to understand code [12] -Lack of well-articulated code	-Replace procedural code segments with object oriented design
feature envy	-Accessing data of another object often [6] -Occurs when fields are moved to data class -Data defined in class A, however operations defined in class B	-High volume of requests to access a class and its objects -Numerous read and write to remote object	-Move method -Extract method
shotgun surgery	-Single behavior defined across multiple classes [9]	-Requires multiple changes in different locations (e.g., multiple files) of the code in order to make a single modification [9] -Existing behavior in multiple classes	Inline class
god class	-Class aims to do many activities [9] -Large number of instance variables declared in one class	-Difficult to manage multiple functionalities -Difficult to understand code	-Extract class -Extract Interface

TABLE I: Characteristics and applied refactoring properties of software smells

the methodology of experimental and evaluation processes.

A. Identified Research Questions

The main goal is to determine whether automatic refactoring of smells in cloud-based software impact the resource consumption in terms of CPU and memory. The motivation is derived from the fact that cloud resources are in high demand, and any unnecessary resource usage will incur unnecessary costs. More specifically, we want to determine the refactoring of which smells will result in an increase or reduction of CPU and Memory consumption in the cloud. Based on this goal, we determine the following research questions:

- **RQ1.** How does auto-refactoring of code smells in cloud-based software affect CPU utilization?
- **RQ2.** How does auto-refactoring of code smells in cloud-based software affect memory utilization?

B. Selected Code Smells

A code smell is a software behavior that is indicative of more profound quality issues [9]. We selected the aforementioned seven smells mainly because of the following reasons. Primarily, those smells are studied popularly by software community, and they are considered as classic smells [9]. Secondly, several tools can detect the code smells; however, few tools are available, which can automatically refactor those. Our selected smells could be automatically detected by the refactoring tools we used in this study. Table I provides a summary of the smells which includes the causes, impact on software, and the refactoring policy applied by the tools.

C. Study Methodology

To eliminate biases in our study, we executed each of the apps 24 times and took the mean of the data. To analyze the effect of a given smell, first, we ran the smelly code 3 times. Then we refactored a smell and ran the software 3 times more, hence for one software, we had (7*3=21)+3=24 runs. Running each software 24 times resulted in a total of 144 runs. This required a significant time frame.

We conducted the experiments in $OpenStack$ cloud servers, which had 32 GB RAM, 8 core processors, and 2 TB persistent storage [13]. We set up the cloud and ran the six software in VM instances, which were created using *kernel virtual machine (kvm)*. Every time we refactored, we executed the software in a new instance and deleted all existing data to minimize the impact of caching. We did not run any other cloud VM instance with other services to minimize the effect of external entities. Following this mechanism allowed us to erase all existing data before the software was run. As a result, it helped us achieve the same initial condition for each experimental run. Next, we followed a methodology for refactoring the selected software, which is shown in Figure 1 and discussed here.

- **Smell detection and provenance:** As a first step, we applied *JDeodrant* to detect the three code smells (*god*
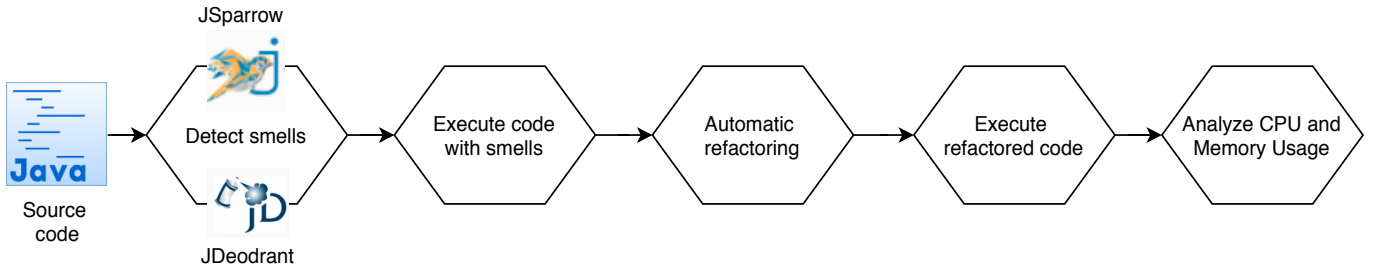
Fig. 1: Automatic detection and refactoring of code smells.

class, feature envy, and *type checking*), followed by *JSparrow* to detect four remaining smells. Since both *JDeodrant* and *JSparrow* have plugins for *Eclipse IDE*, we imported the source codes in to *Eclipse* and built those. After detection of the smells, we preserved provenance of the packages, classes, and methods, which were affected by the smells. Overall, *JDeodrant* and *JSparrow* detected 744 instances of the seven analyzed smells in the six software. The reason for using two tools is to include significant number of smells in the study. None of the tools could single-handedly detect and refactor all the smells considered here.

- **Executing smelly code:** Next, we executed the codes in the cloud. We ran the code three times, each time with the same workload, and collected the CPU and memory utilization during each clock cycle using scripts in $OpenStack$ [3]. This allowed us to eliminate biases.
- **Automatic refactoring of smells:** Afterwards, we refactored the code from the smells. For each smell, we took a new copy of the smelly code and refactored to eliminate the effect of another refactoring. As a result, the projects were imported newly for each smell, refactored and executed. After refactoring, we repeat step 2 and record the CPU and memory consumption for the same workload and following the same method.
- **Logging the data:** We proceeded to log all the CPU and memory consumption in each experimental run. We tried to eliminate any external impact during each execution. We recorded the CPU and memory consumption during each CPU cycle and stored the data persistently.
- **Statistical operation:** Finally, we obtained the arithmetic mean of the CPU and memory usage. Afterward, we tabulated the results and compared whether the resource usage improved or deteriorated after refactoring the smells.

The next section highlights results obtained by following these prescribed steps.

III. RESULTS

The obtained results during experimentation have been provided here. The number of detected smells is showcased which is followed by quantitative analysis of the data achieved during experimentation.

System	Activity	LoC	Commits	NoC
Zimbra	08/05-03/19	24,698	15,052	1,871
JGraph	09/09-02/19	22,758	1,360	187
OnedataShare	11/18-01/20	23,002	1,644	390
Hadoop	12/11-03/20	14,2790	23,611	2,069
JENA	06/10-05/19	70,948	8,287	1,392
JAMES	06/04-08/19	27,003	9,314	340

TABLE II: List of selected systems for the study

A. Selected Software for Analysis

For this study, we identified and selected six open-source software which serve the users in the cloud. Among those, we analyzed 3,11,199 Lines of Code (LoC) and 6,249 class files. *Zimbra* [14] is an OS-independent emailing, and file sharing tool running in the cloud, popularly used by Dell, Rackspace, and Mozilla. *OneDataShare* [14] is a cloud-based data transfer tool that incorporates multiple transfer protocols, including DropBox and GoogleDrive. *GraphHopper* is a GIS application provided as SaaS that incorporates spatial rules in hybrid mode. *Hadoop* is a software framework which was first proposed by Google to facilitate the analysis of large volume of data in the cloud [15]. *Java Apache Mail Enterprise Server (JAMES)* consists of a modular architecture based on state of the art components which provides secure, stable, and end-to-end mail servers running on the JVM [14]. *JENA* is a platform provided by Apache for designing and building linked data applications, giving access to a variety of APIs for serialization, processing, storage, and transfer [16]. Table II shows details related to the software selected for this study. Following is a description of the mechanism followed here to to obtain those software.

To analyze the impact of automatic refactoring of smells on resource usage, we decided to use open-source software that runs in the cloud as Software as a Service (SaaS) for multiple reasons. First, there are many software which are developed as SaaS for cloud since cloud computing has increased in popularity. So the availability of software was satisfied. Second, due to the high demand for the cloud, those software are heavily deployed by the industry. As shown in the previous paragraph, our selected software are used widely by top companies. Hence we believe it is important to analyze the impact of

automatic smell refactoring on resource consumption since those companies will benefit from our research. Third, the authors of this paper felt that the software running in the cloud will contain a significant number of smells since those have complex code blocks that are written to ensure real-time user interactions and serving a large number of users from remote cloud data centers where they are hosted. Finally, one of the main challenges of SaaS is to optimize resource utilization, since provisioning extra resources in the cloud will require additional cost. Hence optimized use of cloud resources will result in cost reduction for the cloud service provider.

We obtained the source code of the software from Github during November 2019. The software source code obtained based on search in Github using the following keywords: *cloud computing software, Software as a Service, pervasive computing,* and *cloud data transfer*. From there, we selected the software sorted by popularity. To ensure that our selection of software is unbiased, we implement a set of checks which software shall satisfy to be selected for analysis. Those checks are described as follows:-

- **Check 1 - Initial list of software:** Initially, we conducted a preliminary search in Github to identify the realistic chance of obtaining software that runs in the cloud. This search was manual and it involved looking out for cloud computing SaaS. It helped us to identify the keywords for searching in Github and also laid the foundation for the following checks.
- **Check 2 - Script to automatically download the software:** We developed a script that will parse through Github projects and automatically download the source codes from the master branch based on the keywords. It resulted in downloading 84 repositories from Github. We required to use automated scripts because cloning the 84 repositories manually would be time-consuming.
- **Check 3 - Refining the search to match tool requirements:** Finally, we refined our search to be in line with the requirements of our selected automatic code smell detection and refactoring tools. Prior to that, we eliminated any source code not written in Java, which left us with a list of 21 repositories. Next, we decided to include source codes that can be compiled in Eclipse as *JDeodrant* and *JSparrow* both have plugins, which can be run using Eclipse. After this, we were left with 11 repositories. Finally, we considered repositories with at least 10,000 lines of code, since otherwise we could run into the risk of considering a prototype software which we aimed to avoid.

B. Analysis of Results

We start with highlighting our findings related to CPU utilization. More specifically, we identify which code smells from the list will impact the CPU consumption of the analyzed Java software. Hence we address the research question *"How does auto-refactoring of code smells in cloud-based software affect CPU utilization?"* Table III identifies the change in CPU

utilization after each smell is refactored. Each column highlights the percent change in CPU utilization after refactoring the smell specified in column heading. The negative sign in front of the numbers show decrease in resource use, whereas the positive sign show increase in resource usage.

After presenting the analysis of obtained results for CPU consumption, we analyze the results for change in memory usage after refactoring the smells. Hence, we aim to answer our second research question, *"How does auto-refactoring of code smells in cloud-based software affect memory utilization?"* Table IV identifies the percent change in memory utilization after refactoring each of the smells. The values of memory usage were measured in MB. Following is an analysis of the obtained results.

As seen in Table III, the refactoring of *cyclic dependency* smells improves CPU utilization for all six software tested. The CPU utilization was reduced by 16.52%, 17.22%, 50.81%, 4.67%, 31.96%, and 24.53% respectively for *Zimbra, GraphHopper, OneDataShare, Hadoop, JENA, and JAMES*. Cyclic dependencies result in direct and indirect dependencies between abstractions [17]. The abstractions were tightly coupled between a large number of direct and indirect cyclic dependencies, so they resulted in a tangled design of the code. Further analysis showed that since the elements were not placed in the correct package, it resulted in cyclic dependencies between packages as well. Co-ordination between the packages became difficult which resulted in multiple calls to the same package. Hence, the existence of this smell resulted in higher CPU and memory usage. To eliminate the smell, JSparrow encapsulated all the packages involved in the chain and assigned it to a single team, which required less processing and could be loaded once into memory, thus reducing resource consumption in terms of both CPU and memory.

A common practice to eliminate the *god method* smells by refactoring tools is to detect and remove those by implementing *extract method* design [18]. This refactoring procedure will improve the quality and maintainability of the software while preserving correctness. However, our results in Tables III and IV show that refactoring *god method* leads to an increase in resource utilization. The extra resource usage we found came from the higher number of message traffic obtained from architecture modification of this smell. Refactoring of this smell enables us to obtain a modular architecture of code where the elements have higher cohesion and lower coupling. Despite the benefits, this refactoring mechanism may not be ideal for cloud software as it might create harmful side effects in terms of sustainability [18]. Refactoring of *god method* smells yielded an increase in CPU utilization of *3.50%, 14.59%, 35.79%, 10.45%, 14.36%, and 3.69%* for *Zimbra, GraphHopper, OneDataShare, Hadoop, JENA, and JAMES*.

Removal of *shotgun surgery* smells contributed to the reduction of resource usage by the software. As we know *shotgun surgery* smells occur when we try to modify a class which in turn makes multiple modifications to several different classes. For example, in *OneDataShare*, removal of all instances of this smell resulted in 52.30% of CPU and

TABLE III: The impact of smell refactoring on percentage of CPU utilization

Table	Affect on CPU utilization after refactoring						
	cyclic dependency	*god method*	*shotgun surgery*	*spaghetti code*	*feature envy*	*type checking*	*god class*
Zimbra	-16.5%	+3.5%	-42.6%	-27.6%	+11.0%	-33.7%	+14.8%
GraphHopper	-17.2%	+14.6%	-30.7%	-0.7%	+27.6%	-10.4%	+66.2%
OneDataShare	-50.8%	+35.8%	-52.3%	-27.4%	+66.8%	-53.6%	+48.8%
Hadoop	-4.7%	+10.5%	-14.4%	-8.2%	+20.1%	-6.8%	+17.4%
JENA	-32.0%	+14.4%	-35.5%	-10.4%	+57.0%	-2.8%	+78.5%
JAMES	-24.5%	+3.7%	-20.7%	-2.7%	+35.0%	-3.3%	+20.6%

TABLE IV: The impact of smell refactoring on percentage of Memory utilization

Table	Affect on Memory utilization after refactoring						
	cyclic dependency	*god method*	*shotgun surgery*	*spaghetti code*	*feature envy*	*type checking*	*god class*
Zimbra	-60.8%	+14.2%	-71.2%	-60.5%	+12.6%	-53.9%	+6.2%
GraphHopper	-55.6%	+18.9%	-57.1%	-4.7%	+66.7%	-22.1%	+54.8%
OneDataShare	-33.2%	+38.2%	-31.4%	-4.8%8	+81.9%	-13.6%	+88.1%
Hadoop	-7.5%	+7.4%	-12.8%	-20.6%	+5.4%	-7.7%	+17.3%
JENA	-7.5%	+48.6%	-1.7%	-14.2%	+28.7%	-2.9%	+42.3%
JAMES	-13.2%	+0.1%	-21.9%	-17.8%	+14.7%	-5.0%	+35.3%

31.43% of memory utilization. We consulted the $diff$ in the commits of $OneDataShare$ and agreed that the smell was introduced due to the practice of overzealous and sudden changing of multiple components in different classes during development to incorporate new requirements without proper documentation. The refactoring tool used *Inline Class* to transfer the diversified behaviors of one operation to one class. This was done for 39 occurrences of this smell. As a result, we obtained the improvement of resource utilization.

OneDataShare was found to have improved CPU utilization by 27.35% when the *spaghetti code* smells were refactored. On the other hand, *Zimbra's* memory utilization improved by 60.47% when this smell was removed from it. Once again taking $OneDataShare$ as a case study, upon analysis of its source code using the refactoring tools, it was seen that the code used a large volume of $GOTO$ statements. Excessive use of $GOTO$ instead of well-articulated code design resulted in software that is convoluted and unmanageable [19]. At the same time, it causes the program to have methods scattered across many classes, which required more memory for file read and write operations.

Eliminating *feature envy* causes increased resource utilization which is a threat to the sustainability of the software, as it would result in provisioning more resources for the same task, thus incurring more cost [20]. More specifically, CPU and memory consumption increased by *66.84%* and *81.87%* respectively when this smells was refactored in *OneDataShare*. Analysis of files in $OneDataShare$ showed that the method calls required access to specific classes each time it requested for certain operations, hence increasing inter-class communication, thus deteriorating memory and CPU usage.

Refactoring the *type checking* smell yielded CPU utilization improvement of 53.6% of *OneDataShare*. Also, notable improvement of memory utilization of 53.9% was observed for *Zimbra*. This is because type checking results in a large

function which should be broken down into multiple smaller functions. Calling the large function numerous times will cause all its functionality to be executed even when it is not necessary, yielding high CPU and memory utilization.

The tested software started to consume an alarming quantity of memory after refactoring *god class* smells. In general, the reason for a similar pattern of memory usage can be summarized to two main causes. The first is despite having a large volume of LoC and effective session cache management, the refactoring divided large classes into multiple sub-classes [18]. Hence, implementing *extract method* on *god classes* caused an increased number of calls the program has to make to perform its tasks, which increased the data volume stored in memory. Secondly, all the software have multiple developers contributing to it. So, from the perspective of community smells, there is a possible reason between these smells and developer viewpoint, which requires further research.

In summary, it is seen that all the software considered here suffered from increased CPU utilization after smells called *god method, feature envy,* and *god class* were refactored. The increase is *3.5%, 11.00%,* and *14.8%* respectively for the three smells. Although the increase in CPU utilization may seem low, however, it is important to remember that when the software runs in the cloud, every CPU cycle is charged to the customer, hence increase in CPU utilization due to refactoring of software smells will lead to the cloud client incurring unnecessary extra cost. On the other hand, it is seen that refactoring the smells called *cyclic dependency, shotgun surgery, spaghetti code,* and *type checking* contributes to reduced CPU usage.

As seen in Table IV, memory consumption is seen to drastically increase after refactoring *god class*. For *Zimbra, GraphHopper, and OneDataShare* this increase in memory consumption is found to be *6.2%, 54.8%, and 88.1%*. Overall, similar observations are made for *feature envy, and god method*

smells as well for all six software. Like CPU, memory resource is also paid in the cloud, hence refactoring these smells will lead to additional cost as extra memory needs to be provisioned. We determine that the refactoring techniques adopted by *JDeodrant* for *god class,* and *feature envy* smells, together with *JSparrow* for *god method* are not useful for the cloud-based applications since all the six software resulted in an increase of CPU utilization and memory utilization.

IV. Related Work

Platform-specific code smells in High-Performance Computing applications were determined by Wang et al. [5]. AST based matching was used to determine smells present in an HPC software by comparing it with a dictionary of smells. The authors claimed that the removal of such smells would increase the speedup of the software when migrated to a new platform. The assumption was that certain code blocks perform well in terms of speedup in a given platform. However, the results show that certain smell detection and refactoring reduced the speedup, thus challenging the claims and showing the importance of further research in this area.

Oliveira et al. conducted an empirical study to evaluate nine context-aware Android apps to analyze the impact of automated refactoring of code smells on resource consumption [21]. They studied three code smells, namely *god class, god method, and feature envy*. They found that for the three smells, resource utilization increases when they are refactored. Although their findings are useful, it is limited to the analysis of three code smells only. At the same time, the importance of analyzing the impact of automated code smell refactoring on cloud computing SaaS applications were not considered.

V. Conclusion

In this paper, we evaluated the impact of automatically refactoring seven code smells on resource usage of software running in the cloud platform. Obtained results highlight that the refactoring techniques adopted by *JDeodrant* for *god class,* and *feature envy*, together with *JSparrow* for *god method* resulted in more message traffic which adversely affected CPU and memory usage. More specifically, cumulative increase of CPU and memory consumption for refactoring these three smells significantly higher as shown in Table III and Table IV. Hence, there exists scope of further research to improve the automatic refactoring mechanisms of existing tools. Also, determining the correlation between refactoring multiple smells and resource consumption needs to be explored. Additionally, impact on resource usage after refactoring smells specific to the cloud should be studied.

Acknowledgment

References

[1] V. V. H. Pham, X. Liu, X. Zheng, M. Fu, S. V. Deshpande, W. Xia, R. Zhou, and M. Abdelrazek, "Paas-black or white: an investigation into software development model for building retail industry saas," in *2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C)*. IEEE, 2017, pp. 285–287.

[2] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica *et al.*, "A view of cloud computing," *Communications of the ACM*, vol. 53, no. 4, pp. 50–58, 2010.

[3] A. Imran, A. U. Gias, R. Rahman, A. Seal, T. Rahman, F. Ishraque, and K. Sakib, "Cloud-niagara: A high availability and low overhead fault tolerance middleware for the cloud," in *16th Int'l Conf. Computer and Information Technology*. IEEE, 2014, pp. 271–276.

[4] R. Verdecchia, R. A. Saez, G. Procaccianti, and P. Lago, "Empirical evaluation of the energy impact of refactoring code smells." in *ICT4S*, 2018, pp. 365–383.

[5] C. Wang, S. Hirasawa, H. Takizawa, and H. Kobayashi, "A platform-specific code smell alert system for high performance computing applications," in *2014 IEEE International Parallel & Distributed Processing Symposium Workshops*. IEEE, 2014, pp. 652–661.

[6] M. Fokaefs, N. Tsantalis, E. Stroulia, and A. Chatzigeorgiou, "Jdeodorant: identification and application of extract class refactorings," in *2011 33rd International Conference on Software Engineering (ICSE)*. IEEE, 2011, pp. 1037–1039.

[7] S. IT-Consulting. (2020) Jsparrow.

[8] S. Sarkar, G. M. Rama, N. N. Siddaramappa, A. C. Kak, and S. Ramachandran, "Measuring quality of software modularization," Mar. 27 2012, US Patent 8,146,058.

[9] M. Fowler, *Refactoring: improving the design of existing code*. Addison-Wesley Professional, 2018.

[10] M. Lanza and R. Marinescu, *Object-oriented metrics in practice: using software metrics to characterize, evaluate, and improve the design of object-oriented systems*. Springer Science & Business Media, 2007.

[11] N. Tsantalis, T. Chaikalis, and A. Chatzigeorgiou, "Jdeodorant: Identification and removal of type-checking bad smells," in *2008 12th European Conference on Software Maintenance and Reengineering*. IEEE, 2008, pp. 329–331.

[12] M. Abbes, F. Khomh, Y.-G. Gueheneuc, and G. Antoniol, "An empirical study of the impact of two antipatterns, blob and spaghetti code, on program comprehension," in *2011 15th European Conference on Software Maintenance and Reengineering*. IEEE, 2011, pp. 181–190.

[13] A. Imran, S. Aljawarneh, and K. Sakib, "Web data amalgamation for security engineering: Digital forensic investigation of open source cloud." *J. UCS*, vol. 22, no. 4, pp. 494–520, 2016.

[14] A. Imran, "Design smell detection and analysis for open source java software," in *2019 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 2019, pp. 644–648.

[15] M. R. Ghazi and D. Gangodkar, "Hadoop, mapreduce and hdfs: a developers perspective," *Procedia Computer Science*, vol. 48, no. C, pp. 45–50, 2015.

[16] E. Tempero, C. Anslow, J. Dietrich, T. Han, J. Li, M. Lumpe, H. Melton, and J. Noble, "Qualitas corpus: A curated collection of java code for empirical studies," in *2010 Asia Pacific Software Engineering Conference (APSEC2010)*, Dec. 2010, pp. 336–345.

[17] G. Samarthyam, G. Suryanarayana, and T. Sharma, "Refactoring for software architecture smells," in *Proceedings of the 1st International Workshop on Software Refactoring*, 2016, pp. 1–4.

[18] R. Pérez-Castillo and M. Piattini, "Analyzing the harmful effect of god class refactoring on power consumption," *IEEE software*, vol. 31, no. 3, pp. 48–54, 2014.

[19] M. Ceccato, P. Tonella, and C. Matteotti, "Goto elimination strategies in the migration of legacy code to java," in *2008 12th European Conference on Software Maintenance and Reengineering*. IEEE, 2008, pp. 53–62.

[20] K. Nongpong, "Feature envy factor: A metric for automatic feature envy detection," in *2015 7th International Conference on Knowledge and Smart Technology (KST)*. IEEE, 2015, pp. 7–12.

[21] J. Oliveira, M. Viggiato, M. F. Santos, E. Figueiredo, and H. Marques-Neto, "An empirical study on the impact of android code smells on resource usage." in *SEKE*, 2018, pp. 314–313.

Towards Fine-Grained Compiler Identification with Neural Modeling

Borun Xie[1,2], Zhenzhou Tian[1,2*], Cong Gao[1,2], Lingwei Chen[3]

[1]School of Computer Science and Technology, Xi'an University of Posts and Telecommunications, Xi'an, China
[2]Shaanxi Key Laboratory of Network Data Analysis and Intelligent Processing, Xi'an, China
[3]College of Information Sciences and Technology, Pennsylvania State University, PA, USA
[*]Corresponding: tianzhenzhou@xupt.edu.cn

Abstract—**Different compilers and optimization levels can be used to compile the source code. Revealed in reverse from the produced binaries, these compiler details facilitate essential binary analysis tasks, such as malware forensics and binary code similarity analysis. Most existing approaches adopt a signature matching based or machine learning based strategy to identify the compiler details, showing limits in either the detection accuracy or granularity. In this work, we propose NeuralCI (Neural modeling-based Compiler Identification) to perform the identification of compiler family and optimization level on individual functions. The basic idea is to formulate sophisticated neural networks to process abstracted instruction sequences generated using a lightweight function abstraction strategy. To evaluate the performance of NeuralCI, a large dataset consisting of 413,119 unique functions collected from real-world projects is constructed. The experiments show that NeuralCI achieves over 99% and 90% accuracy in identifying the compiler family and optimization level respectively, which outperforms most state-of-the-art function level compiler identification methods. Also, we explore for the first time the possibility of conducting compiler identification on binary code snippets rather than complete functions, where NeuralCI still achieves 96% accuracy, indicating its ability of capturing subtle yet significant features.**

Keywords-software forensics; binary code analysis; compiler identification; neural network

I. INTRODUCTION

In the software production process, diverse toolchains and toolchain settings can be adopted to transform the source code to the final binary. For example, different compilers like GCC and Clang as well as different compiler options like O0-O3 can be used by the developers. Besides, it is also a common practice to apply various kinds of code obfuscation techniques [1, 2] and packers [3, 4] in the binary production process.

Usually binaries produced with these different toolchains and toolchain settings exhibit significant differences when viewed in a straight way [5-7]. These differences just indicate that toolchain footprints are preserved during the source code to binary code translation process, enabling the possibility of revealing the toolchain and toolchain setting choices made during the production process of a binary. This task, which in the literature is called binary program provenance analysis, provides ways to spy on the specifics of the binary production process. A major subtask of it, compiler identification, which focuses on the compilation phase, attempts to infer from a piece of binary code the compiler-related details such as the specific compiler family, the optimization options etc.

Overall, relatively few works have been conducted on compiler identification, which mainly fall into two categories: signature matching based methods [8-10] and learning based methods [11-14]. The former, implemented in several reverse engineering tools like IDA [8] and PEiD [9], performs whole program level identification via exact matching of signatures that are manually constructed for certain compilers. Drawbacks of these kinds of methods lie in the stringent expertise in constructing a good enough compiler-specific signature as well as their coarse identification granularity. The latter formulates compiler identification as a machine learning task, which trains models to capture compiler-specific patterns, further with which to infer the compiler details on previously unseen binaries. For this kind of method, syntactic or structural features are extracted based on artificially defined templates such as idioms [11] which are short sequences of instructions with wildcards or graphlets [12] which are small subgraphs within the CFG (Control Flow Graph). The accuracy of these methods greatly depends on the quality of manually-crafted feature extraction strategies, where potential human-bias exists, resulting in capturing lots of irrelevant or redundant features for the compiler identification task meanwhile failing to capture closely relevant ones.

In recent years, tremendous successes have been witnessed of applying natural language processing techniques and deep learning models to various program analysis tasks [16-21]. In this paper, we attempt to adopt some of the most popular neural network structures to achieve fast and accurate fine-grained compiler identification on function level. Specifically, we feed typical Recurrent Neural Network (RNN) and Convolutional Neural Network (CNN) based structures with abstracted assembly instruction sequences to train classification models for inferring the compiler families and the optimization levels. Our intuition is based on the observation that co-occurring instructions together with their orderings that appear even in short instruction sequences form good enough signals of distinguishing different compliers or optimization levels, and here we resort to the neural models to capture them.

Our main contributions are summarized as following:

- We propose to reveal fine-grained compiler details for individual functions by designing a lightweight

function abstraction strategy and adapting typical sequence-oriented neural networks. It alleviates the task complexity and human bias impacts by handing over the professional process of extracting and selecting features significant for compiler identification from the domain experts to the less human intervened neural networks.

- We have implemented the proposed methods as a tool called NeuralCI (**Neural** modeling based **C**ompiler **I**dentification), and evaluated its performance of revealing either the compiler family or the optimization level on a large dataset consisting of 413,119 unique functions that we constructed via processing a set of diverse real world projects. The experiments show that NeuralCI outperforms most state-of-the-art function level compiler identification methods. It achieves above 99% and 90% accuracy in identifying compiler family and optimization level respectively.

- We explore for the first time the possibility of conducting compiler identification on arbitrary binary code snippets rather than complete functions. It shows that NeuralCI achieves 96% accuracy, indicating that it can capture very subtle yet significant features.

The remainder of this paper is organized as follows: Section II summarizes the related works. Section III describes in detail our proposed approach. The experimental evaluation conducted are presented in Section IV. Finally, we conclude the paper in Section V.

II. RELATED WORK

In general, existing works on compiler identification can be divided into two classes: signature matching based methods and learning based methods.

A. Signature Matching Based Methods

The signature matching based methods [8-10] search the binary program against a corpus of manually constructed signatures for exact matching, and attribute to the whole program the compiler label corresponding to the matched signature string. This kind of method has been implemented in several reverse engineering tools, such as IDA Pro [8], PEiD [9] and LANGUAGE 2000 [10], in consideration of its high detection efficiency and low cost. Drawbacks of these methods lie in the stringent expertise and labor work in constructing a good enough compiler-specific signature, as well as the easily affected accuracy due to slight differences between signatures. Besides, the signatures usually depend on the metadata or details of program headers, which can be easily altered or become unavailable in stripped binaries. Moreover, these tools identify compilers on the whole binary, while a program can be produced with multiple compilers in scenarios such as statically linking library code to produce the final binary program.

B. Learning Based Methods

This type of method formulates compiler identification as a machine learning task performed on (in most instances stripped) binaries, based on the belief that the resulting binaries implicit features reflecting design and implementation decisions of the certain compiler which are used to produce the binaries. Specifically, they train models that capture compiler-specific patterns, further with which to infer the compiler provenance on previously unseen binaries.

The pioneering work [11] adopting this type of approach was conducted by Rosenblum et al. that manually defined a set of idioms (short sequences of instructions with wildcards) and utilized mutual information calculation to capture and select significant patterns indicative of the source compiler of the program binaries. High accuracy is observed for inferring the compiler families, but we have no idea of its performance on optimization levels identification as no evaluation was conducted. ORIGIN [12] achieved superior accuracy in recovering the compiler details by introducing graphlets (small and non-isomorphic subgraphs within the CFG) in addition to idioms so as to capture additional structural features. Hidden Markov models were learnt via observing the differences in the type and frequency of instructions comprising the binaries compiled with different compilers, and proved to be accurate in identifying the compiler family for a whole program [14, 15]. However, for each individual compiler family a corresponding separate model needs to be learnt. Also, these models do not extract information regarding the optimization levels. To improve efficiency in terms of computational resources and detection time, BinComp [13] adopted a stratified approach to infer different compiler details on different granularity. It identifies compiler family for the whole program via exact matching of signatures, and conducts compiler version and optimization level detection for compiler-related functions. However, the compiler-related functions constitute only a very small portion of all functions, making it largely impractical in handling real world programs where user defined functions hold the principal status. Basically, accuracy of these methods greatly depends on the quality of manually-crafted feature extraction strategies, where potential human-bias exists, resulting in capturing lots of irrelevant or redundant features for the compiler provenance task meanwhile failing to capture closely relevant ones.

In recent years, significant successes have been witnessed of applying deep learning techniques to the domain of binary program analysis [16-21]. BinEye [16] is one of the few works that utilize neural models to achieve compiler identification. It combines word embedding and position embedding to encode the raw bytes of an object file, and then utilizes CNN to learn a model that supports optimization level recognition on each individual object file. Our work differs in that we achieve finer grained identification of both the compiler family and the optimization level for each individual function by adopting an abstraction strategy that operates on assembly instructions rather than the raw bytes. Structure2Vec [19] utilizes a graph embedding network to transform the function CFGs into vectors, which are then fed into a dense layer to train a classifier for compiler family identification. Compared to this work, we operate directly on the instructions comprising a function with a lightweight abstraction strategy, and adopt the much faster sequence-oriented neural networks to train models for identifying the optimization level besides just the compiler family as did by Structure2Vec.
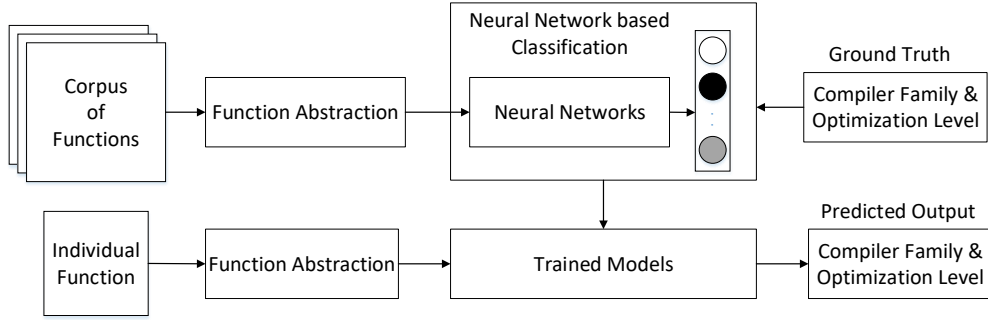
Figure 1. The basic framework of NeuralCI

III. THE APPROACH

Figure 1 depicts the overall architecture of NeuralCI, which consists of two phases: the training phase as illustrated in the top half subfigure, and the detection phase as illustrated in the half bottom subfigure. The training phase consists of three steps. As a deep learning-based method, the first step is to construct a high-quality database comprised of labeled functions which shall be discussed in detail in Section IV.A. The second step takes as input each raw function and outputs an abstracted instruction sequence via a light-weight abstraction strategy implemented in the function abstraction module. Then these abstracted sequences together with their ground truth labels are fed into the neural network based classification module to train compiler identification models. The detection phase is much simpler, which takes in an individual function, processes it with the function abstraction and utilizes the trained model to produce a predicted output. In the following, we discuss the details of the function abstraction module and neural network based classification module respectively.

A. Function Abstraction

A function must be represented in certain forms such that it can be processed for further analysis. The typical ways include using the raw byte sequence, the assembly instruction sequence or the control flow graph [19] to depict a function. According to the findings in [15], different compilers tend to use distinguishable assembly instructions. For example, *call* instruction occurs frequently in GCC-generated assembly code, while the Clang assembly uses *callq* instead. Also, as indicated by the pretty good compiler family identification accuracy in [12], short assembly instruction sequences successfully capture compiler-related features. Thus, in this work we choose to use the assembly instruction sequence as the representation of each function, and we use IDA Pro for the parsing [1]. That is, a function f will be represented as $f = \{ins_1, ins_2, \cdots, ins_n\}$, where n denotes the number of instructions within the function, and each instruction ins_i consists of an opcode (i.e. mnemonic) and a list of operands.

---

[1] We assume a reliable way of identifying the function boundaries, the instruction boundaries, as well as the correct parsing of each instruction, by using the best commercial reverse engineering tool IDA Pro. The correct disassembly of binaries is still a complex and open problem, but are beyond the scope of this paper.

However, as has been confirmed in many existing binary analysis tasks [19, 20], it is usually not wise to work directly on the raw assembly instructions. For our case, we want to capture features reflecting the compiler details rather than the functionality of the function. That is, we do not care whether a value 6 is assigned to register *eax* or a value 10 is assigned. So the two instructions "*mov eax, 6*" and "*mov eax, 10*" should be considered the same. Besides, the memory addresses (e.g. the target of *jmp* instructions) are meaningless but just noises that bring adverse impacts. Meanwhile, to prevent introducing too much human bias, we choose to process the raw instruction sequence with a light-weight abstraction strategy.

Specifically, we do abstraction to each assembly instruction in a function with the following rules:

- The mnemonics remain unchanged.

- All registers in the operands remain unchanged.

- All base memory addresses in the operands are substituted with the symbol *MEM*.

- All immediate values in the operands are substituted with the symbol *IMM*.

As an example, with the above abstraction rules, the instruction "*add eax, 6*" will become "*add eax, IMM*", the instruction "*mov ebx, [0x3435422]*" will become "*mov ebx, MEM*", while the instruction "*mov eax, [ebp-20]*" will become "*mov eax, [ebp-IMM]*".

B. Neural Models for Compiler Identification

Given a set of abstracted assembly instruction sequences, it is promising to utilize skip-gram [22] to learn the embedding for each instruction, explore max-pooling, averaging or concatenation to aggregate the embeddings for each sequence, and then feed them to any classification model for compiler identification. However, it still faces the following two limitations: (1) skip-gram assigns each instruction a static embedding vector, which is not context-aware to different sequences it interacts with; this may fail to learn the compiler-related features; (2) since instruction sequences are abstracted from functions, they may not only enjoy local instruction associations and correlations, but also global or long-range instruction dependency; in this respect, it calls for sequence learning models to better capture the representative compiler-specific patterns and features from instruction sequences for compiler identification. As advanced neural network structures,

both RNN and CNN have achieved great success in sequence learning. As such, in this work, we design an RNN model and a CNN model respectively to learn the semantic and structural information of instruction sequences and thus leverage these advances to identify their compilers.

1) RNN Model: RNN is known to learn the sequential dependency, and strict to align the positions and contexts for the instances in the input sequences. Considering that some instructions may play more significant roles in the function or some instructions may be uniquely generated by specific compilers, RNN is able to attend such instructions and learn a comprehensive and contextualized embedding for the whole instruction sequence. In this work, we employ Long Short-term Memory (LSTM) or Gated Recurrent Unit (GRU), either of which is an architecture designed for RNN to address the vanishing and exploding gradient issue. The structure of our RNN model is shown in Figure 2. Each instruction in the input sequence is first embedded in vector space. Afterwards, the model reads the input instruction sequence through LSTM/GRU units to obtain the summary vector, which is then fed to a Softmax layer to predict the real compiler. The training loss is adopted to measure the correctness of sequence learning and compiler prediction. During the training process, dropout is also applied to prevent the neural network from overfitting.

2) CNN Model: Different from RNN, CNN is known to learn the local correlations with shared weights and utilize pooling mechanism to greatly reduce the number of parameters needed to find important local patterns. In other words, CNN is able to attend those frequently co-occurring instructions in the short sequences. In our model formulation, we further take advantage of different kernel-size filters to thoroughly extract interacted salient features among different instruction grams to capture the behaviors of compilers. The structure of our CNN model is shown in Figure 3. Each instruction sequence is first transformed into a matrix, where each row of the matrix is the instruction's embedding. We take such an embedding matrix as input fed to the CNN architecture to learn the higher-level concept. In the convolutional layer, the raw instruction feature matrix gets convoluted by different kernels of size 2, 3 and 4 such that different views of feature patterns can be extracted in parallel, which are then passed through 1-maxpooling layers for dimensionality reduction. The resulting representations are concatenated through a dense layer to be fed to a Softmax layer for compiler prediction. The CNN model is trained using the instruction sequences with ground truth.

IV. EXPERIMENTS AND EVALUATION

A. Dataset Construction

To evaluate the performance of NeuralCI, we collected 9 widely used C/C++ open source projects, including coreutils 8.31, curl 7.65.3, FreeImage 3.17, git 2.22, libpng 1.6.3, pigz 2.4, x264, vim 8.1.19 and sqlite 3.22, as the basics to construct the dataset. To be specific, we process these projects with the following steps:

- Two different compilers involving multiple versions including GCC (4.6.3, 4.7.4, 4.8.5, 4.9.4, 5.5.0, 6.5.0, 7.4.0) and Clang (3.8, 5.0, 6.0), as well as varying

compiler optimization levels (O0, O1, O2, O3) are used as the toolchain settings to compile each project.

- IDA Pro is then used to identify and extract functions from each binary. Also, we get rid of trivial functions (functions containing just a few instructions, such as the stub functions) that are meaningless to analyze. We consider functions containing less than 10 instructions as trivial in our current setting.

- To avoid the neural models see during the training phase functions that are really similar to the ones in the testing phase, which if not properly accounted for can inflate the performance metrics, we only keep unique functions. Specifically, a function is considered redundant if it has the same abstracted instructions as any other functions'. Then we label each remaining function with the compiler settings used to compile the binary that the function resides in.

With these settings, we finally construct a dataset comprised of totally 413,119 unique functions.
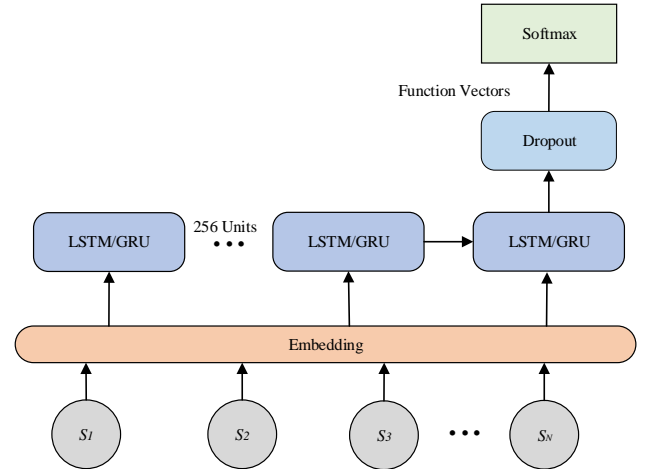
Figure 2. RNN-based model
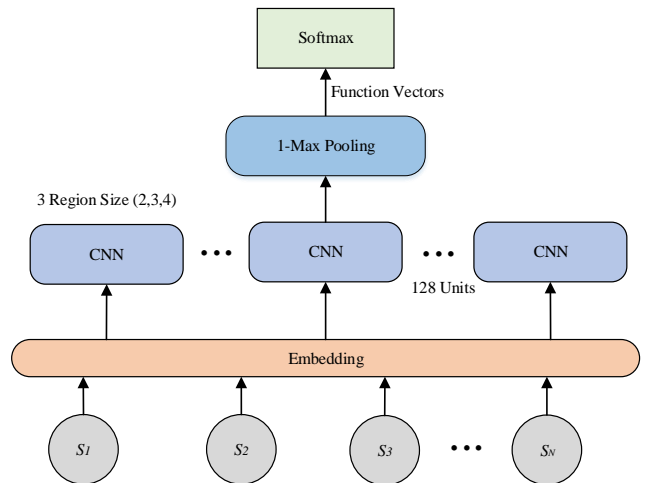
Figure 3. CNN-based model

B. Implementation Details and Experimental Settings

We have implemented NeuralCI as a prototype tool. It utilizes IDA Pro for the parsing of binaries to obtain functions as well as the raw assembly instructions. The function abstraction module is implemented in Java, and the neural modeling module is implemented using Python and the Tensorflow framework. Skip-gram model provided by gensim is used to generate the instruction embedding vectors with the embedding size setting to 100.

For the experimental settings, we randomly split the whole dataset into training set, validation set and testing set according to a percentage of 70%, 15% and 15% respectively. The neural models are trained with a RTX2080Ti GPU card using a batch size of 500, learning rate 0.001 and Adam optimizer for 100 epochs (Note that for each epoch the training samples are shuffled and accuracy on the validation set is calculated). Then we take the model with the best validation accuracy as the final model to be further evaluated on the testing set with respect to performance metrics including accuracy, precision, recall and f1-score.

C. Evaluation

In the following parts, firstly we evaluate the performance of NeuralCI in identifying the compiler family and optimization level respectively and get it compared against state-of-the-art function level compiler identification methods. Then we further explore for the first time the applicability of NeuralCI on compiler identification of arbitrary binary code snippets which can be just a part of a complete function. Considering that several different neural structures are implemented in NeuralCI, we use NeuralCI$_x$ where x can be LSTM, GRU or CNN to get them distinguished.

1)Performance of Identifying Compiler Family: In this experiment, we take the compiler family that each function is compiled with as the ground-truth, and get the NeuralCI models trained and evaluated by adopting the experimental settings as described in Section IV.B. Also, we compare them against existing methods including Structure2Vec, Idioms, Graphlets and BinComp that support function level compiler family identfication. Table I summarizes the experimental results[2]. As it shows, the three NeuralCI models achieve nearly perfect accuracy and f1-score in identifying the compiler family and they all outperform existing methods in terms of detection accuracy. Also, the CNN based one Neural$_{CNN}$ exhibits the best performance with an accuracy of 99.5% and f1-score of 0.995, an improvement of 6.3% against Idioms and the least improvement of 0.7% against Graphlets.

2)Performance of Identifying Optimization Level: In this experiment, the optimization levels of certain compilers are

---

[2] It should be noted that most existing methods do not provide a public access to their source code implementation or the dataset they used. So we give the best performance exhibited in their original evaluations rather than conducting a direct comparison with them. Also, not all performance metrics as we evaluated against are used in their original work, we mark these missing data with a symbol '-' in Table I as well as the following tables.

---

taken as the ground-truth to train and evaluate NeuralCI. We do not use the default 4-level optimization option setting, but adopt the same strategy as in works [12, 13] that condense the 4 optimization levels to 2 classes 'low' and 'high', considering the findings presented in existing studies [12, 13] that it is difficult to distingush between O2 and O3 compiled binaries. That is, O0 and O1 will be considered as the low optimization class, while O2 and O3 will be considered as the high optimization class. Similarly, we compare NeuralCI with methods having function-level compiler optmization level identification results. As summarized in Table II the detection results on GCC optimzation levels, relatively good detection accuracy of above 90% are observed for the NeuralCI models, while the Graphlets method achieves the best accuracy of 97.1% according to their original evaluation result. But as the reproduction evaluation conducted by BinComp on a different dataset shows, the Graphlets method achieves an f-score of just 0.62, much lower than those of NeuralCI's. Similar results can be observed for evaluation on the Clang optimization levels in Table III, where an accuracy of above 90% is achieved for the NeuralCI models. There's no data for other methods, as they didn't evaluate on the Clang compiler.

TABLE I
COMPILER FAMILY IDENTIFICATION RESULTS

Model	Accuracy	Precision	Recall	F1-Score
NeuralCI$_{LSTM}$	99.4%	0.994	0.993	0.994
NeuralCI$_{GRU}$	99.3%	0.994	0.99	0.994
NeuralCI$_{CNN}$	**99.5%**	**0.995**	**0.995**	**0.995**
Structure2Vec	98.2%	0.98	0.98	0.98
Idioms	93.2%	-	-	-
Graphlets	98.7%	-	-	-
BinComp	97.0%	-	-	-

TABLE II
OPTIMIZATION LEVEL IDENTIFICATION RESULTS FOR GCC

Model	Accuracy	Precision	Recall	F1-Score
NeuralCI$_{LSTM}$	91.3%	**0.914**	0.912	**0.914**
NeuralCI$_{GRU}$	91.2%	0.913	**0.916**	**0.914**
NeuralCI$_{CNN}$	90.9%	**0.914**	0.907	0.910
Graphlets	**97.1%**	-	-	-
BinComp	91.0%	-	-	-

TABLE III
OPTIMIZATION LEVEL IDENTIFICATION RESULTS FOR CLANG

Model	Accuracy	Precision	Recall	F1-Score
NeuralCI$_{LSTM}$	**0.914**	0.920	**0.917**	0.908
NeuralCI$_{GRU}$	0.903	**0.925**	0.912	**0.924**
NeuralCI$_{CNN}$	0.900	0.915	0.905	0.903

3)Evaluation on Isolated Binary Code Snippet: The prologue (the preparation of stacks and registers to be used)

and epilogue (the lines of code appearing at the end of a function for restoring the stack and registers to the state before the function is called) of a function play important roles in many binary analysis tasks. Also, as shown by the analysis presented in the works of Austin[15] and Toderici [16], *push*, *mov* and *pop* instructions (which are the main components of prologues and epilogues) show significant effect in distingushing different compilers. So in this experiment we explore whether NeuralCI still works on arbitrary binary code snippets that may not contain function prologue and epilogue. To achieve that, we apply NeuralCI to identify compiler family for each basic block containing no less than 10 instructions. As depicted in Table IV, reduced performance is observed compared with the results on individual functions, but still pretty good accuracy (around 96%) and f1-score (around 0.96) are achieved. It indicates that NeuralCI can capture very subtle yet significant features which may otherwise be missed by artifically crafted feature extraction and selection strategies.

TABLE IV
COMPILER FAMILY IDENTIFICAITON ON INDIVIDUAL BASIC BLOCKS

Model	Accuracy	Precision	Recall	F1-Score
NeuralCI$_{LSTM}$	**0.962**	**0.962**	**0.961**	**0.961**
NeuralCI$_{GRU}$	0.960	0.959	0.959	0.960
NeuralCI$_{CNN}$	0.961	0.960	0.960	**0.961**

V. CONCLUSION

In this work, we attempt to solve the problem of fine-grained compiler identification by feeding in typical neural networks with abstracted instruction sequences generated with a light-weight function abstraction strategy. We implement our methods in a prototype tool NeuralCI, and get its performance evaluated on a large dataset consisting of totally 413,119 unique functions. As the experimental evaluation shows, NeuralCI outperforms most state-of-the-art function level compiler identification methods. It achieves above 99% and 90% accuracy in identifying compiler family and optimization level respectively. Moreover, the evaluation we conducted of applying NeuralCI for the tougher task that infers compiler family from arbitrary binary code snippet achieves rather good accuracy of 96% and f1-score of 0.96. Future works will focus on the model interpretability as well as designing more powerful neural nets such as network structures with attention mechanism.

ACKNOWLEDGMENT

REFERENCES

[1] Schrittwieser S, Katzenbeisser S, Kinder J, Merzdovnik G, Weippl, E. Protecting software through obfuscation: Can it keep pace with progress in code analysis?[J]. ACM Computing Surveys (CSUR), 2016, 49: 1-37.

[2] Schrittwieser S, Katzenbeisser S. Code obfuscation against static and dynamic reverse engineering[C]//International workshop on information hiding. Springer, Berlin, Heidelberg, 2011: 270-284.

[3] Roundy K A, Miller B P. Binary-code obfuscations in prevalent packer tools[J]. ACM Computing Surveys (CSUR), 2013, 46(1): 1-32.

[4] Ugarte-Pedrero, X, Balzarotti, D, Santos, I, & Bringas, P. G. SoK: Deep packer inspection: A longitudinal study of the complexity of run-time packers[C]//2015 IEEE Symposium on Security and Privacy. IEEE, 2015: 659-673.

[5] F. Zuo, X. Li, P. Young, L. Luo, Q. Zeng, and Z. Zhang. Neural machine translation inspired binary code similarity comparison beyond functions pairs[J].arXiv preprint arXiv:1808.04706,2018.

[6] Z. Z. Tian, T. Liu, Q. H. Zheng, E. Zhuang, M. Fan, and Z. J. Yang. Reviving sequential program birthmarking for multithreaded software plagiarism detection[J]. IEEE Transactions on Software Engineering, 2017, 44(5): 491-511.

[7] Z. Z. Tian, Q. H. Zheng, T. Liu, M. Fan, E. Y. Zhuang, and Z. J. Yang, Software plagiarism detection with birthmarks based on dynamic key instruction sequences[J]. IEEE Transactions on Software Engineering, 2015, 41(12): 1217-1235.

[8] IDA. Available: https://www.hexrays.com/products/ida/index.shtml.

[9] PEiD. Available: https://www.aldeid.com/wiki/PEiD.

[10] LANGUAGE 2000. Available: https://farrokhi.net/language/.

[11] Rosenblum N E, Miller B P, Zhu X. Extracting compiler provenance from program binaries[C]//Proceedings of the 9th ACM SIGPLAN-SIGSOFT workshop on Program analysis for software tools and engineering. 2010: 21-28.

[12] Rosenblum N, Miller B P, Zhu X. Recovering the toolchain provenance of binary code[C]//Proceedings of the 2011 International Symposium on Software Testing and Analysis. 2011: 100-110.

[13] Rahimian A, Shirani P, Alrbaee S, Wang L, & Debbabi M. Bincomp: A stratified approach to compiler provenance attribution[J]. Digital Investigation, 2015, 14: S146-S155.

[14] Toderici A H, Stamp M. Chi-squared distance and metamorphic virus detection[J]. Journal of Computer Virology and Hacking Techniques, 2013, 9(1): 1-14.

[15] Austin T H, Filiol E, Josse S, et al. Exploring hidden markov models for virus analysis: a semantic approach[C]//2013 46th Hawaii International Conference on System Sciences. IEEE, 2013: 5039-5048.

[16] Yang S, Shi Z, Zhang G, et al. Understand Code Style: Efficient CNN-Based Compiler Optimization Recognition System[C]//2019 IEEE International Conference on Communications (ICC). IEEE, 2019: 1-6.

[17] Shin E C R, Song D, Moazzezi R. Recognizing functions in binaries with neural networks[C]//24th USENIX Security Symposium (USENIX Security 15). 2015: 611-626.

[18] Chua Z L, Shen S, Saxena P, & Liang Z. Neural nets can learn function type signatures from binaries[C]//26th USENIX Security Symposium (USENIX Security 17). 2017: 99-116.

[19] Massarelli L, Di Luna G A, Petroni F, et al. Investigating graph embedding neural networks with unsupervised features extraction for binary analysis[C]//Proceedings of the 2nd Workshop on Binary Analysis Research (BAR). 2019.

[20] Zuo F, Li X, Young P, et al. Neural machine translation inspired binary code similarity comparison beyond function pairs[C]//Proceedings of the Network and Distributed Systems Security Symposium (NDSS), 2019.

[21] G. Zhao and J. Huang, "Deepsim: deep learning code functional similarity," in Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, 2018, pp. 141-151: ACM.

[22] Mikolov T, Chen K, Corrado G, et al. Efficient estimation of word representations in vector space[J]. arXiv preprint arXiv:1301.3781, 2013.

Evaluating the Relationship of Personality and Teamwork Quality in the Context of Agile Software Development

Alexandre Gomes
Federal University of Campina Grande
Intelligent Software Engineering Group
alexandre.gomes@virtus.ufcg.edu.br

Manuel Silva
Federal University of Campina Grande
manuel.silva@virtus.ufcg.edu.br

Dalton Cézane Gomes Valadares
Federal University of Campina Grande
Federal Institute of Pernambuco
dalton.valadares@embedded.ufcg.edu.br

Mirko Perkusich
Intelligent Software Engineering Group
mirko@virtus.ufcg.edu.br

Danyllo Albuquerque
Federal University of Campina Grande
Intelligent Software Engineering Group
danyllo@copin.ufcg.edu.br

Hyggo Almeida, Angelo Perkusich
Federal University of Campina Grande
Intelligent Software Engineering Group
(hyggo@dsc.ufcg.edu.br
perkusich@dee.ufcg.edu.br)

Abstract—The software industry is increasingly adopting agile software development (ASD). A characteristic of ASD is of focusing on people over processes. Given this, the literature presents models to evaluate teamwork quality for agile teams. Another perspective is to predict the team's behavior, given the members' personality. This study aims to evaluate the effect of the personality of a team on its teamwork quality. For this purpose, we executed an empirical study collecting data from 38 subjects from five software teams, using a psychometric and a teamwork quality instrument presented in the literature. We triangulated the data from both instruments to check their agreement through correlation analysis. As a result, the soft skills expected given the psychometric instrument were observed given the metrics presented in the teamwork quality instrument, evidencing the impact of the team's personality on its efficiency. Moreover, we observed that the personality of the project manager has a direct impact on the behavior of the team. The results presented herein show that personality instruments might be used to predict the team's behavior having several applications such as assisting in forming teams.

Keywords: Agile team; Psychometric instruments; Personalities; Bayesian networks.

I. Introduction

Recently, Agile Software Development (ASD) has become the mainstream development method of choice [10]. It consists of a change-driven approach to developing software in the context of volatile requirements [11]. ASD relies highly on people factors, as evidenced in the Agile Manifesto: six out of the twelve principles are teamwork related factors such as collaboration, communication, and motivation.

The literature presents several studies exploring the relationship between personal characteristics and ASD. Misra et al. [16] explored success factors in adopting ASD and concluded that personal characteristics of the team members, including interpersonal and communication skills, collaborative attitude, and sense of responsibility, are a crucial success factor. Sheffield and Lemétayer [19] claim the empowerment of the team members is one of the factors that characterize agility. This claim leads to the importance of the team members having proper personal characteristics, as identified by Misra et al. [16]. These claims are confirmed by other studies such as Sahibuddin et al. [18] and Dhir et al. [4].

Researchers have also proposed models to assess ASD teamwork quality. For this purpose, Freire et al. [7] presented a Bayesian network, Lindsjørn et al. [14] applied the Structural Equation Modeling-based instrument, previously proposed by Hoegl and Gemuenden [12], to agile teams, and Moe et al. [17] proposed a Radar Plot. Each of these studies proposes a construct to assess teamwork quality based on expected personal characteristics such as communication, collaboration, and cohesion.

Another perspective is, instead of modeling the team's characteristics or expected behavior, predicting them given the team members' personality. Several researchers have explored this theme in software engineering, such as Kosti et al. [13], Yilmaz et al. [21], Capretz [2], Farhangian et al. [5], and Cruz et al. [3]. For instance, Kosti et al. [13] demonstrated that personality is a predictor for the individual's preference regarding software engineering tasks. Given this, we hypothesize the personality is also a predictor for the teamwork quality.

For this purpose, we executed an empirical study collecting data from 39 subjects from five software teams, using a psychometric and a teamwork quality instrument presented in the literature. We used the 16 Personality Factors questionnaire and the TeamWork Quality (TWQ) model proposed by Freire et al. [7]. We triangulated the data from both instruments to check their agreement through correlation analysis. This study synthesizes the findings of our empirical study focusing on the analysis of the level of agreement of the applied instruments and discussing its implications.

The remaining of this paper is organized as follows: Section II presents an overview of psychometric instruments and the TWQ model proposed by Freire et al. [7]. Section III describes the methodology adopted in this research. Section IV discusses our findings; Section V discusses the study's threats to validity. Finally, Section VI presents our conclusions and

future works.

II. BACKGROUND

This section presents an overview of psychometric instruments (Section II-A) and the teamwork quality (TWQ) model proposed by Freire et al. [7] (Section II-B).

A. Psychometric Instruments

One of the main views of personality psychology is the personality can be described by a set of characteristics, being a fixed set of patterns of how a person behaves, feels, and thinks[15]. These characteristics can be used to summarize, explain, or even predict how a person will act in different situations [6]. To determine these personality characteristics, analysts use to apply **psychometric instruments**. The psychometric instruments act as identifiers for personalities. Among them, the most used instruments by psychologists and coaches are the following:

- The **Myers-Briggs Type Indicator (MBTI)**, based on Jungian theory and, to the best of our knowledge, it is the most used psychometric instrument. The MBTI has four dimensions: (i) Extroversion vs. Introversion, (ii) Sensing vs. Intuition, (iii) Thinking vs. Feeling, and (iv) Judging vs. Perceiving [8]. Based on 93 forced-choice items (only two options of which one has to be chosen), a licensed MBTI assessor can identify the type of a person based on the largest score for each bipolar dimension. In theory, each of the 16 different personality types measured by MBTI can be viewed as collections of packaged traits [21].
- The **Big Five Inventory (BFI)** is a structure that considers five factors (i.e., Openness, Consciousness, Extroversion, Kindness, and Neuroticism) which are essential for classifying individual differences in terms of personality characteristics [9]. Based on five comprehensive dimensions (i.e., the personality characteristic), this model suggests a personality visualization. It is worthy to mention that this instrument is one of the most reorganized by personality researchers [13]
- The **16 Personality Factors (16PF)** questionnaire is a psychometric instrument, from the same family as the BFI, that presents a reliable measure of 16 personality characteristics, describing and predicting a person's behavior in several contexts. It is used to select, develop, and motivate people to make organizations thrive. With over 50 years of research, the insights selected by the 16PF instrument are authenticated by more than 2,700 independent research articles, reviewed by experts, displaying a highly reliable and accurate indicator of future behavior and presumably success [5].

These psychometric instruments, in addition to revealing the personality of the individual, also establish the soft skills determined for each personality type [6]. Although the software industry has become a vital force in society, attracting people of all psychological types, specific characteristics are more represented than others in the software engineering field [20]. For instance, according to Barroso et al. [1], the software field is dominated by introverts, who typically have difficulty communicating with the software end-users.

Since in this study we use the 16PF questionnaire, we describe it in more detail in what follows. We discuss the reasoning for choosing this psychometric instrument in Section III-A. The 16PF generates 16 types of personalities, formed by acronyms generated from the dichotomies emitted by the psychometric instrument [2]. The acronyms are generated from the combination of the initial letters of the ten psychological preferences [3]. For instance, INTJ is obtained from a combination of **IN**tuitive, **T**hinking and **J**udgment. Likewise, ISTP is obtained from **I**ntrover**S**ion, **T**hinking and **P**erception. In the following is presented five dichotomies used in the psychometric instrument applied in the present study:

- Mind: Extroversion; Introversion.
- Energy: Intuitive; Sensitive.
- Nature: Thinking; Feeling.
- Tactic: Judgement; Perception.
- Identify: Assertive; Cautious.

According to combination of aforementioned characteristic, the personalities are organized in the following four groups:

- **Analyst personalities**: INTJ-A/ INTJ-T; INTP-A / INTP-T; ENTJ-A / ENTJ-T; ENTP-A / ENTP-T.
- **Diplomat personalities**: INFJ-A / INFJ-T; INFP-A / INFP-T; ENFJ-A / ENFJ-T; ENFP-A / ENFP-T.
- **Sentinel personalities**: ISTJ-A / ISTJ-T; ISFJ-A / ISFJ-T; ESTJ-A / ESTJ-T; ESFJ-A / ESFJ-T.
- **Explorer personalities**: ISTP-A / ISTP-T; ISFP-A / ISFP-T; ESTP-A / ESTP-T; ESFP-A / ESFP-T.

B. Bayesian Networks for Teamwork Quality Estimation

Freire et al. [7] proposed a Bayesian Network (BN) that models the causal relationships between the key factors that influence TWQ. A Bayesian Network is a probabilistic graphical model that exposes conditional dependency or causality, representing random variables by nodes and conditional dependence by edges in a Directed Acyclic Graph (DAG).

The model outputs a probability distribution that represents the current TWQ and can assist in its process of continuous improvement. Each node in the network represents a key factor of the TWQ, and an edge between two nodes represents a causal relationship (i.e., influence). Also, each key factor has possible states, and each has an associated probability [7]. Figure 1 presents the main variables that make up the model.

To feed the model, the user inserts data related to each input node, according to the observations made by the team during the software development time-frame (e.g., sprint in ASD context). Then, the outputs must be calculated using a Bayesian inference tool, such as AgenaRisk[7]. The calculated results represent probabilities for each node state regarding the quality level of each factor in the current state of the project.

III. METHODOLOGY

In this section we present our approach, Personality-based Teamwork Skills (PTS), which uses psychometric instruments to identify the personalities of team members. Based on this classification, we estimate some teamwork skills interesting for
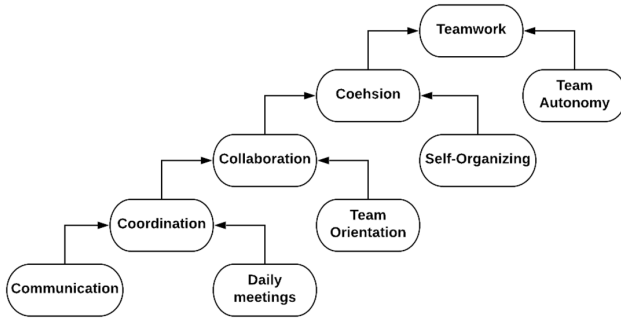
Fig. 1. Key factors that influence agile teamwork

the software development process, comparing our results with the ones found through the Bayesian Network model proposed by Freire et al [7]. Next, we describe our methodology, as well as the environment in which we obtained the data and how we performed the validation.

A. Methods

Among the psychometric instruments presented in the previous session, the 16 Personality Factors (16PF) Questionnaire was chosen to be applied in this study. This choice occurred due to several factors as (i) its free availability on the internet, (ii) its ease of use, and its data analysis to have a vocabulary that is easy to understand [5], (iii) It is one of the most used instruments in the field of psychology [6]. This instrument generates a descriptive analysis of the personality and the soft skills, associated with the one obtained, according to the responses collected and inserted in the instrument, for each individual that composes the work team.

The 16PF is available on the internet and it has about 60 questions. With an application time average about 15 minutes, each one has a scale of markup variation, composed of seven marking circles, which vary from: "Agree"; "Neutral" and "Disagree". The choice of marking directly influences the results obtained. With this, the questionnaire was adapted/transcribed in the text editor and a short header was added, which asked the individual to identify the sex and function performed in the project. Then, printed and applied in loco, with the work teams, of five ongoing projects in the company, formed by a project manager, scrum master; test developers and analysts, totaling 39 subjects (6 women and 33 men) with ages ranging from 23 to 35 years with working experience.

After applying the psychometric instrument, the answers obtained from the printed questionnaires were inserted, one by one, in the same online questionnaire available on the 16PF website, after inserting the questionnaires, descriptive reports of the personalities were generated and saved in a digital folder (i.e., in .pdf format). Then, it aims to perform the analysis of the personalities described in the reports generated by the 16PF, the percentages of the dichotomies issued were generated. These data were correlated with the percentage of nodes and states of the Bayesian network generated from the same agile teams of the applied projects of this research.

B. Environment

We apply the psychometric instrument in a Brazilian software company that works with Scrum-based projects, centralizing its workforce mainly in the development of Web Systems. The company focuses on executing RD projects, in cooperation with other Information Technology and Communication companies, demanding temporary efforts with predefined objectives to create new products, services, or processes. Nowadays, the company executes more than 40 projects together with multinational partners. The projects follow agile methods and each team has 5 to 10 members, depending on the project requirements. Each person spends about 15 minutes - on average - to complete the questionnaire, which was estimated by the 16PF psychometric instrument itself. The questionnaire was applied at the same time to all members, where each answered his own. One of the authors organized this activity and received support from two other authors for coordinating and controlling this activity.

C. Validation

Our validation was performed by analyzing the data generated in the reports issued by the 16PF psychometric instrument. Then, we correlated the aforementioned data with other ones generated from the Bayesian network (See figure 1). In addition to the information described the individual's personalities, the 16PF issues five dichotomies (each one with two partitions that help to describe the personality): mind, energy, nature, tactics and, identity. As seen in Table I.

For each dichotomy, a graph is generated with values, in percentages, based on the factorial analysis, these values are generated from the variations of the responses of the markup variation scale: "I agree"; "Neutral" and "Disagree", contained in the 16PF questionnaire. This instrument uses the matrix of inter-correlations between these variables as a starting point, in an attempt to discover the underlying traits of human personality.

IV. RESULTS AND DISCUSSION

After the application of the psychometric instrument, the obtained data were entered into the 16PF web platform. Then, after the individual generation of the personalities report of each team member, the data were inserted into an electronic spreadsheet aiming at organizing them. According to that, it was possible to analyze which were the most predominant personalities of the agile teams by members as well as by function.

The personalities of the agile team members were organized by function, then we made a comparative analysis with the Bayesian model applied to the same projects. From the accomplishment of the previous task, this allowed us to make the correlation of the personalities with the Bayesian model under study.

Figure 2 shows how the types of personalities of the agile teams found in the project 1 (P1) were organized, being composed by the personalities of the project manager (PM) and the work team. During the analysis, we noticed in the same team there were several types of personalities, being
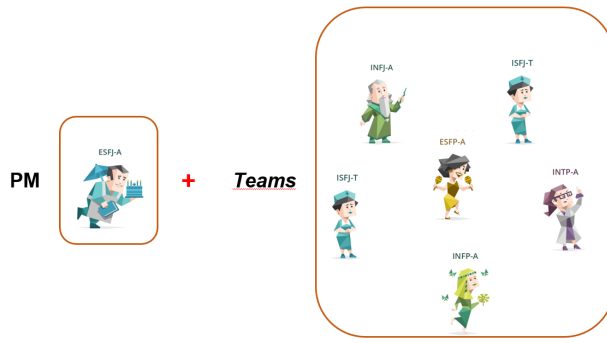
Fig. 2. Types of personalities from the agile project 1 team

able to contemplate the four personality groups presented in section II, named by the instrument under study.

After identifying the personalities of the team, the average of the percentages of the dichotomies obtained from each member was calculated and consolidated in a single graph containing all percentages of the 10 psychological preferences, attributed by the 16PF. The result of the aforementioned task can be seen in Table I below.

TABLE I
DICHOTOMIES OF THE FIVE AGILE PROJECTS (P)

DICHOTOMIES		P1	P2	P3	P4	P5
MIND	EXTROVERTED (%)	40.6	75.5	46	71.7	60
	INTROVERT (%)	52.4	24.5	51	28.3	40
ENERGY	INTUITIVE (%)	45	28.5	42.7	48.9	52.2
	OBSERVER (%)	55	71.5	57.3	51.1	47.8
NATURE	THINKING (%)	41.6	32	34.3	30.3	39.5
	FEELING (%)	58.4	68	65.7	69.7	60.5
TACTIC	JUDGER (%)	63.6	79	71.9	58.6	53.5
	EXPLORER (%)	36.4	21	28.1	41.4	46.5
IDENTITY	ASSERTIVE (%)	69.3	63	57.9	56.6	55.3
	CAUTIOUS (%)	30.7	37	42.1	43.4	44.7

Through the percentages used for each psychological dichotomy, it was possible to identify which were the predominant psychological users in the agile teams for each project. We performed the same analysis for each evaluated project, taking into account the data obtained from the dichotomies of each one. Next, the analysis of the dichotomies prevalent in each project will be shown, according to the data provided in Table I. In what follows, we described the analysis of the predominant dichotomies of software projects:

- **Mind:** The psychological preference *Introvert* - which covers the team members who have the following characteristics: Receptors, Contained, Reflexive and Quiet - had a *slightly higher percentage in projects 1 and 3*. On the other hand, the psychological preference *Extroverted* - which concerns the team members who have characteristics such as Initiatives, Expressive, Active and enthusiastic - was predominant in the *projects 2, 4 and 5*.
- **Energy:** The psychological preference *Observer* - which covers the team members who have the following characteristics: Concrete; Realistic, Practical and Traditional - had a *slightly higher percentage in projects 1, 2, 3 and*

4. Additionally, the psychological preference *Intuitive* - which concerns the team members who have characteristics such as Imaginative, Conceptual, Theoretical and Original - obtained *greater occurrence in project 5*.

- **Nature:** The psychological preference *Feeling* - which covers the team members who have the following characteristics: Empathic, Sensitive and Receptive - achieving a *high occurrence in all evaluated projects*. On the other hand, the psychological preference *Thinking* - which concerns the team members who have characteristics such as: Logical, Questioners, Reviews and Reasonable - obtained *greater occurrence in project 1 and 5*.
- **Tactic:** The psychological preference *Judger* - which covers the team members who have the following characteristics: Systematic, Planned, Anticipated and Methodical - had a *higher percentage in projects 2 and 3* whereas the psychological preference *Explorer* - which concerns the team members who have characteristics such as Informal, Open, Situational and driven by pressure - obtained *greater occurrence in projects 4 and 5*.
- **Identify:** The psychological preference *Assertive* - which covers the team members who have characteristics such as Objectives and Direct - achieving *higher percentage in all evaluated projects*. In addition, the psychological preference *Cautious* - which concerns the team members who have the following characteristics: Weighted, Moderates and Cautions - obtained *greater occurrence in projects 4 and 5*.

Figure 3 shows the following current TWQ of the agile project team 1. The Bayesian networks have the following key factors: *Cohesion; Teamwork; Self-Management; Collaboration; Team orientation; Coordination; Daily Meetings and Communication*. Each factor is composed of the states: *(i) Very low; (ii)Low; (iii) Medium; (iv) High and (v) Very high*. Each one of these factors contains the corresponding percentages of occurrence. These data were acquired by the answers raised by the questionnaire applied to the members of the agile team. The input nodes assigned to their states in project 1 by the author [7] were: Team Autonomy; Team Learning; Team leadership; Expertise; Personal Attributes; Presence of all members; Monitoring; Team Communication and Distribution. For each node, how their state's probabilities are identified and represent the level and quality of each factor in the current state of the project.

Observing the Figure 3, we can conclude the *team members belonging to project 1 had a good performance*, despite the team's autonomy having negatively influenced the teamwork, the other factors were assessed as high quality, which contributed to a high value of TWQ.

From that, it was possible to make a direct correlation between (i) the influence of the predominant psychological preference - using the percentages obtained utilizing the psychometric instrument 16PF - in contrast to (ii) the percentages shown for each state with their respective key factors, associated with the input nodes of the model shown in Figure 3.

after analyzing the project 1, we perform the same analyses for other projects (i.e, projects 2, 3, 4 and 5). For doing
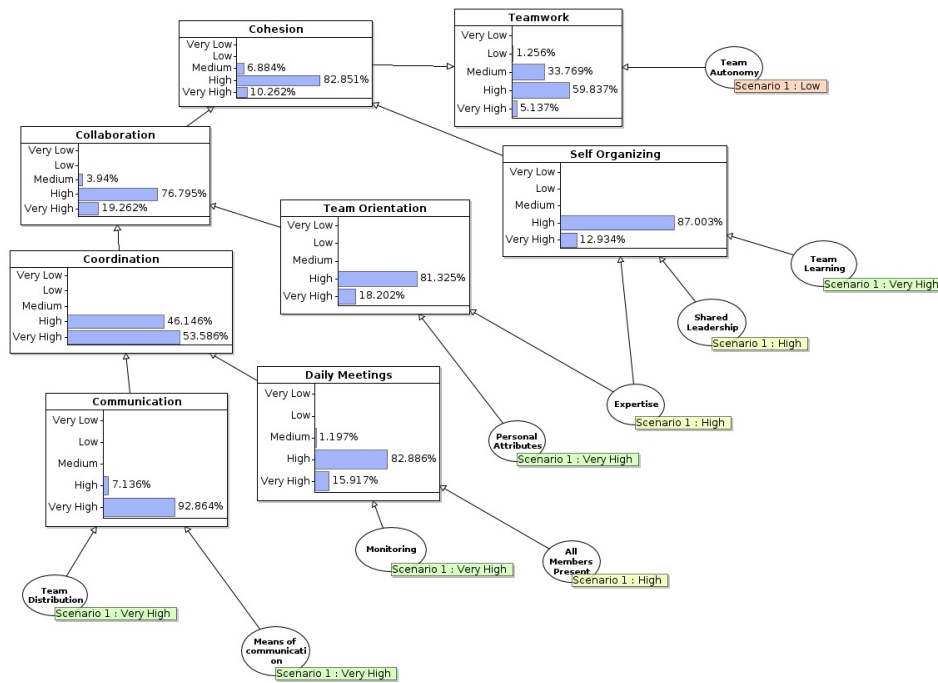
Fig. 3. Bayesian Network Diagram Project 1

so, Bayesian networks were generated with the same input nodes, main factors and states, according to Bayesian Network generated from project 1 (See Figure 3). Due to space limitations, further details of these empirical activities can be accessed through the supplementary material available at https://bit.ly/39rMTQI .

It was possible to correlate the input node "Team autonomy" in contrast to the dichotomy "mind". According to previous results, the team members of project 1 has a 'Receptive' profile because there was a greater prevalence in the psychological preference 'Introvert'. Additionally, we do not realize a negative impact on the team's performance. it occurs due to the (i) 'extrovert' psychological preference had a minimal difference in the percentage obtained and (ii) the profile of the project manager also has this same psychological preference. Therefore, we conclude the psychological preference of the project manager has a direct influence on the performance of the team. The projects 2, 3, 4 and 5 had similar results to those ones obtained by project 1. Even though the nodes had an impact factor very low or medium, the team's results were good. Similarly, the project manager personality had a direct influence on the results.

It was possible correlating the input nodes "Team learning and Shared Leadership" in contrast to the dichotomies "Mind and identity" because the agile team of the project 1, proved to be individuals with characteristics such as "assertive, active and initiative". The aforementioned relationship favored the key factor of "Self Management" to reach a state of 96.57% 'High'. The other projects 2, 3, 4 and 5 had results consistent with those of project 1. According to this, we conclude these two dichotomies (i.e. "Mind and identity') directly influenced the results.

It was possible to correlate the input nodes "Expertise and Personal Attributes" in contrast to the dichotomies "Identity and Tactics", since the agile team of the project 1 proved to be individuals with characteristics such as "assertive, planned, anticipated". The aforementioned relationship favored the key factors "Self Management and Team Orientation" to reach their respective percentages of '96, 57% High 'and '81, 33% High'. The projects 2, 3, 4 and 5 had results consistent with those of project 1.

It was possible to correlate the input nodes "Presence of all members and Monitoring" in contrast to the dichotomy "Tactic", since the agile team of the project 1, proved to be individuals with the following characteristics: planned, methodical, anticipated and systematic. This relationship favored the key factor "Meeting Diaries" to reach the status of '87, 54% High'.The projects 2, 3, 4 and 5 had results consistent with those ones obtained by project 1. We can conclude the dichotomy influenced the results obtained.

It was possible to correlate the input nodes "Means of Communication and Team Distribution" in contrast to the dichotomies "Mind' and 'Tactics", since the agile team of the project 1, proved to be individuals with characteristics such as extroverted, initiators, expressive, planned and anticipated. The aforementioned relationship favored the key factor "Communication" to reach the status of '97, 86% Very High '.The projects 2, 3, 4 and 5 had similar results compared to those ones obtained by project 1. According to this, we conclude these two dichotomies directly influenced the obtained results.

The 16PF psychometric instrument applied in the team members who working on the five analyzed projects supporting us to obtain The main following soft skills: Leader; Nice; Organized; Encouraging; Controller; Responsible; Re-

spectful; Perceptible; Reserved; Adaptable; Altruistic; Generous; Perfectionist; Accurate; Noticeable; Dreamer; Enthusiastic; Dedicated; Friendly; Shy; Realist; Charitable; Opnious; Sensitive; Egalitarian; Stressed; Impatient; Insecure; Joyful; Spontaneous; Energetic; Sociable; Curious; Idealistic; Positive; Honesty; Intuitive; Optimistic; Kind, and Communicative.

After analyzing all the projects, we realized the states of the Bayesian Networks met the expectations of the soft skills of the agile teams. We can highlight the state of the key factor "Communication" exceeded this estimation. Even the team members having low levels of soft skills related to communication performance, the fact of having a project manager with a high level of this soft skill favored the high communication performance of the team members under analysis.

V. THREATS TO VALIDITY

We identified a few threats in our work, which follow from the instruments selected, researcher bias, and data being collected from only one company. The 16PF psychometric was the only means for identifying personalities, which can be considered a threat because there are other instruments to accomplish this task. Further, we only used one instrument to measure the TWQ. Even though there is a risk of bias in the measurement processes, both instruments have been evaluated with industry projects.

In addition, the data generated by the psychometric instrument was analyzed only by the authors of this research, with no contribution from any psychology professional, which leads to the risk of researcher bias. To mitigate this threat, the first author held training sessions with specialists aims to use the psychometric instrument correctly.

Finally, we collect data from a single company. this makes it difficult to generalize our conclusions. Therefore, we cannot generalize the study findings to the entire agile context. Unfortunately, collecting real-world data in the field of Software Engineering is not a trivial task because most software development companies are not likely to contribute to academic research.

VI. CONCLUSIONS AND FUTURE WORKS

In this paper, we evaluated the level of agreement of the 16PF psychometric instrument and the Teamwork Quality (TWQ) model proposed by Freire et al. [7], which is based on a Bayesian network. The correlational analysis was based on the data obtained from the dichotomies generated from the 16PF, compared with the data obtained from the input nodes, states, and key factors of the generated Bayesian networks.

As a result, the soft skills expected given the psychometric instrument were observed given the metrics presented in the TWQ instrument, evidencing the impact of the team's personality on its efficiency. Moreover, we observed the personality of the project manager has a direct impact on the behavior of the team. The results presented herein show that personality instruments might be used to predict the team's behavior having several applications such as assisting in forming teams.

For future work, we seek to carry out new empirical studies on the types of personalities of the work teams and their correlation with the individual performance of each team member. Besides, we also seek to carry out new studies aims to confirm the main findings of this research.

REFERENCES

[1] A. S. Barroso, J. S. Madureira, M. S. Soares, and R. P. do Nascimento. Influence of human personality in software engineering-a systematic literature review. In *International Conference on Enterprise Information Systems*, volume 2, pages 53–62. SCITEPRESS, 2017.

[2] L. F. Capretz. Personality types in software engineering. *International Journal of Human-Computer Studies*, 58(2):207–214, 2003.

[3] S. Cruz, F. Q. da Silva, and L. F. Capretz. Forty years of research on personality in software engineering: A mapping study. *Computers in Human Behavior*, 46:94–113, 2015.

[4] S. Dhir, D. Kumar, and V. Singh. Success and failure factors that impact on project implementation using agile software development methodology. In *Software Engineering*, pages 647–654. Springer, 2019.

[5] M. Farhangian, M. Purvis, M. Purvis, and B. T. R. Savarimuthu. Personalities and software development team performance, a psycholinguistic study. In *24th European Conference on Information Systems*, 2016.

[6] R. Feldt, L. Angelis, R. Torkar, and M. Samuelsson. Links between the personalities, views and attitudes of software engineers. *Information and Software Technology*, 52(6):611–624, 2010.

[7] A. Freire, M. Perkusich, R. Saraiva, H. Almeida, and A. Perkusich. A bayesian networks-based approach to assess and improve the teamwork quality of agile teams. *Information and Software Technology*, 100:119–132, 2018.

[8] V. Garousi and A. Tarhan. Investigating the impact of team formation by introversion/extraversion in software projects. *Balkan Journal of Electrical and Computer Engineering*, 6(2):132–140, 2018.

[9] N. Gorla and Y. W. Lam. Who should work with whom? building effective software project teams. *Communications of the ACM*, 47(6):79–82, 2004.

[10] R. Hoda, N. Salleh, and J. Grundy. The rise and evolution of agile software development. *IEEE Software*, 35(5):58–63, 2018.

[11] R. Hoda, N. Salleh, J. Grundy, and H. M. Tee. Systematic literature reviews in agile software development: A tertiary study. *Information and Software Technology*, 85:60–70, 2017.

[12] M. Hoegl and H. G. Gemuenden. Teamwork quality and the success of innovative projects: A theoretical concept and empirical evidence. *Organization science*, 12(4):435–449, 2001.

[13] M. V. Kosti, R. Feldt, and L. Angelis. Personality, emotional intelligence and work preferences in software engineering: An empirical study. *Information and Software Technology*, 56(8):973–990, 2014.

[14] Y. Lindsjørn, D. I. Sjøberg, T. Dingsøyr, G. R. Bergersen, and T. Dybå. Teamwork quality and project success in software development: A survey of agile development teams. *Journal of Systems and Software*, 122:274–286, 2016.

[15] L. G. Martínez, G. Licea, A. Rodríguez-Díaz, and J. R. Castro. Experiences in software engineering courses using psychometrics with ramset. In *Proceedings of the fifteenth annual conference on Innovation and technology in computer science education*, pages 244–248, 2010.

[16] S. C. Misra, V. Kumar, and U. Kumar. Identifying some important success factors in adopting agile software development practices. *Journal of Systems and Software*, 82(11):1869–1890, 2009.

[17] N. B. Moe, T. Dingsøyr, and T. Dybå. A teamwork model for understanding an agile team: A case study of a scrum project. *Information and Software Technology*, 52(5):480–491, 2010.

[18] M. H. N. Nasir and S. Sahibuddin. Critical success factors for software projects: A comparative study. *Scientific research and essays*, 6(10):2174–2186, 2011.

[19] J. Sheffield and J. Lemétayer. Factors associated with the software development agility of successful projects. *International Journal of Project Management*, 31(3):459–472, 2013.

[20] Z. Stojanov, T. Zoric, and I. Hristoski. Human factor in software requirements engineering: Preliminary review of qualitative empirical studies. *ZBORNIK RADOVA UNIVERZITETA SINERGIJA*, 19(4).

[21] M. Yilmaz, R. V. O'Connor, R. Colomo-Palacios, and P. Clarke. An examination of personality traits and how they impact on software development teams. *Information and Software Technology*, 86:101–122, 2017.

ISC-FS: An Improved Spectral Clustering with Feature Selection for Defect Prediction

Xuan Zhou[1], Lu Lu[1,2*], and Yexia Qin[2,3]

[1]School of Computer Science and Engineering, South China University of Technology, Guangzhou, China
[2]Modern Industrial Technology Research Institute, South China University of Technology, Meizhou, China
[3]School of Environment and Energy, South China University of Technology, Guangzhou, China
*Corresponding author email: lul@scut.edu.cn

Abstract—We notice the lack of historical data with labels always exists in software defect prediction (SDP). Unsupervised learning and cross-project defect prediction (CPDP) have tried to address this problem. However, traditional unsupervised learning always requires manual intervention while CPDP faces the challenge of heterogeneity between different projects. Therefore, this paper proposed a framework called Improved Spectral Clustering with Feature Selection (ISC-FS) to conduct unsupervised learning for defect prediction without human effort in this paper. First, ISC-FS clusters the software entities and gets pseudo-labels. Second, we do a feature selection, of which the key idea is different clusters hold the different magnitude of features. Last, the selected features are fed to a spectral clustering method based on connectivity-distance. To validate the proposed method, experiments were carried out on 28 projects from PROMISE and NASA datasets, and comparisons were made with the other five unsupervised methods. The results show the promising performance that ISC-FS can outperform referential methods.

Keywords—Software defect prediction, Unsupervised learning, Spectral clustering

I. INTRODUCTION

With the expansion scale of contemporary software, software defect prediction (SDP) has attracted more and more attention, which can help reduce the test burden and optimize the allocation of testers and developers. It is usually believed that the cost of fixing bugs after deployment is higher than that during development. Given the huge cost and limited budget, it is important that SDP is involved in the early stage of the software life cycle.

Researchers have investigated the different algorithms and features to improve the performance of SDP. SDP tasks usually consist of two steps: capturing features from software entities and training a classifier through machine learning methods. Supervised learning methods predominated in the previous research related to SDP. But unfortunately, SDP is not widely used in industry [1]. That is mainly because that the historical data with labels required in supervised learning is often lack in pre-release software; more importantly, it is also expensive and difficult to collect in post-release software [2].

One way to solve this problem is cross-project defect prediction (CPDP), which attempts to use prediction models built

by other projects with sufficient historical data [3]. The main challenge CPDP faces are heterogeneity. On the one hand, different projects may have dissimilar features [4]. On the other hand, even if a source project with the same features as the target project is selected, CPDP still faces the differences in data distribution between source projects and target projects [5]. As shown in Figure 1, the classifier trained by source projects may not be suitable for target projects. Moreover, existing CPDP researches mainly focused on the establishment of a usable target prediction model. However, a large number of irrelevant data usually makes the prediction model perform worse. That explains why CPDP is challenging to achieve promising performance in practice [6].
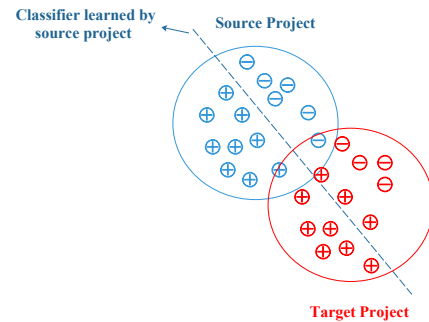
Fig. 1. Divergent data distribution between different projects.

Meanwhile, unsupervised learning has also been used in defect prediction to solve the shortage of labeled data. Nevertheless, early studies on unsupervised learning for SDP often need human effort. To solve this problem, [7] proposed novel approaches showing the defective possibility for software entities by using the magnitude of features. More recently, [8] presented a connectivity-based unsupervised classifier, different from traditional distance-based methods.

As far as this paper is concerned, with the heuristic of [8], we proposed our unsupervised framework for SDP called the Improved Spectral Clustering with Feature Selection (ISC-FS). First, given the importance of data quality [9], the feature selection considering data distribution is added. Second, compared with traditional clustering methods, spectral clustering has the adaptability to different data distributions.

To make the adjacency matrix used in spectral clustering closer to the physical meaning, we present a new adjacency matrix definition, which will be explicated in Section III. Furthermore, following the previous point of view [10], a particular threshold is adopted when labeling entities to avoid human intervention in the classification process. At last, we get the final prediction.

In summary, the contributions of this paper are listed as follows:

- We proposed a novel framework called ISC-FS for SDP on unlabeled datasets.
- To access our proposed method, experiments were conducted on 23 projects in the PROMISE dataset and 5 projects in the NASA dataset. Results showed that ISC-FS performs better over traditional unsupervised methods and state-of-the-art unsupervised approaches for SDP.

The remaining of this paper is structured as followed: In section II, we introduce the related work about SDP and unsupervised learning. Section III presents our method in details. Section IV describes the experimental setup and Section V illustrates the experimental result. In Section VI, we point out the potential threats to validity. The last section concludes and discusses future work.

II. RELATED WORK

We introduce the related work on SDP and unsupervised learning in this section.

A. Software Defect Prediction

SDP is a process predicting whether the software entities have defects or not. There exists many studies on defect prediction. [11] combined semantic and structural scattering to capture project and human characteristics as resource features to build a prediction model. [12] proposed cross-entropy, which carries information representing the difference between two probability distributions. However, all of the above work are using supervised methods.

That is because that people believe unsupervised classifiers usually shown disappointing performance. However, according to the meta-analysis in [2], generally speaking, unsupervised models do not seem to perform worse than supervised models. Under this circumstance, and taking into account that it is easier to collect unlabeled data in the big data era, an effective unsupervised model for SDP when lacking the labeled data is necessary.

B. Unsupervised Learning

Unsupervised learning is a machine learning technique classifying instances to different classes without labeled data. Due to acquiring labeled data is a difficult task, unsupervised learning has been applied in many fields. [13] employed it to the end-to-end training of visual features on large-scale datasets. In SDP, [14] first attempted expert-based unsupervised learning including k-means and neural-gas. Besides, [15] labeled the cluster with a certain threshold. The typical process of unsupervised learning for SDP mainly consisted of two

steps, 1) clustering software instances into k clusters; and 2) identifying each class is defect-prone or not.

Recently, spectral clustering has been used in SDP [8]. Spectral clustering, based on Laplacian mapping, uses minimal cuts to characterize the original data with special orientations that carry the cutting information, and then clusters. The main idea is treating data as vertices (V) in space, and these vertices can be connected by edges (E). In SDP, edges represent the connection between software instances, and its weight is determined by the connection between two instances. For clustering, the graph $G = (V, E)$ will be cut into two disjoint sets. The key idea to this cut is that the edge weights between the subgraphs are as low as possible, and the edge weights within the subgraphs are as high as possible.

III. METHOD

In this section, we explicate our method in detail. Figure 2 illustrates the framework of ISC-FS. To be more specific, ISC-FS includes five steps: (1) preprocessing, (2) spectral clustering, (3) labeling cluster and get pseudo-labels (4) feature selection using pseudo-labels; and (5) re-clustering and labeling by selected features to obtain the final predictions.
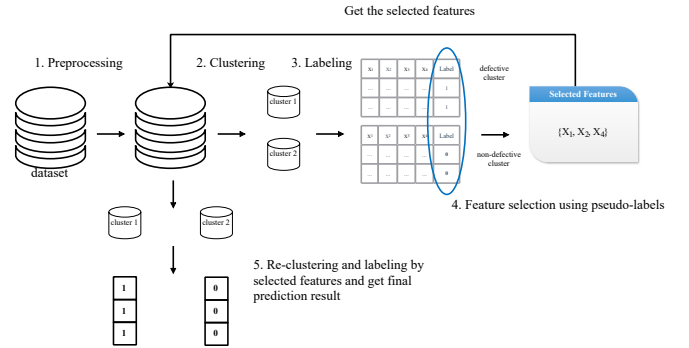
Fig. 2. Overview of ICS-FS framework.

A. Preprocessing software features

Software features usually have various sizes. Take the features in the PROMISE dataset as an example, loc means the line of code, which is usually greater than one hundred,or even thousands. And dam, the data access metrics, is the ratio of all private or protected attributes in the class to all attributes, which means its range is [0,1]. Consequently, ISC-FS used the z-score normalization, which makes data to a normal distribution with a mean of 0 and a standard deviation of 1. For every specific feature x, the normalized features x' can be shown as:

$$x' = \frac{x - mean(x)}{std(x)} \qquad (1)$$

where $mean(x)$ is the average value of x, and $std(x)$ means the standard deviation of x.

Moreover, in some datasets, we observe there are some missing values, which we assign to the average of all existing values of the corresponding features.

B. Spectral Clustering

Guided by the idea of spectral clustering, an adjacency matrix $W \in \mathbb{R}^{n \times n}$ to represent the weight of edges is required at first and we note that n is the number of entities. The previous work [8] adopted the dot product to generate adjacency matrix, as shown in Equation 2.

$$w_{ij} = x_i \cdot x_j = \sum_{k=1}^{m} (a_{ik} * a_{jk}) \tag{2}$$

where m represents the number of features, and a_{ij} represents the value of the j_{th} feature of the i_{th} entity.

However, since spectral clustering assumes that values in the adjacency matrix are non-negative, it simply sets the negative value to zero, which will lead to the loss of some original information. Furthermore, according to Equation 2, two more distant points may have larger weights, which is inconsistent with physical meaning. Thus, we proposed a new adjacency matrix definition as shown in the Equation 3.

$$w_{ij} = \begin{cases} 0 & \text{if } i = j \\ \sum_{k=1}^{m} exp(-(a_{ik} - a_{jk})^2) & \text{if } i \neq j \end{cases} \tag{3}$$

From Equation 3, each feature has a value ranged in [0,1] represents its similarity, and the sum represents the similarity of two entities. Besides, since it is meaningless to focus on the self-circle, the self-circle value is set to zero.

Second, the Laplacian matrix is calculated by the following formulas:

$$L_{ij}^{sym} = \begin{cases} 1 & \text{if } i = j \\ -\frac{1}{\sqrt{d_i * d_j}} & \text{if } i \neq j \end{cases} \tag{4}$$

where L^{sym} represents the symmetric normalized Laplacian matrix, and $d_i = \sum_{j=1}^{n} w_{ij}$.

Last, we conduct the eigendecomposition on the Laplacian matrix L^{sym}. Follow the normalized cut algorithm proposed by [10], we use the second smallest eigenvector, denoted as v, for clustering.

C. Labeling Cluster

After spectral clustering, entities has been divided into two groups. C_1 and C_2 are used to denote two groups respectively. To determine whether a cluster is a defective one, we use the following heuristic: *entities with more defect-proneness tend to have higher complexity*. The features in our datasets measure complexity, which means that larger values represent higher complexity. In this case, we adopt the average row sums of the normalized features to determine which cluster is defective. Calculated by Equation 5, if AVS_{C_1} is greater than AVS_{C_2}, we mark C_1 as defective cluster. Otherwise, C_2 is considered as cluster containing defective entities. Specially, the labels obtained by the first clustering are called pseudo-labels, which are used to assist feature selection, and the labels obtained by the second clustering are the final results.

$$AVS_{C_i} = \frac{\sum_{entity \in C_i} RowSum(entity)}{size(C_i)} \tag{5}$$

where AVS_{C_i} represents the average row sums of cluster C_i, $RowSum(entity)$ is the sum of all feature values of the specified entity, $size(C_i)$ means the size of cluster C_i.

D. Feature-selection

Feature selection has a greater influence than classifier selection [17]. Since that the irrelevant or redundant features not only require more computational cost but also reduce the performance of prediction models [18], feature selection based on feature violation scores (FVS) is added in ISC-FS by removing features with less relevant information. FVS can be calculated by the following equation:

$$FVS_i = \frac{V_i}{Num} \tag{6}$$

where V_i means the number of violations in the i_{th} features and Num is the number of entities.

A violation is a value that does not follow the defect proneness heuristic. To reduce manual intervention, the median or average value is commonly used as a special threshold. Nevertheless, in the inherently imbalanced SDP field, the median or average tends to weaken the prediction performance. Thus, we selected the corresponding defect percentile from pseudo-labels as the threshold.

Algorithm 1 FeatureSelection(x, pseudoY)

1: $percentileX = percentile(x)$
2: **for** $i = 1$ to row **do**
3:     **for** $j = 1$ to col **do**
4:         **if** $pseudoY(i)$ xor $x(i,j) >= percentileX(j)$ **then**
5:             $violate(j) + +$
6:         **end if**
7:     **end for**
8: **end for**
9: **for** $j = 1$ to col **do**
10:     **if** $violate(j) < percentileX(j)$ **then**
11:         select the corresponding feature
12:     **end if**
13: **end for**

The algorithm is shown in Algorithm 1. We give a specific example in Figure 3, the clean rate is $\frac{4}{6}$, and the corresponding percentile is calculated for each feature, shown in bold. For instance, the Entity D is labeled as clean and its X_5 is 10, which is not less than the corresponding percentile, so it should be considered as a violation. After calculating all the FVS, select the features with FVS less than the clean rate. Therefore, X_1, X_2, X_4 as features are finally selected.

E. Re-clustering and labeling by selected features

After feature selection, the results might differ from the pseudo-labels used in feature-selection. In this case, it is necessary to repeat the step of spectral clustering and labeling clustering.

	X_1	X_2	X_3	X_4	X_5	Label
En. A	3	1	3	7	1	Buggy
En. B	1	1	0	3	9	Clean
En. C	2	3	5	8	8	Buggy
En. D	0	0	1	8	10	Clean
En. E	1	0	9	1	3	Clean
En. F	1	4	7	7	2	Clean

Selected Features

$\{X_1,\ X_2,\ X_4\}$

FVS	$\frac{3}{6}$	$\frac{3}{6}$	$\frac{4}{6}$	$\frac{3}{6}$	$\frac{4}{6}$	$\frac{4}{6}$

Clean Rate
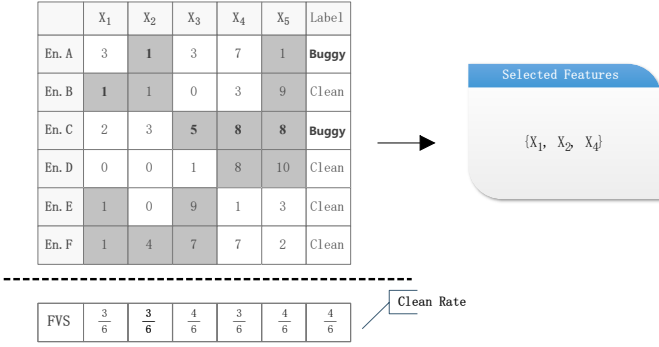
Fig. 3. An instance of computing feature violation scores (FVS), violations are shown in a dark gray shade.

IV. EXPERIMENT SETUP

A. Experiment Datasets

Table I lists the statistical characteristics of the 28 datasets from two groups, the PROMISE and the NASA, used in our experiments, which is commonly used in recent SDP researches [19] [20].

- The PROMISE dataset collected by [21], including data from different versions. As it is likely to be a significant overlap between different versions of a project, we considered a version as a separate project. Each project in PROMISE dataset contains 20 features.
- The NASA dataset is collected from the NASA Metrics Data Program. Each NASA project includes various static code metrics (CMs) of a NASA software system or subsystem, as well as the corresponding defect label data.

B. Comparative Methods

The following five unsupervised learning methods are selected to compare with ISC-FS:

- K-means: A traditional classic clustering algorithm, first used in SDP in [14].
- Partition around medoids (PAM): A method of K-medoids. The method uses medoids as a reference point, which solves the problem that K-means is extremely susceptible to extreme values.
- CLA: An unsupervised method proposed by [7] for SDP. The name is taken from the first letter of the steps are **C**lustering and **LA**beling instances.
- CLAMI: An improved version of CLA by adding **M**etric selection and **I**nstance selection.
- Spectral Clustering (SC): A connectivity-based unsupervised method proposed by [8].

It should be pointed out that we adopted the same heuristic rules as ISC-FS are adopted when labeling the cluster in the first two baselines, while the last three algorithms have their own clear rules in the corresponding researches [7] [8].

C. Experiment Datasets

The classifier has many widely used measures for classifiers, such as accuracy, which is the most traditional measure in classification tasks, and it represents the ratio of correct prediction.

However, accuracy usually does not deal well with the imbalanced datasets. Furthermore, a critical value for the probability of defect-proneness is required when computing accuracy and many other measures (e.g. F-score). The critical value can affect the performance and the default value (i.e. 0.5) may not be the best critical value [22]. Thus, we adopted the Area Under Curve (AUC), which is independent of critical value and has good tolerance for imbalanced datasets, to evaluate the effectiveness of the approaches.

AUC is defined as the area under the ROC (receiver operating characteristic curve). ROC refers to a curve of the false positive rate (FPR) against the true positive rate (TPR). The FPR and TPR can be expressed in Equation 7 and 8. From the above definition, it can be seen the AUC value ranges in [0, 1], and an AUC value of 0.5 indicates that the effect of the classifier is almost the same as the random guessing. The higher AUC is, a better result implies.

$$TPR = \frac{TP}{TP + FN} \tag{7}$$

$$FPR = \frac{FP}{FP + TN} \tag{8}$$

where TP, FN, FP and TN can be calculated from Table II.

V. RESULT

We evaluated our proposed method by the Scott-Knott test [23], using hierarchical clustering to divide different methods into groups with significant statistical differences. In this study, we adopted the normality and effect size aware variant of the standard Scott-Knott test, Scott-Knott test effect size difference (ESD) [24]. Based on the traditional Scott-Knott test, the Scott-Knott ESD test has the following improvements: (1) correct non-normal distribution inputs, which are considered to be normally distributed in traditional Scott-Knott testes; and (2) merge any two statistically different groups with a negligible effect size into one group. We used the function *sk_esd* in the *ScottKnottESD* R package to make the implementation.

The result is shown in Figure 5. After comprehensive observation, ISC-FS outperformed 5 reference methods in our experiments. The following points can be drawn from a detailed observation:

- In the PROMISE dataset, the average AUC of ISC-FS was 0.694, which outperformed K-means, PAM, CLA, CLAMI, SC by 44.58%, 43.68%, 5.15%, 14.33%, 3.74% respectively.
- In the NASA dataset, the average AUC of ISC-FS was 0.685, which outperformed K-means, PAM, CLA, CLAMI, SC by 42.12%, 35.91%, 2.24%, 2.39%, 1.48% respectively.
- The results are divided into groups with different statistical differences in Figure 5. Our results are statistically different from the other comparison methods in both of the two datasets.

Group	Granularity	Project	Version	Avg. instance	Avg. Buggy Rate(%)	# of metrics
PROMISE	class	ant	1.3 1.4 1.5 1.6 1.7	338	19.58	20
		ivy	1.1 2.0	232	34.06	
		jedit	4.0 4.1 4.2 4.3	369	16.29	
		log4j	1.0	135	25.19	
		lucence	2.2 2.4	294	59.00	
		poi	3.0	442	63.57	
		synapse	1.0 1.2	257	21.70	
		velocity	1.5 1.6.1	222	50.21	
		xalan	2.4 2.5 2.7	812	52.23	
		camel	1.2	608	35.53	
NASA	function	CM1	–	505	9.50	37
		KC1		2107	15.42	21
		KC3		458	9.39	39
		KC4		125	50.40	13
		MC2		161	32.30	39

	Predicted defective	Predicted non-defective
Actual defective	True Positive (TP)	False Negative (FN)
Actual non-defective	False Positive (FP)	True Negative (TN)

- K-means and PAM showed the lowest performance because they adopt a different mechanism with spectral clustering, which only clustering based on Euclidean distance between entities.
- We noticed that the CLAMI performed worse than CLA. However, in [7], CLAMI is proposed as an enhanced version of CLA by adding metric selection and instance selection to improve the ability of prediction models. We studied on it and observed that CLAMI only selected features with the minimum metric violation score (MVS), however, in a real dataset with a large number of instances, the MVS of different features often varies. In this case, dozens of features will be selected as few features in some projects, which will weaken the performance.

VI. THREATS TO VALIDITY

We discuss a few threats to the validity in this section.

A. Implementation of CLA and CLAMI

In this study, we compared our approaches with 5 referential methods. Due to the unpublished implementations of CLA and CLAMI, we have reimplemented our version according to the corresponding paper. Although we strictly followed the procedures reported, our implementation may not reflect all details in the comparative method.

B. AUC might not be the only suitable measures

We used AUC as measures to evaluate the performance of SDP models. There are many other measures (e.g., G-measure, MCC) can be used for performance evaluation. In fact, AUC is a widely used evaluation measure in SDP tasks [4] [7] [8].

C. Experimental results might not be generalizable:

In our experiment, we selected 28 projects that have been widely used in SDP. However, diverse software projects have different characteristic, so there is no guarantee that our findings will be applicable to other projects. More validation should be conducted in the future.

VII. CONCLUSION AND FUTURE WORKS

In this paper, we proposed an unsupervised learning method called ISC-FS to solve the problem of lacking historical data in SDP. The main advantage of ISC-FS is that it introduces feature selection considering data distribution and improves spectral clustering. A large number of experiments on 28 projects from 2 groups have been conducted to assess that the proposed method can perform better in terms of AUC than the referential approaches.

In the future, we plan to evaluate our approach with more datasets from different sources. Furthermore, we will try to extend our approach to the semi-supervised version by introducing a few labeled data.

(a) Promise

(b) NASA
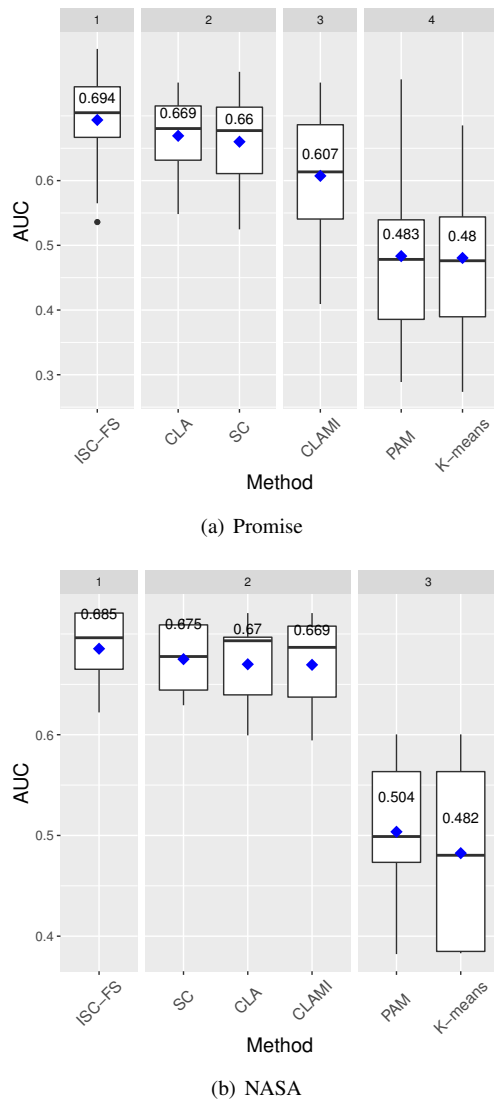
Fig. 4. The Scott-Knott ESD ranking of 6 methods

REFERENCES

[1] R. Rana, M. Staron, C. Berger, J. Hansson, M. Nilsson, and W. Meding, "The adoption of machine learning techniques for software defect prediction: An initial industrial validation," in *Joint Conference on Knowledge-Based Software Engineering*. Springer, 2014, pp. 270–285.

[2] N. Li, M. Shepperd, and Y. Guo, "A systematic review of unsupervised learning techniques for software defect prediction," *arXiv preprint arXiv:1907.12027*, 2019.

[3] S. Hosseini, B. Turhan, and D. Gunarathna, "A systematic literature review and meta-analysis on cross project defect prediction," *IEEE Transactions on Software Engineering*, vol. 45, no. 2, pp. 111–147, 2017.

[4] J. Nam, W. Fu, S. Kim, T. Menzies, and L. Tan, "Heterogeneous defect prediction," *IEEE Transactions on Software Engineering*, vol. 44, no. 9, pp. 874–896, 2017.

[5] S. Qiu, H. Xu, J. Deng, S. Jiang, and L. Lu, "Transfer convolutional neural network for cross-project defect prediction," *Applied Sciences*, vol. 9, no. 13, p. 2660, 2019.

[6] T. Zimmermann, N. Nagappan, H. Gall, E. Giger, and B. Murphy, "Cross-project defect prediction: a large scale experiment on data vs. domain vs. process," in *Proceedings of the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*, 2009, pp. 91–100.

[7] J. Nam and S. Kim, "Clami: Defect prediction on unlabeled datasets (t)," in *2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2015, pp. 452–463.

[8] F. Zhang, Q. Zheng, Y. Zou, and A. E. Hassan, "Cross-project defect prediction using a connectivity-based unsupervised classifier," in *Proceedings of the 38th International Conference on Software Engineering*. ACM, 2016, pp. 309–320.

[9] T. M. Khoshgoftaar and N. Seliya, "The necessity of assuring quality in software measurement data," in *10th International Symposium on Software Metrics, 2004. Proceedings.* IEEE, 2004, pp. 119–130.

[10] J. Shi and J. Malik, "Normalized cuts and image segmentation," *Departmental Papers (CIS)*, p. 107, 2000.

[11] D. Di Nucci, F. Palomba, G. De Rosa, G. Bavota, R. Oliveto, and A. De Lucia, "A developer centered bug prediction model," *IEEE Transactions on Software Engineering*, vol. 44, no. 1, pp. 5–24, 2017.

[12] X. Zhang, K. Ben, and J. Zeng, "Cross-entropy: A new metric for software defect prediction," in *2018 IEEE International Conference on Software Quality, Reliability and Security (QRS)*. IEEE, 2018, pp. 111–122.

[13] M. Caron, P. Bojanowski, A. Joulin, and M. Douze, "Deep clustering for unsupervised learning of visual features," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 132–149.

[14] S. Zhong, T. M. Khoshgoftaar, and N. Seliya, "Unsupervised learning for expert-based software quality estimation." in *HASE*. Citeseer, 2004, pp. 149–155.

[15] C. Catal, U. Sevim, and B. Diri, "Clustering and metrics thresholds based software fault prediction of unlabeled program modules," in *2009 Sixth International Conference on Information Technology: New Generations*. IEEE, 2009, pp. 199–204.

[16] J. Yang and H. Qian, "Defect prediction on unlabeled datasets by using unsupervised clustering," in *2016 IEEE 18th International Conference on High Performance Computing and Communications; IEEE 14th International Conference on Smart City; IEEE 2nd International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*. IEEE, 2016, pp. 465–472.

[17] A. Agrawal and T. Menzies, "Is better data better than better data miners?: on the benefits of tuning smote for defect prediction," in *Proceedings of the 40th International Conference on Software engineering*. ACM, 2018, pp. 1050–1061.

[18] W. Liu, S. Liu, Q. Gu, X. Chen, and D. Chen, "Fecs: A cluster based feature selection method for software fault prediction with noises," in *2015 IEEE 39th Annual Computer Software and Applications Conference*, vol. 2. IEEE, 2015, pp. 276–281.

[19] S. Wang, T. Liu, J. Nam, and L. Tan, "Deep semantic feature learning for software defect prediction," *IEEE Transactions on Software Engineering*, 2018.

[20] X.-Y. Jing, F. Wu, X. Dong, and B. Xu, "An improved sda based defect prediction framework for both within-project and cross-project class-imbalance problems," *IEEE Transactions on Software Engineering*, vol. 43, no. 4, pp. 321–339, 2016.

[21] M. Jureczko and L. Madeyski, "Towards identifying software project clusters with regard to defect prediction," in *Proceedings of the 6th International Conference on Predictive Models in Software Engineering*. ACM, 2010, p. 9.

[22] F. Zhang, A. Mockus, I. Keivanloo, and Y. Zou, "Towards building a universal defect prediction model with rank transformed predictors," *Empirical Software Engineering*, vol. 21, no. 5, pp. 2107–2145, 2016.

[23] E. Jelihovschi, J. C. Faria, and I. B. Allaman, "The scottknott clustering algorithm," *Universidade Estadual de Santa Cruz-UESC, Ilheus, Bahia, Brasil*, 2014.

[24] C. Tantithamthavorn, S. McIntosh, A. E. Hassan, and K. Matsumoto, "An empirical comparison of model validation techniques for defect prediction models," *IEEE Transactions on Software Engineering*, vol. 43, no. 1, pp. 1–18, 2016.

A Semantic Convolutional Auto-Encoder Model for Software Defect Prediction

Zhihan Wang[1] and Lu Lu[1,2*]

[1]School of Computer Science and Engineering South China University of Technology Guangzhou, China
[2]Modern Industrial Technology Research Institute, South China University of Technology, Meizhou, China
[*]Corresponding author email: lul@scut.edu.cn

Abstract—In traditional software defect prediction, previous researches mainly focused on manually designing complex features and building classifiers based on features. However, such traditional features often fail in capturing rich syntactic and semantic information in programs. Thus, an effecient prediction model is unable to be constructed in some cases. In this study, a framework called semantic convolutional auto-encoder (SCAE) is proposed to effectively extract semantic features from source code. Token vectors are extracted from Abstract Syntax Trees (ASTs) of programs and then encoded as numerical vectors. Convolutional auto-encoder (CAE) can learn semantic features from the numerical vectors by decreasing the reconstruction error between input and output. After that, the CAE-based features are utilized to train a classifier. To enhance the transferability of CAE-based features for different projects, we perform domain adaptation by matching kernel embedding of layer representations across domains in reproducing kernel Hilbert spaces. Extensive experimental results verify that the SCAE yields referential methods on ten open-source projects.

Keywords—Software defect prediction, Transfer learning, Semantic feature learning, Convolutional auto-encoder

I. Introduction

As one of the research hotspots in software engineering, software defect prediction (SDP) technology can detect potential bugs in an application, then help developers assign limited testing resources and effectively enhance software reliability. An ideal defect prediction model can accurately determine whether there are defects in program, and thus plays an important role in improving software quality, shortening the development cycle, and decreasing maintaining costs.

Traditional software defect prediction (SDP) technologies utilize features representing software complexity and software scale [1] to find the potential defects in program such as Halstead metric [2], which was based on the number of operators and operands in program, and McCabe metric [3], which was calculated by loop complexity. Besides, CK metric [4] and other object-oriented metrics [5] were also adopted for defect prediction. Previous

researchers mainly focused on elaborately designing and choosing handcrafted features that reflect the software characteristic for better performance of defect prediction. Asad et al. [6] evaluated the quality of software design and presented several coupling metrics for defect prediction. Conducting an empirical study for 32 feature selection methods, Xu et al. [7] showed that the selection of features has great effects on classification performance.

A key issue in SDP is the extraction of the syntax and semantic information from program. Deep learning technique is also used in the latest studies. Wang et al. [8] and Li et al. [9] leverage deep belief network (DBN) and convolutional neural network (CNN) for semantic feature generation respectively. However, compression of higher features is often accompanied by the information loss, which may cause a bad effect on making classifications. As an unsupervised method in deep learning, auto-encoder (AE) can automatically learn features from a large amount of unlabeled data and is widely used in various tasks [10], [11]. The convolutional auto-encoder (CAE) is an effective variant of the basic auto-encoder, and can reserve as much program semantics as possible by reconstructing input. The convolution and pooling operations in CAE can capture local patterns more effectively during the period of feature generation. When transfering knowledge from a labeled source domain to an unlabeled target domain, different domains may exhibit distributional discrepancy in transfer learning, which caused cross-domain knowledge adaptation problems [12]. In cross project defect prediction, distribution discrepancy between different projects also hinders the transerability of semantic feature, and degrades the performance of classifier.

In this work, a framework named semantic convolutional auto-encoder (SCAE) is developed, which uses the convolutional auto-encoder to effectively capture semantic information from ASTs of program. Specifically, we first parse source code into ASTs, then extract AST node to form token vectors and map token into number according to the mapping table. The embedded numerical vectors are fed into convolutional auto-encoder. By decreasing the reconstruction error between input and output, the CAE model can automatically learn high-level representation of input. To bridge the substantial distributional discrepancy

between different projects, a domain confusion loss based maximum mean discreapancy (MMD) is introduced in feature extraction. Enhancing the semantic feature transferibility generalizes the model of SCAE to the domain adaptation scenario. Evaluating the proposed approach on ten open-source java projects, experimental results indicate that the SCAE can improve the performance on both within project defect prediction (WPDP) and cross project defect prediction (CPDP).

This work makes the following contributions:

- To capture semantic information hidden in ASTs, convolutional auto-encoder is utilized for a better feature generation in defect prediction by reconstructing the input and output.
- Considering the domain discrepancy, we optimized the model of CAE by minimizing the distributional discrepancy between source and target projects to obtain transferable semantic features.
- For WDDP and CPDP, extensive experiments are conducted to verify the effectiveness of our method, and the results shows that our approach can enhance the classification performance of SDP.

The rest of this paper is organized as follows. The related work of SDP is described in Section II. Section III illustrates the proposed approach and the experiment setup is given in Section IV. The proposed approach is evaluated and the experimental results are analyzed in Section V. Section VI presents the threats to validity. In Section VII, we present the conclusion of this work and possible directions in the future.

II. Related work

Over the past few decades, software defect prediction becomes a hotspot research area in software engineering and there are many researches in the literature [13], [14]. Traditionally, most researchers mainly leverage traditional features for defect prediction, such as Halstead features [2], McCabe features [3], CK features [4], etc. The selection of features plays a vital role in classification performance [7] and Jacob et al. [15] also proposed a method to identify and remove redundant features for SDP. Ni et al. [16] investigated multi-objective features selection for SDP and proposed a method taking feature selection and construction of prediction models into account. Moreover, different algorithms of classification in machine learning are also utilized to build classifiers for defect prediction. Support Vector Machine [17], Logistic Regression [9], Naive Bayes [18] were leveraged to build classification models in SDP respectively. Researchers also build prediction models on CPDP, in which the training set and the test set are from different projects. Turhan et al. [19] took the distance of different data into account, and proposed nearest neighbor filter to remove irrelevant instances in source project. Considering the difference of data distribution, Nam et al. [20] presented an approach, i.e. TCA+, using Transfer Component Analysis (TCA) to make the source and target project distributions similar in feature space. To improve CPDP, Qiu et al. [21] constructed an ensemble classifier for the target project, which is trained on multiple components of source project data.

Recently, several deep learning techniques are also applied to build models for defect prediction. Wang et al. [8] utilized a deep learning technique, i.e. Deep Belief Network, to learn the semantic representation in source code of program. Convolutional Neural Network were leveraged [9] for defect prediction, and achieved significant results on seven open-source projects in terms of F1-score. Liang et al. [22] also employed Long Short Term Memory (LSTM) network in defect prediction. Although previous studies [8], [22] utilized neural network to learn nonlinear high-dimensional features for CPDP, but they overlooked the impact of domain discrepancy existing in different projects. Besides, convolution and pooling operations were also taken advantage of [9], but the proposed method in this work mainly has the following two differences. Firstly, the CAE does not require the information of label in the period of semantic feature generation. Secondly, we perform extensive experiments for CPDP in this work.

III. The proposed approach

As shown in Figure 1, the proposed approach consists of the following major three steps: 1) Parsing source code into ASTs; 2) Mapping tokens and embedding vectors; 3) Using convolutional auto-encoder to generate high-level semantic features and making classifications.

A. Parsing source code

The proposed approach takes source code files as input, and we need to transform the input to learn semantic information. It is proved that ASTs can be successfully applied to various tasks. In this paper, we firstly transform source code into ASTs so as to perform the following steps. Source files are parsed into ASTs and appropriate token nodes are selected from ASTs to generate token sequences. Following this work [9], four types of AST nodes are mainly chosen for both WPDP and CPDP.

B. Encoding Tokens and word embedding

In part III-A, each source file is parsed into a token sequence. Since the proposed model requires numerical vectors as input, each token sequence needs to encoded as numerical vectors based on the mapping table. Specifically, a mapping relation between tokens and integers is constructed, which means that for every token in token sequences, it has a unique integer identifier. According to the mapping table, the token sequence can be converted into an integer vector. In this way, different integer vectors may differ from each other in length, so zero is appended at the end of integer vectors to keep the same length with the longest vector. Following the work [8], infrequent tokens that occurs less than three times are also filtered out during this process. Word embedding is also performed

1. Parsing source code to AST

2. Mapping tokens and embedding vectors

3. Feature extraction and making classification
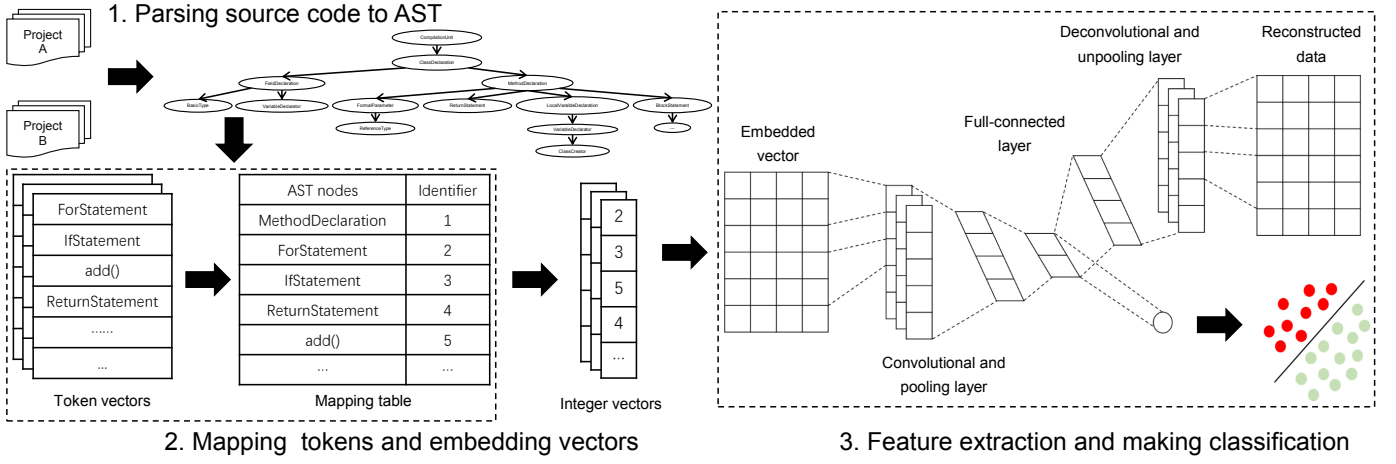
Fig. 1. The framework of SCAE

before training the model. It is difficult to draw the distance of different AST nodes only using an integer index. Word embedding technique maps a token to a fixed-length vector, and tokens in similar context tend to have similar representations. Here, the embeded size is chosen as 30, and each token is mapped to a vector whose length is 30 on the real-number field.

Similar to other classification tasks in machine learning, there also exists the issue of class imbalance in SDP. Specifically, the number of defective instances is less than the non-defective instances in a project, which makes the classifier tend to predict non-defective. Data sampling technique is also used in this work by randomly duplicating the minority class instances, i.e. Random Over-Sampling, to obtain more balanced training samples.
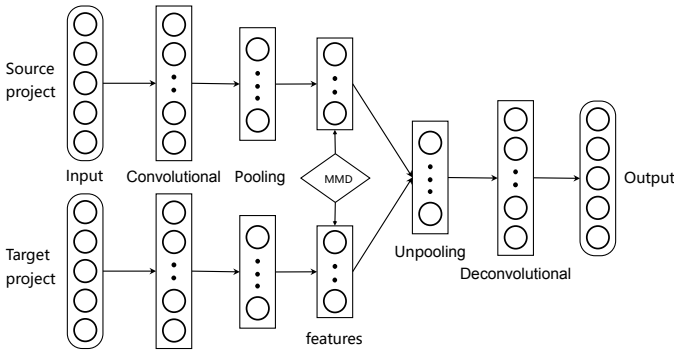
Fig. 2. Overview of CAE-based feature generation for CPDP

C. Building convolutional auto-encoder and making prediction

1) Training CAE for WPDP: Comparing with basic auto-encoder, CAE integrates the convolution and pooling operations for the advantage of CNN in feature extraction to better capture the syntactic and semantic information in program. In particular, CAE includes an encoder and a decoder. The encoder consists of a convolutional layer,

a pooling layer, a fully-connected layer, and the decoder also includes an unpooling layer, a deconvolutional layer.

The embedded vector $x \in \mathbb{R}^{l \times e}$, where l and e are the length of input and the embedding size respectively, passes through the convolutional layer and the pooling layer and is encoded as the feature $y = f_e(x) \in \mathbb{R}^h$, where h denotes the feature size. During the step of decoding, the hidden representation y also moves through unpooling and deconvolutional layer, and is decoded as $\tilde{x} = f_d(f_e(x))$. The reconstruction error, $J_{ae} = \sum_{i=1}^{n} \|\boldsymbol{x} - \tilde{\boldsymbol{x}}\|_2^2$, can be calculated by the input x and output $\tilde{x}$. In addition, a regularization item J_{wd} is added to avoid overfitting during model training. The objective function to be minimized is shown in Equation 1, where W, b denote the parameters in CAE and factors λ, γ are the tradeoff parameters for the J_{wd} penalty and J_{mmd} penalty respectively. In WPDP, we do not perform the domain adaptation, which means that the regularization hyperparameter γ is set as 0.

$$\min_{W,b} \mathcal{J}(W, b) = J_{ae} + \lambda J_{wd} + \gamma J_{mmd}$$

$$= \frac{1}{2n_s} \sum_{i=1}^{n_s} \|x - \tilde{x}\|_2^2 + \lambda \sum \|\omega\|_2^2 + \gamma MMD(X_s, X_t) \tag{1}$$

CAE is an unsupervised learning method, and cannot directly be utilized for classification tasks. Therefore, an additional classifier needs to be constructed. After feature extraction, we use the CAE-based features of training set with labels to learn a classifier, and test its performance on the test set. In this work, the algorithm of Logistic Regression is utilized to established the classifier for both WPDP and CPDP.

2) Training CAE for CPDP: To promote transferability of learned features, we need to reduce the distribution difference between source and target projects. By mapping distribution X_s and X_t to a reproducing kernel Hibert space $\mathcal{H}_k$, maximum mean discreapancy (MMD) measures the mean value of these two distributions in the $\mathcal{H}_k$. Figure 2 shows the CAE-based feature generation for CPDP, and

the MMD loss, i.e. J_{mmd}, is to be minimized during model training. The MMD distance can be resolved as follows,

$$MMD(X_s, X_t) = \left\| \frac{1}{n_s} \sum_{i=1}^{n_s} \phi(\boldsymbol{x_i}) - \frac{1}{n_t} \sum_{i=1}^{n_t} \phi(\boldsymbol{x_i}) \right\|_{\mathcal{H}_k}^2$$
$$= \frac{1}{n_s^2} \sum_{i=1}^{n_s} \sum_{j=1}^{n_s} k(\boldsymbol{x_i^s}, \boldsymbol{x_j^s}) + \frac{1}{n_t^2} \sum_{i=1}^{n_t} \sum_{j=1}^{n_t} k(\boldsymbol{x_i^t}, \boldsymbol{x_j^t}) \quad (2)$$
$$- \frac{2}{n_s n_t} \sum_{i=1}^{n_s} \sum_{j=1}^{n_t} k(\boldsymbol{x_i^s}, \boldsymbol{x_j^t})$$

where $\phi(\cdot)$ is a nonlinear feature mapping function on the reproducing kernel hilbert spaces $\mathcal{H}_k$. The most important property is that $MMD(X_s, X_t) = 0$ when $X_s = X_t$. The characteristic kernel function $k(\boldsymbol{x_i}, \boldsymbol{x_j}) = \langle \phi(\boldsymbol{x_i}), \phi(\boldsymbol{x_j}) \rangle$ calculates the dot product of $\boldsymbol{x_i}$ and $\boldsymbol{x_j}$ in $\mathcal{H}_k$. In this study, we adopted Gaussian kernel: $k(\boldsymbol{x_i}, \boldsymbol{x_j}) = exp(-\|\boldsymbol{x_i} - \boldsymbol{x_j}\|^2 / 2\sigma)$, and σ is the kernel width.

IV. EXPERIMENTS

This section describes the experiment setup in detail. Extensive experiments are carried out on ten open source project (listed in Table I) to evaluate effectiveness of the proposed approach. Here are two research questions:

- *RQ1: Does the semantic features extracted by the proposed approach outperforms traditional features in WPDP?*
- *RQ2: Does the distribution adaptation method can improve the performance of CPDP?*

TABLE I
THE DESCRIPTION OF PROJECTS FROM PROMISE REPOSITORY

Project Name	Versions	Avg. files	Avg. Bug Rate(%)
ant	1.6, 1.7	521	21.4
camel	1.4, 1.6	888	18.5
ivy	1.4, 2.0	284	7.2
jedit	4.0, 4.1	269	21.2
lucene	2.2, 2.4	272	60.7
poi	2.5, 3.0	391	63.9
synapse	1.0, 1.1	190	20.0
velocity	1.5, 1.6	218	49.1
xalan	2.4, 2.7	759	61.5
xerces	1.3, 1.4	370	35.0

A. Dataset

To evaluate the performance of the approach, experiments are performed on ten different open-source projects from the PROMISE repository which has been used in prior studies [8], [22]. Table I shows the necessary information about projects in experiments, including project name, project versions (including the training set and the testing set), the number of files and the bug rate. Traditional features for these projects are also widely used in conventional defect prediction [13], [20], [21]. Besides, the projects in the dataset have different data sizes, i.e.

different numbers of files and defect rates, and we assure that it can verify the generalization of the model.

B. Evaluation metric

In this study, F1-score (also F-measure) is adopted to assess the performance of prediction. As a composite measure, F1-score is the harmonic average of precision and recall, which has been widely used in binary classification.

C. Baseline of methods

The proposed approach is compared with the following methods, LR based on 20 traditional features, and other two deep learning methods, including DBN and CNN. Besides, NNFilter, TCA and TCA+ are also performed for CPDP to verify the validity of domain adaptation method.

- **LR:** A logistic regression classifier is built based on 20 traditional features.
- **DBN [8]:** This method utilizes Deep Belief Network to capture semantic information in source code.
- **CNN [9]:** Supervised Convolutional Neural Network is utilized to extract semantic features from ASTs of programs.
- **NNFilter [19]:** This method uses KNN algorithm to select instance from multi source projects .
- **TCA [23]:** Transfer Component analysis, the state-of-the-art method in transfer learning.
- **TCA+ [20]:** This method optimizes the normalization process of TCA for enhancing CPDP.

When implementing these two deep learning algorithms, their optimal parameters are selected according to their papers and we ensure that the experiments are conducted with the same data processing for a fair comparison, including parsing source code into ASTs, mapping tokens, as well as data imbalance preprocessing. The proposed model is built from a training set using Adam as the optimizer. Same network architectures and parameters are adopted when implementing CAE. We set the epoch size as 30, the batch size as 16, the filter size as 3, the number of filter as 10, the number of nodes as 20.

V. RESULTS AND ANALYSIS

This section presents the results of experiments conducted on PROMISE dataset. According to analysis of the results, we answer the two questions raised in Section III.

RQ1: Does the semantic features extracted by the proposed approach outperforms traditional features in WPDP?

With 20 traditional features, a classifier is constructed using Logistic Regression algorithm for WPDP. We train the model with the instances from an old version of project, and make classifications on a new version of the same project. Additional two deep learning methods, i.e. DBN and CNN are also performed to extract semantic features. Conducting experiments on ten projects as listed
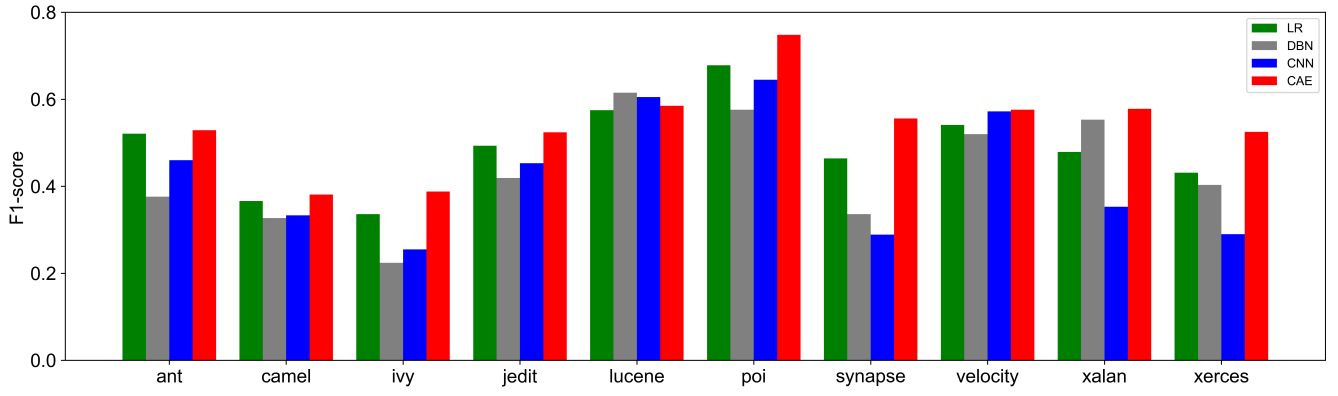
Fig. 3. F1-scores of traditional method using LR and deep learning methods using DBN, CNN, CAE respectively.

in Table I, these three methods are compared with our method, built on CAE-based features using LR algorithm.

As we can see from Figure 3, our approach can obtain higher F1-scores in most cases. Taking project synapse as an example, we use 1.0 version for model training, 1.1 version for testing. The F1-score are respectively 0.464, 0.336, 0.289, 0.556 for LR, DBN, CNN and CAE, and the CAE can obtain higher F1-scores than other methods on project synapse. The average of F1-scores over ten projects are 0.488, 0.435, 0.426 and 0.539 for these four methods, and CAE outperforms 10.36%, 23.94%, 26.67% than LR, DBN and CNN on F1-score respectively.

RQ2: Does the proposed distribution adaptation approach for CAE can improve the performance of CPDP?

In CPDP, the domain adaptation is performed on CAE, and we call it CAE+. CAE and CAE+ are compared with other three cross-project defect prediction techniques, including NNFilter, TCA and TCA+. Different from WPDP in RQ1, we only use the new version of projects for CPDP. For ten projects from PROMISE dataset, each CPDP task takes a project from these projects as a target project, and another project as source project. Therefore, 90 sets of CPDA task are constructed for each method.

Giving a target project, 9 CPDP tasks can be established respectively using the remaining 9 projects, and we calculate the average F1-score for each target project. Table II illustrates the average results of our methods and other compared approaches on PROMISE dataset. Similarly, each row in the table represents the average of F1-scores of a project, and the best result are also maked in bold. The next-to-last row reports the w/t/l, which means that our CAE+ wins w projects, ties t projects, loses l datasets, versus other methods at coressponding column. Also taking a project, e.g. xerces as an example, the highest F1-score is 0.608 achieved by CAE+. Compared with NNFilter method, CAE+ wins 9 projects, ties 0 projects, loses 1 projects, and compared with TCA method, CAE+ wins 8 projects, ties 0 projects, loses 1 projects respectively.

As Table II reports, the F1-score of CAE and CAE+ is 0.503 and 0.521, which indicates that our proposed models obtains better performances for CPDP. The CAE+ outperforms LR, NNFilter, TCA, TCA+, DBN, CNN, and CAE by 8.8%, 11.3%, 4.4%, 13.8%, 17.9%, 26.2%, and 3.6% respectively. The results also proves that, in general, domain adaptation method can enhance the performance of CPDP.

VI. Threats to validity

A. Implementation of compared methods

The proposed model is compared with other deep learning methods, DBN [8] and CNN [9]. These two methods are reproduced according to their papers. However, we can not guarantee that all the implementation details have been taken into account. Considering the randomness involved in batch shuffle, we repeated the experiment ten times, recording the average of F1-score.

B. Pamameter selection

During the training of the proposed model, we adjust the hyperparameters of CAE to get promising performance. Considering the large space of parameters, experiments cannot be done on all combinations of parameters, which may make a difference in experimental results.

VII. Conclusion

In this work, a framework of SCAE is proposed to extract semantic features from source code for defect prediction. In particular, token nodes are deliberately selected from ASTs of program. Each token in token sequence is mapped into an integer, and then word embedding is performed on numerical vector. The data then are fed into the convolutional auto-encoder to capture the intermediate representation of syntactic and semantic information of source code. Considering the distributional discrepancy of semantic representations between source and target project, an additional domain loss item is introduced during feature generation in CPDP. Conducting experiments on ten open-source PROMISE projects, the results prove

TABLE II
Average cross-project defect prediction results on PROMISE dataset

Target	LR	NNFilter	TCA	TCA+	DBN	CNN	CAE	CAE+
ant	0.49	0.487	0.464	0.387	0.336	0.38	0.483	**0.495**
camel	0.348	0.348	0.343	0.341	0.312	0.278	0.32	**0.349**
ivy	0.27	0.275	0.265	0.21	0.176	0.227	**0.291**	0.263
jedit	0.452	0.445	0.448	0.211	0.337	0.378	0.445	**0.471**
lucene	0.603	0.592	**0.664**	0.646	0.582	0.528	0.644	0.646
poi	0.611	0.605	0.627	0.602	0.625	0.573	**0.699**	0.675
synapse	0.49	0.478	0.456	0.468	0.396	0.384	**0.501**	0.494
velocity	0.53	0.493	0.56	0.514	0.435	0.429	0.542	**0.564**
xalan	0.49	0.49	0.64	0.629	**0.663**	0.545	0.643	0.647
xerces	0.509	0.47	0.523	0.576	0.562	0.405	0.466	**0.608**
CPCAE:(w/t/l)	9/0/1	9/0/1	8/0/2	9/1/0	9/0/1	10/0/0	7/0/3	
Average	0.479	0.468	0.499	0.458	0.442	0.413	0.503	**0.521**

that the proposed SCAE can improve performance on both WPDP and CPDP in terms of the evaluation metric, i.e. F1-score. In future work, we plan to explore other domain adaptation methods for CPDP, and our future investigation involves applying the proposed approach to more projects.

VIII. Acknowledgements

References

[1] T. Menzies, Z. Milton, B. Turhan, B. Cukic, Y. Jiang, and A. Bener, "Defect prediction from static code features: current results, limitations, new approaches," *Automated Software Engineering*, vol. 17, no. 4, pp. 375–407, 2010.

[2] M. Halsted, "Elements of software science (operating and programming systems series)," 1977.

[3] T. J. McCabe, "A complexity measure," *IEEE Transactions on Software Engineering*, no. 4, pp. 308–320, 1976.

[4] R. Subramanyam and M. S. Krishnan, "Empirical analysis of ck metrics for object-oriented design complexity: implications for software defects," *IEEE Transactions on Software Engineering*, vol. 29, no. 4, pp. 297–310, 2003.

[5] S. R. Chidamber and C. F. Kemerer, "A metrics suite for object oriented design," *IEEE Transactions on Software Engineering*, vol. 20, no. 6, pp. 476–493, 1994.

[6] A. A. Asad and I. Alsmadi, "Evaluating the impact of software metrics on defects prediction. part 2." *Computer Science Journal of Moldova*, vol. 22, no. 1, 2014.

[7] Z. Xu, J. Liu, Z. Yang, G. An, and X. Jia, "The impact of feature selection on defect prediction performance: An empirical comparison," in *2016 IEEE 27th International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 2016, pp. 309–320.

[8] S. Wang, T. Liu, and L. Tan, "Automatically learning semantic features for defect prediction," in *Proceedings of the 38th International Conference on Software Engineering, ICSE 2016, Austin, TX, USA, May 14-22, 2016*, 2016, pp. 297–308.

[9] J. Li, P. He, J. Zhu, and M. R. Lyu, "Software defect prediction via convolutional neural network," in *2017 IEEE International Conference on Software Quality, Reliability and Security, QRS 2017, Prague, Czech Republic, July 25-29, 2017*, 2017, pp. 318–328.

[10] T. Bao, C. Ding, S. Karmoshi, and M. Zhu, "Video anomaly detection based on adaptive multiple auto-encoders," in *Advances in Visual Computing - 12th International Symposium, ISVC 2016, Las Vegas, NV, USA, December 12-14, 2016, Proceedings, Part II*, 2016, pp. 83–91.

[11] W. Xu, H. Sun, C. Deng, and Y. Tan, "Variational autoencoder for semi-supervised text classification," in *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

[12] M. Long, Y. Cao, Z. Cao, J. Wang, and M. I. Jordan, "Transferable representation learning with deep adaptation networks," *IEEE transactions on pattern analysis and machine intelligence*, vol. 41, no. 12, pp. 3071–3085, 2018.

[13] X.-Y. Jing, S. Ying, Z.-W. Zhang, S.-S. Wu, and J. Liu, "Dictionary learning based software defect prediction," in *Proceedings of the 36th International Conference on Software Engineering*. ACM, 2014, pp. 414–423.

[14] Z. Li, X. Y. Jing, and X. Zhu, "Progress on approaches to software defect prediction," *IET Software*, vol. 12, no. 3, pp. 161–175, 2018.

[15] S. Jacob and G. Raju, "Software defect prediction in large space systems through hybrid feature selection and classification," *Int. Arab J. Inf. Technol.*, vol. 14, pp. 208–214, 2017.

[16] C. Ni, X. Chen, F. Wu, Y. Shen, and Q. Gu, "An empirical study on pareto based multi-objective feature selection for software defect prediction," *Journal of Systems and Software*, vol. 152, pp. 215–238, 2019. [Online]. Available: https://doi.org/10.1016/j.jss.2019.03.012

[17] D. Gray, D. Bowes, N. Davey, Y. Sun, and B. Christianson, "Using the support vector machine as a classification method for software defect prediction with static code metrics," in *Engineering Applications of Neural Networks*, D. Palmer-Brown, C. Draganova, E. Pimenidis, and H. Mouratidis, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 223–234.

[18] H. JI, S. HUANG, X. LV, Y. WU, and Y. FENG, "Empirical studies of a kernel density estimation based naive bayes method for software defect prediction," *IEICE Transactions on Information and Systems*, vol. E102.D, pp. 75–84, 01 2019.

[19] B. Turhan, T. Menzies, A. B. Bener, and J. Di Stefano, "On the relative value of cross-company and within-company data for defect prediction," *Empirical Software Engineering*, vol. 14, no. 5, pp. 540–578, Oct 2009.

[20] J. Nam, S. J. Pan, and S. Kim, "Transfer defect learning," in *2013 35th International Conference on Software Engineering (ICSE)*. IEEE, 2013, pp. 382–391.

[21] S. Qiu, L. Lu, and S. Jiang, "Multiple components weights model for cross-project defect prediction," *IET Software*, vol. 12, no. 4, pp. 345–355, 2018.

[22] H. Liang, Y. Yu, L. Jiang, and Z. Xie, "Seml: A semantic LSTM model for software defect prediction," *IEEE Access*, vol. 7, pp. 83 812–83 824, 2019.

[23] S. J. Pan, I. W. Tsang, J. T. Kwok, and Q. Yang, "Domain adaptation via transfer component analysis," *IEEE Transactions on Neural Networks*, vol. 22, no. 2, pp. 199–210, 2010.

Revisiting Dependence Cluster Metrics based Defect Prediction

Qiguo Huang*,Xiang Chen*†,Zhengliang Li*,Chao Ni*,Qing Gu*‡
*State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing 210023, China
†School of Computer Science and Technology, Nantong University, Nantong 226019, China

Abstract—A dependence cluster is a set of program elements that all depend upon each other. Prior empirical studies have found that the dependence cluster based metrics are useful in effort-aware defect prediction. However, it is still unknown whether they are useful in non-effort-aware defect prediction. In this paper, we perform empirical studies to investigate this issue. We use the product, process, and network metrics to build the "B" model (baseline model), and then use the product, process, network and dependence cluster metrics to build the "B+C" model. Our experimental results, based on five well-known open-source systems, show that the dependence clusters are useful for non-effort-aware defect prediction. These findings help us better understand how dependence clusters influence non-effort-aware defect prediction.

Index Terms—dependence cluster, metrics, defect prediction, revisiting, non-effort-aware

I. Introduction

A dependence cluster is a set of program elements that all depend upon each other [1], [2]. Prior studies showed that large dependency clusters are widely existed in all kinds of source code. The existence of the large dependent clusters will result in ripple effects — a code change in one element of the dependent cluster will produce potential impact on other elements of the cluster [1], [2]. A high-quality software system should reduce or even eliminate large dependency clusters since they not are easier to develop, maintain, and reuse. Therefore, The detection and analysis of dependency clusters is the key point. Binkley et al. [1] used the technology of program slicing to solve this problem by defining the program system dependence graph. Based on their observation, dependence clusters can influence software quality. Yang et al. [3] first applied dependence cluster metrics to software defect prediction, and their empirical studies have found that the dependence cluster based metrics are useful in effort-aware defect prediction. However, it is still unknown whether they are useful in non-effort-aware defect prediction. In this paper, our study attempts to fill this gap.

Based on this motivation, we investigate the effectiveness of dependence cluster based metrics in terms of non-effort-aware performance indicators. Our main contributions are the following:1) We investigate whether dependence clusters are useful for non-effort-aware defect prediction. Our results show that the dependence clusters are still useful for non-effort-aware defect prediction. 2) We examine whether our conclusions change if the potentially confounding effect of module size is excluded. The results show that the "B+C" model still performs better than the "B" model. 3) We examine whether our conclusions change if the class imbalanced method is used. The results show that the "B+C" model still performs better than the "B" model.

The rest of this paper is organized as follows. In Section II, we summarize related work. In Section III, we present the research questions and research method. We describe experiment setup, including studied projects, data collection, and performance indicators in Section IV. In Section V, we report out experimental results. Section VI discusses our findings. Finally, we conclude the paper and direct future work.

II. Related Work

In this section, we summarize related work on non-effort-aware defect prediction and dependence clusters.

A. Non-effort-aware Defect Prediction

The non-effort-aware defect prediction is also called traditional defect prediction [4]. It includes within-project defect prediction and cross-project defect prediction. For within-project defect prediction, Hall et al. [5] investigated the effect of model independent variables and model techniques on the performance of defect prediction model. Their results showed that simple modeling techniques, such as Logistic Regression, tended to perform well. For cross-project defect prediction, To the best of our knowledge, the earliest study on CPDP(Cross-Project Defect Prediction) was performed by Briand et al. [6], they used logistic regression to build defect prediction models based

on the Xpose project. The results showed that the CPDP model is better than the random model, but lower than the within-project defect prediction performance. Ulike the above stduies, we investigate whether dependence clusters have practical value in non-effort-ware defect prediction.

B. Dependence Clusters

The concept of dependency clusters based on program slicing at the statement level was first proposed by Binkley et al. [1]. The dependence cluster is a set of program elements that all depend upon each other. Later, Harman et al. [2] extended Binkley's study to modules with coarser granularity. Their results showed that the accuracy of the "same slice size" method proposed by Binkley is very high. In addition, they found that large dependency clusters are widely existed in analyzed software projects. Yang et al. [3] first applied dependence cluster based metrics to effort-aware software defect prediction. Their empirical result showed the combination of the product, process, network and dependence cluster metrics produce more effective models for the prediction of post-release defect than the combination of the product, process and network metrics alone. Our study is different from their studies, we study the defect prediction model in non-effort-aware evaluations with respect to within-project and cross-project.

III. RESEARCH METHODOLOGY

In this section, we first introduce the research questions, then give the research method for the research questions.

A. Research Questions

In order to be easily understand research questions, we use the system dependence clusters shown in Figure 1 [3] to illustrate the questions. In Figure 1, the nodes indicate functions and the directed edges indicate dependencies between functions, which includes the data dependencies and the function call dependencies. Such as, from $f1$ to $f16$ are functions, labeled "d" and "c" indicate data dependencies and function call dependencies, respectively. In this dependency graph, there are 16 functions and 3 dependency clusters which are $dc1$, $dc2$ and $dc3$. In Figure 1, the functions are divided into two groups: functions inside dependence clusters and functions outside dependence clusters. Such as, from $f1$ to $f4$ are functions inside $dc1$, and from $f12$ to $f16$ are functions outside dependence clusters. Functions inside dependence clusters and functions outside dependence clusters form the subgraphs $SubG_{in}$ and $SubG_{out}$, respectively.

Based on the above the preliminary knowledge, we aim to investigate whether dependence clusters are useful for non-effort-aware defect prediction Therefore, our research questions are set up as follows:

System dependence graph at function-level

d: data dependency; c: call dependency;
dc_i: the i-th dependence cluster
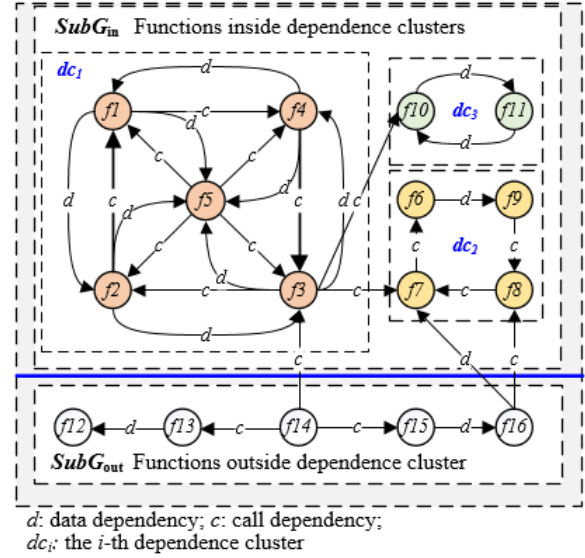
Fig. 1: An SDG with dependence clusters

RQ1. In the scenario of within-project defect prediction, are dependence cluster based metrics useful for non-effort-aware prediction?

RQ2. In the scenario of cross-project defect prediction, are dependence cluster based metrics useful for non-effort-aware prediction?

These research questions are important to both software researchers and practitioners, as they help us better understand the effects of dependence clusters on software quality.

B. Research Method

In order to answer RQ1 and RQ2, we use AIC(Akaike Information Criterion) as the criteria to perform a forward stepwise variable selection procedure to build the following two types of multivariate logistic regression models: (1) the "B" model (using product, process and network metrics); (2) the "B+C" model (using product, process, network and dependence clusters metrics). The logistic regression has been widely used for building defect prediction models [7], [8]. We choose the forward stepwise variable selection rather than the backward stepwise variable selection because the forward stepwise variable selection is less time consuming on stepwise variable selection especially for plenty of independent metrics. The AIC criteria is a widely used variable selection [9].

IV. EXPERIMENTAL SETUP

In this section, we first introduce the projects used in our study and the method of collection the data. Then,

we give a description of the performance indicators in this study.

A. Experimental Subjects

We use the five well-known open source projects to investigate the predictive capability dependence clusters based metrics for non-effort-aware defect prediction: Gstreamer (GSTR), Glibc (GLIB), Gimp (GIMP) and Bash (BASH). They are all GNU projects. In Table I, from the second to seventh columns are respectively the version number, the release date, the total source lines of code in the each studied project, the number of functions, the number of faulty functions, and the percentage of faulty functions. From eighth to the ninth columns are the previous version number and the release date of the previous version, which used to compute the process metrics. The last two columns are the fixing version number and the release date of the fixing release, which are used to determine the faulty or not faulty label for each function.

B. Data Collection

We used the Understand[1] tool and R package igrah[2] to collect the data from the above-mentioned five projects. Metrics for each project consist of: 1) 16 product metrics (i.e. SLoC metrics); 2) 3 process metrics (i.e. code churn metrics); 3) 21 network metrics (i.e. Ties metrics); 4) Collected the dependence clusters for each system ; 5) Collect the importance metrics for dependence clusters [3]; and 6) the faulty or not-faulty labels of the functions after version release.

Table II describes the dependence clusters in the experimental systems. The second to the fifth columns respectively show the number of functions, the number of clusters, the percentage of functions inside dependence clusters, and the size of the largest cluster in each experimental system. We can see that there exist many clusters in these systems from Table II. Table III describes the importance metrics for dependence clusters in the experimental systems. These metrics are widely used network metrics [10].

C. Performance Indicators

At present, most of the existing research work regards the problem of the defect prediction as a binary classification problem. We set defective functions as positive and non-defective functions as negative. We combine the real results of function with the predicted results of model and divide into true positive (TP), false positive (FP), true negative (TN) and false negative (FN). Let TP, FP, TN and FN denote the corresponding numbers of functions, respectively. These values are stored in the confusion

[1]https://scitools.com/
[2]https://igraph.org/r/

matrix, and the confusion matrix is used to compute the Precision, Recall, and F-measure performance indicators. These indicators are defined as follows:

- Precision: The ratio of correctly predicted defective functions over all the functions predicted as being defective. It is calculated as:

$$Precision = \frac{TP}{TP + FP} \qquad (1)$$

- Recall: The ratio of correctly predicted defective functions over all of the true defective functions. It is calculated as:

$$Recall = \frac{TP}{TP + FN} \qquad (2)$$

- F-measure: The indicator is harmonic mean of the precision and recall. It is calculated as:

$$F - measure = \frac{2 \times precision \times recall}{precision + recall} \qquad (3)$$

These indicators are widely used for non-effort-aware defect prediction [11].

V. EXPERIMENTAL RESULTS

In this section, we first describe the models ("B" model and "B+C" model), then we present the experimental results for RQ1 and RQ2.

A. The Models

Figure 2 [3] provides an overview of analysis method for RQ1 and RQ2. In order to answer RQ1 and RQ2, we first use the procedure described in section 3.2 to build "B" model and "B+C" model on each data set, respectively. The introduction of "B" and "B+C" models as follows:

(1) **The "B" model.** It is the baseline model, which is built with product, process, and network metrics. These metrics are described in Table IV. We choose the metrics as the baseline metrics since they are widely used in defect prediction [12].

(2) **The "B+C" model.** The functions are divided into two groups: functions inside dependence clusters and functions outside dependence clusters, the "B+C" model is segmented model which consists of two independent models. They are the "B+C_{in}" model and the "B+C_{out}" model. "B+C_{in}" and "B+C_{out}" models are respectively used for predicting the probability that a function inside and outside dependence clusters are faulty. They are both built with product, process, network and the importance metrics described in Table III. After building the models, we can verify RQ1 and RQ2.

System name	Subject release						Previous release		Fixing release	
	Version number	Release Date	Total SLoC	# functions	# faulty functions	% faulty functions	Version	Release Date	Version	Release Date
Gstreamer	1.0.0	2012-09-24	75985	3946	146	3.70%	0.11.90	2011-08-02	1.0.10	2013-08-30
Glibc	2.1.1	1999-05-24	172599	5923	417	7.04%	2.0.1	1997-02-04	2.1.3	2000-02-25
Gimp	2.8.0	2012-05-12	557436	19978	818	4.10%	2.7.0	2009-08-15	2.8.16	2015-11-21
Gcc-core	4.0.0	2005-04-21	422182	13612	430	3.16%	3.4.0	2004-04-20	4.0.4	2007-01-31
Bash	3.2	2006-10-11	49608	1947	68	3.49%	3.1	2005-12-08	3.2.57	2014-11-07
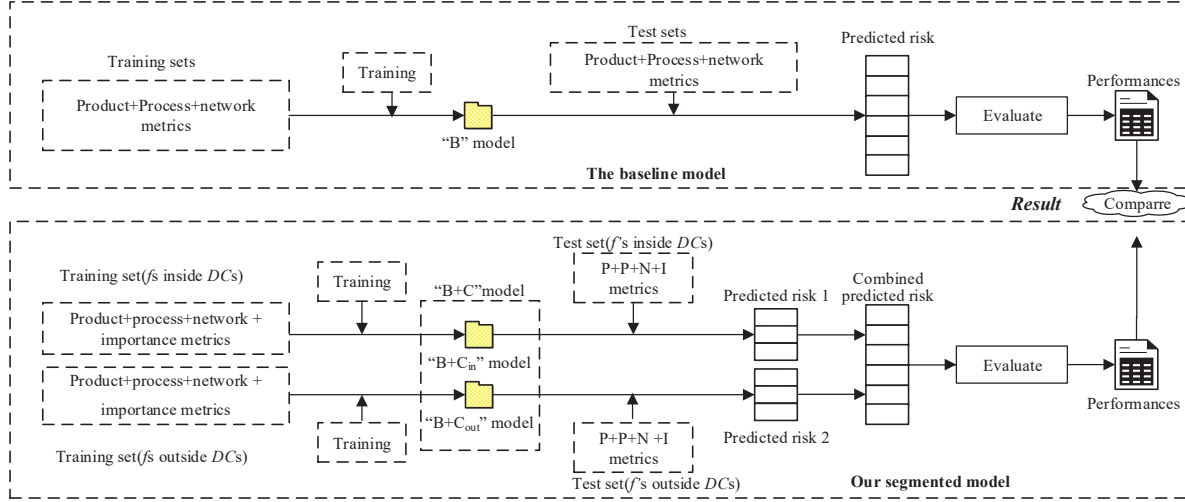
Fig. 2: Overview of the analysis method for "B" VS "B+C" in within-project and "B" VS "B+C" in cross-project

TABLE II: The dependence clusters in experimental systems

System name	# functions	# clusters	% functions Inside clusters	Size of Largest cluster
Gstreamer	3946	59	15.2	170
Glibc	5923	105	11.6	277
Gimp	19978	363	14.2	158
Gcc-core	13612	139	34.9	4083
Bash	1947	41	46.2	483

TABLE III: The importance metrics for dependence clusters

Metrics	Description
Betweenness	# shortest paths through the vertex
Centr_betw	Centrality score according to betweenness
Centr_clo	Centrality score according to the closeness
Centr_degree	Centrality score according to the degrees
Centr_eigen	Centrality score according to eigenvector
Closeness	How close to other vertices
Constraint	The Burt's constraint
Degree	# v's adjacent edges
Eccentricity	Maximum graph distance to other vertices
Page_rank	Google page rank score

TABLE IV: Baseline metrics in this study

Category	Description
Product	SLOC, FANIN, FANOUT, NPATH Cyclomatic, CyclomaticModified, CyclomaticStrict, Essential, Knots, Nesting,MaxEssentialKnots, MinEssentialKnots, n1, n2, N1, N2
Process	Added, Deleted, Modified
Network	Size, Ties, Pairs, Density, nWeakComp, pWeakComp, 2StepReach, ReachEffic, Broker, nBroker, EgoBetw, nEgoBetw, effsize, efficiency, constraint, Degree, Closeness, dwReach, Eigenvector, Betweenness, Power

B. Experimental Result

In the following, we describe the experimental results for RQ1 and RQ2, respectively.

(1) **RQ1.** *In the scenario of within-project defect prediction, are dependence cluster based metrics useful for non-effort-aware prediction?*

For RQ1, we use 30 times 3-fold cross-validation to evaluate the effectiveness of the prediction models. We use the same training/test set to train/test our segmented model (i.e., the "B+C" model) and the baseline model (i.e., the "B" model). On each fold, we first divide the training set into two groups: functions inside dependence clusters and functions outside dependence clusters. Then, we train the "B+C$_{in}$" model and the "B+C$_{out}$" model, respectively. We also divide the test set into two groups and subsequently use the "B+C$_{in}$" model and the "B+C$_{out}$" model to predict the probability of those functions that contain faults. After that, we combine the predicted values to derive the final predicted values to compute the performance indicators.

Based on F-measure predictive values, we use the Wilcoxon's signed-rank test to examine whether two models have a significant difference in their predictive effectiveness. Then, we use the Bonferroni correction method to adjust p-values to examine whether a difference is significant at the significance level of 0.05 [13].Furthermore, we use Cliff's δ to examine whether the magnitude of the difference between the prediction performances of two models is important from the viewpoint of practical application [14]. By convention, the magnitude of the difference is considered either trivial ($|\delta| < 0.147$), small (0.147-

0.33), moderate (0.33-0.474), or large ($|\delta| > 0.474$) [15] . From Table V, we find that the "B+C" models have larger F-measure values than the "B" model in all the five systems except in Gcc-core , and most of cliff's $|\delta|$ values are more than 0.147 except in Gcc-core .That is to say, the dependence cluster based importance metrics are useful for non-effort-aware prediction under within-project evaluation.

(2) **RQ2.** *In the scenario of cross-project defect prediction, are dependence cluster based metrics useful for non-effort-aware prediction?*

TABLE V: The experimental results for RQ1

Projects	"B"	"B+C"	%↑	$|\delta|$
Gstreamer1.0.0	0.139	**0.166**	**19.4%**	0.153 √
Glibc2.1.1	0.068	**0.180**	**164.7%**	0.997 √
Gimp2.8.0	0.065	**0.159**	**144.6%**	0.992 √
Gcc-core4.0.0	0.094	0.086	-8.5%	0.111
Bash3.2	0.187	**0.229**	**22.5%**	0.556 √
Average	0.111	**0.164**	68.5%	0.562

Cross-project defect prediction uses a predicted model trained on one project to predict defect in another projects [8]. From Table VI, we find that the "B+C" models have larger F-measure values than the "B" model, and the cliff's δ values are more than 0.147. That is to say, the dependence cluster based importance metrics are useful for non-effort-aware prediction under cross-project evaluation.

TABLE VI: The experimental results for RQ2

Source	Target	"B"	"B+C"	↑%	$|\delta|$
Gimp2.8.0	Glibc2.1.1	0.132	**0.135**		
	Gstreamer1.0.0	0.071	**0.075**		
	Gcc-core4.0.0	0.061	**0.067**		
	Bash3.2	0.067	0.062		
Glibc2.1.1	Gimp2.8.0	0.107	0.051		
	Gstreamer1.0.0	0.111	0.080		
	Gcc-core4.0.0	0.159	0.069		
	Bash3.2	0.027	0.012		
Gstreamer1.0.0	Gimp2.8.0	0.079	**0.159**	18.5%	**0.203** √
	Gcc-core4.0.0	0.061	**0.106**		
	Bash3.2	0.021	**0.026**		
	Glibc2.1.1	0.132	**0.143**		
Gcc-core4.0.0	Gimp2.8.0	0.025	**0.042**		
	Bash3.2	0.011	**0.052**		
	Glibc2.1.1	0.014	**0.067**		
	Gstreamer1.0.0	0.101	0.099		
Bash3.2	Gimp2.8.0	0.021	**0.078**		
	Glibc2.1.1	0.023	**0.081**		
	Gstreamer1.0.0	0.013	**0.064**		
	Gcc-core4.0.0	0.059	**0.066**		

Overall, the above experimental results show that the non-effort-aware defect prediction capability of "B+C" model is better than that "B" model under the settings of within-project and cross-project prediction.

VI. DISCUSSION

In this section, we further discuss our findings. First, we analyze whether our conclusions will change if the potentially confounding effect of module size is excluded for the "B" and the "B+C" models. Then, we analyze whether we have similar conclusions if the class imbalanced method is used.

(1) Will our conclusions change if the potentially confounding effect of module size is excluded?

In our study, we did not take into account the potentially confounding effect of function size on the associations between those metrics with fault-proneness [16], when building a fault-proneness prediction model. Therefore, it is not readily known whether our conclusions will change if the potentially confounding effect of module size is excluded. In the following, we use the method proposed by Zhou et al. [16] to remove the confounding effect of module size and then rerun the analyses for RQ1 and RQ2. From Table VII, we find that the "B+C" models have larger F-measure values than the "B" model in all the five systems except in Gcc-core based on Within-Project Defect Prediction and most of cliff's δ values more than 0.147 except in Gcc-core. Table VIII, we find that the "B+C" models have most of larger F-measure values than the "B" model based on Cross-Project Defect Prediction ,and the cliff's δ values are more than 0.147. This indicates that "B+C" model still performs better than the "B" model.

TABLE VII: F-measure values after excluding the potentially confounding effect of module size: the "B" model vs "B+C" model based on Within-Project Defect Prediction

Projects	"B"	"B+C"	%↑	$|\delta|$
Gstreamer1.0.0	0.106	**0.162**	52.8%	0.875 √
Glibc2.1.1	0.070	**0.176**	151.4%	0.923 √
Gimp2.8.0	0.085	**0.155**	82.3%	0.728 √
Gcc-core4.0.0	0.084	0.072	-14.2%	0.187
Bash3.2	0.161	**0.221**	37.2%	0.421 √
Average	0.101	0.157	61.9%	0.627

TABLE VIII: F-measure values after excluding the potentially confounding effect of module size: the "B" model vs "B+C" model based on Cross-Project Defect Prediction

Source	Target	"B"	"B+C"	↑%	$|\delta|$
Gimp2.8.0	Glibc2.1.1	0.096	**0.123**		
	Gstreamer1.0.0	0.084	**0.101**		
	Gcc-core4.0.0	0.051	**0.092**		
	Bash3.2	0.043	**0.098**		
Glibc2.1.1	Gimp2.8.0	0.073	**0.091**		
	Gstreamer1.0.0	0.091	0.072		
	Gcc-core4.0.0	0.087	0.057		
	Bash3.2	0.082	**0.107**		
Gstreamer1.0.0	Gimp2.8.0	0.052	**0.087**	18.2%	0.303 √
	Gcc-core4.0.0	0.077	0.052		
	Bash3.2	0.042	0.031		
	Glibc2.1.1	0.081	**0.102**		
Gcc-core4.0.0	Gimp2.8.0	0.072	0.056		
	Bash3.2	0.052	**0.071**		
	Glibc2.1.1	0.081	0.067		
	Gstreamer1.0.0	0.103	0.071		
Bash3.2	Gimp2.8.0	0.041	**0.071**		
	Glibc2.1.1	0.057	**0.101**		
	Gstreamer1.0.0	0.036	**0.052**		
	Gcc-core4.0.0	0.043	**0.087**		

(2) Will our conclusions change if the class imbalanced method is used?

We did not take into account removing the imbalanced data in our study, when building a fault-proneness prediction model. Therefore, it is not readily known whether our conclusions will change if removing imbalanced data. In the following, we use the random under-sampling method proposed by Kamei et al. [17] to remove imbalanced data and then rerun the analyses for RQ1 and RQ2. From

Table IX, we find that the "B+C" models have larger F-measure values than the "B" model except in Gcc-core based on Within-Project Defect Prediction ,and most of cliff's δ values are more than 0.147 except in Gcc-core. Table X, we find that the "B+C" models have most of larger F-measure values than the "B" model, and the cliff's δ values are more than 0.147. This indicates that "B+C" model still performs better than the "B" model.

TABLE IX: F-measure values after removing the imbalanced data: the "B" model vs "B+C" model based on Within-Project Defect Prediction

Projects	"B"	"B+C"	%↑	$|\delta|$
Gstreamer1.0.0	0.177	**0.190**	7.34%	0.256 √
Glibc2.1.1	0.203	**0.276**	36.0%	0.421 √
Gimp2.8.0	0.168	**0.198**	17.9%	0.556 √
Gcc-core4.0.0	0.156	0.152	-2.6%	0.187
Bash3.2	0.122	**0.134**	9.8%	0.375 √
Average	0.165	**0.190**	13.7%	0.359

TABLE X: F-measure values after removing the imbalanced data: the "B" model vs "B+C" model based on Cross-Project Defect Prediction2

Source	Target	"B"	"B+C"	↑%	$|\delta|$
Gimp2.8.0	Glibc2.1.1	0.132	**0.207**		
	Gstreamer1.0.0	0.091	**0.196**		
	Gcc-core4.0.0	0.102	**0.201**		
	Bash3.2	0.097	**0.182**		
Glibc2.1.1	Gimp2.8.0	0.155	0.087		
	Gstreamer1.0.0	0.186	**0.205**		
	Gcc-core4.0.0	0.113	0.067		
	Bash3.2	0.105	**0.109**		
Gstreamer1.0.0	Gimp2.8.0	0.079	**0.127**	12.1%	0.215 √
	Gcc-core4.0.0	0.062	**0.102**		
	Bash3.2	0.061	**0.112**		
	Glibc2.1.1	0.131	**0.192**		
Gcc-core4.0.0	Gimp2.8.0	0.202	0.162		
	Bash3.2	0.167	0.134		
	Glibc2.1.1	0.236	0.195		
	Gstreamer1.0.0	0.232	0.195		
Bash3.2	Gimp2.8.0	0.160	**0.201**		
	Glibc2.1.1	0.203	0.171		
	Gstreamer1.0.0	0.177	**0.181**		
	Gcc-core4.0.0	0.103	**0.105**		

VII. CONCLUSION AND FUTURE WORK

In this paper, we investigate whether dependence clusters are useful for non-effort-aware defect prediction. We use the product, process, and network metrics to build the "B" model (baseline model), and use the product, process, network and dependence cluster metrics to build the "B+C" model. Our experimental results, based on five well-known open-source systems, show that the dependence clusters are useful for non-effort-aware defect prediction. In the future, we plan to build the model for dependence clusters at different granularities and examine their effectiveness.

References

[1] D. Binkley and M. Harman, "Locating dependence clusters and dependence pollution," in *21st IEEE International Conference on Software Maintenance (ICSM'05)*. IEEE, 2005, pp. 177–186.

[2] M. Harman, D. Binkley, K. Gallagher, N. Gold, and J. Krinke, "Dependence clusters in source code," *ACM Transactions on Programming Languages and Systems (TOPLAS)*, vol. 32, no. 1, pp. 1–33, 2009.

[3] Y. Yang, M. Harman, J. Krinke, S. Islam, D. Binkley, Y. Zhou, and B. Xu, "An empirical study on dependence clusters for effort-aware fault-proneness prediction," in *2016 31st IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2016, pp. 296–307.

[4] Y. Yang, Y. Zhou, J. Liu, Y. Zhao, H. Lu, L. Xu, B. Xu, and H. Leung, "Effort-aware just-in-time defect prediction: simple unsupervised models could be better than supervised models," in *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, 2016, pp. 157–168.

[5] T. Hall, S. Beecham, D. Bowes, D. Gray, and S. Counsell, "A systematic literature review on fault prediction performance in software engineering," *IEEE Transactions on Software Engineering*, vol. 38, no. 6, pp. 1276–1304, 2011.

[6] L. C. Briand, W. L. Melo, and J. Wust, "Assessing the applicability of fault-proneness models across object-oriented software projects," *IEEE transactions on Software Engineering*, vol. 28, no. 7, pp. 706–720, 2002.

[7] Y. Yang, Y. Zhou, H. Lu, L. Chen, Z. Chen, B. Xu, H. Leung, and Z. Zhang, "Are slice-based cohesion metrics actually useful in effort-aware post-release fault-proneness prediction? an empirical study," *IEEE Transactions on Software Engineering*, vol. 41, no. 4, pp. 331–357, 2014.

[8] F. Rahman, D. Posnett, and P. Devanbu, "Recalling the" imprecision" of cross-project defect prediction," in *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering*, 2012, pp. 1–11.

[9] F. Rahman and P. Devanbu, "How, and why, process metrics are better," in *2013 35th International Conference on Software Engineering (ICSE)*. IEEE, 2013, pp. 432–441.

[10] S. Wasserman, K. Faust *et al.*, *Social network analysis: Methods and applications*. Cambridge university press, 1994, vol. 8.

[11] X. Chen, Y. Mu, Y. Qu, C. Ni, M. Liu, T. He, and S. Liu, "Do different cross-project defect prediction methods identify the same defective modules?" *Journal of Software: Evolution and Process*, 10 2019.

[12] T. Zimmermann and N. Nagappan, "Predicting defects using network analysis on dependency graphs," in *30th International Conference on Software Engineering (ICSE 2008), Leipzig, Germany, May 10-18, 2008*, 2008.

[13] Y. Benjamini and Y. Hochberg, "Controlling the false discovery rate: a practical and powerful approach to multiple testing," *Journal of the Royal statistical society: series B (Methodological)*, vol. 57, no. 1, pp. 289–300, 1995.

[14] E. Arisholm, L. C. Briand, and E. B. Johannessen, "A systematic and comprehensive investigation of methods to build and evaluate fault prediction models," *Journal of Systems and Software*, vol. 83, no. 1, pp. 2–17, 2010.

[15] J. Romano, J. D. Kromrey, J. Coraggio, and J. Skowronek, "Appropriate statistics for ordinal level data: Should we really be using t-test and cohen'sd for evaluating group differences on the nsse and other surveys," in *annual meeting of the Florida Association of Institutional Research*, 2006, pp. 1–33.

[16] Y. Zhou, H. Leung, and B. Xu, "Examining the potentially confounding effect of class size on the associations between object-oriented metrics and change-proneness," *IEEE Transactions on Software Engineering*, vol. 35, no. 5, pp. 607–623, 2009.

[17] Y. Kamei, T. Fukushima, S. McIntosh, K. Yamashita, N. Ubayashi, and A. E. Hassan, "Studying just-in-time defect prediction using cross-project models," *Empirical Software Engineering*, vol. 21, no. 5, pp. 2072–2106, 2016.

Guidelines for Quality Assurance of Machine Learning-based Artificial Intelligence

Koichi Hamada
DeNA Co., Ltd
Tokyo, Japan
koichi.hamada@dena.com

Fuyuki Ishikawa
National Institute of Informatics
Tokyo, Japan
f-ishikawa@nii.ac.jp

Satoshi Masuda
IBM Research
Tokyo, Japan
smasuda@jp.ibm.com

Mineo Matsuya
LIFULL Co., Ltd.
Tokyo, Japan
matsuyamineo@lifull.com

Tomoyuki Myojin*
Japan Aerospace Exploration Agency
Tsukuba, Japan
myojin.tomoyuki@jaxa.jp

Yasuharu Nishi
University of Electro-Communications
Tokyo, Japan
Yasuharu.Nishi@uec.ac.jp

Hideto Ogawa
Hitachi, Ltd.
Yokohama, Japan
hideto.ogawa.cp@hitachi.com

Takahiro Toku
OMRON Corporation
Kyoto, Japan
takahiro.toku@omron.com

Susumu Tokumoto
FUJITSU LABORATORIES LTD.
Kawasaki, Japan
tokumoto.susumu@fujitsu.com

Kazunori Tsuchiya
FUJITSU LTD.
Kawasaki, Japan
ktsuchiya@fujitsu.com

Yasuhiro Ujita
OMRON Corporation
Kyoto, Japan
yasuhiro.ujita@omron.com

Abstract—Great efforts are currently underway to develop industrial applications for artificial intelligence (AI), especially those using machine learning (ML) techniques. Despite the intensive support for building ML applications, there are still challenges when it comes to evaluating, assuring, and improving the quality or dependability. The difficulty stems from the unique nature of ML: namely, that the system behavior is derived from training data, not from logical design by human engineers. This leads to black-box and intrinsically imperfect implementations that invalidate many of the existing principles and techniques in traditional software engineering. In light of this situation, the Japanese industry has jointly worked on a set of guidelines for the quality assurance of AI systems (in the QA4AI consortium) from the viewpoint of traditional quality-assurance engineers and test engineers. We report the initial version of these guidelines, which cover a list of the quality evaluation aspects, a catalogue of current state-of-the-art techniques, and domain-specific discussions in four representative domains. The guidelines provide significant insights for engineers in terms of methodologies and designs for tests driven by application-specific requirements.

Index Terms—software quality, testing, artificial intelligence, machine learning, guidelines

I. Introduction

Machine learning (ML) is a key driving force for industrial innovation in the form of artificial intelligence (AI) systems. ML-based AI systems consistently display unique characteristics in engineering because components (models) are constructed by training with data in an inductive manner. The obtained components are intrinsically imperfect, i.e., they tend to have limited accuracy, and they are black-box in the sense that the learned behavior is too complex to understand or reason about, especially in the case of deep learning. Further difficulties emerge as such AI systems work with fuzzy requirements regarding human perception or the open real world. One survey showed that more than 40% engineers feel the difficulty of quality assurance for AI systems is at the highest level in the sense that existing approaches are no longer working [1].

At the same time, there is an increasing demand for high-quality and dependable AI systems because they work closely with humans. It is therefore crucial to provide clear guidance for understanding and tackling the difficulties inherent in high-quality AI systems. In response to such industry demands, we established the Consortium of Quality Assurance for Artificial Intelligence-based products and services (QA4AI Consortium), made up of experts from both industry and academia. The objectives of the consortium are to form a societal consensus on quality of AI systems by researching issues and solutions relating to them, and to contribute to the diffusion of ML developments into a safe and secure society.

In this paper, we report the first version of the guidelines for the quality assurance of ML-based AI systems [2]. These guidelines define the general concept and technologies for the quality assurance of AI systems including concrete guidelines relating to the quality characteristics, test architecture, and test viewpoints in each typical domain.

The remainder of this paper is organized as follows: In Section II, we first describe the consortium and the methodology to work on the guidelines. In Sections III and IV, we describe the guidelines in terms of the common core part and domain-specific parts, respectively. We evaluate the

---

guidelines in section V, and discuss the threats to validity of the evaluation in section VI. Section VII introduces the related work of this paper. We conclude the paper with future perspective in Section VIII.

II. METHODOLOGY

A. The QA4AI Consortium

The QA4AI Consortium is a voluntary group to discuss the quality assurance of ML-based AI systems in Japan. Its objectives are to promote the application of ML-based AI systems by reducing the risks associated with AI/ML and to foster common social understanding of their quality, including limitations.

When the first version of the guidelines was released, the consortium consisted of 39 experts and three organizations from both academia and industry. Members include researchers and practitioners in various technical fields including software engineering, system safety, machine learning, and quality assurance. The application domains of the participants are also diverse, covering the entertainment, automotive, factory automation, electrics and electronics, communications, software, IT solutions, consumer devices, web systems, aerospace and more.

B. Structure of Guidelines

The consortium facilitated two types of discussion to formulate the guidelines. In the first, quality assurance-related issues in specific application domains were discussed. The purpose was to derive concrete insights, as general insights might be too abstract for the various domains with different demands. For the first version of the guidelines, there were four working groups: one each for generative systems, operational data in process systems, voice user interface, and autonomous driving.

The second type of discussion was for organizing and summarizing the common core concepts of the quality assurance of ML-based AI systems. These discussions were facilitated by expert members and their output was reviewed by the entire consortium. The common concepts consist of two parts: axes of quality evaluation and a technical catalogue.

The first version of the guidelines was published on the QA4AI Consortium's web site[1] in May 2019. It has the structure below corresponding to the two discussion types:

- Core parts of guidelines, including (1) Axes of Quality Evaluation and (2) Technical Catalogue
- Guidelines for specific domains for (1) Generative Systems, (2) Operational Data in Process Systems, (3) Voice User Interface, and (4) Autonomous Driving

III. CORE PARTS OF GUIDELINES

A. Axes of Quality Evaluation

The quality assurance of ML-based systems has unique aspects in contrast to the quality assurance of traditional, non ML-based systems. Specifically, ML-based systems usually include a complex, nonlinear model constructed in the inductive

[1] http://www.qa4ai.jp

development style for stakeholders, who may be unfamiliar with ML-based system development.

Software development can be divided into the deductive style and the inductive style. The former is that, for traditional software, engineers have rich knowledge on development from their experiences. Quality assurance applies the knowledge such as process assessment, measurement, reviews, and testing. The latter is for ML-based systems, because engineers have poor knowledge how to develop ML-based systems as they are automatically generated, nonlinear and too complex. Traditional process assessment, measurement and reviews are hence ineffective. Frequent, Entire, and Exhaustive Testing (FEET) still works. Engineers have to adopt both the inductive development style for the core ML models and the deductive development style for entire ML-based system.

These guidelines extract five aspects of quality evaluation for ML-based systems: Data Integrity, Model Robustness, System Quality, Process Agility, and Customer Expectation.

Data Integrity relates to the quality assurance of samples of inputs and outputs. This guideline has 11 general checkpoints for statistical considerations, privacy, intellectual property rights, online learning, and quality of the data generator, such as volume and cost, meaningfulness and requirements, relationships between population and sample, bias and contamination, complexity, multicollinearity, outliers and missing values, privacy and confidentiality, intellectual property rights, independence of validation data, and effect of online learning.

Model Robustness relates to the quality assurance of a model generated automatically. This guideline has 11 general checkpoints for the characteristics of neural networks, model performance, generalization, noise, local optima, architecture, hyper parameters, cross validation, data diversity, and degradation.

System Quality relates to the quality assurance of the whole system. This guideline has eight general checkpoints for system-level quality including system performance, validation scope, criticality and frequency of accidents, controllability of the system in accidents, functional safety, security, contribution and localizability of ML components, and explainability and assurability.

Process Agility relates to the quality assurance from the viewpoint of development process. This guideline has 11 general checkpoints for quickness of exploration including short iterations and immediate feedback, scalability, automatability, FEET, appropriate skills and deep understanding, and teamwork.

Customer Expectation relates to the quality assurance for various stakeholders, who may be unfamiliar with ML-based system development. This guideline has eight general checkpoints for extravagant expectation for AI, acceptance of probabilistic behaviour, severity of expectation, optimism for huge data, ambiguity of requirements, compliance, linear and deterministic thinking, and bureaucracy. This axis is the baseline for the other. The higher Customer Expectation is, the higher the other axes need to be.
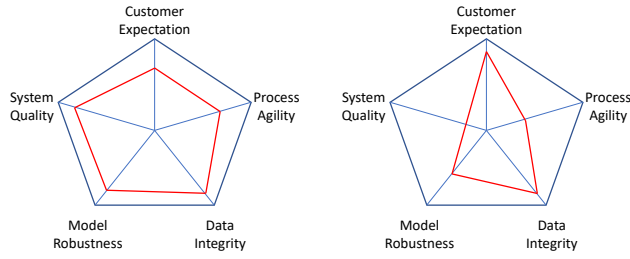
Fig. 1. Examples of Well-balanced and Ill-balanced Quality Pentagon

The total quality of ML systems should be evaluated from the viewpoint of balance among the axes according to Customer Expectation. The development organization of ML-based systems should also establish a well-balanced quality assurance fabric, an organization structure, and a quality management system. Fig. 1 shows examples of a well-balanced and an ill-balanced quality pentagon, consisting of the axes. Furthermore, the total quality of ML-systems usually depends on development phases such as Proof of Concept, Beta Release and deployment of service to a large number of users. The later the phase of development, the better the quality should be.

B. Technical Catalogue

Typically, technical guidelines generalize and summarize techniques and practices being successfully employed in the industry, at least in leading companies. However, for the quality assurance of ML models or ML-based systems, techniques or practices are only just emerging and remain under active investigation. We therefore collected trends from state-of-the-art research papers in the Software Engineering community. We also listed the standard concepts established in the ML community, primarily for performance evaluation, e.g., precision/recall, over/under-fitting, and cross validation.

The state-of-the-art trends we included in the first version of the guideline are as follows.

- Use of pseudo oracle, e.g., [3]
- Metamorphic testing, e.g., [4], [5]
- Robustness evaluation and search for adversarial examples, e.g., [4], [6]
- Structural coverage for neural network [3], [7]
- Methods for explainable AI including local explanation for each output, e.g., [8], [9] and global explanation of the trained model, e.g., [10].

Noted that we endeavor to generalize the concepts as well as decompose multiple aspects combined in one research paper or tool, e.g., in [3].

IV. GUIDELINES FOR SPECIFIC DOMAINS

The five axes provide common guidelines for the quality assurance of ML-based systems, but it is necessary to design a concrete scheme of quality assurance with an appropriate understanding of the characteristics of each system. Therefore, we examined four popular domains in which ML-based systems are used to discuss the characteristics required quality, and the quality assurance viewpoint for each domain.

Fig. 2. Image generation from given pose specification.

A. Generative Systems

There have been outstanding advances in techniques for generative models, which learn "what happens with what probability", particularly in techniques for generative adversarial networks (GANs) [11]. With these techniques, applications that create images, videos, essays, or dialogue can be constructed. We focus on such emerging applications because they have a unique focus when it comes to quality: for example, how natural and diverse the outputs are. Such quality attributes are intrinsically fuzzy and difficult to assess automatically.

Our objective in this domain is to uncover potential approaches to automated evaluation of such quality attributes for emerging generative systems. We defined a concrete application that generates an image or video of an anime character, which is inspired by the technique in [12]. Such functions help create attractive interface agents and videos. We defined five use cases for this application. Two of them are shown below and Fig. 2 illustrates the first example.

1) Generate diverse natural character images of a specified pose given as 2D-coordinates of key body parts
2) Generate a natural character video given two images for the start and end points

For these use cases, we enumerated the quality attributes that should be investigated, which are summarized as follows.

- **Naturalness**, e.g., the outputs let human users feel they are created by human creators.
- **Clearness and Smoothness**, e.g., there is no noise, collapse, or discontinuity in the outputs.
- **Diversity**, e.g., poses (when not specified) or clothing in the outputs have a certain degree of diversity.
- **Social Appropriateness**, e.g., no discriminatory or obscene output is generated.
- **Specification Conformance**, the output follows the given instruction such as genders or color of cloths.

Although they are fuzzy intrinsically due to human perception, the possibilities of automated evaluation should be explored. Three primary approaches for evaluating these quality attributes and some of the examples are shown below:

Approach 1 - Metrics: Define and use metrics that represent the target quality attribute, even approximately. For example, we can leverage the evaluation metrics of GANs for naturalness and diversity [13], [14]. As another example, we can evaluate statistical values and distributions of optical flow, which capture the movement of each part in the frames of the video to detect obviously too drastic movement.

Approach 2 - Evaluation AI: Construct an AI that evaluates the target quality attribute. Pose estimation techniques [15]

can be used to judge whether a generated output matches the specified pose. We can also build our own model for pose estimation, as the training data for the generative model originally includes mappings between poses and images, which can be used as training data for a pose estimation model. We can also investigate a dedicated model and data, for the target quality attributes. For example, we may construct a classifier to detect noisy images by creating training data that includes images with noises automatically added.

Approach 3 - Evaluation Rules: Construct a rule-based AI or traditional software to evaluate the target quality attribute. For example, we can implement an analyzer that checks if the specified clothing color is dominant inside the character in the output image.

B. Operational Data in Process Systems

In industrial systems, ML technologies have been applied and practically used in various fields, such as abnormality detection, parameter recommendation and visual inspection. Quality assurance requires the following three characteristics.

- Stakeholder Diversity: Industrial systems consist of multiple subsystems. Data integrity depends on various stakeholders, operations , and contracts.
- Environmental Dependency: Systems are exposed to unrepeatable and unpredictable changes of 5M + E (man, machine, method, material, measure, and environment).
- Accountability: To operate the whole system, we need to endorse the validity for all of system standards and rules.

Considering the three characteristics and the inductive manner in building machine learning model, we defined a development process model for machine learning system, named Intelligent eXperimental Integration (IXI) model as shown in Fig. 3. This model is divided into three phases: proof of concept (PoC), development, and operation. Major risks should be identified and verified in the PoC phase. In the development phase, industrial systems using machine learning are developed, based on the results of the PoC. In operation phase, the output of the deployed machine learning and the behaviour of the system are monitored and maintain its own quality. The results in each phase are collected to explain to stakeholders, and the machine learning model of the system should be evaluated using risk identified data and during operation and would be updated as necessary. The reason why it is difficult to proceed each phase is there are no rational guideline of evaluation in each activities. So we modeled all of mandatory development and operation activities in IXI model, and defined evaluation viewpoints with relationship of quality model below.

- Customer Expectation: Coordinate intangible assets such as software and involved various stakeholders.
- Data Integrity: Repetitive data confirmation process for environmental changes or deterioration of facilities.
- Model Robustness: Condition of data collection and evaluation process and measurements.
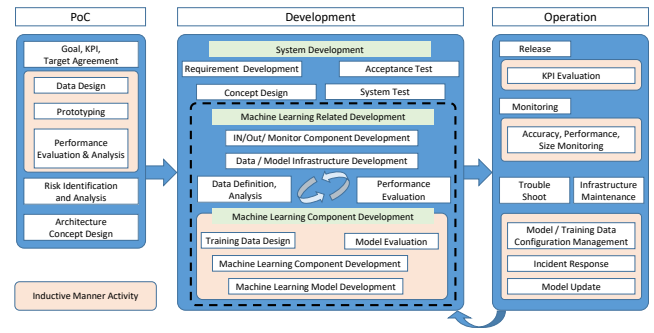
Fig. 3. IXI model : Intelligent eXperimental Integration model

- System Quality: System quality would depend on data and model quality. This criteria shows the evaluation process of each change and explanation to stakeholders.
- Process Agility: Because of above criteria, we emphasized the importance of adapt any changes. We picked up important agile practices.

We also discussed about a real example and added the results to guideline content. It is a system of built-in machine learning system in an industrial machine [16].The system has the three characteristics, so we take it as an good example for our guideline (quality model criteria and IXI model). We discussed criteria, review process and test viewpoints for the case. According to the discussion, we found that there are following pros(+) /cons(-) in our guideline.

- (+) easy to cover quality criteria. It covers all of ML related review points and test viewpoints.
- (+) easy to plan using the IXI phase model. The model helps to understand the necessity of iteration.
- (-) To conclude specific criteria of thresholds and methodology to measure the metrics, still we need ML expert in the project.

Finally, we conclude that our guideline has benefit to cover and plan the quality assurance for industrial system.

C. Voice User Interface System

The voice user interface (VUI) system such as a smart speaker recognizes the user's voice sentence, understands the intent and performs the actions as requested by using the ML technologies as follows.

- Speech recognition: Converts speech signals captured with a microphone into texts
- Natural language understanding: Interprets the converted texts to generate the commands to act
- speech synthesis: Converts texts that are results of the commands to speech signals

We discussed quality of VUI according to the axes of quality evaluation shown in Section III-A. For "Data Integrity", the system requires to perform the same action for the same intention with different voices or expressions. For "Model Robustness", quality of model update is typically important since even new words are created day by day. For "System

TABLE I
EXAMPLE OF TEST ARCHITECTURE FOR VUI SYSTEMS

Test Level	Test Target	Test Viewpoint
Unit test	System modules other than ML	Unit test for each module
	Speech recognition, Natural language understanding and Speech synthesis	Accuracy test for data and ML models
Integration test	APIs	Functional test of integrated modules
System test	Features	Specification-based testing
		Exploratory testing
		Scenario-based testing

Fig. 4. Methodology for analysis of uncertainty and items to verified in AD development process.

Quality", profiles and daily lives of users are of importance because smart speakers are usually placed home. For "Customer Expectation", it is necessary to determine target users for each function and to evaluate whether the users are satisfied.

The test architecture for the smart speakers consists of several test viewpoints in several test levels as shown in Table I. It is, however, difficult to clearly evaluate the conformity to the requirements due to various requirements for VUIs. The n-level evaluation method will solve such difficulty: Various engineers evaluate whether output behaviors are suitable to various intentions and specifications. An example of five-levels evaluation for the smart speaker is shown below:

1) Perform unintended and different function.
2) The intended function is performed, but the content is unintended.
3) The intended function is performed, but unintended information is returned.
4) The intended function is performed and the intended content is returned, but it must be incorrect.
5) The intended function is performed and the intended content is returned.

Quality assurance levels of the whole system of smart speaker can be defined in the following two levels:

1) Behavior level: The results of tests that can be answered with Yes/No meet the specified acceptance criteria
2) Contents level: The results of tests that evaluate attractiveness of the product meet defined acceptance criteria

D. Autonomous Driving

Autonomous driving (AD) utilizes ML-based systems as core technologies for object recognition, path planning, and manipulation decisions. We investigated ideas, approaches, technologies and methodologies that assure the quality of the ML-based systems, focusing on object recognition for Autonomous Emergency Braking (AEB) as a concrete function of AD and scenario of AEB for the first version of our guidelines. It supports automated steering and acceleration capabilities, which correspond to level 2 of the Society of Automotive Engineers (SAE) standard [17].

We Identified three challenges with the quality assurance of AD: AD is expected to reduce crashes compared to human driving, scenes in real driving are now beyond the scope when AD was first developed, and ML models are difficult to replace on the fly after AD is deployed to real driving. As a solution to these challenges, we developed a methodology consisting of the following phases:

1) Analyze a use-case for object recognition of AEB based on a framework to manage uncertainty for AD [18] and structuring-validation [19]
2) Develop fail-safe functions against mis-recognition based on analysis of failures
3) Store knowledge of uncertainty gaps on the real road and feed it back to the analysis at the beginning of the development

An example of this methodology that includes an AD development process, analysis of uncertainty, and items to be verified in the development process is shown in Fig. 4.

This methodology helps to create test cases for the AEB. For the results of analyzing uncertainty of AEB for pedestrians, test cases are represented as a pedestrian who does not look like a pedestrian (false negative) and an object that looks like a pedestrian (false positive). The false negatives include, for example, pedestrians who wear a coat the same color as the wall or who stand behind a pole, and the false positives include a pedestrian reflected in a window and a painting that looks like a pedestrian. The test cases require expected results. A false negative means the AEB will not work, so the driver needs to operate the brake. A false positive means that AEB will work (the car will decelerate) against the driver's expectation, so the driver cannot avoid the deceleration.

V. EVALUATION

We administered a questionnaire survey to evaluate usefulness of guidelines. The respondents were 31 of the readers, including 13 persons had participated in developing the guidelines, since the authors are also users of the guidelines. Table II shows the professions of respondents.

The questionnaire utilized 5-point Likert scale ranging from "strongly deny" to "strongly agree." The summary of questions is listed in Fig. 5. The result of question 1 shows that the whole of users can understand the characteristics of ML-based

Target	Research	Devel.	Testing	Quality	Other	Total
ML/AI	1	11	3	2	0	17
Software	1	0	4	8	0	13
Procurement	0	0	0	0	1	1
Total	2	11	7	10	1	31
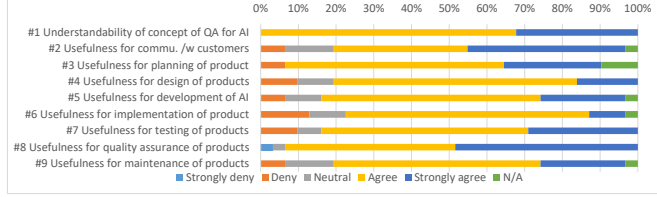
Fig. 5. Responses to questionnaires

Fig. 6. Differences of responses to question 1 between authors and the others

products that completely differs from that for software and the proposed concept of quality assurance of them.

Questions from 2 to 9 address the usefulness of the guidelines at each phase of AI-based system development. Over 77% of respondents agreed or strongly agreed with the usefulness at every phase, especially 94% of them did at the quality assurance phase. These results mean that the QA4AI guideline meets the objective of clarifying general concept of quality assurance of AI-based systems.

VI. THREATS TO VALIDITY

There were few respondents to the questionnaire, so more readers are needed to properly evaluate the guidelines. Moreover, because this was an open web questionnaire, it is possible that only readers who felt positively responded.

Fig. 6 shows the difference of the response to question 1 between the authors and the others. The authors rate the understandability of the guidelines more highly than the others The authors may have a weaker assessment of the guidelines than the others, otherwise, the results may indicate the effect of the consortium's deeper understanding of machine learning properties as they were involved in the development of the guidelines.

VII. RELATED WORK

Reports on practices or case studies are emerging from the industry. Most are general, such as [20], [21], and aspects of quality assurance or testing are very limited. Simple questions to evaluate testing activities were provided in [22]. These questions provide significant guidance on which aspects should be considered, e.g., monitoring input features. Our guidelines, which cover these questions, provide more detailed guidance including the investigation of specific domains in depth.

VIII. CONCLUDING REMARKS

We have reported the active efforts for the quality assurance of ML models and ML-based systems in the QA4AI Consortium driven by the Japanese industry. The first version of a set of guidelines was published, including five axes of evaluation,
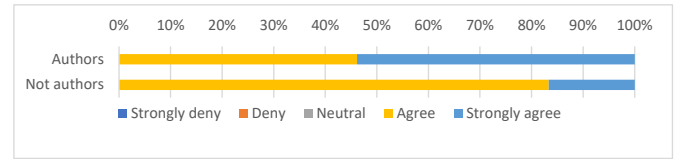
a technical catalogue, and specific insights for four application domains. Testing is the most significant aspect of the guidelines as testing is the most significant activity in practice. The guidelines provide insights from quality-assurance engineers and test engineers. This direction complements specific testing techniques that have been actively investigated, which are also introduced in the guideline.

Given the high demands of the industry, we opted for a quick release and frequent cycles of updates. We are aware that the current guidelines are insufficient for some aspects of the industry. The first version was constructed in a bottom-up, best-effort way to identify what is missing in the guidelines or in the knowledge from research communities. For example, we found there is very little discussion on how to make use of explainability tools such as LIME [8] in engineering activities.

We are continuously working to extend and enhance the guidelines. Current activities include case studies to uncover more insights in each domain as well as to clarify mapping with other standards such as the Ethics Guidelines in the European Commission[2] and quality standards for general software systems (SQuaRE, ISO/IEC 250XX series).

Acknowledgements

REFERENCES

[1] F. Ishikawa and N. Yoshioka, "How do engineers perceive difficulties in engineering of machine-learning systems? - questionnaire survey," in *Joint International Workshop on Conducting Empirical Studies in Industry and 6th International Workshop on Software Engineering Research and Industrial Practice (CESSER-IP 2019)*, May 2018.

[2] QA4AI consortium, "Guideline for quality assurance of ai-based products (in japanese)," http://www.qa4ai.jp/QA4AI.Guideline.201905.pdf, Japan, May 2019.

[3] K. Pei, Y. Cao, J. Yang, and S. Jana, "Deepxplore: Automated whitebox testing of deep learning systems," in *The 26th Symposium on Operating Systems Principles (SOSP 2017)*, October 2017, pp. 1–18.

[4] Y. Tian, K. Pei, S. Jana, and B. Ray, "DeepTest: automated testing of deep-neural-network-driven autonomous cars," in *The 40th International Conference on Software Engineering (ICSE 2018)*, May 2018, pp. 303–314.

[2]https://ec.europa.eu/futurium/en/ai-alliance-consultation/guidelines

[5] A. Dwarakanath, M. Ahuja, S. Sikand, R. M. Rao, R. P. J. C. Bose, N. Dubash, and S. Podder, "Identifying implementation bugs in machine learning based image classifiers using metamorphic testing," in *The 27th ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA 2018)*, July 2018, pp. 118–120.

[6] X. Huang, M. Kwiatkowska, S. Wang, and M. Wu, "Safety verification of deep neural networks," in *The 29th International Conference on Computer Aided Verification (CAV 2017)*, July 2017, pp. 3–29.

[7] L. Ma, F. Juefei-Xu, F. Zhang, J. Sun, M. Xue, B. Li, C. Chen, T. Su, L. Li, Y. Liu, J. Zhao, and Y. Wang, "Deepgauge: Multi-granularity testing criteria for deep learning systems," in *The 33rd ACM/IEEE International Conference on Automated Software Engineering (ASE 2018)*, r 2018, pp. 120–131.

[8] M. T. Ribeiro, S. Singh, and C. Guestrin, ""why should I trust you?": Explaining the predictions of any classifier," in *The 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD 2016)*, August 2016, pp. 1135–1144.

[9] P. W. Koh and P. Liang, "Understanding black-box predictions via influence functions," in *The 34th International Conference on Machine Learning (ICML 2017)*, August 2018, pp. 1885–1894.

[10] E. Angelino, N. Larus-Stone, D. Alabi, M. Seltzer, and C. Rudin, "Learning certifiably optimal rule lists for categorical data," in *The 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD 2017)*, August 2017, pp. 35–44.

[11] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Advances in Neural Information Processing Systems 27 (NIPS 2014)*, December 2014, pp. 2672–2680.

[12] K. Hamada, K. Tachibana, T. Li, H. Honda, and Y. Uchida, "Full-body high-resolution anime generation with progressive structure-conditional generative adversarial networks," in *The 1st Workshop on Computer Vision for Fashion, Art and Design*, Septembe 2018.

[13] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen, "Improved techniques for training GANs," in *The 29th International Conference on Neural Information Processing Systems (NIPS 2016)*, December 2016, pp. 2234–2242.

[14] M. Heusel, H. Ramsauer, T. Unterthiner, B. Nessler, and S. Hochreiter, "GANs trained by a two time-scale update rule converge to a local nash equilibrium," in *The 30th International Conference on Neural Information Processing Systems (NIPS 2017)*, December 2017, pp. 6626–6637.

[15] K. Sun, B. Xiao, D. Liu, and J. Wang, "Deep high-resolution representation learning for human pose estimation," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2019)*, June 2019, pp. 2672–2680.

[16] Y. H. Tsuruta Kosuke, Minemoto Toshifumi, "Development of ai technology for machine automation controller (1)," OMRON technics, Tech. Rep., 2018.

[17] S. O.-R. A. D. Committee *et al.*, "Sae j3016. taxonomy and definitions for terms related to driving automation systems for on-road motor vehicles," tech. rep., SAE International, Tech. Rep., 2016.

[18] K. Czarnecki and R. Salay, *Towards a Framework to Manage Perceptual Uncertainty for Safe Automated Driving: SAFECOMP 2018 Workshops, ASSURE, DECSoS, SASSUR, STRIVE, and WAISE, Västerås, Sweden, September 18, 2018, Proceedings*, 01 2018, pp. 439–445.

[19] L. Gauerhof, P. Munk, and S. Burton, *Structuring Validation Targets of a Machine Learning Function Applied to Automated Driving: 37th International Conference, SAFECOMP 2018, Västerås, Sweden, September 19-21, 2018, Proceedings*, 01 2018, pp. 45–58.

[20] M. Zinkevich, "Rules for reliable machine learning: Best practices for ML engineering," NIPS 2016 Workshop on Reliable Machine Learning in the Wild, December 2017.

[21] S. Amershi, A. Begel, C. Bird, R. DeLine, H. Gall, E. Kamar, N. Nagappan, B. Nushi, and T. Zimmermann, "Software engineering for machine learning: A case study," in *The 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP 2019)*, May 2019, pp. 291–300.

[22] E. Breck, S. Cai, E. Nielsen, M. Salib, and D. Sculley, "What's your ML test score? a rubric for ML production systems," NIPS 2016 Workshop on Reliable Machine Learning in the Wild, December 2017.

Call Sequence List Distiller for Practical Stateful API Testing

Koji Yamamoto, Takao Nakagawa, Shogo Tokui, Kazuki Munakata

Fujitsu Laboratories Ltd.

{yamamoto.kouji,nakagawa-takao,tokui.shogo,munakata.kazuki}@fujitsu.com

Abstract

Necessary and sufficient combinatorial testing is important especially for continuous development to provide stateful service APIs that are invoked by an unspecified number of users. Listing API call sequences for this type of test cases is an important factor in achieving both high test coverage and short time required for test execution. This paper proposes a method to list fewer call sequences without reducing API coverage, and a method to measure the degree of adequacy of an API sequence for testing. Evaluations of more than 400 services show that the listing method reduces the number of sequences for half of the services, and that the measurement method can determine whether the reduction is possible or not for each service with high probability.

Keywords: test for microservices; call sequence listing; stateful API; API specification; API fuzzing

1. Introduction

In the development of application systems using microservices, stateful fundamental functions on remote computing nodes are combined to realize more advanced and valuable functionality. The continuous development process to provide remote side services of fundamental functions involves testing to ensure that any combination of function calls works as intended by the developers.

Each test case of the combinatorial tests for this purpose consists of three parts: a sequence of APIs to invoke functions, input parameter values of the functions, and expected output values returned by the functions. Among them, sequences ("seqs" hereinafter) of APIs are most important because the seqs determine most of the test coverage and the time required to complete the test process.

Representative previous work to list API call seqs for services is RESTler [1] to our knowledge. It lists API call seqs by appending an API to the previously listed seq that outputs values required for the API.
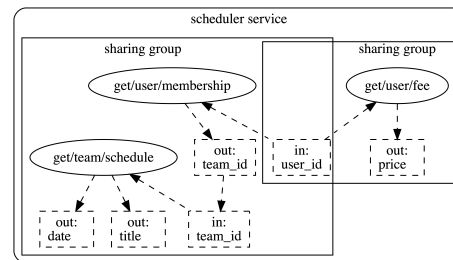
Figure 1. APIs for motivating example and value-sharing groups

From another perspective, test case enumeration pursues two types of aims. One is to find unexpected defects. The other is to ensure the functionalities are (still) as expected. The former is important for testing newly created features. The latter is crucial to continuous development of services. RESTler has achieved the former aim. So the method lists *all* the API seqs in which the value that each API takes is emitted by the predecessor APIs. However, it is necessary to reduce the number of seqs for the latter aim.

Let us see a visualized version[1] of API specification ("spec" hereinafter) in Figure 1 for a certain service. The spec includes APIs that have little to do with each other. Though some APIs in it should be called one after the other for test cases, others need not. It is hypothesized that the values handled by APIs reflect the developers' intent as to which API call should or *should not* follow a particular API call. For instance, the API get/user/membership

---

[1]Ovals in the figure represent APIs. Dashed rectangles stand for values emitted or consumed by APIs. Dashed arrows indicate data flow.

API spec is assumed to be written in a common format such as OpenAPI specification[2] (OAS). For example, the oval named "get/user/membership" represents the API spec in YAML style of OAS2 as follows:

```
paths:
  /user/membership:
    get: {tags: [scheduler service]
        parameters: [{name:user_id, type:string, required:true}]
        responses: {200: {
          schema: {type: array,
                items: {type: object,
                      properties: {team_id: {type: string}},
                      required: [team_id]}}}}}
```

("API$_M$" for short) emits value named team_id; the API get/team/schedule ("API$_S$") takes the value. These indicate a call of API$_M$ can be followed by a call of API$_S$. Therefore API call seq "API$_M$ and then API$_S$" should be a candidate test case. The API get/user/fee ("API$_F$"), contrarily, does not emit values that others take, nor does it take values that others emit. So API$_F$ should follow nothing and vice versa. To decrease in seqs, the above hypothesis can be used to avoid API seqs that are not intended by the developers.

Value-sharing groups (SGs).   In order to determine the degree of adequacy of API call seqs for test cases, we propose a method to construct groups in which APIs can pass values to each other. We call the groups as *value-sharing groups*. More precisely, a value-sharing group is defined as *a minimum disjoint set of APIs that exchange values by emitting only to or receiving only from other member APIs in the same set*. In this paper, values are identified *by name*.

Value-sharing groups could be a method to measure to what extent each API call seq is adequate for a test case by counting the number of sharing groups that APIs in each seq belong to. Formally, a seq is the most adequate iff $|\{sg \in \text{SharingGroups}(spec)|sg \cap seq \neq \emptyset\}| = 1$ by using function SharingGroups in Algorithm 1 where $spec$ is a set of API specs and seq is a set of APIs contained in the seq. For example, APIs in Figure 1 are divided into two value-sharing groups drawn as two rectangles. That is, if the seq is the most adequate, the set seq of APIs in the seq holds $seq \in \mathcal{P}(\{\text{API}_M, \text{API}_S\}) \cup \mathcal{P}(\{\text{API}_F\})$ instead of $seq \in \mathcal{P}(\{\text{API}_M, \text{API}_S, \text{API}_F\})$.

This determination could help filter out API call seqs that previous work lists to reduce the time required for testing.

We preexamined API specs for 2,157 cases[2] of 410 REST services. Half of cases have more than one sharing groups. Therefore, We have developed a method to list API call seqs for testing so that each of the listed seqs is associated with one value-sharing group for almost all seqs.

This method lists API call seqs with exactly one sharing group for all the investigated cases. For 1/3 of the cases, our method lists fewer seqs than previous work. Nevertheless, API coverage by our method is equivalent to the previous work, except for one of the 2,157 cases investigated. We suppose our method reduces the number of seqs for testing without reducing test coverage.

Contributions.   Contributions of this work are:

- We have developed a measurement method to decide the adequacy degree of API call seqs for testing.

- We have developed a way to list fewer API call seqs.

- We have performed quantitative evaluation of the proposed listing method using 2,157 cases of REST services. The evaluation results show our method lists fewer API call seqs for testing than previous work without reducing API coverage.

We show the proposed method and evaluation in sections 2 and 3 resp., discuss related work in 4, then conclude in 5.

2. Proposed Method

To reduce API call seqs for testing, two methods have been developed. One is to divide APIs to value-sharing groups (SGs), which appears in subsection 2.2. It is used to measure a set of API specs and an API call seqs by counting the number of associated sharing groups. Another method is to list the reduced number of API call seqs, which appears in subsection 2.3. The method is also based on the relationships between values emitted or taken by APIs.

2.1. Prerequisites

Suppose you have API specs for a service obtained by parsing the API spec file (in OpenAPI Specification [2] or other formats). Each parsed API spec corresponds to a specific API, and consists of the following information[3]:

- A set $ivals$ of tuples of the API input values. A tuple consists of a value name $name$, a boolean $reqd$ indicating that the value must be input, and a value type.

- A set $ovals$ of tuples of the API output values. The tuple type is the same as in $ivals$, but $reqd$ indicates the value must be outputted.

2.2. Value-sharing group listing function

Function SharingGroups in Algorithm 1 receives a set of API specs each of which is of type described in subsection 2.1 to output a set of SGs for the spec set.

The function creates SGs one by one. Variables $group$, ref, and $nams_N$ contain the SG being created, a set of names for values emitted or taken by at least one member API of the SG, and a set of names for values emitted or taken only by members newly added to the SG, respectively. The function attempts to select new members of the SG (ln. 5). If no member are selected, the function decides to create another SG with any API in $spec$ as an initial member (ln. 7, 11). Otherwise, the function adds selected members to the SG (ln. 11). In either case, the function adds the names for the values that the new members emit or take to

---

Algorithm 1 SharingGroups

Input: A set $spec$ of API specs.

Output: A set $groups$ that stores all the sharing groups as pairs of sets. The 1st set is of APIs in a sharing group. The 2nd set is of value names that the APIs in the group take or emit.

1: $ivals \leftarrow \text{NS}(\underset{api \in spec}{\cup} api.ivals); ovals \leftarrow \text{NS}(\underset{api \in spec}{\cup} api.ovals)$
2: $ungot \leftarrow ivals \setminus ovals; unused \leftarrow ovals \setminus ivals$
3: $groups \leftarrow \emptyset; nams_N \leftarrow \emptyset$
4: **while** $spec \neq \emptyset$ **do**
5:     $mems_N \leftarrow \{api \in spec | \text{VALNS}(api) \cap nams_N \neq \emptyset\}$
6:     **if** $mems_N = \emptyset$ **then**
7:         $api \leftarrow$ select an element from $spec; mems_N \leftarrow \{api\}$
8:         $group \leftarrow \emptyset; ref \leftarrow \emptyset$ ▷ allocate new memories
9:         $groups \leftarrow groups \cup \{\langle group, ref \rangle\}$ ▷ stores $group$
    and ref as references to reflect changes after that in $groups$.
10:     **end if**
11:     $group \leftarrow group \cup mems_N$
12:     $nams_N \leftarrow \cup_{api \in mems_N} \text{VALNS}(api) \setminus ref$
13:     $ref \leftarrow ref \cup nams_N$
14:     $nams_N \leftarrow nams_N \setminus (ungot \cup unused)$
15:     $spec \leftarrow sepc \setminus mems_N$
16: **end while**
17: **return** $groups$
18: **function** VALNS(api)
19:     **return** $\text{NS}(api.ivals) \cup \text{NS}(api.ovals)$
20: **end function**

Algorithm 2 Common functions

21: **function** NS($vals$)
22:     **return** $\{v.name | v \in vals\}$
23: **end function**

---

ref (ln. 13), then removes the members from $spec$ (ln. 15), and replaces $nams_N$ with a name set for the values emitted or taken only by newly added members (ln. 14).

The time complexity of Algorithm 1 is $O(S^2 N)$ for S API specs and N value names because the most expensive part, ln. 5, needs $O(SN)$ at each run and is run $O(S)$ times.

Solid rectangles in Figure 1 shows the result for example.

2.3. Sequence (seq) listing algorithm

Function ListAPISeqs in Algorithm 3 takes API specs $spec$, and builds an API call seq list for testing.

First the function lists the initial API seqs (ln. 27), and stores them into the queue $todo$. Each element in the queue $todo$ is a triple of an API seq and two sets of value names that APIs in the seq take and emit resp. The function picks an API seq (ln. 29), and checks for executability by calling INVOKE[4] (ln. 30). If all the APIs in the seq have been run, the function stores it to the result list $seqlist$ (ln. 32). If the last API call has ended successfully[5], the function extends

---

[4]The definition of the function is omitted.
[5]For REST APIs, ListAPISeqs uses HTTP status code to judge success.

Algorithm 3 ListAPISeqs

Input: A set $spec$ of API specs, a max count N_{list} of seqs, and a max length N_{seq} of a seq.

Output: $seqlist$ that stores all listed API seqs.

24: $seqlist \leftarrow [\,]$ ▷ $seqlist$ is a list of API lists.
25: $todo \leftarrow [\,]$ ▷$todo$ is a queue for triples of an API list, and two sets of names for values taken or emitted by APIs in the list.
26: $given \leftarrow \text{NS}(\cup_{api \in spec} api.ivals) \setminus \text{NS}(\cup_{api \in spec} api.ovals)$
27: EXTEND($[\,], \emptyset, \emptyset$)
28: **while** $todo \neq [\,] \wedge |seqlist| \leq N_{list}$ **do**
29:     dequeue $\langle seq, taken, emitted \rangle$ from $todo$
30:     $\langle done\_whole, last\_result \rangle \leftarrow$ INVOKE(seq)
31:     **if** $done\_whole$ **then**
32:         append seq to $seqlist$
33:         **if** $last\_result$ is successful $\wedge |seq| < N_{seq}$ **then**
34:             EXTEND($seq, taken, emitted$)
35:         **end if**
36:     **end if**
37: **end while**
38: **procedure** EXTEND($seq, taken, emitted$)
39:     $ref \leftarrow emitted \cup taken; feedable \leftarrow given \cup ref$
40:     **for** each $next \in spec$ **do**
41:         $starving \leftarrow \text{NS}(\{v \in next.ivals | v.reqd\})$
42:         **if** $starving \not\subset feedable$ **then**
43:             **continue** to process rest of $next$-s
44:         **end if**
45:         $taking \leftarrow \text{NS}(next.ivals)$
46:         **if** $seq \neq [\,] \wedge (ref \setminus given) \cap taking = \emptyset$ **then**
47:             **continue** to process rest of $next$-s
48:         **end if**
49:         $seq_N \leftarrow seq + [next]$
50:         $taken_N \leftarrow taken \cup (feedable \cap taking)$
51:         $emitted_N \leftarrow emitted \cup \text{NS}(next.ovals)$
52:         enqueue $\langle seq_N, taken_N, emitted_N \rangle$ to $todo$
53:     **end for**
54: **end procedure**

---

it (ln. 34) for longer seqs using procedure EXTEND.

Procedure EXTEND appends a API $next$ to the specified seq seq to get a longer seq seq_N. Not all APIs are used for appending. The procedure uses the following value name sets to pick APIs to append to the seq: **(1)** $given$ – A name set of values taken by at least one APIs in $spec$ and emitted by no API. The values are treated as coming from outside the APIs in $spec$; **(2)** ref – A name set of values emitted or taken by at least one APIs in seq; **(3)** $feedable$ – A name set of values supplied by APIs in seq or externally supplied. EXTEND picks APIs that meet both of the following conditions (Note previous work employs condition **I** alone):

I All the values needed by the API are in $feedable$ (ln. 42).

II If the specified seq seq is not empty, $(ref \setminus given)$ contains at least one value taken by the API (ln. 46).

EXTEND appends each API that holds the conditions to seq

to make a new seq seq_N (ln. 49), then queues seq_N to $todo$ besides names of values taken or emitted by seq_N (ln. 52).

The time complexity of Algorithm 3 is $O(M^Q SN)$ for max API seq length Q, S API specs, N value names, and at most M members for a SG, because EXTEND, which is the most expensive and consumes $O(SN)$, is run $O(M^Q)$ times. On the other hand, the previous work needs $O(S^{Q+1}N)$. It is larger than the former complexity because $O(S) = O(GM) \geq O(M)$ where G is the number of SG.

The output seqs for APIs in Figure 1 are "API$_F$", "API$_M$", and "API$_M$, API$_S$" for example. Besides them, previous work outputs "API$_F$, API$_M$", and "API$_M$, API$_F$".

2.4. Implementation

We have implemented the functions SharingGroups and ListAPISeqs in Python3. We also have made an OAS2 parser required for the prerequisites in section 2.1. It also decomposes arrays[6] in OAS2 to obtain value names of array items, such as team_id in the motivating example API spec. The implementation includes code in which the condition **II** in section 2.3 is disabled to emulate the way of previous work like RESTler for comparison purposes. In the following, the implementation of the proposed method is called *DC* and the previous work is called *SC*.

3. Evaluation and discussion

Using the implementation above, we aim to answer the following research questions:

Q1: Does DC (proposed method) list fewer API call seqs than SC (previous work)?

Q2: Does DC decrease value-sharing groups (SGs) per seq? If so, is the decrease related to the decrease in the listed seq?

Q3: Does DC achieve the same API coverage as SC?

Q4: Is the decrease in listed call seq for a case related to the number of value-sharing groups (SGs) in that case?

To answer these fairly, we have examined *all* the API specs described in OAS 2.0 collected by APIs.guru[3][7].

This examination omits the actual API call portion of ListAPISeq[8] due to lack of access rights to the services. We listed API call seqs up to length 3. We canceled listing if the queue $todo$ was still non-empty after 2,000 seqs had been listed for each service[9]. The distribution of the examined cases with each number of SGs is shown in Figure 2.

---

[6]Object types are not supported yet.

[7]These specs may not have been created by the developers.

[8]It was replaced with a function that always returns $\langle true, true \rangle$.

[9]We gave up 125 of 2,282 cases in 32 of 442 services.
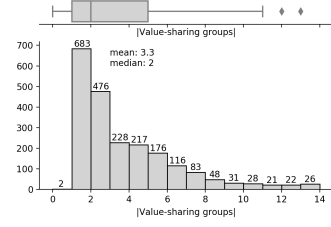
Figure 2. The examined cases distribution
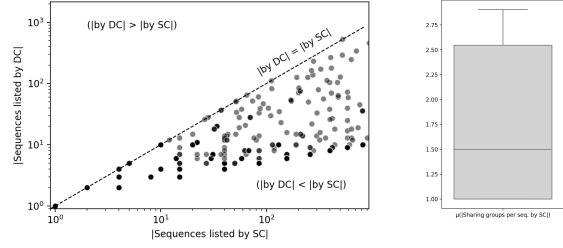
Figure 3. Plots for seqs and groups per seq

3.1. Decrease in seqs and sharing groups (Q1 & Q2)

The rows for n_s in Table 1 and the scatter plot in Figure 3 show Q1 as yes. The number n_s of seqs listed by DC is less than or equal to the number of seqs listed by SC. Each grey dot in the figure indicates how much the number of seqs for each case is reduced by DC.

The rows for n_g in the table say DC sets n_g to 1 in almost all cases. On the other hand, the box plot in Figure 3 shows the numbers of SGs per seq listed by *SC* vary from 1 to 3. Table 2 shows n_s is related to decrease of n_g. These respond affirmatively to the both questions of Q2.

Category		|Cases|
Examined cases		2,157
The number n_s of listed seqs	increased by DC	0
	equivalent	651
	decreased by DC	1,506
The number n_g of sharing groups per seq (mean value)	increased by DC	0
	equivalent (both are 1)	623
	decreased to 1 by DC	1,498
	other; no mean value	36
API coverage (The number of APIs that appear in the listed seqs)	increased by DC	0
	equivalent	2,154
	(both are 100%)	(2,033)
	(both are < 100%)	(121)
	decreased by DC	1
	other; no seqs listed	2

Table 1. The numbers of cases

|Cases|	n_g decreased to 1	otherwise
n_s not decreased	0	651
n_s decreased	1498	8

(The p-value for χ^2 test is 0.0.)

Table 2. Relation of n_g and n_s

|Cases|	Cases having multiple SGs	single SG
n_s not decreased	39	612
n_s decreased	1435	71

(The p-value for χ^2 test is 0.0.)

Table 3. Relation of the number of SGs and n_s

The answer to Q1 indicates DC reduces API call seqs. The answer to Q2 says the reduction may be due to DC creating seqs containing only APIs of a single sharing group.

3.2. API coverage (Q3)

The rows for API coverage in Table 1 show that method DC holds the number of APIs that appear in the listed seqs in almost all cases while the method reduces the listed seqs.

3.3. Relationship between seq and SGs (Q4)

Table 3 shows that the fact that a case has multiple sharing groups (SGs) is related to the fact that DC lists fewer call seqs than SC. Thus, the number of SGs for a service can indicate the possibility of pruning the call seqs for a service by using the proposed method.

3.4. Threats to the validity

One threat to the validity is the services to be examined. We use API specs in APIs.guru[3] alone. The API specs may be biased while the distribution of the number of SGs in Figure 2 appears natural and an evidence of unbiased to us. Besides, we did not evaluate our method with finer grained measures (ex. code coverage) since internal information like code on the examined services is not available.

Another threat is we have not actually called APIs to examine call seqs. Even if the method is based on static analysis, the result should be confirmed by actual execution results. In particular, each seq listed by the proposed method must be checked by actual calls to see it is actually practical.

An important internal threats to the validity is that categorization by shared values may not capture the essential characteristics of API specs. There may be more intuitive and obvious factors. The scatter plot in Figure 3 implies that there can exist other drivers to control the number of seqs listed, even if the shared values is one of the drivers.

4. Related Work

Our algorithm is based on RESTler [1]. It aimed at finding unexpected results. To address another aim, seq reduction, we have to add the idea of condition **II** in section 2.3.

We suppose the proposed approach also improves methods aiming at finding unexpected results since our approach can support effective testing by reducing redundant call seqs. RESTler calls itself an API fuzzing tool. One definition of fuzzing is "the execution of the program under test (PUT) using input(s) sampled from an input space that *protrudes* the expected input space of the PUT"[4] (the emphasis is also by [4]). API seq listing without restriction does not only protrudes the input space[10] but may enlarge it explosively. Our method can control its degree.

MoonShine[5], which lists API call seqs for OS kernels, took a similar approach to ours. Its static analysis has the algorithm for cond. **I** in section 2.3, though cond. **II** is missing. As another advantage, ours depends only on API specs to support PUTs written in any programming languages.

5. Conclusion

We have developed a method to list fewer API call seqs than previous work without losing API coverage. This reduces the number of seqs in half of cases. Another developed method determines whether a seq list is reducible for each case. Nevertheless, we are afraid that these methods alone are insufficient for more practical testing of stateful service APIs. One key to improving the methods is to use attributes of values that APIs input and output more deeply (ex. on the types and the degree of necessity of the values).

References

[1] V. Atlidakis et al. RESTler: Stateful REST API fuzzing. In *IEEE/ACM 41st International Conference on Software Engineering (ICSE)*, pages 748–758, 2019.

[2] OpenAPI specification. http://swagger.io/resources/open-api.

[3] APIs-guru - Wikipedia for web APIs. https://github.com/APIs-guru/openapi-directory.

[4] V. J. M. Manès et al. The art, science, and engineering of fuzzing: A survey. *IEEE Transactions on Software Engineering (Early Access)*, pages 1–1, 2019.

[5] Shankara Pailoor et al. Moonshine: Optimizing os fuzzer seed selection with trace distillation. In *Proceedings of the 27th USENIX Conference on Security Symposium*, SEC '18, page 729–743, 2018.

---

[10]Call seqs are also inputs for combinatorial testing of services.

Impact of Label Noise and Efficacy of Noise Filters in Software Defect Prediction

Shihab Shahriar Khan, Nishat Tasnim Niloy, Md. Aquib Azmain, Ahmedul Kabir

Institute of Information Technology
University of Dhaka
Dhaka, Bangladesh
(bsse0703,bsse0723,bsse0718,kabir)@iit.du.ac.bd

Abstract—A well-established fact in the domain of software defect classification is that dataset labels collected using automated algorithms contain noise. Several prior studies examined the impact of noise and proposed novel ways of dealing with this issue. Those studies, however, relied on randomly simulated artificial noise on clean datasets, but real-world noise is not random. Using a recently proposed dataset with both clean labels annotated by experts and noisy labels obtained by heuristics, this paper revisits the question of how label noise impacts the defect classification performance and demonstrate how the answer varies among several types of classification algorithms. Based on a diverse set of 9 different noise filters, this paper empirically investigates their ability to improve the performance of classifiers trained with label noise. Contrary to previous findings, we observe that the noise filters mostly struggle to improve performance over unfiltered noisy data. Lastly, we conduct several small-scale experiments in a bid to explain our findings and uncover actionable insights.

Index Terms—Defect prediction, Label Noise, Class Imbalance, Noise detection

I. INTRODUCTION

Like any machine learning task, the quality of defect prediction models depends highly on the quality of data used to train them. Unfortunately, noise is pervasive in binary defect prediction datasets, particularly label noise [1], [2]. Label noise occurs when a defective artifact is labeled as non-defective, or vice-versa. There are many possible sources of such noise: mislabeling of issues [3], failure to link issues to relevant code changes [4] or bias of human annotators [5].

Several studies suggest that the presence of mislabeled samples can significantly impact the performance of defect prediction models [1], [6]. Studying the impact of noise is a tricky problem, however, as it requires access to a set of samples that are known to be reliable. Evaluating the model on noisy test data might provide an unreliable estimate of its generalizability. A common approach is to first clean data [7], [8] or to only choose datasets believed to be of high-quality [5], [6], and then randomly introduce artificial noise only in the training subset. But the label noise of defect prediction dataset is not random, and random noise tends to overestimate noise's impact compared to naturally occurring noise [9].

According to [10], the gold standard for label noise research is to use a dataset where both naturally occurring noisy labels,

and their clean, reliable counterparts (produced for example with the help of domain experts) are available. Recently, Yatish *el al.* [11] presented such a defect prediction dataset, and concluded, somewhat surprisingly, that the impact of realistic noise on classifier performance is "modest". To resolve this apparent disparity between results from artificial and clean noise, using the same dataset as [11], this paper revisits the question of how noise impacts classifier performance. We show that except for Naive Bayes, noise has a high impact on most classifiers. Decision tree and random forest were found to lose comparatively the most performance.

Another objective is to find out whether this loss of performance can be mitigated by noise handling techniques. There are several ways to handle dataset noise such as cost-sensitive learning [12], robust classifiers [13] or noise filters. We restrict ourselves to noise filter in this study, and most of the noise handling approaches proposed in defect prediction literature fall under this category. In this paper, 9 filters of diverse properties are investigated to analyze how they individually perform, and how classifiers react to them. To the best of our knowledge, this is the first paper to empirically compare such a broad array of filters in defect prediction setting.

The result of the application of filter is somewhat mixed. On the one hand, across all classifiers and datasets, noise filters barely improve performance over unfiltered noisy labels. On the other hand, from a classifier standpoint, performance improves quite significantly when *best* filter performance is considered. The takeaway is that while filtering *can* improve performance, it crucially depends on the right combination of classifier and filter.

Our final objective is to explain our findings and extract actionable insights from them. We conduct several small-scale experiments to that end. These revealed that the noise model we study is far more challenging than artificially introduced random noise, which may explain the observed big impact of noise on classifiers. They also reveal how traditional approaches to fight the natural imbalance of defect datasets is particularly inadequate in the high-level noisy setting that we study.

II. STUDY DESIGN

The scope of this paper is limited to the "within-project" post-release defect prediction scenario, and the main focus

here is on prediction, not interpretation. Throughout the paper, the noise level (NL) of a dataset refers to the percentage of its total samples that are mislabeled. Imbalance ratio (IR) refers to the ratio of the number of samples in the majority (non-defective) class to the number of samples in minority (defective) class for clean labels, nIR does the same but for noisy labels. The term "model" refers to an instance of the combination of classifier, imbalance-method and filters used. P→N denotes fraction of originally positive (defective) samples that have been flipped to negative (non-defective) in heuristic-based labeling, and N→P denotes the opposite.

All mentions of averages in this study are actually trimmed mean, calculated after trimming away 5% of extreme values from each side. Wilcoxon signed-rank test [14] is used to test statistical significance, with the p-value set to .01. Benjamini-Hochberg procedure is used to correct for multiple comparisons. To compute the effect size, Hedges' g [15] is used with an interpretation of values according to [16], that is $|g| < 0.2$ "negligible", $|g| < 0.5$ "small", $|g| < 0.8$ "medium", otherwise "large"

A. Dataset Description:

This paper uses 32 datasets from 9 open-source software systems presented in [11]. To identify defects introduced by a release, traditionally used heuristic methods rely on strong assumptions like "all bugs reported after release X is introduced in release X" or that "all defect-fixing commits that affect the release X occur within (say) 6 months after it's release", etc. Furthermore, to link a bug report to a defect-fixing commit, they assume that the logs of all bug-fixing commits contain a specific set of keywords (e.g. Bugs, Fix etc.) along with relevant issue ID.

In contrast, datasets used here are all collected from JIRA issue tracker, which allows developers of a software to define what releases were affected by a given bug, earliest of which can be assumed to be the one that introduced it. This makes the produced defective/non-defective labels comparatively far more reliable. For the 32 datasets studied here, summary statistics for their key characteristics is presented in Table 1. #ND and #defective denote number of non-defective and defective samples respectively in clean labels.

TABLE I: Summary statistics of key dataset characteristics

Dataset Property	Min	Median	Max
Size	731	1717	8846
NL(%)	2.35	13.32	28.99
IR	1.97	8.24	45.07
nIR	3.09	12.65	56.87
#defective	26	197	669
#ND	609	1455	8654
P→N(%)	19.23	63.27	93.43
N→P(%)	1.05	4.12	20.97

B. Classifiers

For subsequent experiments, 6 classifiers have been used: Decision Tree (DT), K Nearest Neighbor (KNN), Naive Bayes (NB), Logistic Regression (LR), Support Vector Machine (SVM) and Random Forest (RF). Each of these classifiers has been combined with 5 data balancing techniques: Wilson's editing (ENN), Random Under-Sampling (RUS), SMOTE, Tomek Links (TOMEK) and None (representing no sampling), totaling $6*5 = 30$ models. Two bagging classifiers using Naive Bayes and Decision Tree are also used as base learners, both of which apply RUS independently at each base learner (BagNB and BagDT). This brings the total number of models studied to 32.

C. Noise Filters

All the 9 noise filters operate under the assumption that mislabeled instances are harder to predict than clean ones. These methods can be roughly divided into two groups. The first group, ensemble-based methods, assume a mislabeled training instance will be frequently misclassified by a committee of classifiers. Iterative Partitioning Filter (IPF) [17], Random Forest Filter (IHF) [18] and Instance Hardness Threshold (IHT) [19] belong to this group. IPF partitions the training data into n subsets to train a decision tree on each, and all n classifiers are used to predict the label of each instance (and thereby detect mislabeling) in the training set. RFF takes a cross-validation approach- it uses $n - 1$ subsets of training data to train a random forest classifier, and uses its base trees to predict on remaining subset. IHT differs from IHF only in that it down-samples the majority class, leaving minority class untouched.

The second group uses an instance's nearest neighbors to predict its label. This group includes Neighborhood Cleaning Rule (NCL) [20], SPIDER2 [21], Edited Nearest Neighbor (ENN) [22], SMOTE_ENN [23] and SMOTE_IPF [24] and Closest List Noise Identification (CLNI) [6]. CLNI finds for each instance the percentage of its K nearest neighbors that have different class values, instances for which this percentage cross a certain threshold are removed and repeats this procedure unless some stopping criterion is met. For space consideration, the discussion about the rest of the filters is omitted.

Six of these filters belong to the family of focused imbalance-methods- noise identification and removal are integrated into them. Since the noise detectors have a tendency to overestimate the noise likelihood of minority class [25], for the rest 3 types- IPF, CLNI and RFF, this paper relies on the previously discussed imbalance-methods to balance dataset before applying filtering. All these filters are compared against the baseline "NoF", meaning no filtering.

D. Evaluation:

Following the guidelines of [26], our primary choice for a model's performance evaluation is Matthews Correlation Coefficient (MCC). For a given confusion matrix, MCC is defined as:

$$MCC = \frac{TP \times TN - FP \times FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}}$$

Since this is threshold-dependent, the area under the Precision-Recall curve (APRC) is also used to complement MCC, but we note its performance only when APRC result diverges significantly from MCC. APRC is used over the more popular alternative area under the receiver operating characteristic curve (roc-auc) following the recommendation of [27].

In all of the following experiments, only clean labels are used for evaluation. 5-fold cross-validation has been repeated 5 times for measuring all reported performance values. The experiments are conducted with the help of scikit-learn [28] and imbalanced-learn [29] libraries. For all classifiers, the default hyper-parameter values provided in these libraries are used. The dataset and source code of this study can be found online [1].

III. RESULTS & DISCUSSION

A. How does the presence of label noise impact bug detection?

Figure 1 shows how the performance of examined classifiers compare between clean and noisy labels. As expected, all classifiers fare worse when noise is included. But some, for example, decision tree (DT), reacts quite poorly to noise. This is expected since unpruned decision trees are known to easily overfit [30]. But performance loss of random forest (RF) was somewhat surprising as it has been frequently found to be noise resistant in previous studies [31], [32]. On the opposite spectrum is Naive Bayes and its balanced bagging version BagNB. In fact, as Table II shows, these are the only two classifiers where label noise seems to have had a small effect in terms of MCC. Among others, only SVM has a medium effect, all else are highly impacted by noise. The impact is comparatively less severe for APRC, which is expected since it's threshold-independent. But overall, the impact is statistically significant for all of the results in Table II except one. This result contradicts [2] or is slightly at odds with [11] previous studies, which reported noisy labels having an only modest effect on classifiers.

This performance of Naive Bayes under noise is consistent with previous studies [33] that demonstrated Naive Bayes' superiority in the defect classification task. This is slightly at odds with the findings from general-purpose (i.e. not domain-specific) datasets like the ones from UCI [34], where random Forest, SVM or Boosting approaches perform well. As noise label is pretty ubiquitous in bug prediction datasets used in those studies [7], [8], this finding suggests that Naive Bayes' particular success in defect classification domain may have been primarily due to its unique robustness in the face of label noise.

To summarize:
- The impact of label noise is statistically significant for all classifiers, and the impact is quite large for most.
- Random forest and similar balanced bagging with decision tree perform best when trained with clean labels.
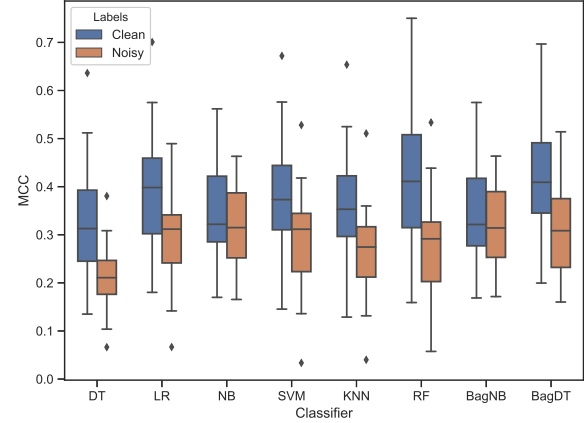
Fig. 1: Comparison of performances of classifiers trained with clean and noisy labels

- NB and bagging with NB are two of the best performing classifiers with noisy labels, particularly owing to NB's surprisingly good inherent robustness to noise.

Discussion: Why is the impact of noise high? One obvious explanation is that the amount of noise in studied datasets is comparatively high compared to previous studies. All of the filters that we studied were tested with the assumption that noise level will not cross 50%. Even many theoretical guarantees we have about learning from corrupted labels is grounded on the same assumption [12], [35]. But as Table I shows, at least for defective class, this assumption does not hold. In 23/32 datasets, noise level in defective class (P→N) is higher than that 50% mark. While overall noise level is well below 50% for all the datasets, several prior studies show using artificial noise that the P→N noise is relatively far more harmful than N→P.

We demonstrate this in a slightly different way: we start with clean labels, for each class we select the samples that are mislabeled by heuristic method, and then flip a certain percentage of them, while keeping the labels of other class

TABLE II: Effect of label noise on classifier performance. "Delta" denotes average *loss* of performance across all datasets and imbalance methods. Results that are NOT statistically significant are marked with *

Classifier	MCC			APRC		
	Delta	Effect	Interpret.	Delta	Effect	Interpret.
DT	.116	1.312	Large	0.082	0.611	Medium
LR	.103	1.003	Large	0.079	0.515	Medium
NB	.022	0.277	Small	0.013	0.125	Neglig.
SVM	.081	0.755	Medium	0.089	0.551	Medium
KNN	.097	1.000	Large	0.086	0.607	Medium
RF	.141	1.224	Large	0.118	0.760	Medium
BagNB	.021	0.268	Small	0.011*	0.087	Neglig.
BagDT	.097	0.962	Large	0.097	0.586	Medium

constant. We then compute how this two types of intentionally introduced noise degrades performance w.r.t clean data, aggregated over all datasets and models.

(a) Impact of each noise type on APRC (b) Impact of each noise type on MCC
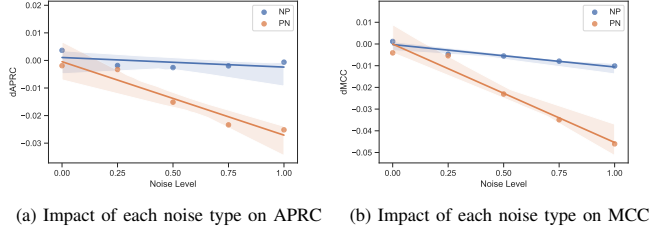
Fig. 2: Impact of P→N vs N→P noise

As Figure 2 shows, performance decreases much more rapidly with P→N noise than N→P. This implies where noise occurs is more important than overall noise level, and that's why these datasets pose a very difficult task for any learner.

B. How much can noise filters recover performance?

As previously mentioned, 9 noise filters are applied to the training dataset. Figure 3 boxplot shows how much each of them improves performance over unfiltered noisy labels, i.e the difference in MCC and APRC, across 32 models and 32 datasets.
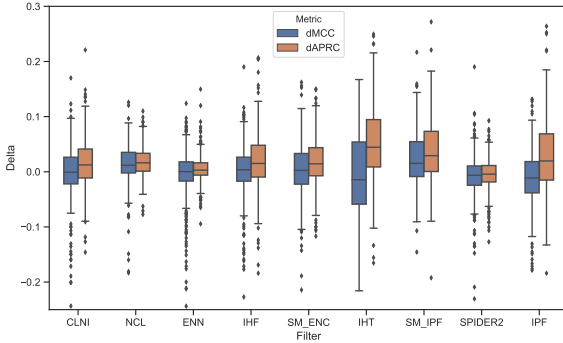
Fig. 3: Performance *improvement* by noise filters on noisy labels across all models and datasets w.r.t baseline no-filering (NoF)

One salient finding is that the median for most of the filters lies pretty close to zero, a value that indicates no improvement over non-filtered data. In fact, the boxes quite often go below the zero mark, these are the cases where filtering actually decreased performance. Some outliers go even below the range presented in the figure.

As both Figure 3 and Table III show, the results vary slightly among the evaluation metrics. Among all the filters, classifiers and datasets, APRC performance on average is improved by .0203, for MCC this value is 0.0012. With MCC, only 2 out of 9 filters, NCL and SMOTE_IPF, bring statistically significant improvement. With APRC, this is true for all except SPIDER2. But irrespective of evaluation metric, the effect of

performance improvement remains small even for the best performing filters.
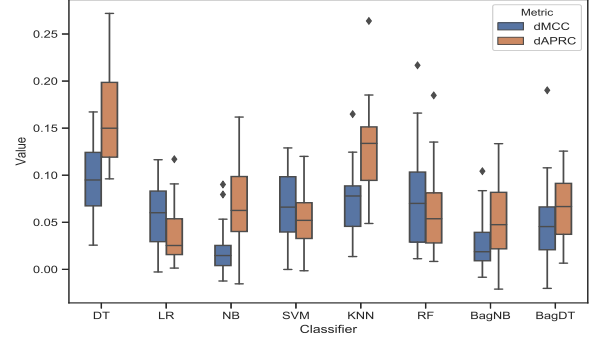
Fig. 4: Performance improvement of each classifier using the best filter for that classifier

In Figure 4, for each classifier, we plot the improvement brought by the best among all 9 filters for that classifier. This represents an upper bound on how much a classifier can recover performance from the application of filtering. The figure reveals that KNN and DT classifiers can benefit most from noise filters. Using only the best value among filters, performance improvement across all models and datasets on average is .057 for MCC and .077 for APRC. This implies filtering *can* positively impact performance, as long as right classifier-filter combination is used.

Discussion: So, Why do filters struggle to improve performance? Apart from high noise in defective class discussed before, we believe a big part of this answer lies in the fact that these datasets are highly class-imbalanced. Next, we consider 3 of the most common ways to address imbalance in turn: no sampling, under-sampling and over-sampling, and provide preliminary evidence to demonstrate why they might be inadequate for the datasets at hand.

1) No Sampling: We begin with the baseline where we do not do anything to address class imbalance. [26] suggests that class balancing is important for moderate or highly imbalanced datasets, where moderate imbalance was defined as having

TABLE III: Performance *improvement* for each noise filter across all datasets and models. Statistically significant improvements are marked with *.

| Filter | MCC | | | APRC | | |
	Delta	Effect	Interpr.	Delta	Effect	Interpr.
SM_IPF	0.0237*	0.275	Small	0.0359*	0.274	Small
SPIDER2	-0.0080	-0.121	Neglig.	-0.0046	-0.04	Neglig.
IHF	0.0001	-0.061	Neglig.	0.0188*	0.138	Neglig.
NCL	0.0163*	0.156	Neglig.	0.0172*	0.127	Neglig.
ENN	-0.0042	-0.079	Neglig.	0.0057*	0.045	Neglig.
IHT	-0.0066	-0.079	Neglig.	0.0496*	0.356	Small
CLNI	-0.0031	-0.099	Neglig.	0.0159*	0.119	Neglig.
IPF	-0.0116	-0.130	Neglig.	0.0264*	0.205	Small
SM_ENC	0.0041	0.038	Neglig.	0.0175*	0.119	Neglig.
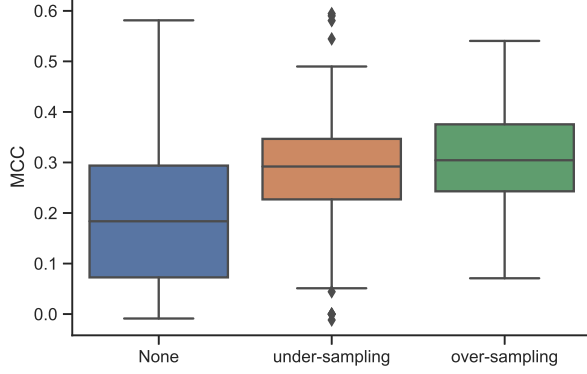
Fig. 5: Comparison of 3 approaches to data balancing accross all datasets and models. under-sampling includes ENN and RUS, over-sampling only includes SMOTE.

IR> 3.94. Using the same threshold, 27 out of 32 datasets we use is moderately or highly imbalanced, suggesting this option might not be optimal. We illustrate this using IPF, one of the filters which does not have data balancing baked in. As Figure 5 shows, both of the alternatives clearly outperform no-sampling.

2) Under-Sampling: These techniques remove samples only from majority class, unless a desired level of balance with minority class is reached. One problem with these approaches is that when imbalance is high, this means throwing out a big portion of dataset. For example, given our median nIR=12.65, under-samplers will have to throw out 85.4% of overall data to create perfect balance.

Some techniques try to minimize the impact of data loss by filtering out uninformative or noisy samples. In fact, 3 of our 9 filters: IHT, NCL and ENN, belong to this group. As we'll show, even when they work exactly as intended i.e. even in the best case scenario, they still struggle to noticeably improve performance over simple random under-sampling. To test this, we first removed each mislabeled sample from majority class of training set, and then kept randomly removing samples until class balance is achieved. While the improvement (across all datasets and models) over random under-sampling is statistically significant, effect size reveals that the improvement is "negligible" for both MCC and APRC (.155 and .129 respectively).

3) Over-Sampling: As a representative of this family of samplers, we choose perhaps the most widely used data balancing technique: SMOTE [36]. In short, SMOTE randomly selects one of the minority (defective) samples and one of it's nearest neighbors, then samples a random point from the line connecting those two samples in feature space. This new sample naturally gets labeled defective. Therefore, quality of labels of new samples depends crucially on the quality of existing minority samples' labels.

Unfortunately, majority of the samples labeled as defective

are actually non-defective due to noise. While the ratio of N→P noise as shown in Table I is comparatively small, even that small noise level can overwhelm minority class due to high imbalance. For example, in a dataset with nIR=12.65, P→N=63.27% and N→P=4.12% (all are median values taken from Table I), only 41.3% of samples labeled as defective will actually be defective. This in turn means that only 17% of SMOTE-generated defective samples is expected to originate from a pair of defective samples, whereas for around 34% samples both of their parents will actually be non-defective. About half ($\sim 49\%$) will originate from one clean and one mislabeled sample. Using PCA transformation, we illustrate this idea with JRuby-1.5.0 dataset in Figure 6.
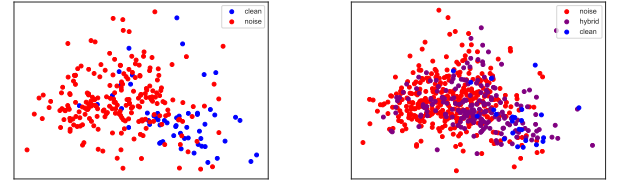
(a) Original defective samples   (b) Samples generated by SMOTE

Fig. 6: Distribution of mislabeled and clean samples in defective class. Explained variance for the figures is 57% and 59% respectively.

IV. THREATS TO VALIDITY

Our study crucially depends on the assumption that the labels we assume to be clean are actually clean. One issue is that these labels are human-annotated. While this makes the so-called clean labels far more reliable than their heuristics-based counterparts, they are still susceptible to human error, due to phenomenons like snoring [37] or annotator bias [5] for example.

Also, a few prominent classifiers like Neural Network or XGBoost [38] are excluded in this study, mainly due to their computationally extensive training procedures. Similar consideration also led us to skip extensive hyper-parameter tuning, something that can impact a classifier's performance [39]. We note however, that any such optimization would have to be carried out on a subset of noisy training data, and this can lead to poor choice of hyper-parameter values [40].

V. CONCLUSION AND FUTURE WORK

By using a diverse set of classifiers, imbalance-methods and noise filters, this study empirically investigates how the presence of label noise in post-release defect prediction datasets affect performance and evaluates the effectiveness of noise filters in minimizing the adverse effects of noise. The principal conclusions of this study are (1) Label noise in bug prediction datasets do have large impact on most classifier's performance, (2) Noise filtering isn't guaranteed to improve performance, but with the right choice of classifier-filter combination, it can yield significant improvement especially for classifiers

that easily overfit, e.g., decision tree. This study also revealed the highly robust nature of the Naive Bayes algorithm, the surprising brittleness of Random Forest and took the first steps towards explaining these findings.

For future work, we plan to investigate several alternatives to filtering for noise handling. The relatively higher cost of P→N noise suggests while designing any auto defect-labeling algorithm, recall of defect class should be prioritized over precision. We also plan to explore this idea in future.

REFERENCES

[1] C. Bird, A. Bachmann, E. Aune, J. Duffy, A. Bernstein, V. Filkov, and P. Devanbu, "Fair and balanced?" in *Proceedings of the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering on European software engineering conference and foundations of software engineering symposium - ESEC/FSE '09*. ACM Press, 2009. [Online]. Available: https://doi.org/10.1145/1595696.1595716

[2] K. Herzig, S. Just, and A. Zeller, "It's not a bug, it's a feature: how misclassification impacts bug prediction," in *Proceedings of the 2013 international conference on software engineering*. IEEE Press, 2013, pp. 392–401.

[3] J. Aranda and G. Venolia, "The secret life of bugs: Going past the errors and omissions in software repositories," in *Proceedings of the 31st international conference on software engineering*. IEEE Computer Society, 2009, pp. 298–308.

[4] A. Bachmann, C. Bird, F. Rahman, P. Devanbu, and A. Bernstein, "The missing links: bugs and bug-fix commits," in *Proceedings of the eighteenth ACM SIGSOFT international symposium on Foundations of software engineering*. ACM, 2010, pp. 97–106.

[5] F. Rahman, D. Posnett, I. Herraiz, and P. Devanbu, "Sample size vs. bias in defect prediction," in *Proceedings of the 2013 9th joint meeting on foundations of software engineering*. ACM, 2013, pp. 147–157.

[6] S. Kim, H. Zhang, R. Wu, and L. Gong, "Dealing with noise in defect prediction," in *2011 33rd International Conference on Software Engineering (ICSE)*. IEEE, 2011, pp. 481–490.

[7] J. Van Hulse, T. M. Khoshgoftaar, and A. Napolitano, "Skewed class distributions and mislabeled examples," in *Seventh IEEE International Conference on Data Mining Workshops (ICDMW 2007)*. IEEE, 2007, pp. 477–482.

[8] C. Seiffert, T. M. Khoshgoftaar, J. Van Hulse, and A. Folleco, "An empirical study of the classification performance of learners on imbalanced and noisy software quality data," *Information Sciences*, vol. 259, pp. 571–595, 2014.

[9] C. Tantithamthavorn, S. McIntosh, A. E. Hassan, A. Ihara, and K. Matsumoto, "The impact of mislabelling on the performance and interpretation of defect prediction models," in *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, vol. 1. IEEE, 2015, pp. 812–823.

[10] B. Frénay and M. Verleysen, "Classification in the presence of label noise: a survey," *IEEE transactions on neural networks and learning systems*, vol. 25, no. 5, pp. 845–869, 2013.

[11] S. Yatish, J. Jiarpakdee, P. Thongtanunam, and C. Tantithamthavorn, "Mining software defects: should we consider affected releases?" in *Proceedings of the 41st International Conference on Software Engineering*. IEEE Press, 2019, pp. 654–665.

[12] N. Natarajan, I. S. Dhillon, P. Ravikumar, and A. Tewari, "Cost-sensitive learning with noisy labels." *Journal of Machine Learning Research*, vol. 18, pp. 155–1, 2017.

[13] T. G. Dietterich, "An experimental comparison of three methods for constructing ensembles of decision trees: Bagging, boosting, and randomization," *Machine learning*, vol. 40, no. 2, pp. 139–157, 2000.

[14] R. Woolson, "Wilcoxon signed-rank test," *Wiley encyclopedia of clinical trials*, pp. 1–3, 2007.

[15] L. V. Hedges and I. Olkin, *Statistical methods for meta-analysis*. Academic press, 2014.

[16] J. Cohen, "A power primer." *Psychological bulletin*, vol. 112, no. 1, p. 155, 1992.

[17] T. M. Khoshgoftaar and P. Rebours, "Improving software quality prediction by noise filtering techniques," *Journal of Computer Science and Technology*, vol. 22, no. 3, pp. 387–396, 2007.

[18] M. Sabzevari, G. Martínez-Muñoz, and A. Suárez, "A two-stage ensemble method for the detection of class-label noise," *Neurocomputing*, vol. 275, pp. 2374–2383, 2018.

[19] M. R. Smith, T. Martinez, and C. Giraud-Carrier, "An instance level analysis of data complexity," *Machine learning*, vol. 95, no. 2, pp. 225–256, 2014.

[20] J. Laurikkala, "Improving identification of difficult small classes by balancing class distribution," in *Conference on Artificial Intelligence in Medicine in Europe*. Springer, 2001, pp. 63–66.

[21] K. Napierała, J. Stefanowski, and S. Wilk, "Learning from imbalanced data in presence of noisy and borderline examples," in *International Conference on Rough Sets and Current Trends in Computing*. Springer, 2010, pp. 158–167.

[22] D. L. Wilson, "Asymptotic properties of nearest neighbor rules using edited data," *IEEE Transactions on Systems, Man, and Cybernetics*, no. 3, pp. 408–421, 1972.

[23] G. E. Batista, R. C. Prati, and M. C. Monard, "A study of the behavior of several methods for balancing machine learning training data," *ACM SIGKDD explorations newsletter*, vol. 6, no. 1, pp. 20–29, 2004.

[24] J. A. Sáez, J. Luengo, J. Stefanowski, and F. Herrera, "Smote–ipf: Addressing the noisy and borderline examples problem in imbalanced classification by a re-sampling method with filtering," *Information Sciences*, vol. 291, pp. 184–203, 2015.

[25] J. Van Hulse and T. Khoshgoftaar, "Knowledge discovery from imbalanced and noisy data," *Data & Knowledge Engineering*, vol. 68, no. 12, pp. 1513–1542, 2009.

[26] Q. Song, Y. Guo, and M. Shepperd, "A comprehensive investigation of the role of imbalanced learning for software defect prediction," *IEEE Transactions on Software Engineering*, 2018.

[27] T. Saito and M. Rehmsmeier, "The precision-recall plot is more informative than the roc plot when evaluating binary classifiers on imbalanced datasets," *PloS one*, vol. 10, no. 3, p. e0118432, 2015.

[28] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg *et al.*, "Scikit-learn: Machine learning in python," *Journal of machine learning research*, vol. 12, no. Oct, pp. 2825–2830, 2011.

[29] G. Lemaître, F. Nogueira, and C. K. Aridas, "Imbalanced-learn: A python toolbox to tackle the curse of imbalanced datasets in machine learning," *The Journal of Machine Learning Research*, vol. 18, no. 1, pp. 559–563, 2017.

[30] M. Bramer, "Using j-pruning to reduce overfitting in classification trees," in *Research and Development in Intelligent Systems XVIII*. Springer, 2002, pp. 25–38.

[31] T. M. Khoshgoftaar, J. Van Hulse, and A. Napolitano, "Comparing boosting and bagging techniques with noisy and imbalanced data," *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans*, vol. 41, no. 3, pp. 552–568, 2010.

[32] C. Pelletier, S. Valero, J. Inglada, N. Champion, C. Marais Sicre, and G. Dedieu, "Effect of training class label noise on classification performances for land cover mapping with satellite image time series," *Remote Sensing*, vol. 9, no. 2, p. 173, 2017.

[33] S. Wang and X. Yao, "Using class imbalance learning for software defect prediction," *IEEE Transactions on Reliability*, vol. 62, no. 2, pp. 434–443, 2013.

[34] A. Asuncion and D. Newman, "Uci machine learning repository," 2007.

[35] N. Manwani and P. Sastry, "Noise tolerance under risk minimization," *IEEE transactions on cybernetics*, vol. 43, no. 3, pp. 1146–1151, 2013.

[36] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "Smote: synthetic minority over-sampling technique," *Journal of artificial intelligence research*, vol. 16, pp. 321–357, 2002.

[37] A. Ahluwalia, D. Falessi, and M. Di Penta, "Snoring: a noise in defect prediction datasets," in *Proceedings of the 16th International Conference on Mining Software Repositories*. IEEE Press, 2019, pp. 63–67.

[38] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system," in *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*. ACM, 2016, pp. 785–794.

[39] C. Tantithamthavorn, S. McIntosh, A. E. Hassan, and K. Matsumoto, "The impact of automated parameter optimization on defect prediction models," *IEEE Transactions on Software Engineering*, vol. 45, no. 7, pp. 683–711, 2018.

[40] D. I. Inouye, P. Ravikumar, P. Das, and A. Dutta, "Hyperparameter selection under localized label noise via corrupt validation," in *NIPS Workshop*, 2017.

Using Deep Learning Classifiers to Identify Candidate Classes for Unit Testing in Object-Oriented Systems.

Wyao Matcha
Software Engineering
Research Laboratory
Department of Mathematics
and Computer Science
University of Quebec,
Trois- Rivieres, Quebec,
Canada
Wyao.Matcha@uqtr.ca

Fadel Touré
Software Engineering
Research Laboratory
Department of Mathematics
and Computer Science
University of Quebec,
Trois- Rivieres, Quebec,
Canada
Fadel.Touré@uqtr.ca

Mourad Badri
Software Engineering
Research Laboratory
Department of Mathematics
and Computer Science
University of Quebec,
Trois- Rivieres, Quebec,
Canada
Mourad.Badri@uqtr.ca

Linda Badri
Software Engineering
Research Laboratory
Department of Mathematics
and Computer Science
University of Quebec,
Trois- Rivieres, Quebec,
Canada
Linda.Badri@uqtr.ca

Abstract — **This paper aims at investigating the use of deep learning to suggest candidate classes to be tested rigorously during unit testing. The approach is based on software unit testing information history and source code metrics. We conducted our experiments using data collected from five (5) successive versions of the open source Java Apache software system ANT. For each version, we collected various source code metrics from the source code of the Java classes. We then extracted testing coverage measures for software classes for which dedicated JUnit test classes have been developed. We considered instruction and method level coverage granularities. Based on the different datasets collected, we trained several deep neural network models. We validated the constructed classifiers using Cross Version Validation technique. The obtained results strongly support the viability of our approach with an average accuracy greater than 87%.**

Keywords- Unit Testing, Tests Prioritization, Source Code Metrics, Testing Coverage Measures, Machine Learning, Deep Learning.

I. INTRODUCTION

Testing is the stage of software development where developers (testers) assess the conformity of the system developed (under development) with specifications [1]. Testing plays a crucial role in software quality assurance. It is, however, a time and resource-consuming process. Unit testing is one of the main phases of the testing process where each software unit is early and individually tested using dedicated unit test cases. In object-oriented (OO) software systems, units are software classes and testers usually write a dedicated unit test class for each software class they decided to test [2]. The main goal of unit testing is to early reveal the faults of software classes. In the case of large-scale OO software systems, because of resources limitation and time constraints, it is difficult, even unrealistic, to test rigorously all the classes [3]. The unit testing efforts are often focused. Testers (developers) usually prioritize unit tests by selecting and focussing the unit testing effort on a limited set of software classes (most critical) for which they write dedicated unit tests. Tests case prioritization is a method for prioritizing and planning test cases [3, 5]. This technique is used to run higher priority test cases (focus on the most critical components) to minimize time, cost and effort during the software testing process [3-6].

In this paper, we focus on unit testing of classes and particularly on how to automatically suggest suitable classes for unit testing using deep learning and information on both unit testing history and source code metrics. Several research related to test cases prioritization [7] using different OO metrics (The Chidamber and Kemerer metric suite in particular [8]) have been proposed in the literature. Some of these metrics, related to different software class attributes, have already been used in recent years to predict unit testability of classes [9-13]. These studies analyzed in particular various open source Java software systems and the corresponding JUnit test classes. One of the observations that have been made in these studies is that unit test cases were written only for a subset of classes [11-13].

Currently, many software repositories are available and can be used (among others) to analyze and predict software quality. Based on these repositories, can we extract valuable information that can be used to help software testers in prioritizing unit tests? Is it possible to automate this process? The main goal of our research is to propose an approach to identifying suitable classes for unit testing using deep learning techniques [14,15] (artificial neural networks). The approach is based on software unit testing information history and source code metrics. We conducted our experiments using data collected from five (5) successive versions of the

open source Java Apache software system ANT [16]. For each version, we first collected various source code metrics from the source code of the Java classes. We then extracted testing coverage measures for software classes for which dedicated JUnit [17] test classes have been developed. We considered instruction and method level coverage granularities. With the collected data, we trained several deep neural network models. We validated the constructed classifiers using Cross Version Validation technique. The obtained results strongly support the viability of our approach with an average accuracy greater than 87%.

II. RELATED WORK

Touré et al. [7] investigated an approach based on software information history to support the prioritization of classes to be tested. They have analyzed different class attributes of ten Java open source software systems for which JUnit unit test classes were developed. Using different techniques, they first characterized the classes for which JUnit test classes were developed by the testers. Secondly, they built two classifiers using OO metrics and unit tests information collected from the selected systems. The classifiers provide, for each software, a set of classes on which the unit testing efforts must be focused. The sets of obtained candidate classes were compared to the sets of classes for which JUnit test classes were developed by the testers. The results have shown that: (1) average values of the metrics of the classes tested are very different from average values of the metrics of the other classes (2) there is a significant relationship between the fact that a JUnit test class was developed for a class and its source code attributes, and (3) the sets of classes suggested by the classifiers correctly reflect the selection of testers.

Mirarab et al. [18] used Bayesian networks to create a unified model based on information provided by the CK metrics suite [8], changes and testing coverage rates. The defined approach optimizes coverage and improves the fault detection rate compared to the random planning of test scenarios.

Helge Spieker et al. [19] present Retecs (Reinforcement Learning for Automatic Test Cases Prioritization and Selection in Continuous Integration), which is a method for selecting and prioritizing test cases based on machine learning. It is used in cases of continuous integration with the aim of minimizing the round-trip delay between code validation and developer comments in the event of failure of the test cases. The Retecs method uses reinforcement learning to select and prioritize test cases according to their duration, the last execution history and failure. In an environment where new test cases are created and obsolete test cases are deleted, the Retecs method learns to prioritize error-prone tests using a reward function and build on previous cycles of integration. By applying Retecs on data extracted from three industrial case studies, Helge Spieker et al. have shown for the first time that reinforcement learning allows fruitful selection and automatic prioritization in continuous integration and regression tests.

III. DATA COLLECTION

For our investigations, we carried out our experiments on Apache ANT software system, which is a command-line tool used to control the execution processes described in mutually dependent XML files (build files) [16]. The versions 1.3, 1.4, 1.5, 1.6 and 1.7 of ANT system have been extracted to carry on our experiments.

A. Data Collection Tools

The source code of ANT's considered versions has been grabbed from GitHub [20] repository. Under IntelliJ [21] IDE, we used Code Mr [22], Cover [23] plugins as well as the JUnit Framework [17] to run the unit test suites and collect source code metrics and testing coverage measures.

B. Source Code Metrics

Four our study, we considered six (6) source code metrics, five from the well-known metrics suite proposed by Chidamber and Kemerer [8] and the widely used SLOC metric. These metrics capture various OO attributes such as coupling, inheritance, cohesion, complexity and size. They have received particular attention in empirical software engineering research and are computed by Code Mr plugin according to the following definitions:

Coupling metrics: (1) CBO (Coupling between objects) calculates the number of classes to which a class is linked and vice versa.

Inheritance metrics: (2) DIT: (Depth of Inheritance) counts the number of classes between the measured class and the root of its inheritance hierarchy. (3) NOC: (Number of Children) calculates the number of subclasses that inherit from the measured class.

Cohesion metrics: (4) LCOM: (Lack of cohesion in Methods) assesses the lack of cohesion in a class. It counts the number of methods pairs whose similarity is 0 minus the number of method pairs whose similarity is different from zero.

Complexity metrics: (5) WMC: (Weighted Method Complexity) sums up the cyclical complexities of all the methods of the measured class. (6) RFC: (Response for classes) calculates the number of possible methods that can be called in response to the method invocation of the measured class.

Size metrics: (7) SLOC: (Source Lines of Codes) calculates the number of lines of code in measured class.

C. Descriptive statistics

The descriptive statistics on the different versions of the ANT system [13] show that the number of classes increases from 1093 for version 1.3 to 1145 classes for version 1.7. The average complexity (WMC) and source lines of code (SLOC) vary slightly from one version to another (around 19 for WMC and 156 for SLOC). This small variation is also observed for the other metrics average values. The great variability of metrics within the same version indicated by the

standard deviations (σ), reflects the variability level of the software classes characteristics.

TABLE 1:Descriptive statistics of the used source code metrics - ANT

Vers.	obs.	Stat	CBO	DIT	LCOM	SLOC	NOC	RFC	WMC
V1.3	1096	Min	0	1	0	4	0	1	1
		Max	672	7	30	1899	143	326	236
		Mean	10.97	2.4	1.85	156.64	0.67	26.06	19.74
		σ	1111.2	1.82	5.36	49022.6	27.09	1130	864.26
V1.4	1102	Min	0	1	0	6	0	1	1
		Max	674	7	30	2204	143	321	235
		Mean	10.84	2.39	1.84	155.65	0.67	25.78	19.51
		σ	1104.3	1.82	5.35	50129.7	26.97	1106.4	841.08
V1.5	1103	Min	0	1	0	6	0	1	1
		Max	674	7	30	2204	143	321	235
		Mean	10.84	2.39	1.84	155.73	0.67	25.79	19.52
		σ	1104.3	1.82	5.35	50304.1	26.97	1108.1	843.18
V1.6	1140	Min	0	1	0	6	0	1	1
		Max	694	7	30	2214	145	321	252
		Mean	10.8	2.38	1.82	155.01	0.66	25.63	19.44
		σ	1123	1.82	5.18	50743.9	27.72	1096.2	840.85
V1.7	1143	Min	0	1	0	4	0	0	1
		Max	694	7	30	2214	145	321	255
		Mean	10.81	2.39	1.83	157.33	0.66	25.65	19.45
		σ	1125.1	1.83	5.25	53739.4	27.71	1100.1	843.81

D. Empirical Analysis

We have collected the source code of the five different versions of ANT [16] considered for the study, grabbed from GitHub repository [20]. We followed the steps described below for each version:

Step 1: Extracting the considered source code metrics using Code Mr [22] plugin.
Step 2: Running unit test suites and collecting the testing coverage rates by using Cover [23] plugin and JUnit [17] framework.
Step 3: Filtering outlier observations, which are mainly OO artefacts with 0 complexity (interfaces, some of abstract classes, enumerations and constants collecting classes).
Step 4: Computing the rank of each value of the metrics. This step is motivated by the fact that the classifier will be trained on several versions of different sizes. Ranks will help the classifiers to mitigate the metrics' values.
Step 5: Labelling testing coverage data. We have labelled each class with a binary value, 1 or 0, depending on whether its unit testing coverage reaches a given threshold percent or not. 50%, 30% and 0% thresholds have been considered. For example, by setting the threshold at 50%, all the classes having a testing coverage greater or equal to 50% are labelled as 1 (considered as tested), and the rest of classes is labelled as 0 (considered as not tested).

For each class, the attributes formed by the source code metrics, the associated rank values as well as the binarized coverage rates, form a labelled observation. With the collected data, our goal is to train a deep neural network model to build classifiers that could automatically predict (suggest) the label of each observation. In addition, we wanted to investigate if a classifier built from the dataset of version n, correctly predicts the level of testing coverage in the successive n + 1 version.

TABLE 2: Distribution of tested classes - method granularity

Versions	Obs	0%	30%	50%
13	1096	179	108	81
14	1102	187	114	82
15	1103	193	117	85
16	1140	193	116	85
17	1143	196	119	87

TABLE 3: Distribution of tested classes - instruction granularity

Versions	Obs	0%	30%	50%
13	1096	179	98	60
14	1102	187	102	61
15	1103	193	105	65
16	1140	193	103	62
17	1143	196	106	64

IV. EXPERIMENTAL METHOD

A. Neural networks

Neural networks belong to the family of machine learning classifier models [24, 25]. The building blocks for neural networks are artificial neurons [26]. These are simple computational units that have weighted input signals and produce an output signal using an activation function [27].

Neurons are arranged into networks of neurons. A row of neurons is called a layer. Neural networks may have multiple layers. The first layer, called input or visible layer, takes input from the dataset. The following layers are referred as hidden layers. They are responsible for compressing and building internal representation of datasets. The last layer, also called output layer, is responsible for classifying the output on the required classes.

Deep neural networks refer to networks containing more than one hidden layer. Training those layers requires large amount of data and specific techniques that prevent vanishing gradient issue [25].

The neural net training process relies on forward propagation technique that consists of feeding input values to the neural network and getting an output (predicted value). On each training epoch, error is computed using back-propagation technique [28].

B. Construction and Architecture of the Neural Network

Our deep learning architectures (ANN) are built using Python programing language, under TensorFlow [29] and Keras [30] frameworks supported by Pandas [31] library for data manipulation. We carried out several tests before coming to the following configuration that produced the obtained results.

Our input layer contains fourteen (14) neurons to match the 14 characteristics (7 for source code metrics plus 7 for rank values associated with each metric value). Thirteen (13) hidden layers follow the input layer containing 169 neurons each. And finally, 2 neurons compose the output layer to match our binary classification problem. We used ReLu as activation function for input and hidden layers. We relied on "Adam" as optimizer and the mean square as loss function. The model has been evaluated using confusion matrix.

C. Classifier Validation

We validated our classifier using Cross Version Validation (CVV) technique inspired by classical cross validation techniques. CVV approach consists of training the neural network model on dataset of a version V of the ANT system, then validating the obtain classifier on the dataset of the successive version V+1.

V. RESULTS AND INTERPRETATIONS

The CVV technique has validated the suggestions of classes to be tested for a given version, made by a classifier trained on the previous version. The following results were obtained by this validation technique applied to the different considered versions of ANT.

TABLE 4: CVV results on Method granularity level

	CM50	CM30	CM0
13->13	93.20%	91.77%	88.28%
13->14	92.97%	91.19%	87.99%
14->14	94.04%	90.84%	88.79%
14->15	93.77%	90.57%	88.26%
15->15	93.51%	92.62%	89.06%
15->16	93.64%	92.35%	88.57%
16->16	94.67%	94.07%	89.86%
16->17	94.25%	93.56%	89.70%

From Table 4 (CVV results on method granularity level), it can be seen that:

The threshold of 50% produces accuracy scores that are greater than 92%. The lowest accuracy is obtained for a validation on version 1.4 (92.97%) and the highest is obtained for a validation on version 1.7 (94.25%).

For a threshold of 30%, the obtained accuracy scores are greater than 90%. The lowest accuracy is obtained while validating on version 1.5 (90.57%) and the highest for the validation on version 1.7 (93.56%).

For a threshold of 0%, the accuracy scores obtained are greater than 87%. The lowest accuracy is obtained when validation is done on version 1.4 (87.99%) and the highest for validation on version 1.7(89.70%).

After all validations, we can notice the trends of the accuracy to increase. Indeed, the highest accuracy scores obtained are noticed for a threshold set at 50%. This indicates that there exists a relationship between classes metrics and labelled classes (classes labelled 1 or 0). Other results obtained with threshold sets at 30% and 0% are good but show that their classifiers are not as accurate as the one obtained with the threshold set at 50%.

From these results, we can conclude that by using method granularity level, we are able to predict correctly candidate classes for unit testing.

TABLE 5: CVV results on Instruction granularity level

	CLOC50	CLOC30	CLOC0
13->13	94.63%	93.47%	88.37%
13->14	94.48%	93.06%	87.28%
14->14	95.37%	94.22%	89.86%
14->15	95.02%	93.95%	89.32%
15->15	94.48%	94.57%	90.12%
15->16	94.67%	94.24%	89.69%
16->16	95.79%	94.76%	88.57%
16->17	95.62%	94.42%	88.58%

From Table 5 (CVV results on instruction granularity level), it can be seen that:

For a threshold set at 50%, the accuracy scores obtained are greater than 94%. The lowest accuracy is obtained for a validation on version 1.4 (94.48%) and the highest is obtained for a validation on version 1.7 (95.62%).

For a threshold set at 30%, the accuracy values obtained are greater than 93%. The lowest accuracy is obtained while validating on version 1.4 (93.06%) and the highest for validation on version 1.7 (94.42%).

For a threshold set at 0%, the accuracy values obtained are greater than 87%. The lowest accuracy is obtained for validation on version 1.4 (87.28%) and the highest for validation on version 1.6 (89.69%).

As we go from validation on version 1.3 to validation on version 1.7, we observed an increase of the accuracy from 87% to 89%.

As in the case of method granularity level, we obtained the highest accuracy scores for a threshold set at 50%. Hence a classifier obtained by training it on the dataset will be more efficient and accurate than the one obtained with another threshold. We also observe a tendency of the accuracy to increase during the test.

From these results, we can conclude that by using instruction granularity level, we are also able to predict correctly candidate classes for unit testing.

By comparing the results obtained for the different levels of granularities (methods and instruction), we see that we obtain high accuracy scores for the instruction level.

From these two tests, we can conclude that the choice of the level of granularity as well as the threshold set is crucial.

The CVV validation showed that it is possible to predict classes to be tested on a version developed or under development by a classifier trained on the previous version of the same system. Moreover, we can notice that we got higher accuracy rates for the instruction granularity level with a threshold of 50%.

VI. LIMITATIONS

The obtained results are quite significant with regard to the used dataset (information collected from five different versions of a same system) but should nevertheless be considered as exploratory. The selection of the classes to test explicitly (for which JUnit test classes have been developed) is in most cases left to the goodwill of developers. This can lead to partially tested classes and systems with few tested classes, which in turn can influence the results obtained in our various experiments.

The generalization of our results requires additional investigations including tests quality as well as the application's domain of the considered software systems, which can also impact the results and restrict their scope. In our analyzes, we limited ourselves to Java OO language. Even if Java is a reference language in OO programming, our results could be biased by this limitation.

We limited our investigations to a well-known case study system (ANT). The study should be replicated on more software systems in order to draw more general conclusions.

VII. CONCLUSION

In this work, the goal was to provide an approach that supports unit testing decision when selecting the software classes to be tested. For that, we used a classifier build from deep neural network model based on various class source code metrics and the corresponding unit testing coverage data. Two levels of testing coverage measures have been considered in combination. We validated the constructed classifier using cross version validation technique. By obtaining after various tests more than 87% of accuracy rates, we can conclude that the obtained results strongly support the viability of the approach.

These results open the possibility of using software metrics and the developers' experience (particularly in terms of unit testing development) in guiding the distribution of the overall unit testing effort. Hence using current big data analysis techniques (involving artificial intelligence algorithms), it is possible to develop cloud-based tools in order to build a new generation of tests prioritization tools and guidance integrated into software development environments.

REFERENCES

[1] B. Boehm, "A Spiral Model of Software Development and Enhancement", Proc. Int'l Workshop Software Process and Software Environments, ACM Press, 1985; also in ACM Software Eng. Notes, Aug. 1986, pp. 22-42.

[2] B. Boehm. "Software Engineering Economics". Prentice Hall, Englewood Cliffs, NJ, ISBN-10 : 0138221227, edition 01 oct 1981.

[3] S. Elbaum, A. G. Malishevsky and G. Rothermel, "Test Case Prioritization: A Family of Empirical Studies", IEEE Transactions Software Engineering, Vol. 28, No. 2, pp.159-182, 2002.

[4] S. Elbaum, A. G. Malishevsky and G. Rothermel, "Prioritizing test cases for regression testing". Proc. ACM SIGSOFT Int. Symp. on Software Testing and Analysis (ISSTA), Portland, OR, USA, 22-25 August 2000, pp. 102-12

[5] G. Rothermel, R.H Untch, C. Chu and M.J Harrold . "Test case prioritization: an empirical study", International Conference on Software Maintenance, Oxford, UK, pp. 179–188.,1999.

[6] J. Kim, and A. Porter , "A history-based test prioritization technique for regression testing in resource constrained environments", In Proceedings of the International Conference on Software Engineering, 2002.

[7] F. Toure., M. Badri and L. Lamontagne, "Investigating the Prioritization of Unit Testing Effort Using Software Metrics", In Proceedings of the 12th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE'17) Volume 1: ENASE, pages 69-80, 2017.

[8] Chidamber S.R. and Kemerer C.F., "A Metrics Suite for Object Oriented Design", IEEE Transactions on Software Engineering, vol. 20, no. 6, pp. 476–493, 1994.

[9] M. Bruntink , and A.V. Deursen,"Predicting Class Testability using Object-Oriented Metrics", 4th Int. Workshop on Source Code Analysis and Manipulation (SCAM), IEEE, 2004.

[10] V. Gupta, K. K. Aggarwal and Y. Singh, "A Fuzzy Approach for Integrated Measure of Object-Oriented Software Testability", Journal of Computer Science, Vol. 1, No. 2, 2005, pp. 276-282. doi:10.3844/jcssp.2005.276.282.

[11] M. Bruntink and A. Van Deursen, "Predicting class testability using object-oriented metrics", in Proceedings of the 4th IEEE International Workshop on Source Code Analysis and Manipulation (SCAM '04), pp. 136–145, September 2004.

[12] M. Bruntink and A. van Deursen, "An empirical study of class testability", Journal of Systems and Software, vol. 79, no. 9, pp. 1219–1232, 2006.

[13] L. Badri, M. Badri, and F. Toure, "An empirical analysis of lack of cohesion metrics for predicting testability of classes", International Journal of Software Engineering and Its Applications, vol. 5, no. 2, 2011.

[14] K. Jarrett, K. Kavukcuoglu, M. A. Ranzato, and Y. LeCun. "What is the best multi-stage architecture for object recognition?" In International Conference on Computer Vision, pages 2146–2153. IEEE, 2009.

[15] Y. LeCun, K. Kavukcuoglu, and C. Farabet, Convolutional networks and applications in vision. In Circuits and Systems (ISCAS), Proceedings of 2010 IEEE International Symposium on, pages 253–256. IEEE, 2010.

[16] Apache ANT releases, https://github.com/apache/ant/releases, Visited in december 2019.

[17] JUnit Framework, https://junit.org/junit5/. Visited in december 2019.

[18] S. Mirarab, A. Hassouna, and L. Tahvildar, "Using Bayesian belief networks to predict change propagation in software systems" in Proceedings of the 15th IEEE International Conference on Program Comprehension, pages 177-188, 2007.

[19] H. Spieker, A. Gotlieb, D. Marijan and M. Mossige, "Reinforcement learning for automatic test case prioritization and selection in continuous integration", Proceedings of the 26th ACM SIGSOFT International Symposium on Software Testing and Analysis, July 2017.

[20] ANT on Github Repository, https://github.com/apache/ant, Visited in december 2019.

[21] IntelliJ IDE, https://www.jetbrains.com/idea/, Visited in december 2019.

[22] Code Mr plugin, https://plugins.jetbrains.com/plugin/10811-codemr/, Visited in december 2019.

[23] Code-Coverage plugin, https://www.jetbrains.com/help/idea/code-coverage.html, Visited in december 2019.

[24] Artificial Intelligence and life in 2030, one-hundred-year study on artificial Intelligence, report of the 2015 Study panel, September 2016

[25] Y. LeCun, Y. Bengio, G, Hinton. "Deep learning" . Nature. 2015;521(7553):436-444. doi:10.1038/nature14539

[26] F. Rosenblatt, "The perceptron : A probabilistic model for information storage and Organization in the brain", in cognitive systems. Buffalo: Cornell Aeronautical Laboratory, Inc. Rep. No. VG-1196-G-1, 1958.

[27] M. Minsky, and S. Papert, "Perceptrons: An Introduction to Computational Geometry," MIT Press, expanded edition, ISBN-10 : 0262631113, décembre 1987

[28] D. E. Rumelhart, G. Hinton, and R. J. Williams, "Learning representations by back-propagating errors", Cognitive modeling 5.3 (1988):

[29] Tensorflow: https://www.tensorflow.org/

[30] Kera: https://keras.io/

[31] Panda: https://pandas.pydata.org

Unit Testing Effort Prioritization Using Combined Datasets and Deep Learning: A Cross-Systems Validation

Fadel TOURE

Department of Mathematics and Computer Science,
University of Quebec at Trois-Rivières,
Trois-Rivières, Québec, Canada.
Fadel.Toure@uqtr.ca

Mourad BADRI

Department of Mathematics and Computer Science,
University of Quebec at Trois-Rivières,
Trois-Rivières, Québec, Canada.
Mourad.Badri@uqtr.ca

Abstract— **Unit testing plays a crucial role in object-oriented software quality assurance. Software testing is often conducted under tight time and resource constraints. Hence, testers do not usually cover all software classes. Testing needs to be prioritized and testing effort to be focused on critical components. The research we present in this paper is part of the development of a collaborative decision support tool allowing the developers' community to pool their unit testing experiences when selecting the candidate classes for unit tests. To achieve this, we proposed in our previous work a unit tests prioritization approach based on software information histories and software metrics. The goal is to suggest classes to be tested by building a classifier that matches the testers selection. Several machine learning classifiers have been previously considered. The current paper explores the deep neural network models with more software source code metrics including explicitly and implicitly tested classes. The training datasets that have been combined are from different systems. So, we considered metrics ranks. Using a cross systems validation technique, obtained results strongly suggest that deep neural network-based classifiers correctly reflect the tester's selections and could thus help in decision support during the selection of candidate classes for unit tests.**

Key words— *Tests Prioritization; Unit Tests; Source Code Metrics; Deep Neural Network; Deep Learning; Machine Learning Classifiers.*

I. INTRODUCTION

Software testing plays a crucial role in software quality assurance. Unit testing is one of the main phases of the testing process where each software class is individually tested using dedicated test cases. In object-oriented (OO) software systems, units are software classes and testers usually write a dedicated unit test class for each software class they decided to test. The unit tests aim at early reveal faults in software classes. In the case of large-scale OO software systems, because of resource limitations and tight time constraints, the unit testing efforts are often focused. Testers usually select a limited set of software classes for which they write dedicated unit test classes. Knowing that it is often not realistic to equally test all software classes, it becomes important for testers to target the most critical and fault-prone classes. However, the task is not obvious and requires a deep analysis of software. These issues belong to the family of tests prioritization topics. Several existing approaches try to prioritize test suites execution in order to discover the maximum of faults quickly, while others try upstream to focus the developer efforts on suitable classes to be tested. This paper focus on how to automatically target suitable candidate classes for unit tests. The long-term goal is to build a collaborative tool for the developers' community. That tool will collect source code metrics and classes unit test information from different projects in order to improve a unique cloud-hosted classifier performance to match the testers' selection of unit tests candidate classes. For new systems under development, the tool could suggest, after collecting some specific source code metrics, a set of candidate classes for unit tests. Due to the large source code diversity and increasing amount of data that the tool will face, we considered using deep neural network models trained on combined systems' datasets to explore how accurate the classifiers could match the testers' selection.

Many OO metrics, related to internal software class attributes have been proposed in literature [1, 2]. Some of them have already been recently used to predict unit testability of classes in OO software systems [3-10] by analyzing various existing open source Java software systems for which Junit [11] test cases were developed and are accessible in public repositories. For all systems, authors [3-10] found that only a subset of classes have dedicated unit test classes written by developers. In previous work [9, 12], we focused on how the selection of the candidate classes for unit tests was made by testers. Multivariate Logistic Regression, Naive Bayes, Random Forest and K-Nearest Neighbours classifiers have been used to automate the selection of candidate classes for unit tests. They have been validated within systems and between systems using Cross Systems Validation (CSV) and Leave One System Out Validation (LOSOV). The latter validation technique implied the use of combined datasets extracted from different systems.

Based on deep neural network models [13], the current work includes more source code metrics to capture various characteristics that we believe are determinative for a software class to be considered as a good candidate for unit tests from testers' point of view. We also included two ways of labelling tested classes according to the existence of dedicated unit test classes and to the actual unit testing coverage.

The paper is organized as follows. Section II presents some related works. Section III addresses the OO software

metrics we used for this study. Section IV describes the data collection procedure and the considered systems. Section V presents the empirical study we conducted and the results obtained with the related discussions. Section VI reports the main threats to validity relatively to our empirical experimentations. Finally, Section VII concludes the paper, summarizes the contributions of this work and outlines several directions for future investigations.

II. RELATED WORK

Test case prioritization has been widely discussed in the context of regression testing. Various techniques have been proposed in the literature and used different leverages. We can distinguish: (1) coverage rates based techniques, (2) software history information based techniques, and (3) risk analysis based techniques.

Fault detection techniques focus on targeting the most fault prone components using, in practice, fault exposure factors as a proxy. Factors are estimated using different ways from the software artifacts. The results obtained by Rothermel et al. [11] and Yu and Lau [12] indicated that this approach improves the fault detection rates.

The coverage-based techniques run the test suites that cover most modified software artefacts during regression testing. Several machine learning algorithms (Naïve Bayes, Genetic Algorithms) are used to derive a prioritization approach. The investigations [14-16] results showed that coverage-based techniques also lead to fault detection rate improvement.

The history-based prioritization collects previous regression testing assets and current changes information of the same system in order to prioritize the new given test suites. Thus, the technique is unsuitable for the first regression testing of software. Kim and Porter [17] used the historical execution data to prioritize test cases for regression tests, while Lin et al. [18] investigated the weight of used information between two versions of history-based prioritization techniques. The results indicated that the history-based prioritization provides a better fault detection rate. Carlson et al. [19] mixed history and coverage-based techniques using a clustering based prioritization technique.

Lachmann et al. [20] introduced a test case prioritization technique for system-level regression testing based on supervised machine learning. The approach considers black-box metadata, such as test cases history, as well as natural language test case descriptions for prioritizing. They used the SVM Rank machine learning algorithms and evaluate their approach on 2 subject systems. The results outperform a test case order given by a test expert.

Spieker et al. [21] proposed the *Retecs* approach, a method for automatically learning test case selection and prioritization in continuous integration with the goal to minimize the round-trip time between code commits and developer feedback on failed test cases. The approach uses reinforcement learning. The Empirical study shows that reinforcement learning enables fruitful automatic adaptive test case selection and prioritization.

The history and machine learning based techniques prioritize test suites in a regression testing context. Some other techniques allow, upstream, the prioritization of components to be tested. They aim to optimize the testing efforts distribution by targeting the most fault prone components. Shihab et al. [22] explored the prioritization for unit testing phase in the context of legacy systems. Our

previous papers [12] proposed machine learning approaches that aim to suggest candidate classes for unit tests. We used 2 classifiers trained on the dataset formed by source code metrics and labelled by tested/not tested, to build classifiers that match the candidate classes for unit tests. After applying cross systems validation techniques, our results indicated that for a given system, the ability of a classifier, to correctly suggest the candidate classes for unit tests (more than 70% of accuracy). Furthermore, we considered more machine learning algorithms and we focused on affinities between the systems used as training and testing datasets during the cross systems validation. We wanted to determine whether some systems make better training sets for suggesting other specific systems unit test candidate classes. The result showed that the datasets of large systems could be only used to suggest large systems unit test candidate classes, while classifiers trained on small systems fail to suggest the candidate classes for unit tests on large and small systems. In the same study [12], we focused on the ability of combined datasets to suggest candidate classes for unit tests. After applying the leave one system out validation technique, the result show that more than 70% of candidate classes selected by testers were well predicted in the case of large size systems.

The current paper investigates deep neural network classifiers trained on combined datasets as predictor models for unit tests candidate classes selection. Combining different systems as a single training dataset presents several advantages such as diversity of observations and their amount. Indeed, our long-term objective is to build a collaborative IDE plugin, based on unit tests information and some specific metrics to support the unit tests prioritization. Hence, the plugin will collect source code metrics and test information from various software systems. Under such conditions, the ability of learning from combined datasets is of great importance. Combining training datasets may, however, lead to metric dimensionality issues. Indeed, from the tester point of view, a class with a given metric value may be considered as a good candidate or not depending on the metric values of the other classes of the system. The following section presents the software metrics we used in our study.

III. SOFTWARE METRICS

This section presents the considered OO source code metrics. We expanded the previous dataset metrics used in [9, 12] by including more source code attributes. The selected metrics are being adopted by practitioners. Several studies have shown that the considered metrics are related to testability [3-8], maintainability [23-26], and fault proneness [27-29]. The set of metrics is related to inheritance, coupling, complexity and size software attributes. We computed them using the Borland Together (http://www.borland.com).

Depth of Inheritance Tree: DIT metric is the maximum inheritance path from the given class to the root class. ***Coupling Between Objects***: The CBO metric counts for a given class, the number of other classes to which it is coupled and vice versa. ***Fan Out***: The FOUT metric counts the number of other classes referenced by a given class. ***Fan IN***: The FIN metric counts the number of other classes that reference to a given class. ***Weighted Methods per Class***: The WMC metric gives the sum of the complexities of the methods of a given class, where each method is weighted by its cyclomatic complexity [27]. Only methods specified in the

class are considered. **Response For Class***:* The RFC metric measures the class's complexity in terms of method invocations. It sums the number of methods defined in a given class and the number of distinct method invocation made by that method. **Lines of Code per Class**: The SLOC metric counts for a given class, its number of source lines of code.

IV. DATA COLLECTION

A. Selected Systems

The source codes of 10 open source OO software systems developed in Java have been extracted from public repositories and described below. For each system, only a subset of classes has been tested using JUnit framework.

IO [1] is a library of utilities for developing input/output functionalities. It is developed by Apache Software Foundation. ***MATH*** [1] is a library of lightweight, self-contained mathematics and statistics components addressing the most common problems not available in the Java programming language. ***JODA*** [2] is the de facto standard library for advanced date and time in Java. It provides a quality replacement for the Java date and time classes. The design supports multiple calendar systems, while still providing a simple API.
DBU [3] (DbUnit) is a JUnit extension (also usable with Ant) used in database-driven projects that, among others, put a database into a known state between test runs. ***LOG4J*** [1] is a fast and flexible framework for logging applications debugging messages. ***JFC*** [4] (JFreeChart) is a free chart library for Java platform. ***IVY*** [1] is an agile dependency manager characterized by flexibility, simplicity and tight integration with Apache Ant. ***LUCENE*** [1] is a high-performance, full-featured text search engine library. It is a suitable technology for applications requiring full-text search. ***ANT*** [1] is a Java library and command-line tool that drives processes described in build files as target and extension points dependent upon each other. ***POI*** [1] is an APIs for manipulating various file formats based upon the Office Open XML standards and Microsoft's OLE2. It can read and write MS Excel files using Java.

B. Unit Test Data Collection Procedure

The selected systems have been tested using the JUnit framework. JUnit [11] is a framework for writing and running automated unit tests for Java classes. JUnit gives testers some support so that they can write the test cases more conveniently. A typical usage of JUnit is to test each class C_s of the software by means of a dedicated test class C_t. To actually test a class C_s, we execute its test class C_t by calling JUnit's test runner tool. JUnit report how many of the test methods in C_t succeeded, and how many failed.

In [12], we used the prefix/suffix linking approach, as other authors [4, 10, 30], to link each software class to its dedicated JUnit test class if exists. Linked classes are referred as E-TESTED classes. Furthermore, we considered, the level of JUnit Coverage (JUC) score computed by Borland Together Tool to take transitively tested classes into account. Indeed, in [8, 9, 12], we noted that some of software classes were tested by transitive method invocations during unit tests.

Table 1: Percent of tested classes

	MATH	IO	JODA	DBU	LOG4J
% I-TESTED	84.04%	81%	76.62%	46.70%	40.26%
% E-TESTED	61.7%	66%	37.81%	40.1%	19.5%
	JFC	IVY	LUCENE	ANT	POI
% I-TESTED	66.26%	61.68%	52.52%	16.89%	67.73%
% E-TESTED	55.50%	15.62%	18.54%	16.89%	28,00%

The JUC score is based on unit test class invocation, representing for each class the percent of software lines of code covered by the set of unit test classes. Classes with a JUC score greater than 0 are referred as I-TESTED classes. Table 1 summarizes the distribution of E-TESTED and I-TESTED classes.

Descriptive Statistics

Table 2 summarizes the statistics of selected metrics for the 10 systems ordered by increasing sizes in terms of the number of classes.

Table 2 Descriptive statistics

Syst	Obs	Stat	FIN	CBO	DIT	LOC	RFC	FOU	WMC
MATH	94	Min.	0	0	1	2	13	0	0
		Max	13	18	6	660	119	12	174
		Sum	275	306	195	7779	3717	194	1824
		μ	2.93	3.26	2.07	82.76	39.54	2.06	19.40
		σ	2.47	3.72	1.11	97.60	18.64	2.46	25.12
IO	100	Min.	0	0	1	7	17	0	1
		Max	14	39	5	968	202	21	250
		Sum	323	405	214	7604	3782	254	1817
		μ	3.23	4.05	2.14	76.04	37.82	2.54	18.17
		σ	4.07	5.70	1.01	121.56	24.79	3.27	31.75

Syst	Obs	Stat	FIN	CBO	DIT	LOC	RFC	FOUT	WMC
JODA	201	Min.	0	0	1	5	11	0	1
		Max	106	36	6	1760	287	22	176
		Sum	2116	1596	447	31339	17857	1089	6269
		μ	10.53	7.94	2.22	155.92	88.84	5.42	31.19
		σ	16.12	6.44	1.28	210.97	64.21	4.78	30.55
DBU	212	Min.	0	0	1	4	11	0	1
		Max	28	24	6	488	95	19	61
		Sum	517	1316	452	12187	6827	901	1989
		μ	2.43	6.18	2.13	57.22	32.05	4.23	9.34
		σ	3.44	5.32	1.22	60.55	14.54	3.94	9.45
LOG4J	231	Min.	0	0	1	5	11	0	1
		Max	72	107	7	1103	632	47	207
		Sum	966	1698	467	20150	15879	1088	3694
		μ	4.18	7.35	2.02	87.23	68.74	4.71	15.99
		σ	9.29	10.12	1.30	130.42	105.75	5.93	25.70

Syst	Obs	Stat	FIN	CBO	DIT	LOC	RFC	FOUT	WMC
JFC	409	Min.	0	0	1	4	11	0	0
		Max	55	101	7	2041	677	56	470
		Sum	2583	4861	967	67481	50628	3253	13428
		μ	6.28	11.83	2.36	164.19	123.18	7.91	32.67
		σ	8.99	14.07	1.40	228.06	148.28	9.43	46.73
IVY	608	Min.	0	0	0	2	1	0	0
		Max	103	92	6	1039	458	46	231
		Sum	2239	5205	1037	50080	35274	3419	9664
		μ	3.68	370.03	1.71	219.60	58.02	5.62	15.84
		σ	7.89	11.74	1.31	141.80	61.67	7.33	27.38
LUCENE	615	Min.	0	0	1	1	11	0	0
		Max	63	55	6	2644	433	46	557
		Sum	2860	3793	1212	56108	23724	2872	10803
		μ	4.65	6.17	1.97	91.23	38.58	4.67	17.57
		σ	7.18	7.24	1.06	192.87	34.61	5.49	35.70
ANT	663	Min.	0	0	1	1	11	0	0
		Max	300	41	6	1252	550	30	245
		Sum	3228	4613	1563	63548	36282	3294	12034
		μ	4.87	6.96	2.36	95.85	54.72	4.97	18.15
		σ	16.87	7.25	1.28	132.92	46.25	5.41	24.17

Syst	Obs	Stat	FIN	CBO	DIT	LOC	RFC	FOUT	WMC
POI	1382	Min.	0	0	1	2	11	0	0
		Max.	189	168	7	1686	642	62	374
		Sum	5733	9660	2899	130185	66574	5924	23810
		μ	4.15	6.99	2.10	94.20	48.17	4.29	17.23
		σ	9.51	10.78	1.24	154.28	58.44	6.27	28.32

1 https://apache.org/

2 http://joda-time.sourceforge.net/

3 http://dbunit.sourceforge.net/

4 http://www.jfree.org/jfreechart/

The number of lines of code varies from 7,779 lines spread over 94 software classes for MATH system, to more than 130,185 lines of code over 1,382 software classes for POI system. The number of classes and their cyclomatic complexity follow the same trend. Following the descriptive statistics, we grouped the systems into 4 categories relatively to their size in order to better interpret the results: (1) the small-size systems, about 100 software classes such as IO and MATH, (2) the medium-size systems around 200 classes such as LOG4J, DBU and JODA, (3) the large-size systems, between 400 and just over 600 classes such as LUCENE, IVY, ANT and JFC, and (4) the very large-size systems over than 1,000 software classes such as POI.

The average cyclomatic complexity varies widely between systems with similar sizes as for JODA and DBU systems. Indeed, these systems present similar number of classes (around 200) but quite a different average of cyclomatic complexity (31.19 vs. 9.34). We made the same observations for LUCENE and JFC systems.

The DIT metric varies from 1 to 7 in all systems when it average is about 2 for the majority of systems. DIT has the lowest variance values compared to other metrics. The minimum average value of DIT is observed for IVY (1.71) and the maximum average value for JFC and ANT (2.36). A very deep inheritance tree may indicate a bad design while shallow inheritance reflects the lack of code reusability.

JODA software has the highest average value of FIN (10.53) while JFC got the highest average value of FOUT (7.91) and RFC (123.18).

V. Empirical Analysis

A. Research question

The machine learning models in [12] have been trained on combined datasets formed by source code metrics and unit tests information of different systems. With 70% of correct classifications, the generated classifiers well suggested the candidate classes for unit tests as long as the targeted systems was large enough. That result has been mainly explained by the probably missing of strategies when testing small software systems. With more source code metrics, our current work test different deep neural network topologies to improve the results we observed previously. The main research question is:

Can deep neural network-based classifier better fit the candidate classes selected by testers for unit testing?

The main goal remains to use metric information in order to support unit tests prioritization decisions. Our research question allows to validate whether a deep neural network model can produce good classifiers that fit the testers selection of candidate classes for unit tests. The empirical study we conducted is based on combined training datasets from which the system under analysis has been excluded, a technique we referred as Leave One System Out Validation.

B. Deep Neural Network

Deep neural network is a family of Artificial Neural Network (ANN) that contains more than one hidden layer. When well trained (Deep learning), it allows computational models that are composed of multiple processing layers to learn representations of data with multiple levels of abstraction. These methods have dramatically improved the state-of-the-art in speech recognition, visual object recognition, object detection and many other domains such as

drug discovery and genomics. Deep learning discovers intricate structure in large data sets by using the backpropagation algorithm to indicate how ANN should change its internal parameters that are used to compute the representation in each layer, from the representation in the previous layer [13].

In deep neural network models, the layers configuration may strongly impact the performances of classifiers. Unfortunately, there is no systematic approach that may determine the right layers topology for a given dataset. Hence, we adopted the try and error strategy to find the suitable architecture for our datasets.

C. Leave One System Out Validation LOSOV

The LOSOV consists of combining the datasets of different S_i systems excluding S_j to form a unique training dataset for the neural network model. The generated classifier is tested on the remaining S_j system. After many tries, following layers topology has been set for the deep neural network model.

The input layer: We managed a dataset that contains 7 properties formed by the selected metrics which lead us to set 7 neurons on the entry layer.

The hidden layers: The hidden layers organization result from multiple tries/error, and the best results was obtained when setting 6 layers of 175 neurons each of them activated with *relu* function. With fewer neurons, the model trends to misclassify the large and the very large systems, while more neurons conduct to overfitting issues. We tried different compressing topologies by gradually reducing the number of cells along the layers, from entry toward the output layer. The results were inconclusive. We also increased/decreased the number of layers and combined them with different epoch numbers but misclassifications and overfitting issues still persisted.

The output layer: The output layer is composed of 2 neurons to match our binary classification problem. The layer uses *softmax* activation function.

We also found, after many tries, that 350 epochs gave the best results. Increasing that number leads to overfitting with totally unbalanced confusion matrix (classifier tends to suggest all software classes or none of them as candidates for unit tests), while reducing it produces misclassifications.

D. Results & Discussion

We considered both the E-TESTED and I-TESTED unit test perspectives. Table 3 summarizes the results we got by generated classifiers with 350 epochs. On each row that represents evaluated system, LOSVO approach validates the classifier obtained from the dataset composed of all remaining systems by testing it on that system. The *accuracy* column indicates the accuracy percentage, while the *conf. matrix* column holds the confusion matrix produced by the classifier.

We immediately remarked that: (1) the candidate classes for unit tests of larger systems are better predicted with better accuracy compared to our previous works, and (2) the I-TESTED point of view leads to better suggestion results in terms of the number of correctly predicted systems. The relationship between systems' size and classifiers' performances is not surprising but follows the trends we previously observed using other classifiers models. The explanation may come from the lack of strategy when testing

small systems. It may also be related to the training dataset scale. The largest system (POI) predication is weak according to E-TESTED point of view (about 63.6%). Removing POI from the combined dataset may unbalance the training dataset and could explain the weakness of the prediction accuracy.

Table 3: LOSVO trained on metric values

	E-TESTED, Value Only			I-TESTED, Value Only		
	Accuracy	Conf. Matrix		Accuracy	Conf. Matrix	
MATH	38.30%	28	8	64.89%	5	10
		50	8		23	56
IO	52.00%	30	4	68.00%	15	4
		44	22		28	53
JODA	73.13%	97	28	78.61%	34	13
		26	50		30	124
DBU	63.21%	115	12	80.66%	84	29
		66	19		12	87
LOG4J	86.15%	175	12	78.79%	106	32
		20	24		17	76
JFC	83.37%	168	14	82.15%	114	24
		53	173		49	222
IVY	88.49%	472	41	85.20%	188	45
		29	66		45	330
LUCENE	80.98%	436	65	84.55%	243	49
		52	62		46	277
ANT	86.12%	497	54	78.73%	422	129
		38	74		12	100
POI	63.6%	725	270	78.22%	431	84
		233	154		217	650

When considering the I-TESTED point of view, the candidate classes for unit tests are better predicted by classifiers. 8 systems over 10 (against 6 over 10 for E-TESTED) have an accuracy greater than 70%. The associated confusion matrices ensure us that the classifiers are not suggesting no class or all classes (at the same time) as candidate classes for unit tests. Indeed, we faced that situation when using shallow neural networks or when we increased the number of training stages epochs during our investigations.

When deepening our investigations and reviewing the descriptive statistics, we understood that some characteristics of class attributes relatively to other classes in the same software system may have an impact on developer decision to select or not that class as a candidate for unit tests. The raw values of source code metrics considered alone are not sufficient for our classifiers to correctly match the tester selections. Indeed, a WMC score of 50 (for example) may be important when a developer tests a system for which the average class complexity (WMC) is much smaller than 50 which lead him to write an explicit test class for that software class. On the other hand, a class with the same complexity score may be considered as little complex by the developer when it belongs to a large system in which average complexity of classes is largely higher than 50. We thus need an attribute that captures the metric values for a class relatively to the other classes within the same system. When combining datasets, that attribute will mitigate it corresponding source code metrics. The ranks of metric scores are good candidates. In the following steps, we tested whether including metric ranks could improve the results of Table 2. We computed the rank of each metric's value of each class inside each system. Ranks have been included to the datasets. All new datasets contain 14 attributes that constrains us to review our neural network topology.

The Input layer: With the new datasets of 14 properties formed by the metrics and their ranks. We set the number of neurons in the entry layer to 14.

The Hidden layers: We set the number of hidden layers to 13. Now, each layer contains 350 neurons. We kept the *relu* as activation function.

The Output layer: The output remains unchanged. Its 2 neurons still match our binary classification problem. They are activated with *softmax* function. The number of epochs has also been doubled to 750 to prevent misclassification.

Table 4 summarizes the results. We immediately remark that all predictions highly improved compared to Table 3. The large systems are suggested with more than 99% of correctness in both perspectives. MATH results with E-TESTED point of view, slightly improved but remains the only system under 70% of correctness. For several systems (JODA, DBU, LOG4J and LUCENE) all tested classes have been found by the classifier without any false positive or false negative classification. We reached 100% of correctness.

Table 4 LOSVO trained on metric and rank values

	E-TESTED, Value + Rank			I-TESTED, Value + Rank		
	Accuracy	Conf. Matrix		Accuracy	Conf. Matrix	
MATH	52.13%	14	22	76.60%	3	12
		23	35		60	69
IO	93.00%	30	4	99.00%	19	0
		3	63		1	80
JODA	100.00%	125	0	99.50%	46	1
		0	76		0	154
DBU	100.00%	127	0%	100.00%	113	0
		0%	85		0	99
LOG4J	100.00%	187	0	99.57%	137	1
		0	44		0	93
JFC	99.76%	181	1	99.51%	138	0
		0	227		2	269
IVY	99.84%	513	0	98.85%	226	7
		1	94		0	375
LUCENE	100.00%	501	0	99.19%	289	3
		0	114		2	321
ANT	99.85%	550	1	99.85%	551	0
		0	112		1	111
POI	99.78%	994	1	99.35%	510	5
		2	385		4	863

The results we obtained in Tables 3 and 4 strongly support our hypothesis. It's possible to build a prediction classifier based on deep neural network and trained on combined datasets composed by different software systems that correctly suggest classes to be tested. "Correctly" means matching the real testers' selection. E-TESTED and I-TESTED points of view have no impact when we included the ranks values in the datasets. Let's recall that our long-term goal was to build an IDE plugin tool that automatically collects source code metrics of systems under development in order to suggest a set of classes to be tested. The plugin's classifiers would be trained from datasets of various systems. Under such conditions, it was important for us to explore in the current work, the suggestion capability of classifiers trained on such a mixed dataset.

VI. THREATS TO VALIDITY

Obtained results are suggestive and the study we presented was performed on 10 open-source systems containing almost a half million lines of code (453K). The sample is large enough to allow obtaining significant results, but the experimental approaches may present limitations that could restrict the generalization of certain conclusions. Indeed, all systems we used are developed using Java language and tested using JUnit framework. Java and JUnit are popular in the developers' community. The obtained results may not be generalizable to other unit testing frameworks or programming languages. More investigations are required to rule on the issue. Furthermore, it would be

interesting to know, in such a condition, whether mixing dataset from systems built using different languages and unit framework could improve or degrade the results. The neural network topology we identified matches very well the analyzed group of systems. Changing the number of systems and their categories may degrade obtained results. Replicating the study on more systems could help to draw more general neural network topology that fits unit test decision support.

VII. CONCLUSIONS AND FUTURE WORK

Ten open source software systems have been analyzed in this study which totals more than 4400 classes. The testers of each system developed dedicated unit test classes for a subset of classes using the Junit Framework. We explored the possibility of deep neural network models to correctly match developers' selections of the candidate classes for unit tests. To achieve our investigations, we considered explicitly and implicitly tested classes. With the combination of the 10 datasets formed by the considered systems, we tested various deep neural network topologies that we validated using Leave One System Out Validation technique. The objective was to know to what extents the combined information of different systems could be a usable training dataset for deep neural network-based classifiers. Results show that it was possible to correctly match the candidate classes for unit tests proposed by testers. Furthermore, the results indicated that all systems could be well predicted with more than 93% of accuracy. These results are particularly interesting since the long-term goal of our work is to build a collaborative plugin tool that suggests the set of the candidate classes for unit tests by learning from different systems information history. Our next challenge will be to validate this approach using different unit testing frameworks under different programming language before developing the plugin tool.

REFERENCES

1. Chidamber S.R. and Kemerer C.F., 1994. A Metrics Suite for Object Oriented Design, IEEE Transactions on Software Engineering, vol. 20, no. 6, pp. 476–493.

2. Henderson-Sellers B. 1996. Object-Oriented Metrics Measures of Complexity, Prentice-Hall, Upper Saddle River.

3. Bruntink M. and Van Deursen A. 2006. An Empirical Study into Class Testability, Journal of Systems and Software, Vol. 79, No. 9, pp. 1219-1232.

4. Badri L., Badri M. and Toure F., 2010. Exploring Empirically the Relationship between Lack of Cohesion and Testability in Object-Oriented Systems, JSEA Eds., Advances in Software Engineering, Communications in Computer and Information Science, Vol. 117, Springer, Berlin.

5. Badri M. and Toure F., 2011. Empirical analysis for investigating the effect of control flow dependencies on testability of classes, in Proceedings of the 23rd International Conference on Software Engineering and Knowledge Engineering SEKE.

6. Badri M. and Toure F. 2012. Empirical analysis of object oriented design metrics for predicting unit testing effort of classes, Journal of Software Engineering and Applications (JSEA), Vol. 5 No. 7, pp.513-526.

7. Toure F., Badri M. and Lamontagne L., 2014. Towards a metric suite for JUnit Test Cases. In Proceedings of the 26th International Conference on Software Engineering and Knowledge Engineering (SEKE Vancouver, Canada. Knowledge Systems Institute Graduate School, USA pp 115–120.

8. Toure F., Badri M. and Lamontagne L., 2014. A metrics suite for JUnit test code: a multiple case study on open source software, Journal of Software Engineering Research and Development, Springer, 2:14.

9. Toure F., Badri M. and Lamontagne L., 2017. Investigating the Prioritization of Unit Testing Effort Using Software Metrics, In Proceedings of the 12th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE'17) Volume 1: ENASE, pages 69-80.

10. Bruntink M., and Deursen A.V., 2004. Predicting Class Testability using Object-Oriented Metrics, 4th Int. Workshop on Source Code Analysis and Manipulation (SCAM), IEEE.

11. JUnit Framework, https://junit.org/junit5/. Visited in December 2019.

12. Toure F., and Badri M., 2018. Prioritizing Unit Testing Effort Using Software Metrics and Machine Learning Classifiers, In Proceedings of the 30th International Conference on Software Engineering and Knowledge Engineering, SEKE 2018 DOI:10.18293/SEKE2018-146

13. LeCun Y, Bengio Y, and Hinton G., 2015. Deep learning. Nature. 2015, 521(7553):436-444. doi:10.1038/nature14539.

14. Rothermel G., Untch R.H., Chu C. and Harrold M.J., 1999. Test case prioritization: an empirical study, International Conference on Software Maintenance, Oxford, UK, pp. 179–188.

15. Yu Y. T. and Lau M. F., 2012. Fault-based test suite prioritization for specification-based testing, Information and Software Technology Volume 54, Issue 2, Pages 179–202.

16. Mirarab S. and Tahvildari L., 2007. A prioritization approach for software test cases on Bayesian networks, In FASE, LNCS 4422-0276, pages 276–290.

17. Kim J. and Porter A., 2002. A history-based test prioritization technique for regression testing in resource constrained environments, In Proceedings of the International Conference on Software Engineering.

18. Lin C.T., Chen C.D., Tsai C.S. and Kapfhammer G. M., 2013. History-based Test Case Prioritization with Software Version Awareness, 18th International Conference on Engineering of Complex Computer Systems.

19. Carlson R., Do H., and Denton A., 2011. A clustering approach to improving test case prioritization: An industrial case study, Software Maintenance, 27th IEEE International Conference, ICSM, pp. 382-391.

20. Lachmann R., Schulze S., Nieke M., Seidl C. and Schaefer I., 2016 System-Level Test Case Prioritization Using Machine Learning, 2016 15th IEEE International Conference on Machine Learning and Applications (ICMLA), Anaheim, CA, 2016, pp. 361-368.

21. Spieker H., Gotlieb A., Marijan D. and Mossige M., Reinforcement learning for automatic test case prioritization and selection in continuous integration, Proceedings of the 26th ACM SIGSOFT International Symposium on Software Testing and Analysis, July 2017.

22. Shihaby E., Jiangy Z. M., Adamsy B., Ahmed E. Hassany A. and Bowermanx R., 2010. Prioritizing the Creation of Unit Tests in Legacy Software Systems, Softw. Pract. Exper., 00:1–22.

23. Li W., and Henry S., 1993. Object-Oriented Metrics that Predict Maintainability Journal of Systems and Software, vol. 23 no. 2 pp. 111-122.

24. Dagpinar M., and Jahnke J., 2003. Predicting maintainability with object-oriented metrics – an empirical comparison, Proceedings of the 10th Working Conference on Reverse Engineering (WCRE), IEEE Computer Society, pp. 155–164.

25. Zhou Y., and Leung H., 2007. Predicting object-oriented software maintainability using multivariate adaptive regression splines, Journal of Systems and Software, Volume 80, Issue 8, August 2007, Pages 1349-1361, ISSN 0164-1212.

26. Basili V.R., Briand L.C. and Melo W.L., 1996. A Validation of Object-Oriented Design Metrics as Quality Indicators, IEEE Transactions on Software Engineering. vol. 22, no. 10, pp. 751-761.

27. Aggarwal K.K., Singh Y., Kaur A., and Malhotra R., 2009. Empirical Analysis for Investigating the Effect of Object-Oriented Metrics on Fault Proneness: A Replicated Case Study, Software Process Improvement and Practice, vol. 14, no. 1, pp. 39-62.

28. Shatnawi R., 2010. A Quantitative Investigation of the Acceptable Risk Levels of Object-Oriented Metrics in Open-Source Systems, IEEE Transactions On Software Engineering, Vol. 36, No. 2.

29. Zhou Y. and Leung H., 2006. Empirical Analysis of Object-Oriented Design Metrics for Predicting High and Low Severity Faults, IEEE Transaction Software Engineering, vol. 32, no. 10, pp. 771-789.

30. Mockus A., Nagappan N. and Dinh-Trong T. T., 2009. Test coverage and post-verification defects: a multiple case study, in proceedings of the 3rd International Symposium on Empirical Software Engineering and Measurement (ESEM), pp. 291– 301.

An Empirical Investigation on the Relationship Between Bug Severity and Bug Fixing Change Complexity

Zengyang Li[1], Dengwei Li[1], Peng Liang[2,*], Ran Mo[1]

[1] School of Computer Science & Hubei Provincial Key Laboratory of Artificial Intelligence and Smart Learning,
Central China Normal University, Wuhan, China

[2] School of Computer Science, Wuhan University, Wuhan, China

zengyangli@mail.ccnu.edu.cn, 762396001@qq.com, liangp@whu.edu.cn, moran@mail.ccnu.edu.cn

Abstract —**Fixing bugs requires changing source code in most cases. The complexity of code changes for fixing bugs has an important impact on release planning. This work intends to investigate whether there are significant differences between bugs with different severity levels with respect to the complexity of code changes for fixing the bugs. We performed a case study on 13 Apache open source software (OSS) projects using commit records and bug reports. The study results show that (1) for bugs of high severity levels, there is no significant difference on the complexity of code change for fixing bugs of different severity levels for most projects, while (2) for bugs of low severity levels, fixing bugs of a higher severity level needs significantly more complex code change than fixing bugs of a lower severity level for most projects. These findings provide useful insights for effort estimation and release planning of OSS development.**

Keywords-bug severity; code change complexity; commit records

I. INTRODUCTION

Bugs of a software project are managed in an issue tracking system, in which the severity of a bug can be indicated by the development team members or external reporters. In practice, developers use severity levels, such as Blocker, Critical, Major, Minor, and Trivial in JIRA, to prioritize the urgency of bugs and to estimate influence and impact of bugs [1]. The bug severity data play an important role in release planning and task assignment [2, 3].

The complexity of required code change for fixing a bug also influences release planning in terms of effort estimation [3]. More complex code change is required for fixing a bug means more effort is needed for this bug fixing task. Effort estimation for tasks is a key aspect in release planning [3].

Both the severity of a bug and required effort for fixing the bug should be taken into consideration during release planning of a project. A natural question arises: *is bug severity in line with actual code change complexity*? The answer to this question will provide meaningful insights for effort estimation of project development.

To investigate whether bug severity is in line with complexity of actual code change, we performed a case study on 13 non-trivial Apache open source software (OSS) projects. Data on bugs were exported from JIRA – an issue tracking system deployed by Apache Software Foundation. Data on the complexity of code change for fixing bugs can be obtained by analyzing the commit records extracted from the code repositories of the OSS projects.

Our main findings are as follows: (1) There is no significant difference on the complexity of code change for fixing Blocker and Critical bugs for most projects. The situation is similar for Critical and Major bugs. (2) Code change for fixing Major bugs has a significantly higher complexity than fixing Minor bugs for most selected projects. The situation is similar for Minor and Trivial bugs.

The rest of this paper is organized as follows. Section II reports related work to this study. Section III describes the design of the case study. Section IV presents the results of the case study. Section V discusses the results of the case study and Section VI presents the threats to validity of the results. Section VI concludes this work with future directions.

II. BACKGROUND AND RELATED WORK

This section presents background of this study and related work on bug severity and required code change for fixing bugs.

A. Background

In the JIRA issue tracking platform, issues are classified in multiple types, including Bug, Improvement, New Feature, Task, Sub-task, Test, and Wish. In particular, according to its severity (i.e., priority to be fixed), a bug can be labeled a level, from high to low severity, as Blocker, Critical, Major, Minor, or Trivial. These severity levels are defined as follows, according to JIRA [4].

- **Level 5 – Blocker**: a time-sensitive issue that is hindering a basic function of a project.
- **Level 4 – Critical**: a time-sensitive issue that is disrupting the project, but does not hinder basic functions.
- **Level 3 – Major**: this issue needs attention soon, but is not hindering basic functions. Most requests for new resources fall into this category.

- **Level 2 – Minor**: this issue needs attention, but is not time-sensitive and does not hinder basic functions.
- **Level 1 – Trivial**: this issue is minimal and has no time constraints.

Besides, in JIRA, the status of a bug can be one of the following: Open, In Progress, Reopen, Resolved, and Closed.

B. Related Work

Many studies proposed various methods to predict bug severity automatically. For instance, Roy *et al.* used text mining and machine learning techniques to improve bug severity classification [5]. Lamkanfi *et al.* applied text mining algorithms to analyze descriptions of bug reports for predicting bug severity [6]. Menzies *et al.* proposed to use standard text mining and machine learning techniques to automate severity assessment based on software defect reports [7]. Tian *et al.* used multi-factor analysis to automatically predict bug priority [8]. However, our work is not aimed to predict severity levels of bugs, but to investigate whether bugs of a higher severity level require more complex code change to fix.

A number of studies looked into the delay of bug fixing from the perspective of bug severity and required code change. Zhang *et al.* found that a larger total lines of changed code can delay bug fixing, and bugs of a high severity level were fixed earlier than bugs of a low severity level [9]. Saha *et al.* revealed that bug priority (i.e., severity) has significant impact on the delay of bug fixing [10]. However, the relationship between bug severity and complexity of changed code was not discussed in these study.

Some works on effort estimation took bug severity into consideration. For instance, Weiss *et al.* took the average time and effort of previous bugs with similar severity as an early estimation of required effort for new bugs [11].

III. STUDY DESIGN

In order to investigate the relationship between bug severity and code change complexity, we performed a case study on fifteen Apache OSS projects written in Java. In this section we describe the case study, which was designed and reported according to the guidelines proposed by Runeson and Höst [12].

A. Objective and Research Questions

The goal of this case study is to investigate: whether there is a significant difference on the complexity of changed source code for fixing the bugs with different bug severity levels.

In this study, the complexity of changed source code is measured in three dimensions: (i) number of modified lines of code, (ii) number of modified source files, and (iii) number of modified packages. It is convenient to extract such information on a bug by analyzing commit records and bug reports.

Based on the abovementioned goal and considering the three dimensions of the complexity of changed source code, we formulated the following three research questions (RQs):

- **RQ1**: Is there a significant difference between the numbers of **lines of modified code** for fixing bugs with different severity levels?

- **RQ2**: Is there a significant difference between the numbers of **modified source files** for fixing bugs with different severity levels?
- **RQ3**: Is there a significant difference between the numbers of **modified packages** for fixing bugs with different severity levels?

B. Case and Unit Analysis

According to [12], case studies can be characterized based on the way they define their cases and units of analysis. This study investigates multiple OSS projects, i.e., cases, and each bug and changed source code for fixing it is a single unit of analysis.

C. Case Selection

In this study, we only investigated Apache OSS projects written in Java. For selecting each case (OSS project) included in our study, we apply the following criteria:

(1) Over 70% of the source code of the project is written in Java.
(2) The age of the project is over 5 years.
(3) The number of stars of the project on GitHub is over 500.
(4) The number of revisions of code repository of the project is over 2,000.
(5) The number of bugs of the project is over 1,500.

These selection criteria were set to ensure that the selected cases are non-trivial and the resulting dataset is big enough to be statistically analyzed.

D. Data Collection

This section presents the data to be collected and the process for collecting required data.

1) Data to be Collected

To answer the RQs formulated in Section III-A, we collected the data items listed in TABLE I, which also provides the mapping between the data items and the target RQ(s).

TABLE I.   DATA ITEMS TO BE COLLECTED

#	Data Item	Description	Target RQ
D1	Severity label of a bug	The priority of an issue in JIRA (the issue tracking system used by Apache), i.e., Blocker, Critical, Major, Minor, or Trivial.	RQ1, RQ2, RQ3
D2	Number of lines of modified code for fixing a bug	The number of lines of source code that is changed to fix a bug.	RQ1
D3	Number of modified source files for fixing a bug	The number of source files that is changed to fix a bug.	RQ2
D4	Number of modified packages for fixing a bug	The number of packages (for Java) that is changed to fix a bug.	RQ3

TABLE II. DEMOGRAPHIC INFORMATION OF SELECTED APACHE OSS PROJECTS

#	Name	Age(year)	Java%	#(Star)	#(Revision)	#(Committer)	#(Bug in JIRA)
P1	Accumulo	9	98.6	782	10,431	113	2,250
P2	Activemq	15	95.9	1,694	10,519	97	4,771
P3	Camel	13	98.6	3,129	43,282	626	4,729
P4	CXF	12	98.9	632	15,524	151	5,114
P5	Flink	10	76.7	12,254	20,738	573	5,797
P6	Hadoop	11	92.7	10,177	23,606	291	23,373
P7	Ignite	6	72.2	3,034	26,624	241	5,575
P8	Maven	17	99.5	2,041	10,639	89	3,230
P9	Nifi	6	86.6	1,930	5,675	296	3,136
P10	Pig	13	93.1	597	3,691	28	3,099
P11	Struts	14	91.2	990	5,836	72	2,894
P12	Wicket	16	88.2	505	20,796	89	4,066
P13	Zookeeper	13	73.7	7,730	2,102	94	1,910

TABLE III. AVERAGE NUMBER OF MODIFIED LINES OF CODE, SOURCE FILES, AND PACKAGES PER BUG

Project	#(LOC)/Bug					#(File)/Bug					#(Package)/Bug				
	Blocker	Critical	Major	Minor	Trivial	Blocker	Critical	Major	Minor	Trivial	Blocker	Critical	Major	Minor	Trivial
P1	368.46	229.20	492.77	211.97	333.62	7.67	5.80	8.85	3.87	7.15	4.38	4.38	4.23	2.72	3.76
P2	398.41	178.21	199.16	109.92	49.39	6.03	3.74	4.02	2.35	1.79	2.97	2.40	2.56	1.81	1.45
P3	19.50	202.36	129.65	100.84	62.16	2.17	5.73	4.20	3.54	3.51	1.67	3.31	2.64	2.34	2.43
P4	185.07	85.05	90.98	85.20	124.47	4.37	3.39	3.48	3.59	3.97	3.15	2.39	2.47	2.53	2.67
P5	255.21	187.04	157.21	90.18	34.66	5.16	4.53	3.93	3.03	3.36	3.30	2.70	2.67	2.01	2.50
P6	145.68	122.66	106.65	49.24	23.99	4.44	3.58	3.14	2.50	1.79	3.15	2.66	2.27	1.86	1.50
P7	172.96	282.02	215.47	150.55	29.08	4.64	6.30	5.29	8.97	1.56	3.27	4.07	3.37	5.46	1.44
P8	197.69	97.94	97.08	86.76	42.75	4.14	3.06	3.03	3.48	1.83	3.00	2.48	2.25	2.43	1.67
P9	157.87	240.55	202.16	79.88	7.53	3.80	3.94	4.17	3.31	3.67	2.79	2.53	2.62	2.29	1.64
P10	78.25	53.71	152.08	79.70	30.29	2.75	2.12	3.98	2.52	2.00	2.25	1.86	2.40	1.91	1.32
P11	70.16	266.16	136.43	67.52	175.45	2.61	7.98	2.96	2.74	3.30	2.19	4.14	2.28	2.11	2.45
P12	64.36	84.70	97.46	68.39	27.10	2.27	2.88	2.87	2.59	1.39	1.91	2.10	2.14	1.85	1.24
P13	176.43	185.34	113.30	47.33	16.30	4.12	4.44	2.84	2.15	1.15	2.39	2.29	1.83	1.57	1.15

2) Data Collection Process

The process of collecting the data items (listed in TABLE I) for an Apache OSS project includes the following four steps.

Step 1: Export commit records. Commit records of the project were extracted from its Git repository. We developed a simple tool to extract commit records and save them in a text file.

Step 2: Export issues from JIRA. Many Apache OSS projects adopt JIRA (https://issues.apache.org/jira) as their issue tracking system. We manually exported all issues of the project and stored them in a Microsoft Access file. Please note that not all exported issues are bugs and we can get bugs by choosing the issue type 'Bug'.

Step 3: Parse commit records. If a commit is performed to solve an issue, the committer would explicitly tell the issue ID in the message of the commit record. The changed source files and the changed lines of code can also be identified in the commit record.

Step 4: Extract bugs and corresponding number of lines of modified code, number of modified source files, and number of modified packages. With issue IDs obtained in **Step 3**, we picked up bugs, i.e., issues with issue type 'Bug'. Then, we calculated the number of lines of modified code, number of modified source files, and number of modified packages for each bug. Please note that, only resolved or closed bugs were included in our dataset. We found that some bugs were resolved in previous revisions but still with the status OPEN. Such bugs actually are in the REOPEN status, which means that these bugs had not been resolved completely and they may involve more lines of source code, source files, and packages.

In this case study, we filtered out abnormal data points. By an abnormal data point, we mean that a bug whose fixing involves either more than 500 modified source files or over 20,000 lines of modified source code. The abnormal data points can affect the validity of conclusions. For instance, in project Accumulo, we found a few bugs each involving hundreds of thousands of modified lines of source code which is generated

automatically using the model-driven engineering techniques [13]; if such bugs were not excluded, the average number of modified lines of source code for bugs would increase greatly.

E. Data Analysis

To answer the RQs formulated in Section III-A, we need to analyze the collected data on code change history and bug severity. First, we calculated the average number of modified lines of code, modified source files, and modified packages, for each category of bugs (classified according to bug severity). Second, in order to know whether there are significant differences between categories of bugs with respect to the number of modified lines of code, modified source files, and modified packages, we performed Mann-Whitney U tests on the data for each selected OSS project. The test is significant at p-value < 0.05, which means that the tested groups have significant difference.

IV. RESULTS

Following the study design, we performed the case study. In this section, we first present the demographic information of the selected cases, i.e., Apache OSS projects. Then, we report the results regarding the research questions formulated in Section III-A.

TABLE IV.   DISTRIBUTION OF BUGS OVER DIFFERENT SEVERITY LEVELS

Project	#(Blocker)	#(Critical)	#(Major)	#(Minor)	#(Trival)	Total
P1	136	90	587	232	124	1,169
P2	34	121	1410	290	38	1,893
P3	6	59	2198	927	49	3,239
P4	46	98	1951	426	30	2,551
P5	325	299	856	249	50	1,779
P6	605	801	4174	1115	283	6,978
P7	137	286	1193	112	25	1,753
P8	58	48	569	84	12	771
P9	102	139	835	260	55	1,391
P10	12	42	1114	127	34	1,329
P11	31	49	432	176	20	708
P12	11	50	1323	475	82	1,941
P13	102	87	232	84	20	525

A. Selected Cases

Thirteen Apache OSS projects were selected for this case study and their demographic information is shown in TABLE II. The age of the projects is from 6 to 17 years, and 9 out of 13 projects are 10+ years old. All these projects are mainly written in Java, and more than 90% source code of 8 (out of 13) projects are written in Java. Eight out of 13 projects are starred over 1,000 times, and project Flink with 12,254 stars is most starred. Nine out of 13 projects have 10,000+ revisions, demonstrating the vitality of these projects. Each of the selected projects has experienced 2,000+ bugs, and the project Hadoop has the most bugs (23,373).

The average number of modified lines of code, source files, and packages over bugs with different severity levels are presented in TABLE III. The distribution of bugs over different severity levels for the 13 selected OSS project is shown in TABLE IV. Please note that, the total number of bugs in TABLE IV for each project is smaller than the number of bugs in JIRA (as shown in TABLE II). This is because many bugs in JIRA are not recorded in the commit messages of the master branch of the project's code repository, which is also the reason why we selected projects with a relatively large number of bugs (i.e., over 1500 bugs, described in Section III-C).

B. Results on Modified Lines of Code (RQ1)

We performed Mann-Whitney U tests to understand whether there is a significant difference between bugs with distinct severity levels with respect to the number of modified lines of code. Results of the tests are shown in TABLE V, where cells with p-value < 0.05 are filled in gray. Specifically, a cell filled in gray and with a number in bold denotes that the average number of modified lines of code for higher level bugs is significantly smaller than low level bugs; the remaining cells filled in gray mean that the average number of modified lines of code for higher level bugs is significantly larger than low level bugs.

(1) 3 out of 13 (23.1%) projects have a significant difference (p-value < 0.05) between Blocker and Critical bugs, and only in one project (i.e., P5) Blocker bugs have a higher average number of modified lines of code than Critical bugs.

(2) 5 out of 13 (38.5%) projects have a significant difference between Critical and Major bugs, and in 4 out of 13 (30.8%) projects Critical bugs have a higher average number of modified lines of code than Major bugs.

(3) **12 out of 13 (92.3%) projects have a significant difference between Major and Minor bugs, and in all the 12 projects Major bugs have a higher average number of modified lines of code than Minor bugs**.

(4) 10 out of 13 (76.9%) projects have a significant difference between Minor and Trivial bugs, and in 9 out of 13 (69.2%) projects Minor bugs have a higher average number of modified lines of code than Trivial bugs.

TABLE V.   P-VALUES OF MANN-WHITNEY U TESTS FOR MODIFIED LINES OF CODE BETWEEN BUGS WITH DIFFERENT SEVERITY LEVELS

Project	Blocker& Critical	Critical& Major	Major& Minor	Minor& Trivial
P1	0.113	**0.006**	0.010	**0.002**
P2	0.156	0.780	<0.001	0.002
P3	**0.016**	0.005	<0.001	<0.001
P4	0.406	0.801	<0.001	0.143
P5	<0.001	0.957	<0.001	0.001
P6	0.655	<0.001	<0.001	<0.001
P7	**0.027**	<0.001	<0.001	0.003
P8	0.052	0.162	0.722	0.249
P9	0.724	0.849	<0.001	<0.001
P10	0.617	0.315	0.005	0.001
P11	0.988	0.428	0.008	0.122
P12	0.807	0.224	0.001	<0.001
P13	0.671	<0.001	0.001	0.001

C. Results on Modified Source Files (RQ2)

We performed Mann-Whitney U tests to understand whether there is a significant difference between bugs with

distinct severity levels with respect to the number of modified source files. Results of the tests are shown in TABLE VI, where cells with *p-value* < 0.05 are filled in gray.

(1) 1 out of 13 (7.7%) project (i.e., P5) has a significant difference between Blocker and Critical bugs, and in this project Blocker bugs have a higher average number of modified files than Critical bugs.

(2) 5 out of 13 (38.5%) projects have a significant difference between Critical and Major bugs, and in 4 out of 13 (30.8%) projects Critical bugs have a higher average number of modified files than Major bugs.

(3) **11 out of 13 (84.6%) projects have a significant difference between Major and Minor bugs, and in 9 out of 13 (69.2%) projects Major bugs have a higher average number of modified files than Minor bugs**.

(4) 8 out of 13 (61.5%) projects have a significant difference between Minor and Trivial bugs, in 7 out of 13 (53.8%) projects Minor bugs have a higher average number of modified files than Trivial bugs.

TABLE VI.   *P-VALUES* OF MANN-WHITNEY U TESTS FOR MODIFIED FILES BETWEEN BUGS WITH DIFFERENT SEVERITY LEVELS

Project	Blocker& Critical	Critical& Major	Major& Minor	Minor& Trivial
P1	0.263	0.220	<0.001	0.587
P2	0.096	0.782	<0.001	0.008
P3	0.097	0.007	<0.001	0.009
P4	0.330	0.701	**0.005**	0.056
P5	<0.001	0.111	<0.001	0.954
P6	0.107	<0.001	<0.001	<0.001
P7	0.125	**<0.001**	**0.001**	0.003
P8	0.117	0.982	0.944	0.241
P9	0.865	0.995	0.001	**<0.001**
P10	0.676	0.016	<0.001	0.050
P11	0.898	0.365	0.131	0.315
P12	0.424	0.181	<0.001	<0.001
P13	0.765	0.001	0.011	0.003

D. Results on Modified Packages (RQ3)

We performed Mann-Whitney U tests to understand whether there is a significant difference between bugs with distinct severity levels with respect to the number of modified packages. Results of the tests are shown in TABLE VII, where cells with *p-value* < 0.05 are filled in gray.

(1) 1 out of 13 (7.7%) project (i.e., P5) has a significant difference between Blocker and Critical bugs, and in this project Blocker bugs have a higher average number of modified packages than Critical bugs.

(2) 6 out of 13 (46.2%) projects have a significant difference between Critical and Major bugs, and in 5 out of 13 (38.5%) projects Critical bugs have a higher average number of modified packages than Major bugs.

(3) **11 out of 13 (84.6%) projects have a significant difference between Major and Minor bugs, and in 10 out of 13 (76.9%) projects Major bugs have a**

higher average number of modified packages than Minor bugs.

(4) 9 out of 13 (69.2%) projects have a significant difference between Minor and Trivial bugs, in 7 out of 13 (53.8%) projects Minor bugs have a higher average number of modified packages than Trivial bugs.

TABLE VII.   *P-VALUES* OF MANN-WHITNEY U TESTS FOR MODIFIED PACKAGES BETWEEN BUGS WITH DIFFERENT SEVERITY LEVELS

Project	Blocker& Critical	Critical& Major	Major& Minor	Minor& Trivial
P1	0.191	0.031	<0.001	0.388
P2	0.175	0.294	<0.001	0.006
P3	0.112	0.004	0.012	**0.041**
P4	0.275	0.419	0.114	**0.024**
P5	<0.001	0.084	<0.001	0.976
P6	0.191	<0.001	<0.001	<0.001
P7	0.124	<0.001	**0.019**	0.002
P8	0.215	0.408	0.762	0.479
P9	0.692	0.550	<0.001	<0.001
P10	0.569	**0.070**	0.001	0.003
P11	0.921	0.537	0.042	0.221
P12	0.611	0.268	<0.001	<0.001
P13	0.728	<0.001	0.010	0.026

E. Summary

According to the results presented above, there is no significant difference on the complexity of code change for fixing Blocker and Critical bugs in terms of the average number of modified lines of code, source files, and packages for most selected projects. The situation is similar for Critical and Major bugs.

Code change for fixing Major bugs has a significantly higher complexity than fixing Minor bugs in terms of the average number of modified lines of code, source files, and packages for most selected projects. The situation is similar for Minor and Trivial bugs.

V.   DISCUSSION

A. Interpretation of Study Results

The results have shown that there is no significant difference on the complexity of code change for fixing Blocker and Critical bugs in most selected OSS projects. As defined in Section II-A, both Blocker and Critical bugs are time-sensitive issues and of a high level of urgency, and they have a serious impact on the projects. Thus, Blocker and Critical bugs may have a similar level of change impact when fixing the bugs.

The results have also shown that code change for fixing Major bugs has a significantly higher complexity than Minor bugs in most selected OSS projects. There is a relatively clear boundary between bugs of these two severity levels. The code change impact on the software system for fixing Major bugs is significantly higher than Minor bugs. The complexity of code change for fixing Minor and Trivial bugs can be interpreted in a similar way to Major and Minor bugs.

It is consistent in general for the results of the Mann-Whitney U tests on the significant difference between code

change complexity in terms of modified lines of code, source files, and packages, for fixing bugs with different severity levels.

B. Implications

There is no significant difference of average number of modified lines of code, source files, and packages between Blocker and Critical bugs for most selected OSS projects. This implies that Blocker and Critical bugs may have similar change impact and difficulty when fixing them. Hence, when estimating required effort for fixing Blocker and Critical bugs, they can be put in the same category.

Major bugs need to modify a significantly higher average number of lines of code, source files, and packages than Minor bugs for most selected OSS projects. Also, fixing Minor bugs involves code change of higher complexity than Trivial bugs. Hence, for Major, Minor, and Trivial bugs, their severity levels are in line with the complexity of code change for fixing such bugs. Therefore, when estimating needed efforts for fixing Major, Minor, and Trivial bugs, they should be placed in different categories.

VI. THREATS TO VALIDITY

There are several threats to the validity of the study results. We discuss these threats according to the guidelines in [12]. Please note that internal validity is not discussed, since we do not study causal relationships.

Construct validity. Since a bug is closed or resolved, its severity level (i.e., bug priority in JIRA) was confirmed by the development team member of the project. Thus, we believe that the severity levels of closed or resolved bugs can genuinely reflect the actual severity of the bug. In the data collection process, only the changed code written in Java were included. In some cases, the changed code for fixing a bug may involve source files in other programming languages than Java, which threatens the construct validity. To reduce this threat, we filtered out any bug whose fixing entails non-Java source code.

External validity. Since we collected bugs whose fixing requires changing Java code only, the conclusions of this case study may not be generalized to projects not written in Java. Only 13 projects were used in this case study, more projects are needed to establish more solid conclusions.

Conclusion validity. Only descriptive statistics was used in the calculation of the average number of modified lines of code, source files, and packages. The Mann-Whitney U tests were executed in SPSS, which is a widely-used and well-engineered statistical tool. Thus, we believe that the threats to conclusion validity are minimal.

VII. CONCLUSIONS

This work investigates whether there are significant differences between bugs of different severity levels with respect to the complexity of code changes for fixing the bugs. We conducted a case study on 13 Apache OSS projects. Based on the study results, we obtain the following findings:

- In most (>=10/13, 76.9%) projects Blocker (Level 5) and Critical (Level 4) bugs have no significant difference on complexity of code change.

- In most (>=7/13, 53.8%) projects Critical (Level 4) and Major (Level 3) bugs have no significant difference on complexity of code change.
- In most (>=10/13, 76.9%) projects Major (Level 3) bugs have a significantly higher complexity of code change than Minor (Level 2) bugs.
- In most (>=8/13, 61.5%) projects Minor (Level 2) bugs have a significantly higher complexity of code change than Trivial (Level 1) bugs.

Based on the findings of this work, in the next step, we plan to include more software projects (from both commercial and open source) to replicate the case study in this work, in order to establish a more solid foundation for the findings in this work.

REFERENCES

[1] Y. Tian, N. Ali, D. Lo, and A. E. Hassan, "On the unreliability of bug severity data," *Empirical Software Engineering,* vol. 21, no. 6, pp. 2298-2323, 2016.

[2] Y. Tian, D. Lo, and C. Sun, "Information retrieval based nearest neighbor classification for fine-grained bug severity prediction," in *Proceedings of the 19th Working Conference on Reverse Engineering (WCRE'12)*, 2012, pp. 215-224.

[3] M. Svahnberg, T. Gorschek, R. Feldt, R. Torkar, S. B. Saleem, and M. U. Shafique, "A systematic review on strategic release planning models," *Information and Software Technology,* vol. 52, no. 3, pp. 237-248, 2010.

[4] Apache Software Foundation. *Guidelines for creating a Jira ticket.* Available: https://infra.apache.org/pages/jira-guidelines.html, accessed on Dec. 20, 2019.

[5] N. K. S. Roy and B. Rossi, "Towards an improvement of bug severity classification," in *Proceedings of the 40th EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA'14)*, 2014, pp. 269-276.

[6] A. Lamkanfi, S. Demeyer, E. Giger, and B. Goethals, "Predicting the severity of a reported bug," in *Proceedings of the 7th IEEE Working Conference on Mining Software Repositories (MSR'10)*, 2010, pp. 1-10.

[7] T. Menzies and A. Marcus, "Automated severity assessment of software defect reports," in *Proceedings of the 24th IEEE International Conference on Software Maintenance (ICSM'08)*, 2008, pp. 346-355.

[8] Y. Tian, D. Lo, X. Xia, and C. Sun, "Automated prediction of bug report priority using multi-factor analysis," *Empirical Software Engineering,* vol. 20, no. 5, pp. 1354-1383, 2015.

[9] F. Zhang, F. Khomh, Y. Zou, and A. E. Hassan, "An empirical study on factors impacting bug fixing time," in *Proceedings of the 19th Working Conference on Reverse Engineering (WCRE'12)*, 2012, pp. 225-234.

[10] R. K. Saha, S. Khurshid, and D. E. Perry, "An empirical study of long lived bugs," in *Proceedings of the 2014 Software Evolution Week - IEEE Conference on Software Maintenance, Reengineering, and Reverse Engineering (CSMR-WCRE'14)*, 2014, pp. 144-153.

[11] C. Weiss, R. Premraj, T. Zimmermann, and A. Zeller, "How long will it take to fix this bug?," in *Proceedings of the 4th International Workshop on Mining Software Repositories (MSR'07)*, 2007, pp. 1-8.

[12] P. Runeson and M. Höst, "Guidelines for conducting and reporting case study research in software engineering," *Empirical Software Engineering,* vol. 14, no. 2, pp. 131-164, 2009.

[13] X. He, P. Avgeriou, P. Liang, and Z. Li, "Technical debt in MDE: A case study on GMF/EMF-based projects," in *Proceedings of the ACM/IEEE 19th International Conference on Model Driven Engineering Languages and Systems (MODELS'16)*, 2016, pp. 162-172.

Can Language Help in the Characterization of User Behavior?
Feature Engineering Experiments with Word2Vec

Eduardo Lopez
Information Systems
McMaster University
Hamilton, ON, Canada
lopeze1@mcmaster.ca

Kamran Sartipi
Department of Computer Science
East Carolina University
Greenville, NC, USA
sartipik16@ecu.edu

Abstract

[1] *Among the many significant advances in the area of deep learning, the Natural Language Processing (NLP) space holds a special place. The availability of very large datasets along with the existence of powerful computing environments have created a fascinating environment for researchers. One of the algorithms recently developed is Word2Vec, which enables the creation of embeddings (low-dimensional, meaningful representations of language that can be used for machine learning tasks such as prediction or classification). In this study, we experiment with Word2Vec and apply it to a different domain, i.e., representation of user behavior in information systems. We demonstrate how feature engineering tasks for user behavior characterization can be enriched by the use of NLP concepts.*

1 Introduction

Information Technology (IT) has enabled dramatic transformations in the way organizations execute their processes. Using information systems for the delivery of value is a competitive necessity across all industries, producing a trove of digital data that can be used for myriad purposes. Behavior is no longer an abstraction constrained to the physical world, but can also refer to the way in which people use the information systems at their disposal. The vast majority of users' interaction with information systems is captured in electronic documents – known as logs. These are usually system-specific, very large files that store actions, events and/or contextual parameters in the form of unstructured data. An analysis of these files may provide remarkable insights into the use of information systems.

This study experiments with a real-life, anonymized set of logs that capture the behavior of many users over a pe-

riod of continued monitoring spanning 58 days. The objective is to extract from this data the key elements that would allow characterization of user behavior in information systems, and that can be used downstream in tasks such as prediction or classification.

This paper is organized as follows. Section 2 explores the different foundational concepts that support the analysis in this study. In Section 3 we describe the approach, delving into technology architecture, data structures and techniques. We describe our implementation in Section 4 and conclude our discussion with a summary of our contributions in Section 5.

2 Background on Machine Learning

In this section, we articulate some of the concepts that support the experiments described in this study. Under the broader umbrella of Artificial Intelligence (AI), machine learning – and more specifically deep learning – is demonstrating great success with many real-world applications [4]. A large number of achievements in areas such as computer vision or language have come close or surpassed human performance as measured by standard tests [5].

Perhaps one of the most remarkable developments in the Natural Language Processing (NLP) space is the Word2Vec algorithm. It was created by a team of researchers led by Tomas Mikolov in 2013 [3]. Word2Vec is best explained with an example. The following words (alphabetically indexed) exist in a 5,000-word vocabulary: Man (2203), Woman (4390), King (1348) and Queen (3100). Each word may be represented by a one-hot encoded sparse vector, where only the index position of that word has a value of 1. Figure 1 depicts this scenario.
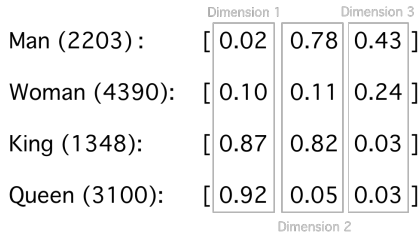
Although these features are numeric, and suitable to feed mathematical models, the sparse representation does not enable comparison between two words as the similarity metrics are meaningless in this context. In contrast, Word2Vec
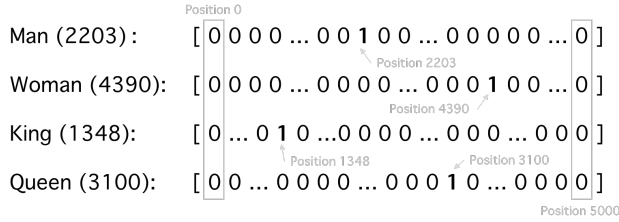
Figure 1. Sparse representation using one-hot encoding for a 5,000 words vocabulary.

Figure 2. Word2Vec 3-dimensional representation of the words.

ingests an existing document (that uses the 5,000-word vocabulary) and produces a dense representation of the words in a lower-dimensional space. The Word2Vec does it by training a shallow neural network with two layers to predict a word given the words (i.e. context) around it in the inputted document. Once the training period is completed, the output layer is discarded, and the final weights are returned as the new representation. For example, a Word2Vec configured to yield three dimensions may produce a vector space that enables comparison between vectors, as per Figure 2.

3   Approach

Using the Word2Vec concepts in a suitable database is a remarkable opportunity that this study pursues. As was explained previously, the intent is to characterize users' behaviors using the tools and techniques from the NLP space. A very large and completed dataset is available from the Los Alamos national laboratory in the United States [2]. It contains the logs from multiple devices running on the same network over a period of 58 days. It includes authentication actions, Domain Name Service (DNS) calls, routing flows and programs started or ended by users. Our attention in this experiment revolves around the programs (i.e. processes) log. It has more than 426 million records uniquely identifying what programs were used by the users.

Manipulating this dataset requires computing power and
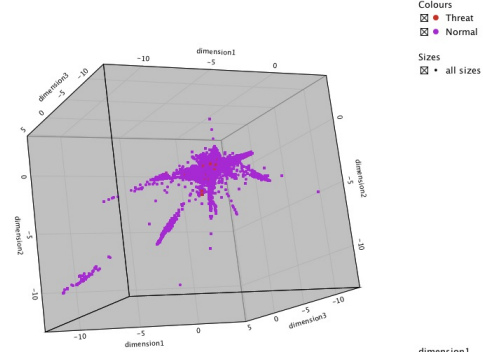
Figure 3. 3-dimensional feature space produced from the raw data through the Word2Vec algorithm.

software beyond the commonly offered in end user workstations. We execute the processes described in this study in a Linux cluster running Apache Spark [1], a unified data science open source tool that implements many of the best-known AI algorithms including Word2Vec.

4   Experimentation

The following experiments are performed using the approach described.

4.1   Feature engineering with Word2Vec

A single program (i.e. process) is coded with the letter 'P' and an integer number. We define a process profile $\vec{PP}$ as a set of processes that are executed by the user in any given hour. The relationship between a process and a process profile is similar to that of a word to a document. Thus, we proceed to use the process profiles as the document to be processed in Word2Vec. There are a total of 2,097,198 records capturing the process profiles ('documents'). We use the Word2Vec implementation in Apache Spark which averages each document when finding the lower-dimensional representation of each word (i.e. process in the case of this study). We experiment with several different dimensional spaces: 1, 2, 3 dimensions (which can be plotted) as well as 10, 100 and 1000.

The 3D vector feature space is depicted in Figure 3. The two classes ('threat' and 'normal') are represented by the red and blue data points.

As the intent of this experiment is to assert whether user behaviors can be extracted using Word2Vec, we proceed to cluster the data in order to best understand if there are regularities that can be detected. The feature set produced by Word2Vec is clustered using the K-means technique. Using
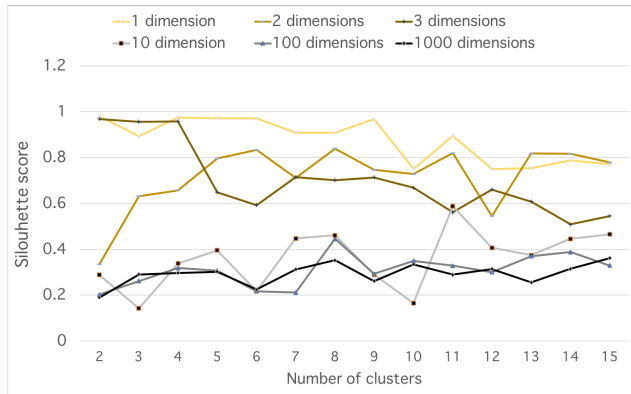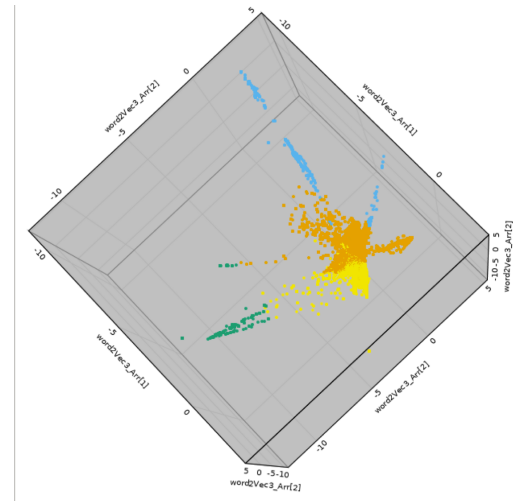
Figure 4. Cluster quality for 1-, 2-, 3-, 10-, 100- and 1000-dimensions, Word2Vec-created vector spaces

Figure 5. K-means clustering for three-dimensional Word2Vec vector space

K-means clustering we establish the silhouette scores from 2 to 15 clusters for each of the Word2Vec vector spaces calculated: 2-, 3-, 10-, 100- and 1000-dimensions. Figure 4 displays the different scores.

There are multiple remarkable elements in this depiction. First: the larger dimensional spaces (10,100 and 1000) found with Word2Vec do not appear to cluster the data well, although they begin to improve as the number of clusters grow (which is to be expected as very high clustering overfits the data). Second: the one dimensional Word2Vec clusters the data well (which is to be expected given the simplicity of clustering scalar numbers). However, using only one dimension neglects the complexity behind user behavior and – in trying to measure it – reduces it to one number only. Using 2 or 3 dimensions enable a better, more textured interpretation and still produce high quality clusters. A third critical point is – knowing that K-means produces different clusters when ran repeatedly as it departs from different centroids – the clustering is performed multiple times, and the results are averaged for generalization purposes.

We select the 3-dimensional vector space, and the maximum number of clusters that yield a 0.95+ score, which is 4 clusters. Three dimensions can represent a wide range of user behaviors, it is easily plot-able and permit the dividing of process profiles into four distinct and well-delineated groups. This can be observed in Figure 5, where the different clusters are depicted with four different colors. We conclude that the features engineered from the data are suitable for clustering activities, reflecting the utility of the feature space estimated.

4.2 Machine learning: classification

The second activity that is performed to assess the utility and effectiveness of the features engineered is classification. The dataset includes labeled data identifying whether the activity was performed by a regular user or by a user belonging to the "red team", i.e., users behaving abnormally. This is a supervised learning model, in which the objective is to verify that the features extracted are suitable for identifying normal vs. abnormal behavior in the feature set.

Every user in the feature set has distinct user behaviors that the classification exercise analyzes. Thus, the logistic regression model needs to be run for each user and not for the total 2.7M records. This means that more than 11,000 logistic regressions (i.e. classifiers) are instantiated and calculated with the labeled data. The feature dataset is partitioned in a training set (80% or approximately 2M records) and a test set (20% or approximately 500K records).

Figure 6 depicts the classified data along the three dimensions calculated with Word2Vec. The color and size convey the probability (blue=1) of an observation being a threat.

It is now possible to calculate how good the classifier was in assigning the correct labels. The following confusion matrix captures the results when each observation is labeled a threat for probabilities higher than 0.5.

The total number of records classified (the test feature set) is 537,280. The ground truth (i.e. known labels) have 161 records labeled as 'threat'. The 3D-Word2Vec logistic classifier predicted 140 records as threats, 87 correctly and 53 incorrectly. Given the rarity of records labeled as threats, the overall accuracy of the classifier is not a good indicator
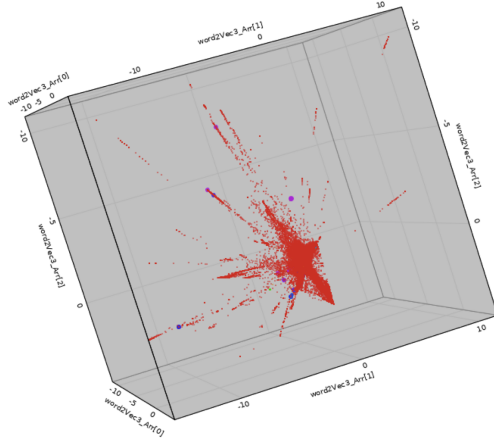
Figure 6. Classified (logistic regression) Word2Vec 3D vector space Color=probability of observation being a threat

Row ID	⊡ Threat	⊡ Normal
Threat	87	74
Normal	53	537066

Figure 7. Confusion matrix (threshold = 0.5) for the logistic classifier.

on its prediction quality. The sensitivity (also called true positive rate) and specificity are calculated as

$$sensitivity = \frac{correct\ threat\ predictions}{total\ number\ threats} = \frac{87}{140}$$

$$specificity = \frac{correct\ normal\ predictions}{total\ number\ normals} = \frac{537,066}{537,140}$$

The classifier has virtually perfect specificity (99.98%), with an adequate 62% sensitivity. It is important to note that the probability of randomly picking an observation labeled as a threat is $\frac{140}{537,280}$ or 0.0002%. It is, therefore, possible to conclude that the classifier built with the features engineered are a very good source of information to identify the threat labeled records.

5 Conclusion

In this study we explore the feature engineering aspects – extraction, transformation and selection – of variables that contain sufficient information for downstream analysis processes such as clustering and classification. We can conclude that using a multidimensional representation of the programs enable suitable characterization of user behavior.

The use of process profiles (i.e. processes started or ended in a given hour by a user) can be equated to language documents that contain words. Extending the analogy to the Word2Vec algorithm allows for the transformation of feature vectors into a dense representation.

Given the results of the different K-means clustering activities, the silhouette scores indicate that three dimensions suffice for the grouping of user behaviors, even enabling plotting for added understanding of the dynamics.

References

[1] Apache Spark™ - Unified Analytics Engine for Big Data. https://spark.apache.org/.

[2] Los Alamos National Lab: National Security Science. https://www.lanl.gov/.

[3] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean. Distributed Representations of Words and Phrases and their Compositionality. In C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 26*, pages 3111–3119. Curran Associates, Inc., 2013.

[4] Raymond Perrault, Yoav Shoham, Erik Brynjolfsson, Jack Clark, and John Etchemendy. Artificial Intelligence index - 2019 annual report.

[5] A. Wang, Y. Pruksachatkun, N. Nangia, A. Singh, J. Michael, F. Hill, O. Levy, and S. R. Bowman. SuperGLUE: A Stickier Benchmark for General-Purpose Language Understanding Systems. *arXiv:1905.00537 [cs]*, Feb. 2020.

Patent Technical Function-effect Representation and Mining Method

Weidong Liu, Piying Zhang, Wenbo Qiao

College of Computer Science, Inner Mongolia University

Inner Mongolia Key laboratory of Social Computing and Data Processing

cslwd@imu.edu.cn, zhangpy.imu@gmail.com, imu.qiaowb@gmail.com

Abstract—**With increasing global competition of intellectual property, a large number of unstructured patent texts are generated for technology protection. The ocean of patent texts include many long sentences about technologies, technical functions, technical effects and complexity relations between them, which make it difficult to textual representation and mining. To solve the above issues, we represent a patent by its technical function-effects, which are mined from the patent. The model represents functions/effects by valence utility-technologies and represents text by association relations between functions and effects. We evaluate our model by comparing with the state-of-the-art models on the patent data set. The results show that our model outperforms other models in evaluation measurement. Such representation can be applied to patent information retrieval and patent text analysis.**

Index Terms—**patent text, representation, function-effect**

I. INTRODUCTION

With the economic globalization, technological innovation are fueling the knowledge-economy increase. To realize innovation-driven economic development, it is urgent to create, protect and applicant for the patents.

Manually reading and comparing the ocean of patents is time-consuming, since the patents have many long sentences which include some technologies, technical functions, technical effects and some complex relations between them. As shown in Fig.1(a), the original claim of patent[1] have many long sentences, which consist of technical terms(marked by blue font) and utility terms(marked by green font). Some implicit relations are included in the patent as well, such as the utility-technology relation between 'control' and 'control device'and the function-effect relation between 'use: wireless communication', 'provide: control device' and 'control: toy client' in Fig.1(b).

Compared with the expected representation in Fig.1(b), most previous representation models lack some aspects of consideration to these technical functions/effects and their relations.

The current textual representation are summarized into three categories:

1) Concept-based models. The concept-based models often represent text by a set of concepts, which include explicit textual representation and implicit text representation. 1) In explicit textual representation, a

Fig. 1. Compared with original claim, the structured claim consist of some utility-technologies and function-effects.

concept, a topic or a text is represented by a vector, where each element corresponds to a clear semantic meanings. The most used models include vector space model(VSM)[1, 2], explicit semantic analysis model(ESA)[3], probability-based topic model, object attribute model(OAR)[4] etc. 2) Implicit textual representation models map the textual information to a latent vector space. The semantic information is often no-interpretable. The most used models include implicit semantic analysis model(LSA)[5], neuron-language model (NLM)[6, 7], segment vector model(PV)[8], hyperspace simulation language model(HAL)[8, 9] and word2vec[10, 11].

2) Relation-based models. Relation based models often represent text as a set of concepts and their relations. Shinmori represents patent claims by six types of relation(process, composition, characteristics, premise, combination)[12]. Okamoto represents patent claims by verb-nouns relations[13]. Luo represents text by association relations[14]. Besides, the relations can form some networks, such as semantic linking network[15], association linking network[16] and knowledge graphs[17–19].

3) Other models. Temporal, citation and other features are considered in textual representation[20–24].

When the above models are used in patent textual representation, the limitations are summarized as follows.

1) Overlooked technical functions/effects in the patent. Most previous models represent a patent by some technical terms(the nouns) other than the functions/effects of the patent.

2) Neglected the relations between functions and effects. Most models only focus on the association relations between technical terms other than the function-effects.

To overcome the above limitations, we propose a patent function-effect representation and its mining method, by which each the patent is represented by some function-effects. The remainders of this paper are organized as follows. Section 2 introduces the preliminary knowledge. Section 3 proposes a function-effect based patent representation. Section 4 gives a mining method of function-effect. Section 5 reports experiments. Section 6 makes conclusion.

II. PRELIMINARY KNOWLEDGE

Valence is a term in chemistry, which refers to the ability of an atom to combine with other atoms. In linguistics, valence refers to the number of arguments(nouns) a verb carries. Inspired by the hypothesis of valence theory, each function/effect of the technology can be represented by the hypothesis of valence theory.

Definition 1: Hypothesis of Valence Theory(HVT)

$$HVT = \{VT^k | 0 \le k < |HVT| - 1\} \tag{1}$$

$$VT^k = \{vt_i^k | 0 \le i < |VT^k| - 1, |vt_i^k| = k\} \tag{2}$$

$$vt_i^k = v_{(i,0)}^k : t_{(1:|vt_i^k|)}^k \tag{3}$$

Where v denotes a utility term; t denotes a technical term; $VT^{(k)}$ is a set of k-valenc utility-technologies; vt_i^k is the ith a utility-technology in $VT^{(k)}$; $|vt_i^k|$ denotes the number of technical term in vt_i^k.

For the claims of the previous patent, the hypothesis of valence theory(HVT) is shown in TableI.

TABLE I
HYPOTHESIS OF VALENCE THEORY OF THE PATENT CLAIMS

$HVT(p)$	$HVT(p) = \{VT^1, VT^2, VT^3\}$				
VT^1	$v_9 : t_4,$	$v_{10} : t_6,$	$v_2 : t_2 t_3,$	$v_3 : t_3 t_4,$	$v_4 : t_4 t_1$
VT^2	$v_5 : t_4 t_5,$	$, v_7 : t_7 t_8,$	$v_8 : t_7 t_1,$	$v_{11} : t_2 t_9,$ $v_1 : t_1 t_9,$	$v_{12} : t_1 t_{10},$ $v_{13} : t_{10} t_{11}$
VT^3	$v_6 : t_2 t_4 t_6$				
Technical Terms: t_1:remote control toy, t_2:client, $\dots t_{11}$: pre-programmed program					
Utility Terms: v_1: control, v_2: use, v_3: provide, $\dots v_{12}$: perform, v_{13}: base					

HVT can represent fucntions/effects of the patent by utility-technologies.

III. FUNCTION-EFFECT BASED PATENT REPRESENTATION

To represent the functions, the effects and their relations, a patent function-effect representation model is proposed.

Definition 2: Function-effect Representation(FR)

$$FR = \{\Phi^k | 0 \le k < |FR| - 1\} \tag{4}$$

$$\Phi^k = \{\phi_i^k | 0 \le i < |\Phi^k| - 1, |\phi_i^k| = k\} \tag{5}$$

$$\phi_i^k = vt_{(i,1:|\phi_i^k|-1)}^k \to vt_{(i,0)}^k \tag{6}$$

where Φ^k denotes a set of k-degree function-effects; ϕ_i^k denotes the ith function-effect in Φ^k; $|\phi_i^k|$ denotes the degree of ϕ_i^k; $vt_{(i,j)}^k$ denotes the jth utility-technology in ϕ_i^k.

TABLE II
FUNCTION-EFFECT REPRESENTATION OF THE PATENT CLAIMS

$FR(p)$	$FR(p) = \{\Phi^{(0)}, \Phi^{(2)}\}$
Φ^0	$\phi_1^0 = vt_4, \phi_2^{(0} = vt_5, \phi_3^0 = vt_6, \phi_4^0 = vt_7, \phi_5^0 = vt_{13}$
$\Phi^{(2)}$	$\phi_1^2 = vt_2 vt_3 \to vt_1, \phi_2^2 = vt_9 vt_{10} \to vt_8, \phi_3^2 = vt_{11} vt_1 \to vt_{12}$
valence utility-technologies: $vt \in HVT$ in tableI	

The function-effect representation(FR) of the previous patent claims is shown in TableII.

For the previous patent, its function-effect representation is shown in TableII.

TABLE III
SYMBOLS AND DESCRIPTION

Symbols	Description
$V = \{v_i | 0 \le i < |V| - 1\}$	a set of utility terms
$T = \{t_i | 0 \le i < |T| - 1\}$	a set of technical terms
$lc_i^k = t_{(i,1:|lc_i^k|-1)}^k \to t_{(i,0)}^k$	a association relation between technical terms
$|lc_i^{(k)}|$	the degree of $lc_i^{(k)}$
$LC^k = \{lc_i^k | 0 \le i < |LC^k| - 1\}$	a set of k-degree relations
$HLC = \{LC^k | 0 \le k < |HLC| - 1\}$	hypothesis of linear concept
$vt_i^k = v_{(i,0)}^k : t_{(1:|vt_i^k|)}^k$	a k-valence utility-technology
$|vt_0^{(k,i)}|$	the valence of vt_i^k
$VT^k = \{vt_i^k | 0 \le i < |VT^k| - 1\}$	a set of k-valence utility-technologies
$HVT = \{VT^{(k)} | 0 \le k < |HVT| - 1\}$	hypothesis of valence theory
$\phi_i^k = vt_{(i,1:|\phi_i^k|-1)}^k \to vt_{(i,0)}^k$	a k-degree function-effect
$|\phi_i^k|$	the degree of ϕ_i^k
$\Phi^k = \{\phi_i^k | 0 \le i < |\Phi^k| - 1\}$	a set of k-degree function-effects
$FR = \{\Phi^k | 0 \le k < |MVFR| - 1\}$	function-effect representation

IV. FUNCTION-EFFECT MINING METHOD

The function-effect representation(FR) is mined by the steps as shown in Algo.1, including 1) generation process of utility-technologies for obtaining functions/effcts in Algo.2, 2) generation process of transaction of functions/effcts in algo.3 and 3) mining association relation between functions and effects for obtaining function-effects of the patent by Eq.7.

In Algo.2, the valence relations between verb terms and noun terms are obtained by pos tagging and dependency parsing[2] from the claims and the abstract of a patent, which will generates different utility-technologies as functions/effects.

Given the functions/effcts, Algo.3 generates some transactions of the functions/effcts, where each sentence can be regards as a transaction consist of functions/effcts.

Given the transactions of functions/effcts, Algo.1 mines the relations of functions and effects with support and confidence large than some threshold values by Eq.7, resulting in the function-effect representation.

$$\left\{ vt_{(i,1:k-1)}^k \to vt_{i,0}^k \quad \middle| \quad \begin{array}{l} vt_{(i,1:k-1)}^k \bigcap vt_{(i,0)}^k = \emptyset \\ sup(vt_{(i,1:k-1)}^k \to vt_{(i,0)}^k) > \theta_s \\ conf(vt_{(i,1:k-1)}^k \to vt_{(i,0)}^k) > \theta_c \end{array} \right\}. \tag{7}$$

V. EXPERIMENTS

In this section, we conduct some experiments to validate the effectiveness of our representation model.

[2]https://nlp.stanford.edu/software/

Algorithm 1: Mining Process of FR

Input: the abstract p^A, the claim p^C, the description p^D of a patent p

Output: the function-effects $FR(p)$ of p

1   initialize $FR = \emptyset$;
2   generate utility-technologies as functions/effects, HVT=Algo.2(p^A, p^C) ;
3   generate transactions of functions/effects, $Trans$=Algo.3(HVT, p^D) ;
4   **if** $\phi_i^k = vt_{(i,1:k-1)}^k \rightarrow vt_{(i,0)}^k$ *is consistent with Eq.7* **then**
5 | $\Phi^k = \Phi^k \bigcup \phi_i^k$;
6   **end**
7   return $FR = \{\Phi^k | 0 \le k < |FR|-1\}$;

Algorithm 2: generation process of utility-technologies as functions/effects

---

Input: $p^A = \{s_i | 0 \le i < |p_A|-1\}$,
        $p^C = \{s_i | 0 \le i < |p_C|-1\}$
Output: $HVT(p^A, p^C) = \{VT^k | 0 \le k < |HVT|-1\}$

1   initialize $\{VT^{(k)} = \emptyset | 0 \le k < |HVT|-1\}$;
2   **for** $s \in p^A \cup p^C$ **do**
3   parse dependency tree $tree(s)$;
4   **if** $\{t_{1:k}\}$ *directly dependent the same* v_0 *in* $tree(s)$ **then**
5 | $VT^k = VT^k \bigcup v_0 : t_{1:k}$;
6   **end**
7   **end**
8   return utility-technologies $HVT = \{VT^k | 0 \le k < |HVT|-1\}$;

Algorithm 3: generation process of function/effect transactions

---

Input: HVT, $p^D = \{s_i | 0 \le i < |p_D|-1\}$
Output: $Trans = \{ts^{(k)} | 0 \le k < |Trans|-1\}$

1   initialize transaction $Trans = \emptyset$;
2   **for** $s \in p^D$ **do**
3     $ts = \emptyset$;
4   **for** $vt \in VT$ *of* HVT **do**
5   **if** $vt \subseteq s$ **then**
6 | $ts = ts \bigcup vt$;
7   **end**
8   **end**
9     $Trans = Trans \bigcup ts$;
10   **end**
11   return $Trans$;

A. Experimental Datasets

Patent data are downloaded from U.S. Patent and Trademark Office(USPTO)[3], which is used in our experiments.

The patents used in our experiments are shown in table IV. The patent CPC codes of each patent are regarded its the multi-label, which are shown in table V. There are 4446 patents from 9 CPC codes.

TABLE IV
THE DESCRIPTION OF EXPERIMENTAL DATA

Source	USTPO								
Data Set	A	B	C	D	E	F	G	H	Y
#code	0	1	2	3	4	5	6	7	8
Number	1256	1508	504	65	279	746	2053	1594	211
Total Number	4446								

TABLE V
PATENT DATA WITH MULTI-LABEL

the number of patent	Multi-Label (ABCDEFGHY)
592	000000100
522	000000110
436	100000000
…	…

B. Baseline Models

We compare function-effect representation(FR) with following state-of-the-art representation models:

1) Vector Space Model (VSM)[1]: VSM is a concept-based model. For VSM, each patent is represented a vector, in which the word is encoded by one-hot.
2) Power Series Representation(PSR)[14]: PSR is a relation-based model. For PSR, each patent is represented by some association relations between technical terms.

C. Evaluation Measurements

Effective representation should have better performance in semantic clustering. The patents with the same CPC code are likely to cluster together. The clustering results are compared with the multi-labels of patent. We use a widely used evaluation measurements in our experiments. The precision, recall, F-measure are used to measure the class code predicted by the model with the reference codes.

Precision is calculated by,

$$P = \frac{TP}{TP + FP} \tag{8}$$

Recall is calculated by,

$$R = \frac{TP}{TP + FN} \tag{9}$$

F-measure is calculated by,

$$F = \frac{2 \times P \times R}{P + R} \tag{10}$$

---

[3]www.uspto.gov
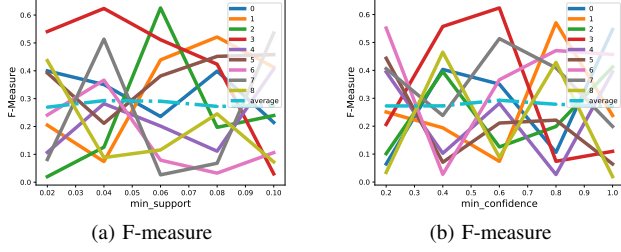
(a) F-measure   (b) F-measure

Fig. 2. (a) F-Measure under different support[0.02, 0.10], confidence 0.6. (b) F-Measure under support 0.04, different confidence[0.2, 1.0].

where TP denotes true positive; TN denotes true negative; FP denotes false positive; FN denotes false negative.

D. Experimental Setups

For each patent p in the data set, we make experiment as follows.

1) Represent the patent by function-effect representation(FR), vector space model (VSM) and Power Series Representation(PSR) in sectionV-B;
2) Cluster each patent by K-means method and each patent p are clustered into top k multi-clusters with most similarity(k equals to the number of different CPC codes in the patent multi-label);
3) Compare and evaluate the results on the evaluation measurement in the section V-C.

E. Experimental Results

The F-measure on 9 different CPC number codes under different support are shown in TableVI. The results are shown respectively in Fig.2 (a). For the experimental data, the highest average F-Measure of the 9 CPC number codes is obtained when the support is 0.04.

TABLE VI
F-MEASURE UNDER DIFFERENT SUPPORT IN RANGE[0.02,0.10] AND CONFIDENCE 0.6

Meas.	sup	0	1	2	3	4	5	6	7	8
F-Measure	0.02	0.3997	0.2048	0.0200	0.5408	0.1071	0.3921	0.2410	0.0815	0.4365
	0.04	0.3506	0.0742	0.1256	0.6237	0.2820	0.2113	0.3663	0.5137	0.0884
	0.06	0.2357	0.4390	0.6256	0.5115	0.2015	0.3814	0.0791	0.0265	0.1158
	0.08	0.3987	0.5211	0.1969	0.4235	0.1113	0.4517	0.0326	0.0680	0.2454
	0.10	0.2145	0.4128	0.2394	0.0300	0.4068	0.4580	0.1061	0.5355	0.0728

The F-Measure on 9 CPC number codes under different confidence are shown int TableVII. The results are shown respectively in Fig.2(b). For the experimental data, the highest average F-Measure of the 9 CPC number codes is obtained when the confidence is 0.6.

TABLE VII
F-MEASURE UNDER SUPPORT 0.04, DIFFERENT CONFIDENCE[0.2, 1.0]

Meas.	sup	0	1	2	3	4	5	6	7	8
F-Measure	0.2	0.0647	0.2508	0.1015	0.2061	0.3945	0.4433	0.5514	0.4061	0.0342
	0.4	0.4033	0.1942	0.3960	0.5572	0.1028	0.0707	0.0275	0.2381	0.4650
	0.6	0.3506	0.0741	0.1256	0.6237	0.2820	0.2113	0.3663	0.5136	0.0884
	0.8	0.1062	0.5705	0.1989	0.0742	0.0268	0.2218	0.4709	0.4089	0.4284
	1.0	0.5465	0.2383	0.4121	0.1098	0.3943	0.0646	0.4569	0.1991	0.0193

TABLE VIII
PRECISION, RECALL AND F-MEASURE FOR THREE MODELS

Meas.	Model	0	1	2	3	4	5	6	7	8
Precision	FR	0.3067	0.0387	0.0673	0.4578	0.1657	0.1566	0.2787	0.3491	0.0463
	PSR	0.3043	0.0111	0.0474	0.3681	0.1648	0.1555	0.2499	0.1290	0.0451
	VSM	0.1626	0.0156	0.0450	0.1580	0.1072	0.1202	0.1188	0.2426	0.0431
Recall	FR	0.4091	0.8642	0.9492	0.9778	0.9453	0.3249	0.5344	0.9716	0.9618
	PSR	0.3931	0.5420	0.5116	0.3401	0.3481	0.2727	0.3214	0.4507	0.3862
	VSM	0.0453	0.0544	0.0457	0.2133	0.1669	0.1303	0.0393	0.0207	0.1503
F-Measure	FR	0.3506	0.0742	0.1256	0.6237	0.2820	0.2113	0.3663	0.5137	0.0884
	PSR	0.3430	0.0218	0.0867	0.3536	0.2296	0.980	0.2812	0.2006	0.0808
	VSM	0.0709	0.0243	0.0453	0.1815	0.1305	0.1250	0.0590	0.0381	0.0670

(a) Precision   (b) Recall
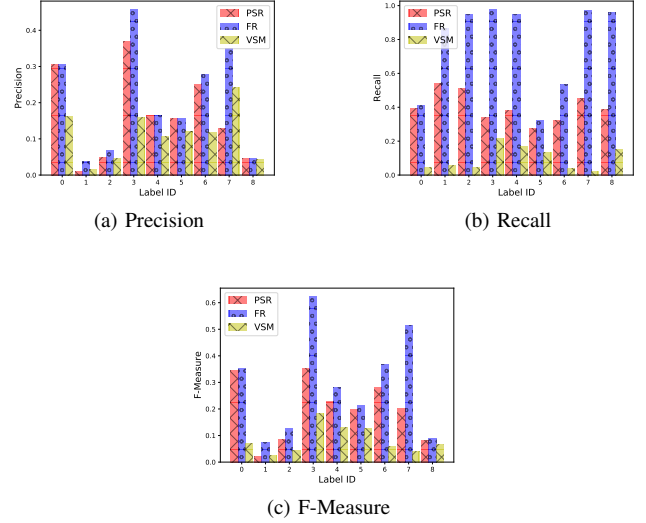
(c) F-Measure

Fig. 3. (a), (b), (c) are the Precision, Recall and F-Measure under different models.

When keep support 0.04 and confidence 0.6, the precision, recall and F-measure on the results obtained by our model compared with that of the baseline models. The comparative results are shown in table VIII.

Fig.3(a), (b) and (c) show the comparative results. The results show that our model outperforms the baseline models in precision, recall and F-measure.

VI. CONCLUSION

In this paper, we propose a function-effect based patent representation model. The contributions of our model are summarized as follow.

1) To represent the functions/effects in a patent, the function-effect representation which is inspired by hypothesis of valence theory, represents functions/effects by multi-valence utility-technologies;
2) To represent the function-effects in a patent, the function-effect representation, which is inspired by hypothesis of linearity concept, represents patent function-effects by multi-degree association relations between functions and effects.

Compared with the baseline models, the function-effect representation exhibits good performance in precise, recall and F-measures in the clustering task.

REFERENCES

[1] G. Salton, A. Wong, and C.-S. Yang, "A vector space model for automatic indexing," *Communications of the ACM*, vol. 18, no. 11, pp. 613–620, 1975.

[2] P. D. Turney and P. Pantel, "From frequency to meaning: Vector space models of semantics," *Journal of artificial intelligence research*, vol. 37, pp. 141–188, 2010.

[3] E. Gabrilovich, S. Markovitch *et al.*, "Computing semantic relatedness using wikipedia-based explicit semantic analysis." in *IJcAI*, vol. 7, 2007, pp. 1606–1611.

[4] G. Dobbie, W. Xiaoying, T. W. Ling, and M. L. Lee, "Ora-ss: An object-relationship-attribute model for semi-stractured data," Tech. Rep., 2000.

[5] S. T. Dumais, "Latent semantic analysis," *Annual review of information science and technology*, vol. 38, no. 1, pp. 188–230, 2004.

[6] Y. Bengio, R. Ducharme, P. Vincent, and C. Jauvin, "A neural probabilistic language model," *Journal of machine learning research*, vol. 3, no. Feb, pp. 1137–1155, 2003.

[7] A. Mnih and Y. W. Teh, "A fast and simple algorithm for training neural probabilistic language models," *arXiv preprint arXiv:1206.6426*, 2012.

[8] L. Azzopardi, M. Girolami, and M. Crowe, "Probabilistic hyperspace analogue to language," in *Proceedings of the 28th annual international ACM SIGIR conference on Research and development in information retrieval*, 2005, pp. 575–576.

[9] D. K. Tayal, L. Ahuja, and S. Chhabra, "Word sense disambiguation in hindi language using hyperspace analogue to language and fuzzy c-means clustering," in *Proceedings of the 12th International Conference on Natural Language Processing*, 2015, pp. 49–58.

[10] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," *arXiv preprint arXiv:1301.3781*, 2013.

[11] X. Rong, "word2vec parameter learning explained," *arXiv preprint arXiv:1411.2738*, 2014.

[12] H. Nanba, H. Kamaya, T. Takezawa, M. Okumura, A. Shinmori, and H. Tanigawa, "Automatic translation of scholarly terms into patent terms," in *Proceedings of the 2nd international workshop on Patent information retrieval*, 2009, pp. 21–24.

[13] M. Okamoto, Z. Shan, and R. Orihara, "Applying information extraction for patent structure analysis," in *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2017, pp. 989–992.

[14] X. Luo, J. Zhang, F. Ye, P. Wang, and C. Cai, "Power series representation model of text knowledge based on human concept learning," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 44, no. 1, pp. 86–102, 2013.

[15] H. Zhuge, "Autonomous semantic link networking model for the knowledge grid," *Concurrency and Computation: Practice and Experience*, vol. 19, no. 7, pp. 1065–1085, 2007.

[16] X. Luo, Z. Xu, J. Yu, and X. Chen, "Building association link network for semantic link on web resources," *IEEE transactions on automation science and engineering*, vol. 8, no. 3, pp. 482–494, 2011.

[17] K. Nakayama, T. Hara, and S. Nishio, "Wikipedia link structure and text mining for semantic relation extraction." in *SemSearch*, 2008, pp. 59–73.

[18] K. Nakayama, M. Pei, M. Erdmann, M. Ito, M. Shirakawa, T. Hara, and S. Nishio, "Wikipedia mining," *Wikimania. Wikimedia*, 2008.

[19] K. Nakayama, T. Hara, and S. Nishio, "Wikipedia link structure and text mining for semantic relation extraction." in *SemSearch*, 2008, pp. 59–73.

[20] L. Zhang, L. Li, and T. Li, "Patent mining: a survey," *ACM Sigkdd Explorations Newsletter*, vol. 16, no. 2, pp. 1–19, 2015.

[21] S. Rogers, S. Tang, and J. Canny, "Acce: automatic coding composition evaluator," in *Proceedings of the first ACM conference on Learning@ scale conference*, 2014, pp. 191–192.

[22] G. A. Miller, "Wordnet: a lexical database for english," *Communications of the ACM*, vol. 38, no. 11, pp. 39–41, 1995.

[23] Z. Dong, Q. Dong, and C. Hao, "Hownet and the computation of meaning," 2006.

[24] Z. Wang, K. Zhao, H. Wang, X. Meng, and J.-R. Wen, "Query understanding through knowledge-based conceptualization," in *Twenty-Fourth International Joint Conference on Artificial Intelligence*, 2015.

Towards A Systematic Derivation Of BPMN Model From Business Process Textual Description

Wiem Khlif[1], Nourchène Elleuch Ben Ayed[2], Faten chihi[1]

[1]Mir@cl Laboratory, University of Sfax, Sfax, Tunisia
[2]Higher Colleges of Technology, ADW, U.A.E
wiem.khlif@gmail.com, nbenayed@hct.ac.ae, chihi.faten.95@gmail.com

Abstract—**Deriving the Business Process (BP) model from its Textual Description (TD) is crucial to its consistent analysis, especially by making process information accessible to various stakeholders. However, establishing or maintaining the TD-BP alignment is not trivial when the enterprise develops a BP. In fact, there is a clear risk that model and text become misaligned when changes are not applied to both descriptions consistently. This paper proposes a new transformation methodology that helps business analyst to build BP model, which is aligned to its textual description. It is based on the use of the business concept's template that is enriched by linguistic-based business rules. Compared to existing methods, our methodology provides more comprehensive alignment, which encompass all BPMN elements. We examined the performance of the transformations through the calculation of recall and precision rates.**

Keywords- Textual Description, BPMN Model, Deriving, Transformation, Business Concept's Template

I. INTRODUCTION

Business processes capture organizational operations and involve numerous actors with various roles [1] [2]. To provide them with the information that they need, organizations have recognized the value of capturing process descriptions in model-based as well as text-based representations [1]. In this context, several methods are proposed to automate the transformation of a given representation into other one: Form model to text and from text to model.

Regarding the model-to-text transformation, [3], the authors have generated textual descriptions of a set of process models using manual and automatic approaches. In Leopold et al. [4], the authors proposed an approach that transforms textual as well as model-based process descriptions into a unified data format to automatically detect inconsistencies between them. [5] define a semi-automated approach that consists of a process model-based procedure for capturing execution-related data in requirements models and an algorithm that takes these models as input for generating natural language requirements.

Apropos of text-to-model transformation, [6] presented an approach to generate BPMN models from natural language text where they faced the complexity of natural language. In [1], the authors presented the first automated approach for the extraction of declarative process models from natural language.

In this paper, we focus on the works related to the generation of BPMN model from its textual description. Nevertheless, the existent works do not cover all BPMN elements. In addition, few works derive automatically the BPMN model. To overcome this gap, we propose a methodology called MONET (a syste**M**atic derivati**O**n of a bpm**N** mod**E**l from business process **T**extual description) that allows generating a BPMN model from the textual description of a business process. To achieve this, the BP description is split into business concepts that achieve a specific business goal. Then, each business concept is specified by using an enriched template [7] that encapsulates the semantic information pertinent to the business logic. Since there are many templates' format, we use the Task and Task & Support Descriptions [8] for the requirements specification to document the business concept. This template is enriched by business rules based on linguistic patterns to support the derivation of all BPMN elements. To evaluate our methodology, we examined the performance of the transformations experimentally through the calculation of recall and precision rates.

The remainder of the paper is organized as follows: Section II introduces the proposed methodology MONET and discusses the transformation definition strategy. Section III describes the BPMN model derivation phase that covers the pre-processing step and the transformation rules, which allow the generation of a BPMN model from the business concept template. Section IV evaluates the quality of the generated BPMN model by considering the recall and precision rates. Section V illustrates our tool MONET that implements the transformation rules and the ontology to generate the BPMN model. Section VI enumerates the threads to validity of our methodology. Section VII discusses the related work and identifies the research gap interest. Finally, Section VIII summarizes the research results and draws the future works.

II. OVERVIEW OF MONET

MONET (a syste**M**atic derivati**O**n of a bpm**N** mod**E**l from business process **T**extual description) is a methodology that derives the BPMN model from a given textual description. Its novelty resides in the production of a BPMN model that is aligned to the input business concepts. More specifically, we propose the business concept as a mean to define the textual description of the business process. Each business concept is enhanced by business rules that transform each linguistic patterns to its corresponding BPMN elements. Fig. 1 depicts our methodology for deriving the BPMN model from a textual description. MONET followed two major phases: BPMN model derivation phase and BPMN model evaluation phase.

The activities of the BPMN model derivation phase are organized essentially in three steps: *A pre-processing step*

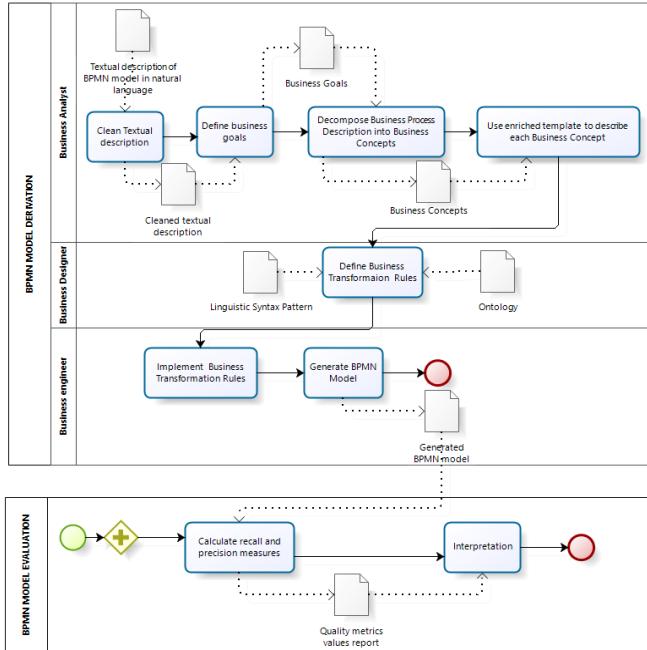during which the Business Analyst receives a textual description of a BPMN model in a natural language.

Figure 1.   Conceptual process of MONET.

The description is cleaned based on a simple NLP technique (Stanford CoreNLP tool) [9]. Then, the Business Analyst uses the output to identify the business goals that are used to divide the business process description into different business concepts. For each business concept, the Business Analyst prepares its textual description according to a specific template. To handle this requirement, we rely on the use of the enriched template presented in [7] (See Section III). Based on this template, the Business Designer defines an ontology in the *transformation-definition step*. The ontology and the linguistic syntactic patterns are used to define the business transformation rules (See Section III). The Business Engineer formalizes/implements the transformation rules in the *transformation-implementation step* which provides for the automated generation of the BPMN model.

The evaluation phase (See Section IV) of our methodology is based on calculating the recall and precision rates in order to assess the performance of the transformations experimentally. Once the calculation is done, a quality report is generated, which is used by the quality interpretation activity.

III.   BPMN MODEL DERIVATION PHASE

A. Natural Langage Pré-processing

We use natural language processing concepts that are syntax parsing and semantic analysis. The syntax parsing consists on obtaining a structured representation of the business knowledge. Therefore, the business analyst has first to clean the textual description by using the Stanford CoreNLP tool [9], and second to organize it according to a specific template's structure. Stanford CoreNLP tool is used to obtain a more manageable and readable text. The tool relies on the following methods:

- *Tokenization* is the task of breaking a character sequence up into pieces (words/phrases) called tokens, and perhaps at the same time throw away certain characters such as punctuation marks [10].

- *Filtering* aims to remove some stop words from the text. Words, which have no significant relevance and can be removed from the documents [11].

- *Lemmatization* considers the morphological analysis of the words

- *Stemming* aims to obtain the root of derived words [12].

- *Part of Speech Tagging* tags for each word (whether the word is a noun, verb, adjective, etc), then finds the most likely parse tree for a piece of text.

The cleaned file is then used to identify the business goals of the business process. By business goal, we mean a collection of business activities that are related to describe a functional process of the BPMN model. Each goal will correspond to a business concept.

To guide and improve the generation of a BPMN model, the business analyst associates to each business concept a template that is described by a set of linguistic patterns. The template covers the semantic and organizational information related to the business logic. It is composed of three blocks (Fig. 2).
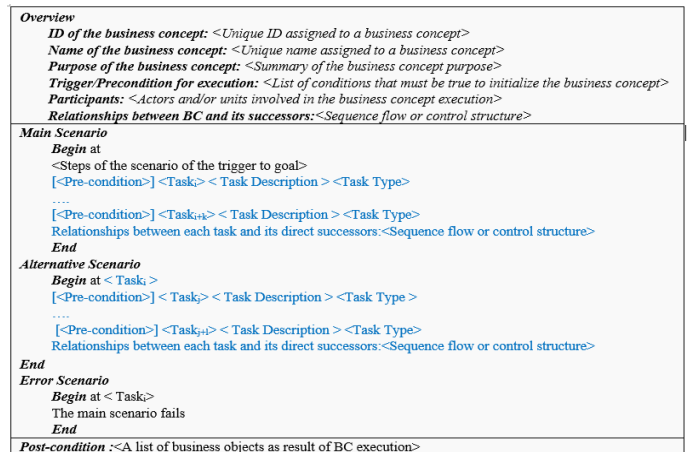
Figure 2.   Detailed description of a business concept.

The first block gives an executive summary of the business concept in terms of its id, name, purpose, pre-conditions, participants involved in its execution, and its relationships with business concept's successors. We defined a specific structure for the triggers, which is **[<Pre-condition>] <Event Description> [< Event Type:{timer | Message | Signal | Conditional}>]**. The event type can be explicitly specified or implicitly extracted from its description. In addition, to formalize the relations among participants, we created a WordNet, which is a lexical database for all business words. It defines a set of synonyms of a participant called Synsets and records the relations among them such as hypernym (Type of), meronym (part of), and antonym (opposite word). The relationships that a business concept has with its successors follows the linguistic pattern: **[<Pre-condition>] <Current Business Concept ID> is related <sequentially | exclusively |**

parallel | inclusively>to<Business Concept ID>, where the **<Precondition>** construct respects this structure **<if>** condition **<then>**.

The second block describes the main, alternative, and error scenarios. These scenarios respect this pattern **[<Pre-condition>] <Task#><Task descriptions> <Task Type >:**

- *Task Description*: We defined a linguistic syntax pattern to describe the tasks: **ActionVerb | CommunicationVerb + BusinessObject | NominalGroup + [[to ReceiverName] | [from SenderName]]** to label the tasks. We mean by *BusinessObject* any entity that describes the business logic. The *NominalGroup* is a set of pre/post-modifiers, which are centered on a Headword that constitutes the *BusinessObject*. The pre-modifiers (respectively post-modifiers) can be a noun, an adjective, or an ed/ing-participle (respectively, a noun, an adjective, or adverb). The *VerbalGroup* indicates the relationship type between *BusinessObjects*. We note that the expression between brackets is optional.

- *Task Type:* The task type can be "ActiveREQ", "ActiveREP", "ActiveRET", "ActivePER" or "Passive" representing respectively "Entry", "eXit", "Read", "Write" or "data manipulation". "ActiveREQ" corresponds to a task representing the act of asking for something. "ActiveREP" corresponds to a reply sent after asking for something.

  "ActiveRET" corresponds to a task allowing data retrieval. "ActivePER" corresponds to a task allowing the data record. "Passive" task does not lead to an exchange of data.

The third block illustrates the list of business objects as result of the execution of the business concept.

For the semantic analysis of the business concepts' template, we propose to create an ontology (See Fig. 3).

For the semantic analysis of the business concepts' template, we propose to create an ontology. It is designed to describe the entities related to the BPMN metamodel. The annotation process is based on the result of the preprocessing task and the defined template. It takes business concept templates of the business process model and define the similarities (the links) between concepts. We use the concept names to produce an expanded list of equivalent or related terms. Each term of the input textual description may be associated with one or more entities from the ontology. To find the similarities, we used the following matching techniques:

- Exact matching identifies the identical entities (String) in the text and in the ontology;

- Morphological matching identifies the entities with a morphological correspondence;

- Syntactical similarities using Levenshtein measure [13];

- Semantic matching identifies the synonyms relations with WordNet ontology.

- Semantic matching identifies the synonyms relations with WordNet ontology.
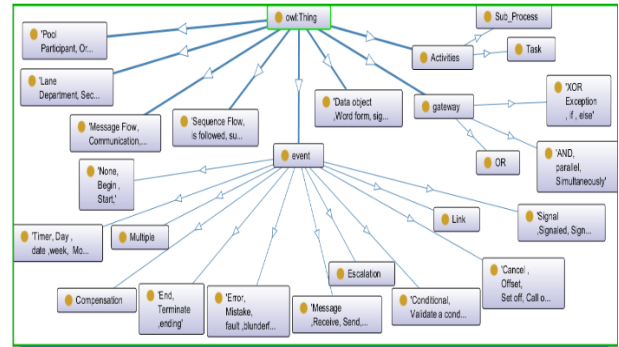

Figure 3. Meta-model ontology.

B. From Textual Description to BPMN Model

We defined eighteen transformation rules. Each transformation rule operates on the different components of the template.

R1. Each trigger is transformed into an event that will be linked to the first element of the current business concept. Based on the trigger type, we add the corresponding event.

  R1.1: If the trigger type describes the time, so add a Timer Event.

  R1.2: If the trigger type describes a certain condition that must be satisfied to start a process, so add a Conditional Event.

  R1.3: If the trigger type describes any action that refers to a specific addressee and represents or contains information for the addressee, so add a Message Event.

  R1.4: If the trigger type describes any action that refers to anyone and represents or contains information for anyone, so add a Signal Event.

R2. Each participant is transformed into pool or lane depending on its type.

  R2.1: If all participants are business workers, then add a pool that has the same name of the business worker. We note that a business worker represents an abstraction of a human that acts within the business to realize a service.

  R2.2: If one of the participants is a metonymy of "department", "unit", "division" or "organizational unit", then add a pool that has the same name of the participant and transform others participants to lanes. Based on the ontology result, if the relation between the participant which is represented by a pool and the other one that is represented by a lane is metonym (part of), so add a lane.

R3. Each relationship between the business concept and its successors respects the linguistic pattern: **[<Pre-condition>] <Current Business Concept ID> is related <sequentially | exclusively | parallel | inclusively>to<Business Concept ID>**.

  R3.1: If the relationship is <sequentially>, then add a sequence flow if the last element of the current business concept and the first element of its successor are in the same pool. Otherwise, add a message flow.

R3.2: If the relationship is <parallel>, then add a parallel gateway between the last element of the current business concept and the first element of its successor.

R3.3: If the relationship is <exclusively> and there is a precondition, then add an exclusive gateway between the last element of the current business concept and the first element of its successor. The precondition expression is associated with the gateway outgoing sequence flow.

R3.4: If the relationship is <inclusively> and there is a precondition, then add an inclusively gateway between the last element of the current business concept and the first element of its successor. The precondition expression is associated with the gateway outgoing sequence flow.

R4. For each step of a BC's scenario respecting the linguistic pattern: **[<Pre-condition>] <Task#> < Task Description > <Task Type >**, then add the following:

R4.1: If the task description is « Action verb + BusinessObject », then add a service task that has the same name of the pattern and a data object.

R4.2: If the task description is « Action verb + NominalGroup », then add a service task that has the same name of the pattern. If the pre/post-modifier is a noun that merely represents a pure value, so there is no data object to add. Otherwise, if the pre/post-modifier is a complex noun (an entity) then add a data object.

R4.3: If the task description is « CommunicationVerb+ BusinessObject|NominalGroup + [[to ReceiverName(s)] | [from SenderName]] », then add send or receive task that has the same name of the pattern and a data object. Then, call R4.4, R4.5, and/or R4.6.

R4.4: If the task type is ActivePER, then add an outgoing object flow between the task and its data object/store.

R4.5: If the task type is ActiveRET, then add an ingoing object flow between the task and its data object/store.

R4.6: If the task type is ActiveREP, then add a message event and an outgoing message flow between the task and message event. The message event name is the concatenation of the Business Object extracted from the task and the past participle of Receive.

R5. Each relationship between the task and its successors respects the linguistic pattern: **[<Pre-condition>] <Current Task ID> is related <sequentially | exclusively | parallel | inclusively>to<Task ID>**.

R5.1: If the relationship is <sequentially>, then add a sequence flow if the current task and its direct successor are in the same pool. Otherwise, add a message flow.

R5.2: If the relationship is <parallel>, then add a parallel gateway between the current task and its direct successor.

R5.3: If the relationship is <exclusively> and there is a precondition, then add an exclusive gateway, if it does not exist, between the current task and its direct successor. The precondition expression is associated with the gateway outgoing sequence flow.

R5.4: If the relationship is <inclusively> and there is a precondition, then add an inclusively gateway, if it does not exist, between the current task and its direct successor. The precondition is associated with the gateway outgoing

sequence flow.

R5.5: If the relationship is <sequentially>, and there is a <complete> construct related to a task, then add an end event.

IV.   BPMN MODEL EVALUATION PHASE

The evaluation of our methodology is based on the experimental comparison activity that calculates for each element type of the BPMN model, the recall and precision rates according the following equations:

$$\text{Precision} = TP/(TP+FP) \qquad (1)$$

$$\text{Recall} = TP/(TP+FN) \qquad (2)$$

Where:

- True positive (TP) is the number of existing real elements generated by our transformation;

- False Positive (FP) is the number of not existing real elements generated by our transformation;

- False Negative (FN) is the number of existing real elements not generated by our transformation.

V.   MONET TOOL

To facilitate the application of our methodology, we developed a tool for deriving the BPMN model from a given textual description, named MONET Tool. Our tool is implemented as an EclipseTM plug-in [14]. It is composed of three main modules : Parser, generator, and evaluator.

The pre-processing engine uses as input the textual description of a BPMN model written in a natural language. It cleaned the file using the Stanford CoreNLP tool. The cleaned file is used by the business analyst to define manually business goals. Then, the latter associates each business goal to its corresponding BC. The business analyst creates the enriched template corresponding to each BC. Fig. 4 shows the BC4's template.

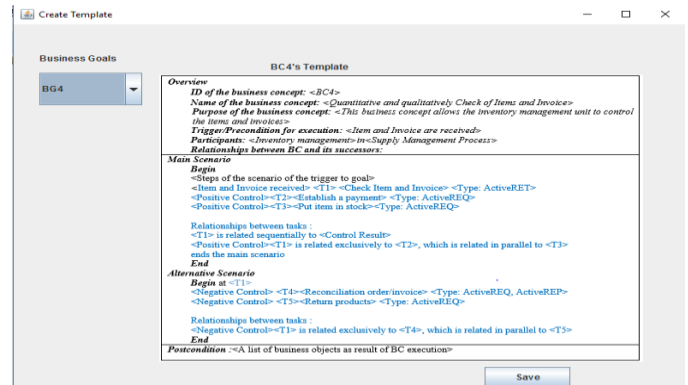

Figure 4.   BC4's Enhanced template.

Next, the analyst selects one or more BCs. If he selects one BC, the corresponding fragment is generated. Else, the business analyst can select all business concepts to transform.

The generator engine uses the ontology and applies the transformation rules to derive the BPMN model. Fig. 5 illustrates the generated BPMN model: "Supply Management Process".
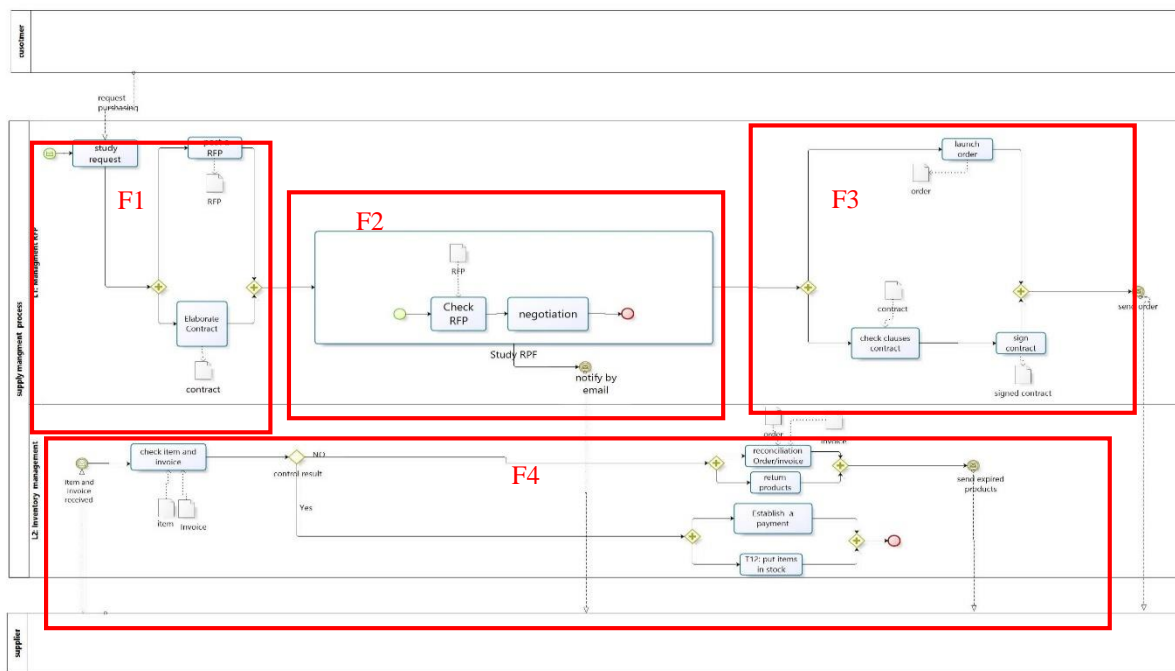
Figure 5.   The generated BPMN model: "Supply Management Process".

The obtained model is generated as follows: First, by applying **R2.2**, we add a lane "PurchaseDepartmentSystem" inside the pool "Supply Management Process". Second, by applying **R1**, we add the message event "Item and Invoice are received" in the corresponding lane.

The transformation of the main scenario calls **R4.2** and **R4.5** that generate a task labelled *"Check Item and Invoice"*, two data objects labelled invoice and item, and add an ingoing object flow between the task and its data objects. Then, **R4.2** produces a task labelled Establish a payment (respectively, Put item in stock).

By applying **R5,** we linked the business task "*check invoice and item*" to the exclusive gateway labelled "*control result*". Then, we applied **R3.3** and added the precondition related to the default outgoing flow expressing the main scenario. By applying **R5**, a parallel gateway is created between establish payment and put item in stock.

Then, by selecting the "Check alignment" button, the generator displays each element in all the business concepts and their corresponding BPMN elements (See Fig. 6).

Figure 6.   The generated BPMN model: "Supply Management Process".

If each element has its correspondence in the BPMN model, then we can deduce that the textual description is aligned to its model.

The BPMN quality evaluator evaluates experimentally the BPMN model through the calculation of recall and precision rates.

Precision==0.86
Recall =0.95

The high scores for both ratios mean that the generated BPMN model covers the whole domain precisely in accordance with the experts' perspective (See Fig. 7). We can deduce that the performance of our methodology approaches the human performance.

Figure 7.   The elaborated BPMN model by the expert

VI. THREATS TO VALIDITY

In our study, threats to validity are relevant to internal validity and external validity [15].

The internal validity threats are related to four issues: The first threat to validity focus on who write the textual description expressing the functional requirements and which template have been used to describe the functional requirements. Expert should well write these requirements based on a particular style to generate a high quality of a BPMN model. The Second problem is addressed when there is a diversity of description of the requirements. In this case, which one can be used to describe the functional requirements? The third issue is related to the impact of an error-prone generation of a BPMN model. This case may lead to misalignment and inconsistency between the textual description and business process model.

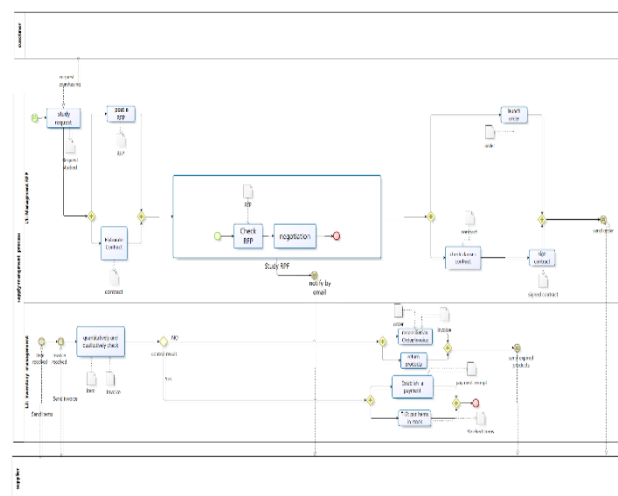The external validity threats deal with the possibility to generalize this study results to other case studies. The limited number of case studies used to illustrate the proposed methodology could not generalize the results. Automation of our methodology needs to be considered even it is easy to use manually given its simplicity.

VII. RELATED WORK

Several methods explicitly emphasis on the generation of process models from different types of text documents. The authors of [6] presented an automatic approach to generate BPMN models from natural language text, where they faced the complexity of natural language.

In [1], the authors present an automated approach for the extraction of declarative process models from natural language. They developed a tailored Natural Language Processing (NLP) techniques that identify activities and their inter-relations from textual constraint descriptions. By considering the semantics of these extracted components, the authors generate declarative constraints aiming to capture the logic defined in the textual description.

In [16] the author uses natural language processing with a focus on the verb semantics, and creates a novel unsupervised technique *TextProcessMiner* that discovers process instance.

In summary, many researchers studied the alignment between BPMN model and textual description. However, they don't cover all BPMN elements.

VIII. CONCLUSION

This paper proposed a transformation-based approach to generate a business process model from its textual description. It provides for the generation of a BPMN model that is aligned to the input business concepts. Compared to existing works, our methodology has the merit of accounting for all BPMN elements and their relationships. To do so, our methodology used the enriched template as the starting point for deriving BPMN model. Then, it defines transformation rules that transform each linguistic patterns to its corresponding BPMN elements. The methodology has been implemented. An evaluation of a business process model shows that our methodology approaches the expert performance and generates BPMN models respecting the quality measurements. Although the current results are very promising, our technique still requires further empirical tests.

We intend to generalize the methodology in order to derive BPEL from the textual description as well as the information system's design models from the textual description and check the alignment between all generated models: BPMN model and information system design models.

REFERENCES

[1] H. Van der Aa, C.D.Ciccio, H. Leopold, H.A. Reijers, "Extracting Declarative Process Models from Natural Language", 31st Conf. Advaneced Information Systems Engineering, Italy pp. 365-338, 2019.

[2] M. Dumas, M. La Rosa, J. Mendling, H.A. Reijers, "Fundamentals of Business Process Management". Springer, ISBN, pp. 1-527, 2018.

[3] S. Zaheer, K. Shahzad, R. M. A Nawab, "Comparing Manual- and Auto-Generated Textual Descriptions of Business Process Models", 6th Conf. on Innovative Computing Technology, Ireland, August. 2016.

[4] Leopold, H., H. van der Aa, F. Pittke, M. Raffel, J. Mendling, H.A. Reijers, "Searching textual and model-based process descriptions based on a unified data format", International Journal of Software and system Modeling. 18, No.2, pp. 1179-1194, 2019.

[5] B. Aysolmaz, , H. Leopold, H.A. Reijers, O. Demirörs, "A semi-automated approach for generating natural language requirements documents based on business process models", International Journal of Information & Software Technology, Vol 93, pp. 14-29, 2018.

[6] F. Friedrich, J. Mendling, F. Puhlmann, F., "Process model generation from Natural Language Text", 23th International Conference on Advanced Information Systems Engineering, LNCS in Computer Science book series, Vol. 6741, London, June, pp. 482–496, 2011.

[7] W. Khlif, A. Sallemi, M. Haoues, H. Ben-Abdallah, "Using COSMIC FSM Method to Analyze the Impact of Functional Changes in Business Process Models", 13th International Conference on Evaluation of Novel approaches to software engineering, Portugal, March, 2018.

[8] S. Lauesen, "Software Requirements: Styles and Techniques", Addison-Wesley, London, 2002.

[9] CD. Manning, M. Surdeanu,, J. Bauer, J., Jenny Rose J.R. Finkel, S.Bethard, D. McClosky, "The Stanford CoreNLP Natural Language Processing Toolkit", The 52nd Annual Meeting of the Association for Computational Linguistics, June 22-27, pp.55-60. 2014.

[10] J. Webster, C. Kit, "Tokenization as the initial phase in nlp", 14th conference on Computational linguistics, Association for Computational Linguistics, Vol.4, pp. 1106-1110, 1992.

[11] Saif, H., Fernandez, M., He, Y., Alani, H. , "On stopwords, filtering and data sparsity for sentiment analysis of twitter", the 9 th Inter. Confe. on Language Resources and Evaluation, European Language Resources Association, Iceland, May 26-31, pp. 810-817, 2014.

[12] J.B.Lovins, "Development of a stemming algorithm", Mechanical Translation and Computational Linguistics, Vol 11, No.1-2, june, pp. 22-31, 1968.

[13] V.I.Levenshtein, "Binary codes capable of correcting deletions, insertions and reversals", *journal of Soviet physics doklady*, Vol. 10, pp. 707-710, 1966.

[14] Eclipse Specification. (2011), Available from: http://www.eclipse.org/

[15] C. Wohlin, P. Runeson, M. Höst, M.C. Ohlsson, B. Regnell, A. Wesslén, "Experimentation in Software Engineering: An Introduction", 2000.

[16] E.V. Epure, P. Martín-Rodilla, C. Hug, R. Deneckère, C. Salinesi, 2015. "Automatic process model discovery from textual methodologies", 9th IEEE International Conference on Research Challenges in Information Science, Greece, May 13-15, 2015.

A Novel Self-Attention Based Automatic Code Completion Neural Network

Bohao Wang, Wanyou Lv, Jianqi Shi, and Yanhong Huang*

National Trusted Embedded Software Engineering Technology Research Center,
East China Normal University
{bohao.wang, wanyou.lv}@ntesec.ecnu.edu.cn
{jqshi, yhhuang}@sei.ecnu.edu.cn

Abstract—**Code completion is one branch of source code modeling tasks. Using a deep learning method to implement it has explored the possibilities of modeling source code with a statistic language model. Recurrent Neural Network (RNN) is a universal feature extractor of Natural Language Processing (NLP), which is used in the code completion field commonly. However, RNN based models are lack of long-range context dependency and have a poor performance in training speed. Besides, some previous models have not handled the issue of out of vocabulary (OOV) well, which hinders further improvements in prediction accuracy. This paper presents a novel automatic code completion neural network, which is based on a self-attention mechanism with open vocabulary to address issues of OOV, slow training speed, and lacking long context-dependency. Experiments in this paper show that our model has a better performance of predicting tokens compared with the traditional N-gram model and RNN based model. In the meantime, we reduced training time significantly. More broadly, the combination of self-attention and open vocabulary has a potential application in the source code modeling field.**

Index Terms—**Code Completion, Self-Attention, Source Code Modeling, Open Vocabulary**

I. INTRODUCTION

As one part of automatic software development, code completion is always a popular research field in software engineering. Code completion, which refers to recommending the next token based on the current context [1], is a technique that allows us to speed up the coding process and to reduce spelling errors during coding. Nowadays, most programmers use Integrated Development Environment (IDE) like Eclipse and IntelliJ IDEA to write code, enjoying the convenient service of code completion which is a basic feature of modern IDE. Traditionally, code completion in IDE relies heavily on compile-time type information to predict the next token [2]. This method only does well in suggesting attributes or methods of classes but fails to predict coding habits of users. Hindle et al. [3] propose that code has a naturalness and is likely to be predictable and repetitive, so they introduce a statistic language model into the field of source code modeling. In the early stage, the N-gram model used to be the statistic language model used in source code modeling

[1], [3]–[5]. Later, when the Recurrent Neural Network (RNN) is introduced into Natural Language Processing (NLP) field, source code modeling has tended to RNN based model [6], [7]. Because the vanilla RNN's variants, Long Short-Term Memory (LSTM) and Gated Recurrent Units (GRU) suppress the problem of gradient exploding and gradient vanishing, they are applied to the field of source code modeling [4], [8]. Although some RNN based deep learning static language models have achieved good results in source code modeling, there are still several defects. It is noted that most models can not predict OOV words, which is called neologisms in [9]. A large number of OOV words will affect the performance of automatic code completion distinctly. Although [10] in ICSE 2020 tried to solve this problem, the method they use aggregates the negative effects of lacking long-range dependency on prediction performance. These defects affect the quality and precision of code completion. This motivates us to propose a novel model to address the issues above. In summary, contributions of this paper include:

- We propose a novel self-attention based automatic code completion neural network. The model addresses issues of unable predicting out of vocabulary tokens, lacking long-range dependency ability, and slow training speed, which are defects of current models.
- We evaluate our model in a real word Java code dataset. Compared with previous work, our model has significant improvements in the metric of Mean Reciprocal Rank (MRR) and entropy in three realistic scenarios. In the meantime, our model spends less time in the training process.
- We design and implement a self-attention based model with Open Vocabulary (SABCCOV), a tool to predict the next token based on current tokens. To the best of our knowledge, we are the first to combine the self-attention with the Open Vocabulary mechanisms in code completion.

The rest of this paper is organized as follows. Section *II* details some defects of existing current models and background knowledge of self-attention mechanism. Section *III* presents the design of SABCCOV and training setup. Section *IV* demostrates experiment details and evaluations of our

model. We discuss related work in Section V and conclude in Section VI.

II. Preliminary

In this section, we mainly talk about existing issues of current models and the basis of self-attention mechanism. Sec.II-A describes a common problem in code completion. The issue of Sec.II-B is a critical factor that affects the performance of the model. And Sec.II-C is an aspect that can be improved continuously. At the end of this section, we will describe the self-attention mechanisms.

A. Out of Vocabulary (OOV)

The vocabulary of natural language processing is commonly formed by top k (assume 50,000) frequency words from a large corpus, which is closed because it can only present limited words. And the indexed vocabulary is used to map a word to the index in the statistic language model. Out of vocabulary words in NLP will be represented by 'UNK'. In natural language, it is feasible that OOV words are replaced by the UNK identifier. The NLP model is rarely affected by it since most words are covered by vocabulary. But the programming language model will be affected seriously on account of many identifiers such as variable names, class names, and method names that are defined by the programmer.

B. Long-Range Dependency

Range dependency means that how many context words does the model need to predict the next word. In [11], LSTM language models use 200 context words on average, which has a longer range commonly than Vanilla RNN and GRU. Empirically, the RNN based model is hard to deal with long-range dependencies that are common in programming language [2]. For example, a class is declared at the top of the file, but it may be used after one hundred lines of declaration. The dependency range of the LSTM model may be enough in NLP, but it is not enough for source code modeling.

C. Slow Training Speed

It is noted that the performance of the deep learning model depends on the scale of data. The more training data, the better the performance of the model. In the source code modeling field, it is an advantage that massive amounts of data are easy to get from some open source communities such as Github. Vanilla RNN calculates hidden state one by one to collect sequence information, that is why Vanilla RNN called Recurrent Neural Network. The architecture of Vanilla RNN demonstrates that it is doomed not to support parallel computing, which will have a strong impact on training speed.

D. Self-Attention Mechanism

The self-attention mechanism is proposed in [12]. An attention function can be described as mapping a query and a set of key-value pairs to an output, where the query, keys, values, and output are all vectors. The matrix of outputs can be computed as:

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V \qquad (1)$$

Intuitively, the self-attention function is to help every token in sequence to calculate how much attention needs to pay to other tokens. In the multi-head version of self-attention, the self-attention function takes n different matrices as input and computed for n times, then output n matrices. Then we concatenate all these matrices and condense them into one matric whose dimension is the same as the original self-attention result.

III. Approach

In this section, we describe the procedure of the data preprocessing, which will get more appropriate data for the model than the raw source code. Then we will present the architecture of the overall network from a block perspective and introduce our training strategy.

A. Dataset and Preprocessing

We use the Gig-token corpus in [13], which has more than 14,000 Java projects from Github. We have a large corpus of source Java code, but we can not feed it into the model directly. In the NLP filed, the common way of input data transformation is mapping some high-frequency token to embedding vectors or one-hot vectors according to a vocabulary. But it can not handle the problem of OOV words described in Sec.II-A. It is hard to solve the issue of OOV completely, but the Open Vocabulary method [10] can solve it partially. Using the Byte Pair Encoding (BPE) algorithm, this method learns a segmentation pattern. BPE is a data compression algorithm that iteratively finds the most frequent pair of bytes in the vocabulary appearing in a given sequence, and then replaces it with a new unused entry [14]. The algorithm is first adapted for word segmentation in [15], it merges pairs of characters or character sequences.

As an illustration, we can segment a Java source code file using a BPE vocabulary file as shown in Fig 1. Some high-frequency tokens are reserved and some low-frequency tokens are split into high-frequency sub-tokens followed by

```
package com.testMain;
public class newClass{
  private String someInformation;
  public newClass(String someinformation){
    this.someInformation = someinformation;
  }
}
```

```
package com . test@@ Main ;
public class new@@ Class {
  private String some@@ Infor@@ mation ;
  public new@@ Class ( String some@@ infor@@
      mation ) {
    this . some@@ Infor@@ mation = some@@
      infor@@ mation ;
  }
}
```

Fig. 1. Java code before/after segmentation with an end-of-subtoken '@@'

'@@'. In such a way, OOV words are decomposed into high-frequency subword, which means they can be represented by word embedding vector or one-hot vectors according to the Open Vocabulary.

TABLE I
DATASET STATISTICS

	Full train	Small train	Test	Valid
Projects	13255	107	38	36
Files	1.9M	12K	8.2K	7.1K
Tokens(original)	1.6B	20M	5.9M	4.6M
Toekns(2K)	2.5B	31.9M	9.4M	7.4M
Tokens(5K)	2.2B	27.8M	8.3M	6.5M
Tokens(10K)	2.1B	25.8M	7.7M	6.1M

It is noted that the BPE algorithm needs code to learn segment patterns and produce a vocabulary file. And the number of BPE merging operation affect the performance of the model, so we set three different value, 2k, 5k, and 10k. As Table I shows, data was divided into four parts: full train, small train, test, and valid, which is the same as [4] except moving a small part randomly from the full train projects as a BPE training corpus. Followed [10], we set the size of BPE training corpus to 1000 projects. We provide the procedure of data preprocessing as follows:

1) Remove comments.
2) Replace non-ascii characters with a special identifier (We use '-UNK-' in experiments).
3) Tokenize the file[1], which means every line is one token (Punctuations such as ';' is also considered as one token).
4) As for BPE data, use the BPE algorithm[2] to get a vocabulary file and a segment pattern file.
5) As for train, test, and valid data, use the BPE algorithm with the segment pattern file on these corpora and get segmented files.
6) Add start and end identifiers (We use '<s>' and '</s>') at the top and bottom of every Java file then integrated all files in train/test/valid projects into one file as train/test/valid file.

After preprocessing, words were split into subwords and subwords can be represented by word embedding, which means most words can be dealt with by the model. Due to the segmentation of the BPE algorithm, we can suppress the **Out of Vocabulary** issue effectively. However, it is noted that the number of tokens has increased at least one-third of the original after the procession of the BPE algorithm. It will magnify the negative effects brought by the problem of **Long-Range Dependency**.

B. Model Architecture

We proposed a self-attention based model with Open Vocabulary as shown in Figure 2.The input and output of the model is a fixed-length sequence. The model has three parts including the **input block**, the **transformer block** and the **output block**. In the input block, the input sequence will be
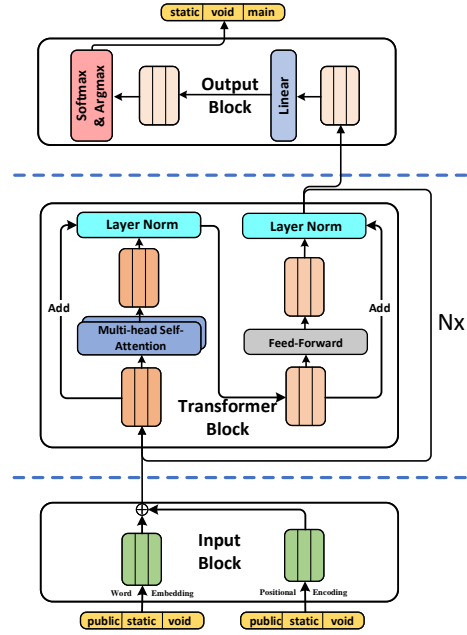
---

[1] The library we use is https://github.com/SLP-team/SLP-Core
[2] We use https://github.com/rsennrich/subword-nmt

Fig. 2. Overall Architecture: Based on the input of 'public static void main', the model predicts 'static void main'. Rectangles composed of 3 small rectangles represents the intermediate vector.

represented by the sum of word embedding and positional encoding, which will be the input tensor of the transformer block. The transformer block has a masked multi-head self-attention layer, a position-wise fully connected feed-forward layer, and some residual connections. The masked multi-head self-attention layer and feed-forward layer are followed by a layer normalization. It looks like the encoder part in [12]. Instead, we replace the multi-head self-attention layer with a masked self-attention layer. The input and output of the transformer block have the same dimension, so it can repeat any time we want. Due to the self-attention mechanism, the transformer block in our model can solve the problems of **Long-Range Dependency** and **Slow Training Speed**. The output block consists of a linear layer and a softmax layer. The output sequence is obtained by the output tensor processed by the argmax function.

As Figure 2 shows, the input is 'public static void' and the model will output the sequence of 'static void main'. Predictions for position i can only depend on the known input at positions less than i. The most critical part of the whole model is the self-attention mechanism. Benefits of the self-attention including two aspects. First, the training process can be shortened observably since the self-attention mechanism supports parallelized-training. The RNN based model computes the next hidden units depending on previously hidden units. Unlike the RNN based model, the self-attention based model does not need previously results, so it can be parallelized by vectorization. The second aspect is the self-attention can learn longer-range dependency than LSTM based model. In order to capture the information between two tokens(noted token i and token j), the RNN based model needs at least $|j-i|$ steps because of the serial computation. And it may lose some information if the distance is too long. But the

vectorization of the self-attention based model can ignore the distance between two tokens in the same segment and correlate them within one step.

The multi-head self-attention layer considers the attention weight of every position of the input, which means the third token in the input sequence may be taken into account when predicting the second position in the output. For example, the model will take 'void' (third token in input) into accounts when it is predicting 'void' (second token in output). But as a prediction task, it is a cheating trick to consider the context that has not shown yet. In [12], the masked mechanism guarantees that the model will not make the prediction based on future information. And this is the reason we replace it with the masked multi-head self-attention layer.

The model we proposed is used to predict sub-tokens, which means it may output part of the correct token (some sub-token is a complete correct token). Further, we use the beam-search-like algorithm [10] to predict complete tokens directly. As Figure3 shows, if the model predicts the token ends with '@@', the model will continue to predict the token until the next predited token without '@@'. Finally, the model will concatenate all the predited subtokens, delete the '@@' and the result is the prediction of the model.

C. Training

As Section III-A described, we have two training sets on a different scale, including small training and full training. Most setup of two training sets is the same. We fix the length of the input sequence feed to the network to 512. If the length of the input is longer than 512, then split it into a few segments and supplement segment whose length is less than 512 with special tokens. The batch size is 32 for the small train and 64 for the full train, respectively. The number of the transformer block is 3, the number of multi-head is 2, and the dropout rate is 0.2. Other hyperparameter of the transfomer block is the same as the encoder in the [12]. The loss function is entropy, which will be discussed in detail in Section IV-B. The optimizer is adam optimizer with an initial learning rate 0.0003 and a learning rate decay strategy. The strategy is that the learning rate will be half if the current epoch's valid loss is bigger than the last epoch. Max training epoch is 50 for the small train and 5 for the full train. If the valid loss of the current epoch has not been decreased for last 4 epoch, the training process will be stopped early.

IV. EXPERIMENT

In this section, we will give a description of our experiment scenarios and evaluation metrics. In these experiments, we used the popular deep learning framework Tensorflow. We implemented and evaluated the proposed model on a Linux PC with an Intel i7-5960X processor @3.0 GHz and an NVIDIA GTX 1080Ti GPU.

A. Scenarios

In the code completion task, the next token is predicted based on current tokens. Whether tokens in the current file
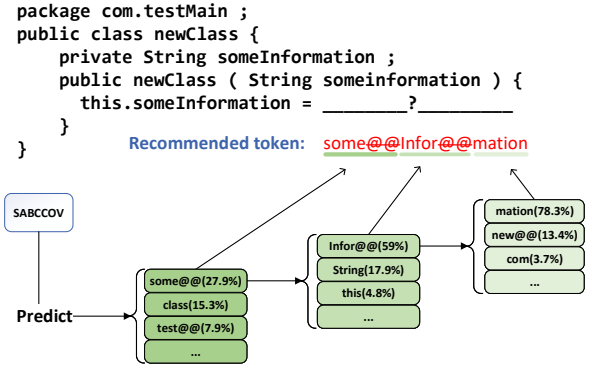

Fig. 3. An example of our model's prediction.

or other files that belong to the same project can be trained for the model is a question. Followed [10], there are also 3 test scenarios including Static, Dynamic, Maintenance in our experiments to check the performance of our model.

Static: This is the basic mode. The model is trained by the large corpus from the internet only. It provides code completion service directly without changed or improved in further usage. In this scenario, local code will not be used for the further training of the model. It may not guarantee to provide a high-quality service of the code completion because everyone has a different code style. The model learned from the large corpus is only a generalization ability.

Dynamic: In this mode, the model has two stage training prcess. The second stage is that, based on the Static mode, the model will be trained by reading the code from the file that the user is editing currently. We denoted it as a local model. With the Dynamic strategy, the model will learn the habits of the coder gradually and prefer to predict the token which reflects the style of the coder finally. The Dynamic strategy here is that the model will read every 512 sub-tokens and executes one step gradient descent to update the model. This mode will have a better performance than the Static mode. However, it will read data that may not be allowed by the user because of security and privacy.

Maintenance: This mode has three stage training process. Based on the Static mode, the model here will use codes in the current project files firstly, which is the second training stage. Lastly, just like Dynamic mode, the model will be trained by reading the code user is editing currently. It is obvious that the model trained in this mode has the best performance than the others since it obtains the largest data set. The disadvantage of this mode is that uploading data in projects may be strictly forbidden in most enterprises.

B. Evaluation Metrics

In our experiments, we evaluated both the intrinsic and extrinsic performance of our model. Intrinsic performance means that evaluating the predicting ability of a language model without other task's inference which may bring some other information. We use entropy as the intrinsic metric, which is a standard measure employed in the previous work. Given a sequence $S = \{t_1, t_2, \ldots, t_{\{|m|\}}\}$, the probability of

TABLE II
ENTROPY RELFECTS THAT THE LOWER ENTROPY, THE MORE REASONABLE THE TOKEN PREDICTED BY THE MODEL.

Entropy		Nested Cache N-gram	Open Vocabulary NLM [10](2k/5k/10k)	SABCCOV(2k/5k/10k)
Small Train	Static	-	4.90 / 4.78 / 4.77	2.62 / **2.57** / 2.63
	Dynamic	2.57	2.33 / 2.27 / 2.54	1.59 / **1.58** / 1.61
	Maintenance	2.23	1.46 / 1.51 / 1.60	1.32 / **1.29** / 1.29
Full Train	Static	-	3.59 / 3.35 / 3.15	1.84 / 1.79 / **1.74**
	Dynamic	2.49	1.84 / 1.72 / 1.70	1.22 / 1.19 / **1.17**
	Maintenance	2.17	1.03 / 1.06 / 1.04	1.08 / 1.02 / **0.99**

TABLE III
MRR REFLECTS THE INVERSE OF THE AVERAGE EXPECTED POSITION IN THE RANK LIST.

MRR		Nested Cache N-gram	Open Vocabulary NLM [10](2k/5k/10k)	SABCCOV(2k/5k/10k)
Small Train	Static	-	62.87% / 63.80% / 63.75%	71.54% / **71.79%** / 71.59%
	Dynamic	74.55%	76.94% / 77.51% / 77.32%	77.54% / **77.72%** / 77.45%
	Maintenance	77.04%	77.48% / 78.49% / 78.69%	78.15% / 78.94% / **78.97%**
Full Train	Static	-	68.69% / 69.87% / 70.84%	77.88% / 78.42% / **78.81%**
	Dynamic	75.0%	78.99% / 79.88% / 80.36%	81.99% / 82.51% / **82.88%**
	Maintenance	77.3%	78.85% / 80.31% / 81.16%	80.71% / 81.78% / **82.30%**

t_i is estimated by $p(t_i|t_1,\ldots,t_{i-1})$. The entropy is defined as:

$$H_p(s) = -\frac{1}{|m|}\sum_{i=1}^{|m|} log\, p(t_i|t_1,\ldots,t_{i-1}) \qquad (2)$$

Entropy corresponds to the average number of bits required in every prediction. But our model predicts a sub-token other than a complete token, we follow [10] and change the subformula in Equation 2 as follows:

$$p(t_i|t_1,\ldots,t_{i-1}) = \prod_{n=1}^{N} p(w_{in}|t_1,\ldots,t_{i-1},w_{i1},\ldots,w_{i,n-1}) \quad (3)$$

In Equation 3, we assume that the token t_i is split into sub-tokens $t_i = \{w_{i1},...,w_{iN}\}$. The combination of Equation 2 and Equation 3 is the loss function of the model to check whether the model is converging.

The extrinsic performance here we use is Mean Reciprocal Rank (MRR). The reciprocal rank of a query response is the multiplicative inverse of the rank of the first correct answer. MRR is the average of reciprocal ranks or results for a sample of queries Q defined as:

$$MRR = \frac{1}{|Q|}\sum_{i=1}^{|Q|}\frac{1}{rank_i} \qquad (4)$$

For example, if the correct word ranks first in the option list, then MRR is 1. If the correct word ranks second, the MRR is 0.5 and so on. In our experiments, we provide ten most likely tokens for users to choose, which means the length of the options list is 10, so the MRR of the predicted word is at least 0.1 and the unpredicted word is 0.

C. Results

The results demonstrate that our model is particularly better than the N-gram model [4] and RNN based Open Vocabulary NLM [10]. Our model has a faster training speed as shown in Table IV. In the small training dataset, one training epoch time of our model was almost half of the Open Vocab NLM's

training time. As for the full training dataset, our model spent an hour less than the previous work every epoch. Open Vocabulary NLM only has one layer of RNN but our model has three transformer blocks, which means our model has more parameters and is more complicated. But our model was still a lot faster than their model. It is noted that the model with a faster training speed is always the first choice when performance is the same.

Furthermore, our model has not only a fast training speed but also a better performance. Table II and III shows that performance comparison statistics of the Open Vocabulary NLM, the Nested Cache N-gram model and our model. In the metric of entropy, our model with 5k BPE operations had the best results among three models in the small training and our model with 10k BPE operations won in the full training no matter in which scenarios. Due to the large number of samples, even minor improvements (e.g., 0.01 bits) in entropy can be statistically significant in language modeling. It was almost same in the metric of MRR. In Table III, there was a great improvement in the Static scenario between our model and Open Vocabulary NLM and a slight improvement in Dynamic and Maintenance scenarios. It was declared in Section IV-A that Dynamic and Maintenance scenarios need the authorization of reading user code, which may be seen as an invasion of privacy. Under normal circumstances, the Static scenario is the most favored option of users. So it is meaningful to improve the performance of the model in the Static scenario.

TABLE IV
TRAINING TIME(MIN/EPOCH)

	BPE Operations	2k	5k	10k
Small Train	Open Vocab NLM	18	16	16
	Our model	**8**	**7**	**8**
Full Train	Open Vocab NLM	717	669	728
	Our model	**561**	**555**	**628**

V. RELATED WORK

Code completion is a basic feature of IDE for a long time. Traditionally, code completion in IDE relies heavily on compile-time type information to predict the next token [1]. The deep learning method finds a way that learns the probability distribution of tokens from a large source code corpora to improve the accuracy of token prediction. In 2012, Hindle et al. [3] first proposed that programming languages have usefully predictable statical properties that can be captured in statistical language models. Based on Hindle's work, Tu et al. [1] put forward that source code has the property of localness and proposed a cache mechanism to improve prediction accuracy. Hellendoorn et al. [4] enhanced established language modeling approaches to handle the special challenges of modeling source code.

Li et al. [2] and Liu et al. [16] proposed that structural information can also be used to improve the performance of the model. They predicted the terminal node and the non-terminal node in source code using the AST tree. It is critical that the AST tree needed to guarantee both semantic information and structural information when modeling the AST tree. In this method, they can predict not only the token itself, namely the terminal node, but also the token's structural information and type which is a non-terminal node. In 2016, Raychev et al. [17] used the decision tree to model the AST sequence of code to predict token. Liu et al. [16] showed that structural information in AST and sequences of the token can learn mutually and get better results with multi-task learning.

Out of vocabulary words used to be called neologisms [9], which means unseen identifier names that have not been used in the training set. In [9], they split OOV words on camel case and underscores and could only handle part of neologisms. And [2] tried to solve the OOV problem by augmenting an RNN with a pointer network [18]. Some researches focused on the techniques for automatic splitting identifiers [19], [20]. One obvious feature of their splitting technique is that human can understand the sub-identifers after splitting. Instead, Karampatsis et al. [10] first proposed Open Vocabulary that split OOV words to incomprehensible sub-words. Based on their technique, we proposed a novel network in this paper to improve the performance and the training speed.

VI. CONCLUSIONS AND FUTURE WORK

In this paper, we proposed a self-attention based code completion model with Open Vocabulary. First, we alleviated the OOV problem by adopting the Open Vocabulary mechanism. Second, We added the self-attention mechanism to address the issue of sequence augment brought by the Open Vocabulary method. In the meantime, we speeded up the training process and improved the performance of the model. To our best knowledge, we are the first to combine the self-attention with the Open Vocabulary mechanisms and get significant results in the code completion field. Besides, we believe our model may inspire other researchers in the source code modeling field. Our embedding layer is trained during the training process now. In the future, we may train the embedding layer using large corpora in advance. Then we insert it directly into the model and fine tune this layer during the training process. And we also plan to improve our model not to predict only one token but a series of tokens that can present user intent.

REFERENCES

[1] Z. Tu, Z. Su, and P. Devanbu, "On the localness of software," in *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, 2014, pp. 269–280.

[2] J. Li, Y. Wang, M. R. Lyu, and I. King, "Code completion with neural attention and pointer networks," *arXiv preprint arXiv:1711.09573*, 2017.

[3] A. Hindle, E. T. Barr, Z. Su, M. Gabel, and P. Devanbu, "On the naturalness of software," in *2012 34th International Conference on Software Engineering (ICSE)*. IEEE, 2012, pp. 837–847.

[4] V. J. Hellendoorn and P. Devanbu, "Are deep neural networks the best choice for modeling source code?" in *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*, 2017, pp. 763–773.

[5] T. T. Nguyen, A. T. Nguyen, H. A. Nguyen, and T. N. Nguyen, "A statistical semantic language model for source code," in *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering*, 2013, pp. 532–542.

[6] M. White, C. Vendome, M. Linares-Vásquez, and D. Poshyvanyk, "Toward deep learning software repositories," in *2015 IEEE/ACM 12th Working Conference on Mining Software Repositories*. IEEE, 2015, pp. 334–345.

[7] M. Sundermeyer, H. Ney, and R. Schlüter, "From feedforward to recurrent lstm neural networks for language modeling," *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, vol. 23, no. 3, pp. 517–529, 2015.

[8] H. K. Dam, T. Tran, and T. Pham, "A deep language model for software code," *arXiv preprint arXiv:1608.02715*, 2016.

[9] M. Allamanis, E. T. Barr, C. Bird, and C. Sutton, "Suggesting accurate method and class names," in *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, 2015, pp. 38–49.

[10] R.-M. Karampatsis, H. Babii, R. Robbes, C. Sutton, and A. Janes, "Big code != big vocabulary: Open-vocabulary models for source code," in *International Conference on Software Engineering (ICSE)*, 2020.

[11] U. Khandelwal, H. He, P. Qi, and D. Jurafsky, "Sharp nearby, fuzzy far away: How neural language models use context," *arXiv preprint arXiv:1805.04623*, 2018.

[12] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in neural information processing systems*, 2017, pp. 5998–6008.

[13] M. Allamanis and C. Sutton, "Mining source code repositories at massive scale using language modeling," in *2013 10th Working Conference on Mining Software Repositories (MSR)*. IEEE, 2013, pp. 207–216.

[14] P. Gage, "A new algorithm for data compression," *The C Users Journal*, vol. 12, no. 2, pp. 23–38, 1994.

[15] R. Sennrich, B. Haddow, and A. Birch, "Neural machine translation of rare words with subword units," *arXiv preprint arXiv:1508.07909*, 2015.

[16] F. Liu, G. Li, B. Wei, X. Xia, M. Li, Z. Fu, and Z. Jin, "A self-attentional neural architecture for code completion with multi-task learning," *arXiv preprint arXiv:1909.06983*, 2019.

[17] V. Raychev, P. Bielik, and M. Vechev, "Probabilistic model for code with decision trees," *ACM SIGPLAN Notices*, vol. 51, no. 10, pp. 731–747, 2016.

[18] O. Vinyals, M. Fortunato, and N. Jaitly, "Pointer networks," in *Advances in neural information processing systems*, 2015, pp. 2692–2700.

[19] E. Enslen, E. Hill, L. Pollock, and K. Vijay-Shanker, "Mining source code to automatically split identifiers for software analysis," in *2009 6th IEEE International Working Conference on Mining Software Repositories*. IEEE, 2009, pp. 71–80.

[20] E. Hill, D. Binkley, D. Lawrie, L. Pollock, and K. Vijay-Shanker, "An empirical study of identifier splitting techniques," *Empirical Software Engineering*, vol. 19, no. 6, pp. 1754–1780, 2014.

An Ensemble Approach to Detect Code Comment Inconsistencies using Topic Modeling

Fazle Rabbi, Md. Nazmul Haque, Md. Eusha Kadir, Md. Saeed Siddik and Ahmedul Kabir

Institute of Information Technology

University of Dhaka, Bangladesh

Email: {bsse0725, bsse0635, bsse0708, saeed.siddik, and kabir}@iit.du.ac.bd

Abstract—In modern era, the size of software is increasing, as a result a large number of software developers are assigned into software projects. To have a better understanding about source codes these developers are highly dependent on code comments. However, comments and source codes are often inconsistent in a software project because keeping comments up-to-date is often neglected. Since these comments are written in natural language and consist of context related topics from source codes, manual inspection is needed to ensure the quality of the comment associated with the corresponding code. Existing approaches consider entire texts as feature, which fail to capture dominant topics to build the bridge between comments and its corresponding code. In this paper, an effective approach has been proposed to automatically extract dominant topics as well as to identify the consistency between a code snippet and its corresponding comment. This approach is evaluated with a benchmark dataset containing 2.8K Java code-comment pairs, which showed that proposed approach has achieved better performance with respect to the several evaluation metrics than the existing state-of-the-art Support Vector Machine on vector space model.

Index Terms—Source Code, Code Comment, Topic Modeling, Software Artifact Analysis

I. INTRODUCTION

Code comments with its corresponding source code are the main artifact of any software systems. For the management of software evolution and maintenance, developers provide comments with a code fragment which give insightful information about a software system. Comments are very important as they are more natural, descriptive and easy to understand than source code [1], [2]. In large projects, new developers are highly dependent on code comments to understand its corresponding source codes. Researchers found that code and comments evolve over time [3] and this evolved codes and comments become inconsistent to each other. Because of changing codes frequently and keeping corresponding comments same, comments become invalid or inconsistent with corresponding source code.

Tracking the inconsistency of source code and its comment, several diverse approaches have been proposed. Where most of the approaches apply Information Retrieval (IR) techniques to collect lexical information with the assumption that the textual information of source code and comment are same. However, that assumption can be violated [4] in several cases, for example, the vocabulary developers use to write source

code can be different from the vocabulary of comment (e.g. synonym). Nevertheless, there is no sufficiently rich literature to track this inconsistency because of lacking standard datasets. A benchmark dataset has been provided [5] with a proposal to measure the coherence between source code and comment. Lexical similarity has been collected by using Vector Space Model to classify the text using tf-idf [6] and finally the code-comment inconsistency is measured using Support Vector Machine (SVM). However, this approach uses all of the vocabulary as features which can take a huge execution time.

By analyzing existing literature, some insights of source code and comments have been found, which are concluded below as the research direction in this domain.

- A single word (topic) is more important than a large number of similar words (features). For example, if a bag of words is found from a java method like, "dropdown", "chrome", "menu", "http" or "browser", a topic related to "browser" can represent these words.
- The size of comments is less than the size of source code. So, the source code and comment need to be represented into a fixed-sized common topic.
- Synonymous words have been chosen by developers while writing comment with respect to source code. So, to capture the semantic information between source code and comment, the vocabulary information needs to be incorporated.

To capture these insightful information, several Research Questions (RQ) have been raised to propose an efficient inconsistency detection approach, which are listed below.

- **RQ1:** How to comprehend the insight meaning of a code and comment pair?
- **RQ2:** How to measure the relation between the code and comment pair?

We focused on the above research questions as our objectives and tried to answer them throughout the newly proposed code comment inconsistency detection technique. This paper proposes an automated approach to identify the inconsistency of source code with its respective comments. The breakdown of the contributions of this paper are listed as follows.

- Datasets are pre-processed to capture more meaningful information about source code and comments, e.g., de-

velopers defined simple name.

- Latent Dirichlet Allocation (LDA) has been used for representing the similar words into topics.
- A fusion approach (ensemble Random Forest) has been proposed for measuring the probability of the inconsistency between code and comments, where SVM is used to discriminate consistent and inconsistent comments.
- The proposed approach has been compared with state of the art baseline classification approaches and it is evident that this approach performs better in terms of Accuracy and Area Under Precision-Recall Curve (AUCPR).

In section II, an overview of the methods are briefed which are important to understand the proposed method. Dataset parsing and pre-processing is discussed in section III-A. Proposed method is explained in section III-B. The dataset description and experimental results are presented in section IV. Related works are reviewed in section V, and finally this work has been concluded in section VII.

II. BACKGROUND

To understand the proposed approach, knowledge about Topic Modeling, Random Forest (RF) and Support Vector Machine (SVM) is needed which are briefly discussed here.

A. Topic Modeling

Topic Modeling is a subfield of Machine Learning and Natural Language Processing. It is one type of statistical model which follows unsupervised machine learning technique to provide abstract topics for a given document. Latent Dirichlet Allocation (LDA) is a type of topic modeling which is used in this paper. LDA is trained using a set of documents and with a given number of topics. It provides a probability distribution of words for a topic and a probability distribution of topics for a document as output.

B. Random Forest

Random Forest [7] is an ensemble learning approach for classifying data. In training time, it builds a multitude of decision trees. Each decision tree predicts a class label of a new input data pattern and RF merges them together to get a more accurate and stable prediction. RF is a fast, simple and flexible machine learning algorithm. In this paper RF receives the topic distribution gained from LDA as input and produces output for the next procedure.

C. Support Vector Machine

A Support Vector Machine [8] is a discriminative classifier formally defined by a separating hyperplane. While training this classifier, it finds the maximum-margin hyperplane that separates the group of data points into two classes. A new incoming pattern is classified in the class according to the side of the hyperplane.

III. PROPOSED APPROACH

The proposed ensemble approach to detect code comment inconsistency is described in this section thoroughly. Before training, the way of pre-processing code and comment pairs is also discussed here.
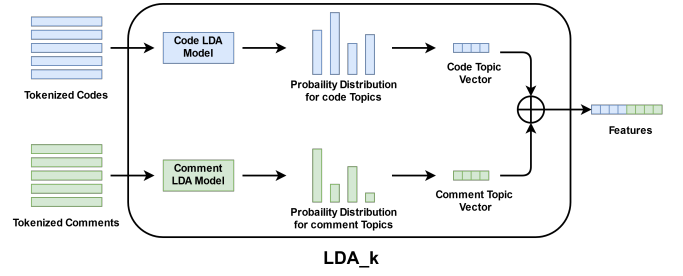

Fig. 1. Topic Modeling (LDA).

A. Code-Comment Parsing

The raw code comment pairs need to be processed to create features for training. The two steps which are followed to make the features during pre-processing are described next.

1) Process Code and Comment Pairs: The pairs of codes and comments need to be parsed into tokens to execute the next steps. At the beginning, all newlines, tabs and special characters like braces, semicolons, full-stops are removed. Extra whitespaces are also removed from the remaining code and comments. Words are also split based on camel cases. After tokenizing, every code and comment is turned into a bag of word tokens. Finally each of the words are lemmatized into root words.

2) Create vectors from code and comments: After processing the code-comment pairs into bag of word tokens, two corpora for codes and comments are built. Each of these corpora turns a code/comment into an index vector. These index vectors of code/comment is passed into its corresponded Latent Dirichlet Allocation (LDA) model. There are two identical LDA models for training code and comment index vectors separately. For every k number of topic, these LDA models provide two separate probability distributions for a pair of code and comment which are concatenated to produce feature vector using Eq. (1).

$$concatenated feature_k = LDA_k(code) \oplus LDA_k(comment) \quad (1)$$

Here, k is the number of topics and $\oplus$ is used for concatenating vectors. This features vector is now ready to be used for Inconsistency Detector described in the next section. The overview of preparing feature vector is illustrated in Fig. 1.

B. Inconsistency Detector

As discussed above, the extracted feature vector of k number of topics is fed into a random forest as input. Based on this input features, the random forest model produces a consistency score for a code-comment pair. A random forest built from k topics, RF_k, provides a consistency score, $score_k$ derived in Eq. (2).

$$score_k = RF_k(concatenated feature_k) \quad (2)$$

As the number of topics for both comment and code can be varied, it is needed to incorporate different number of topics to find the informative and effective features. For different
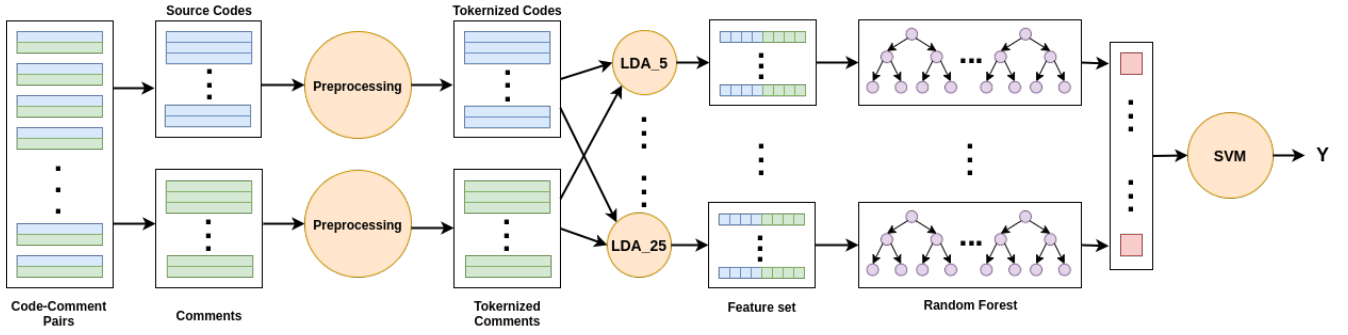
Fig. 2. Overall process of the proposed method.

TABLE I
DESCRIPTIVE STATISTICS OF THE DATASET

Application	Files	Classes	Methods	Methods with comments	Coherent	Non-Coherent	Total	Not Included
CoffeeMaker	7	7	51	47 (92%)	27	20	47	0
JFreeChart-0.6.0	82	83	617	485 (79%)	406	55	461	24
JFreeChart-0.7.1	124	127	807	624 (77%)	520	68	588	36
JHotDraw-7.4.1	575	692	6414	2480 (39%)	762	1025	1787	693
All	788	909	7889	3636 (46%)	1715	1168	2883	753

number of topics ranging from i to j on interval 1, different random forest models are built. Each random forest model returns a consistency score for a code-comment pair. These consistency scores derived from m different random forests are needed to be fused. Here, SVM is used to fuse the consistency scores provided by m random forests and predict the outcome using Eq. (3).

$$\hat{Y} = SVM(\oplus_{k=i}^{j} score_k) \qquad (3)$$

Fig. 2 describes the overall procedure of proposed method.

IV. EXPERIMENTS & RESULT ANALYSIS

We used a dataset provided by Corazza et al. [9] in this experiment. Four versions of three java projects CoffeeMaker[1], JFreeChart[2] and JHotDraw[3] are used in this dataset. Some descriptive statistics of these projects are reported in Table I.

We split the dataset randomly into 90% training and 10% testing set. After training finished, we run our model on testing set. The result is evaluated based on two evaluation metrics namely Accuracy and Area Under Precision-Recall Curve (AUCPR). Table II and III report the performance comparison of our proposed method with other methods in terms of accuracy and AUCPR. We report the average performance by applying 10 fold cross-validation. To find the outputs of the existing approach, we re-implemented it.

From Table II, it can be observed that, the accuracy of the proposed method is better than the existing approach [5] for all of the projects except JHotDraw-7.4.1. At first a single Random Forest having 10 features (RF_10) is used to classify consistent and inconsistent code-comment pairs. As the result

[1]agile.csc.ncsu.edu/SEMaterials/tutorials/coffee_maker
[2]www.jfree.org/jfreechart
[3]www.jhotdraw.org

TABLE II
PERFORMANCE COMPARISON IN TERMS OF ACCURACY

Dataset	RF-10	Coherence	Proposed
CoffeeMaker	0.830	0.873	0.895
JFreeChart-0.6.0	0.879	0.835	0.918
JFreeChart-0.7.1	0.876	0.875	0.898
JHotDraw-7.4.1	0.743	0.811	0.801
All	0.803	0.837	0.841

of this approach was not satisfactory, an ensemble RF with SVM (Proposed) is used. By using this, the accuracy increases for all of the projects and looks very promising.

In Table III, the result is showed based on AUCPR which also denotes that, the result of the proposed approach performs better than the existing one. It can also be observed that, AUCPR values are improved when using the ensemble of RFs for different number of topics instead of a single Random Forest with constant number of topics (e.g. RF_10).

TABLE III
PERFORMANCE COMPARISON IN TERMS OF AUCPR

Dataset	RF-10	Coherence	Proposed
CoffeeMaker	0.878	0.943	0.975
JFreeChart-0.6.0	0.938	0.909	0.941
JFreeChart-0.7.1	0.924	0.941	0.962
JHotDraw-7.4.1	0.837	0.882	0.855
All	0.888	0.888	0.912

We also found that the proposed approach is promising while comparing the training time with state-of-the-art method. For small projects training time is almost same in both cases. However, the proposed approach trains faster than the existing approach for large projects.

V. Related Works

There have been some previous works in this field related to code comment relation. The earliest work related to code comment inconsistency that we studied is "iComment: Bugs or Bad Comments" [10]. In this work, authors proposed an approach to detect code comment inconsistency in locking and calling mechanism. They limited their scope to the comments related to programmers' assumptions and requirements.

Another work for testing Javadoc comments was proposed by Tan et al. to detect comment-code inconsistencies called @TCOMMENT [11]. The authors considered method properties for null values and related exceptions. They set some rules for @param tags in javadoc comments and null pararamer exception statement to detect inconsistencies between codes and comments using Natural Language Processing. The scope of their work is limited to only comments related to null reference and throwing exceptions.

Ratol et al. proposed an approach to detect invalid comments while renaming identifiers in source code [12]. The authors created guidelines to link comments and their responsive codes and defined the scope of comments in a project to link identifiers.

While the above works are related to detect inconsistencies between code and comment, there are some other works to measure the code comments quality. Steidl et al. presented a semi automatic approach for quality analysis and assessment of code comments [13]. Their focus was to evaluate comments quality to improve the readability of source codes. They used machine learning technique to classify comments into categories and based on these categories they developed a comment quality model.

Corazza et al. published a benchmark dataset of java method-comment pairs with corresponding coherent values which they inspected manually [5]. Later they investigated if it is possible to predict whether a code-comment pair is coherent or not. They used Vector Space Model to represent a method or comment based on their tf-idf score. Initially they used lexical similarity to measure the coherence value and later they trained a SVM with grid search algorithm to adjust parameters.

Wen et al. presented a large scale study [14] on code comments and found that, code and comments co-evolve over time. Besides, some approaches are proposed to generate natural language summary or comments from source codes [15], [16]. As per our knowledge, none of the approaches use the insight dominating topics to detect if a code and comment pair conveys the same meaning or not.

VI. Threats to Validity

The most important threat is external validity which is related to the software applications considered in the dataset. All the applications in experimented dataset were implemented in Java which could bias the results. For example, Java is more verbose than other programming languages (e.g., C, C++ etc.) and then the developers of the applications in the dataset could have paid inadequate attention on commenting methods.

VII. Conclusion

This paper proposed a new ensemble approach to measure the effectiveness of detecting code comment inconsistencies. In this approach, features are extracted from codes and comments using topic modeling. Afterwards, proposed model fuses the coherence scores obtained by different sources (Random Forests) to provide the probability of inconsistency between a code and comment. This approach was evaluated in a benchmark dataset of java projects and the result was prominent and satisfactory. This approach can also be applied to detect inconsistencies between code comment pairs of other languages as a future work.

References

[1] T. Tenny, "Program readability: Procedures versus comments," *IEEE Transactions on Software Engineering*, vol. 14, no. 9, pp. 1271–1279, 1988.

[2] S. N. Woodfield, H. E. Dunsmore, and V. Y. Shen, "The effect of modularization and comments on program comprehension," in *Proceedings of the 5th international conference on Software engineering*. IEEE Press, 1981, pp. 215–223.

[3] B. Fluri, M. Wursch, and H. C. Gall, "Do code and comments co-evolve? on the relation between source code and comment changes," in *14th Working Conference on Reverse Engineering (WCRE 2007)*. IEEE, 2007, pp. 70–79.

[4] D. Lawrie, D. Binkley, and C. Morrell, "Normalizing source code vocabulary," in *2010 17th Working Conference on Reverse Engineering*. IEEE, 2010, pp. 3–12.

[5] A. Corazza, V. Maggio, and G. Scanniello, "Coherence of comments and method implementations: a dataset and an empirical investigation," *Software Quality Journal*, vol. 26, no. 2, pp. 751–777, 2018.

[6] H. Schütze, C. D. Manning, and P. Raghavan, *Introduction to information retrieval*. Cambridge University Press Cambridge, 2008, vol. 39.

[7] L. Breiman, "Random forests," *Machine learning*, vol. 45, no. 1, pp. 5–32, 2001.

[8] C. Cortes and V. Vapnik, "Support-vector networks," *Machine Learning*, vol. 20, no. 3, pp. 273–297, 1995. [Online]. Available: https://doi.org/10.1007/BF00994018

[9] A. Corazza, V. Maggio, and G. Scanniello, "On the coherence between comments and implementations in source code," in *2015 41st Euromicro Conference on Software Engineering and Advanced Applications*. IEEE, 2015, pp. 76–83.

[10] L. Tan, D. Yuan, G. Krishna, and Y. Zhou, "/* icomment: Bugs or bad comments?*," in *ACM SIGOPS Operating Systems Review*, vol. 41, no. 6. ACM, 2007, pp. 145–158.

[11] S. H. Tan, D. Marinov, L. Tan, and G. T. Leavens, "@ tcomment: Testing javadoc comments to detect comment-code inconsistencies," in *2012 IEEE Fifth International Conference on Software Testing, Verification and Validation*. IEEE, 2012, pp. 260–269.

[12] I. K. Ratol and M. P. Robillard, "Detecting fragile comments," in *Proceedings of the 32Nd IEEE/ACM International Conference on Automated Software Engineering*. IEEE Press, 2017, pp. 112–122.

[13] D. Steidl, B. Hummel, and E. Juergens, "Quality analysis of source code comments," in *2013 21st International Conference on Program Comprehension (ICPC)*. Ieee, 2013, pp. 83–92.

[14] F. Wen, C. Nagy, G. Bavota, and M. Lanza, "A large-scale empirical study on code-comment inconsistencies," in *2019 IEEE/ACM 27th International Conference on Program Comprehension (ICPC)*. IEEE, 2019, pp. 53–64.

[15] S. Iyer, I. Konstas, A. Cheung, and L. Zettlemoyer, "Summarizing source code using a neural attention model," in *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2016, pp. 2073–2083.

[16] E. Wong, T. Liu, and L. Tan, "Clocom: Mining existing source code for automatic comment generation," in *2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*. IEEE, 2015, pp. 380–389.

A Combined Model for Extractive and Abstractive Summarization Based on Transformer Model

Xin Liu, Liutong Xu

The Academy of Computer Science and Teconology, Beijing University of Post and Telecommunication
Beijing, China
liuxiaoxin@bupt.edu.cn, xliutong@bupt.edu.cn

Abstract—Summary generates by summarizing automatically main information from the critical sentences of the article. The traditional method of generating text summarization uses extractive or abstractive algorithm model built based on neural attention sequence to sequence framework. This kind of model has performance bug and weak parallel computing capability when getting summary, which causes the summary doesn't fit the meaning of the original and has no smooth sentences. Therefore, we put up with a joint summary generation model based on improving transformer. This model can put attention on sentence and provide sequence information for periodical transformer model by recurrent neural network. On the other hand, in the generation stage, Transformer model is used to learn the long distance dependence between words, and the summary statement is more consistent with the original meaning by adding pointer mechanism and consistency loss function. Experiments were carried out on three datasets and a manual evaluation was added to verify that the model has good summary significance.

Index Terms—Text Summarization , Transformer, Sequence Information, Joint Model

I. INTRODUCTION

Summary generation aims to get a simplified input text representation to capture the core meaning of the original content. There are two types of methods: extractive and abstractive. Extractive methods usually selects the original sentence or word [1], such as Lead3, Summarunner [2], Swap-Net [3] model. The summary obtained by these models are not smooth due to the lack of connectives. Abstractive methods can generate new words and phrases that are not included in the source text.But the summary has incorrect fact details and duplicate information, and words that are out of vocabulary. In recent years, the pointer generator model proposed by see et al [4] which has the ability to extract words from the original text and reduce the repetition rate. Hsu et al. [5] proposed the inconsistent loss function which combines extractive methods and abstractive methods.

The transformer model proposed by Ashish et al [9] is effective for capturing the global context semantic relationship and parallel computing. In this paper, we proposed TP-EABS (Transformer added Pointer and combine the Extractive and Abstractive methods) model. It adopted the advantages of two types of algorithms and transformer model. The model uses the hidden layer information of GRU to supplement the sequence

information of transformer position, and dynamically adjusts the attention of words in the second phase through sentence level attention, so as to reduce the probability of words in sentences with lower weight appearing in the abstract. And we add a pointer mechanism to the transformer model, which enables the transformer to copy words from the original text.

In summary, our contributions are as follows:
- We propose a joint model based on improved Transformer.
- We improve the Transformer architecture by adding position information and pointer mechanisms.
- We have made comparative experiments on CNN/Daily, Papers and DUC-2004 datasets, and the results have been improved.

II. RELATED WORK

In recent years, summary generation has been widely studied. Generally,In the extraction method, key sentences or words in the original text are extracted and presented as abstracts. [3] and [4], [6] used recurrent neural network to code the text, and then mark whether the sentence or word belongs to the summary statement. Although some extraction methods [10] can get high Rouge scores, their readability is very low.

Abstrctive methods are mostly based on the sequence to sequence framework based on neural attention [11], [14]. [12] proposed a new model, which first selects key sentences, then rewrites them with abstract algorithm to generate a brief summary of the text. [11] proposed a new model, which can not only retain the ability to generate new words, but also copy words from the original text accurately to reproduce information, and reduce the repetition rate of the generated words in the summary by updating the attention weight. [9] proposed Transformer, which is completely based on attention mechanism and eliminates recursion and convolution. It can solve the problem of long-distance dependence and realize parallel computing. It can obtain the text semantic information and structural information better.

III. OUR MODEL

This chapter introduces three aspects: sentence extraction model, generating model, dynamic word-level mechanism. Fig. 1 gives the overview of TP-EABS model.
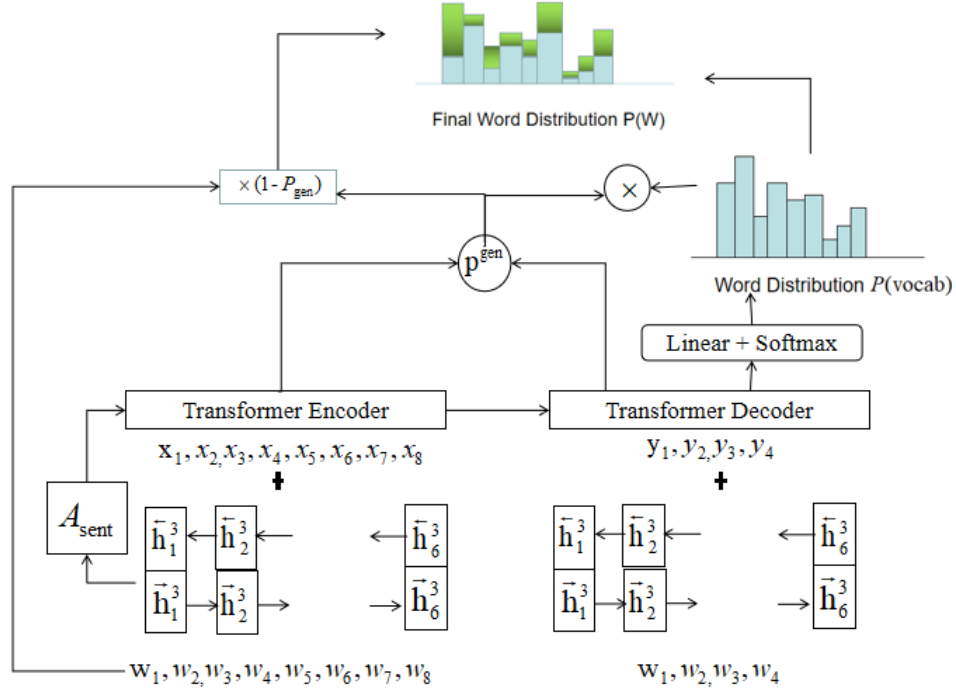
Fig. 1: our model

A. Sentence Extraction Model

The model input is a series of sentences $S = [s_1, s_2, ..., s_m]$, where the representative m is the mth sentence and s_i is the ith sentence, expressed as $s_i = [w_1, w_2, ..., w_n]$, the input sentence maps each word to a vector through the language training model, and the i-th sentence is expressed as $s_i = [x_1, x_2, ..., x_n]$, Where n represents the nth word embedding vector. we use BiGRU (Bi-directional Gated Recurrent Unit) to process the input word sequence.After reading the words of the sentence, we update its word representation $x_i^j = \left[\overrightarrow{h_i^j}; \overleftarrow{h_i^j}\right]$. We use matrix X to represent the input vector.Get the sentence vector by summing the word vectors in the sentence.

Then the sentence vector is input to the second layer of BiGRU, and then calculated by the sigmoid function to obtain sentence-level weight β_n, The extractor loss is calculated using the following cross-entropy loss.

$$L_{ext} = -\frac{1}{n}\sum_{n=1}^{N}(g_n log\beta_n + (1 - g_n)log(1 - \beta_n)) \quad (1)$$

In the above formula, g_n is the n-th sentence a summary, the value belongs to 1 and the value does not belong to 0.To get ground truth labels $g = \{g_n\}_n$.We use the unsupervised method proposed by Nillan et al. [5] to get extracted labels.

B. Generating Model

The benchmark model in this paper uses Transformer. We will extract the vector $Z = [z_1, z_2, ..., z_m, ...]$ of the first layer GRU encoding output in the model as the input of the Transformer layer, we think z_i contains word information and location information.

$$[Q; K; V] = W_Z \cdot Z + B_Z \quad (2)$$

$$Attention(Q, K, V) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V \quad (3)$$

Where W_Z and B_Z are training parameters. We obtain the query, key, and value vectors through Equation 10, d_k represents the size of the key value.

$$MultiHead(Q, K, V) = Concat(head_1, ..., head_h)W^O \quad (4)$$

$$where \quad head_i = Attention\left(QW_i^Q, KW_i^K, VW_i^V\right) \quad (5)$$

We represent the output of the Encoder layer as E and denote the output of the Decoder layer as D.We calculate the attention weight between the encoder and decoder and take it out to calculate the probability p_{gen}, which determines whether the word is copied from the original text or generated in the dictionary.Enter the last layer of the decoder into softmax to get the probability of getting words from the vocabulary p_{vocab}.In the end, we use Beam-search with the Beam-size set to 3.

$$P_{voacb} = softmax(V'(V \cdot D_4 + b) + b') \quad (6)$$

$$[K_E; V_E] = W_E \cdot E + B_E \quad (7)$$

$$Q_d = W_d \cdot D_1 + B_d \quad (8)$$

$$C_t = MulAttn(Q_d, K_E, V_E) \quad (9)$$

We use the following formula to calculate p_{gen}, where X is the input vector and C is the context text. The pointer

generator network is a hybrid between our baseline and the pointer network, because it can both copy words by pointing and generate words from a fixed vocabulary.

$$p_{gen} = \sigma(W_e^T E + W_d^T D + W_C^T C_t + b_{ptr}) \qquad (10)$$

Next, we calculate the probability of the final word through the probability p_{vocab} and p_{gen}, where C_t is the output of the encoder and decoder mutual attention matrix at time t.

$$P(w) = p_{gen} P_{vocab} + (1 - p_{gen}) C_t \qquad (11)$$

$loss_t$ represents the loss function at time t, and L_{abs} represents the loss function of the generated model part.

$$loss_t = -log P(W_t^*) \qquad (12)$$

$$L_{abs} = \frac{1}{T} \sum_{t=0}^{T} loss_t \qquad (13)$$

C. Dynamic word-level mechanism

The dynamic word-level mechanism is to reduce the word-level attention through the sentence's attention weight, so that the generative summary can pay more attention to a certain sentence to generate a word, which also makes the sentence weights of the same information different, which reduces the repeatability of the generated words to a certain probability.

This article uses a BiGRU to obtain sentence-level weights in the sentence extraction model. It needs to be added to the word-level weights. The obtained sentence-level weights are the matrix A_{sent} formed by β, and a sentence vector $\tilde{E}_4^t$ obtained by multiplying the word-level attention matrix and value according to the multi-head attention mechanism , And then obtain the updated encoding matrix using the following formula.

$$E_4^t = (W_{sent} A_{sent} + B_{sent})) \tilde{E}_4^t + B_E \qquad (14)$$

In order to ensure that the two levels of attention can be kept consistent during the training process, a unified loss function is added here. We use the L_{sw} loss function to represent the error function calculated between sentences and words. Where $m(n)$ is a mapping relationship between words and sentences.

$$L_{sw} = -\frac{1}{T} \sum_{T}^{t=1} log(\frac{1}{\kappa}) \sum_{n \in \kappa} E_n^t \times A_{m(n))} \qquad (15)$$

Where κ is the set of the first κ participating words and t is the number of words in the abstract. The loss of inconsistency helps our unified model for end-to-end training benefit both the extractor and the abstractor, and also helps to generate longer digest lengths. Through sentence-level extraction, an improved Transformer generation layer, and a swap mechanism, we finally generate a training loss function L_{sum}

$$L_{sum} = \varepsilon_1 L_{ext} + \varepsilon_2 L_{abs} + \varepsilon_3 L_{sw} \qquad (16)$$

where ε_1, ε_2, ε_3 are hyper-parameters. In our experient, we give L_{ext} a bigger weight (e.g., $\varepsilon_1 = 5$) when end-to-end training with L_{sw} since we found that L_{sw} is relatively large such that the extractor tends to ignore L_{ext}.

IV. Experiments

A. Datasets and Settings

We use three datasets for model evaluation: CNN / Daily Mail, Papers, and DUC-2004.The Papers dataset is a private dataset that we build. We use the introduction of the article as the original information and the multi-sentence abstract of the article as the abstract.

In preprocessing, we use byte pair coding (BPE) algorithm [13] to segment words. In this model, we set the vocabulary size to 50,000. The baseline Transformer model is trained using the same hyperparameters as the basic model in Ashish [14]. The number of heads in the Transformer is 8, the size of the feedforward network is 2048, and the training batch size is 8. We use 256-dimensional in BiGRU, which are stitched into 512-dimensional inputs to the Transformer. During the test, we used a beam search with a size of 3 to generate summary, performed 100,000 iterations, saved a basic model every 1000 times, and dropout is 0.5. We limit the maximum output length to 20 and 30, respectively.

B. CNN/Daily dataset

As shown in Table I, we divide the model into Transformer, TPABS (Transformer added pointer mechanism), and TP-EABS. On CNN/Daily dataset, we can see that our model TP-EABS is about 1.7 percentage points higher than Pgen at ROUGE-1. Later, we compared this model with our own baseline model. It was found that before these mechanisms were added, the evaluation index obtained by our model Transformer was not higher than the baseline model, indicating that these mechanisms have improved the summary quality to a certain extent. The ROUGE score is affected by the short length of the generated summary. We are difficult to know why the model gets a low summary score. To evaluate this hypothesis, we randomly select 40 pairs (articles, abstracts) from a fixed test set for manual evaluation. Articles and model-generated summary were submitted to three relevant professionals for evaluation. Each worker has two model-generated summaries, one from the TP-EABS model and one from the PGen model. Workers were asked to choose a better summary according the four different quality metrics from Celikyilmaz et al. [11]. The results are shown in Table II. Interestingly, compared to the PGen model, the summary of TP-EABS is more favored by humans.

In the experiment, we tried two strategies of location information superposition(1) Direct superposition form (ADD): The bidirectional GRU encoding layer information is directly added on the input vector of the Transformer. (2)Learning Strategy (MLP): Add a layer of neural network to let this layer of neural network learn how to superimpose position information and word vector information.As shown in Table I, it is found that the two superimposed effects have slightly higher learning strategies, but the difference is not large. To reduce the amount of calculation, we add the position vector directly to the model.

TABLE I: ROUGE F1 results for various models and ablations on the CNN/Daily Mail test set.

Model	ROUGE-1	ROUGE-2	ROUGE-L
Attn-S2S [6]	32.75	12.21	29.01
PGen [4]	36.44	15.66	33.42
PGen+Cov [4]	39.53	17.28	36.38
Key information [10]	38.95	17.12	35.68
Transformer	35.26	14.12	31.08
TPABS	36.75	15.89	33.15
TP-EABS(MLP)	38.12	17.36	36.45
TP-EABS(ADD)	38.14	17.29	36.52

TABLE II: Head-to-head comparison between test set outputs of PGenand TP-EABS. Analyses done on summaries for Papers.

Model	PGen	same	TP-EABS
Non-redundancy	65	62	182
Coherence	180	42	145
Focus	140	36	176
Overall	160	41	170

C. Papers and DUC-2004

In Papers data set, the length of the article is long, and its summary is mostly not in the original text. We mainly test our improved Transformer model, as shown in Table III. After adding the pointer mechanism, the model obtains The result is 2 percentage points higher than the Pgen model, which proves that the Transformer model is richer in obtaining context information than the recurrent neural network in the language model. On this data set, we set the parameters of Pgen from See et al[4].We did ablation experiments to evaluate the contributions of different mechanismsTransformer, TEABS (Transformer on the extraction model), TPABS (Transformer with a pointer mechanism added), and TP-EABS.The experimental results of the four models are shown in the Table III.

As shown in Table III, our method has made some progress on the current benchmark on DUC-2004 dataset, and ROUGE-1 and ROUGE-L scores have improved the RAS-LSTM model by absolute 0.3 and 1.5 percentage points, respectively. We also compare the model with Feats. We can see that our model still performs better without introducing external information and reinforcement learning. TPEABS improves the data set

TABLE III: ROUGE F1 results for various models and ablations on the Papers and DUC-2004.

Model	R-1	R-2	R-L	R-1	R-2	R-L
ABS [15]	24.32	7.32	19.24	26.55	7.06	22.05
ABS+ [15]	25.24	7.64	20.32	28.18	8.49	23.81
FeatS2S [7]	26.89	9.52	23.76	28.61	9.42	25.24
RAS-LSTM [8]	28.42	10.77	22.46	28.97	8.26	24.06
PGen [4]	29.70	13.19	27.32	–	–	–
Transformer	28.54	12.04	24.92	–	–	–
TPABS	29.32	12.58	25.45	–	–	–
TEABS	29.46	13.21	26.15	–	–	–
TP-EABS	31.69	14.52	28.18	29.27	9.95	25.54

by 0.13 percentage points and 0.4 percentage points over TPABS. Considering the sequence context information, our model can capture important information and generate high-quality abstracts.

V. Conclusion

We propose a joint abstract generation model based on improved Transformer. Most importantly, we improved the Transoformer model so that it has the ability to copy words from the original text. After adding sequence information and extraction stages, the model in this paper can obtain more complete summary information in the uniformly distributed original text, and it will not ignore its importance because the key information is located later. Through end-to-end training of our model, we conducted experiments on three datasets and conducted reliable human evaluation on private datasets, proving that the model has good summary information significance.

References

[1] Piotr Bojanowski, Edouard Grave,Armand Joulin, and Tomas Mikolov. Enriching word vectors with subword information. Transactions of the Association for Computational Linguistics, 5:135–146, 2017.

[2] Ramesh Nallapati, Feifei Zhai, and Bowen Zhou. 2017. Summarunner: A recurrent neural network based sequence model for extractive summarization of documents. In Proceedings of Thirty-First AAAI Conference on Artificial Intelligence (AAAI-17), pages 3075–3081.

[3] Jadhav A , Rajan V . Extractive Summarization with SWAP-NET: Sentences and Words from Alternating Pointer Networks[C]// Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers). 2018.

[4] Abigail See, Peter J Liu, and Christopher D Manning. 2017. Get to the point: Summarization with pointergenerator networks. In Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics.

[5] Wan-Ting Hsu, Chieh-Kai Lin, Ming-Ying Lee, Kerui Min, Jing Tang, and Min Sun. A unified model for extractive and abstractive summarization using inconsistency loss. arXiv preprint arXiv:1805.06266,2018.

[6] Cao Z, LiW, Li S et al (2016) Attsum: joint learning of focusing and summarization with neural attention[J].arXiv preprint arXiv:1604.00125

[7] Nallapati, R., Zhou, B., dos Santos, C., Gulcehre, C., Xiang, B.: Abstractive text summarization using sequence-to-sequence RNNs and beyond.

[8] Sumit, C., Michael, A., Rush, A.M.: Abstractive sentence summarization with attentive recurrent neural networks. Human Language Technologies, pp. 93–98 (2016)

[9] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In Advances in Neural Information Processing Systems, pages 5998–6008.

[10] Ramesh Nallapati, Bowen Zhou, and Mingbo Ma. 2016a. Classify or select: Neural architectures for extractive document summarization. arXiv preprint arXiv:1611.04244

[11] Shashi Narayan, Shay B Cohen, and Mirella Lapata. 2018. Don't give me the details, just the summary! topic-aware convolutional neural networks for extreme summarization. In EMNLP

[12] Yen-Chun Chen and Mohit Bansal.Fast abstractive summarization with reinforce-selected sentence rewriting. arXiv preprint arXiv:1805.11080,2018.

[13] Sebastian Gehrmann, Yuntian Deng, and Alexander M Rush. Bottom-up abstractive summarization. arXiv preprint arXiv:1808.10792, 2018.

[14] Jiatao Gu, Zhengdong Lu, Hang Li, and Victor O. K. Li. Incorporating Copying Mechanism in Sequence-to-Sequence Learning. In ACL, 2016.

[15] Rush, A.M., Chopra, S., Weston, J.: A neural attention model for abstractive sentence summarization. In: Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, pp. 379–389 (2015)

Modeling Topic Exhaustion for Programming Languages on StackOverflow

Rao Hamza Ali and Erik Linstead

Fowler School of Engineering, Chapman University

Abstract

We apply latent Dirichlet allocation on StackOverflow questions, spanned across ten years, for Python, JavaScript, Java, C++, and R, in order to discover underlying topics of questions asked for these programming languages. We focus on topics that have exhausted over the years; topics that once peaked in terms of the number of questions being asked about them but are now in a decline. Studying these topics provides insight into a language's evolution and its cohesion with other programming languages, that may offer similar features. We also measure the average wait times to get answers for questions from exhausted topics, to highlight if the community also plays a role in making these topics exhausted.

Keywords: Latent Dirichlet Allocation; SOTorrent; Topic Modeling; Stack Overflow

1. Introduction

Stack Overflow (SO) has become one of the most popular platforms for programmers to ask and answer questions for a wide range of topics in software engineering [1]. Starting in 2008, the website has seen constant increase in questions asked everyday and the categories to which they belong [2]. The community, too, has been increasingly active and boasts an average reply time to a question of 11 minutes [3]. With the seemingly nonstop increase in questions asked and new categories sprouting up, we are interested in seeing how topics (many of which were of high interest to the developer community historically) have been performing with the influx of new topics. We are also interested in topics which have seen a peak of interest from programmers and now are in a decline. We term these topics as exhausted, in that they peaked in terms of questions asked about them, but are now seeing continuous decline in new questions being posted.

Questions on SO are assigned tags which categorize the field a question may belong to, but are not helpful in determining the topic of that question. The work in [4] described developer behavior for mobile application development, using tags assigned to SO posts. But two questions with an iOS tag could ask questions on app performance and creating a slide view, which are two relatively different topics. Here we use an unsupervised technique, latent Dirichlet allocation (LDA) [5], to extract topics from questions specific to individual programming languages and study their evolution over time. While there have been studies where LDA is applied to SO questions, the focus has mostly been on identifying topics across all languages [6] and looking at short-term or temporal trends[7] . Another novel approach to using LDA was presented by [8] to automatically categorize software from source code, hinting at the success of using LDA for the analysis of source code and related text using topic generation. Unlike this previous work, here we run LDA separately on questions for popular programming languages and observe the trends across the years. The ultimate goal is to identify which features of the language have been exhaustively discussed on SO and are no longer the main focus of the developer community. We also want to understand how the adoption of programming languages evolves based on the type of questions that are asked about them.

One important question that is raised when looking at topics that have been exhausted is: is the community no longer active in answering a question related to such topics? Long wait times to get an answer or failure to come up with a good answer in time, could dissuade developers from asking further questions about these topics. Once we identify topics that have been exhausted, we will also look at the community wide statistics about the time taken to answer questions across the years, and can then conclude the community's role in exhaustion of topics.

2. Data

We use the December 2018 data set of SOTorrent[9] for our study. It consists of over 42 million posts made on SO from August 2008 to December 2018, including question and answers from the community on different topics. Each question is manually tagged by the poster with a programming language and a related field. We use this attribute

Table 1. Total number of questions per programming language

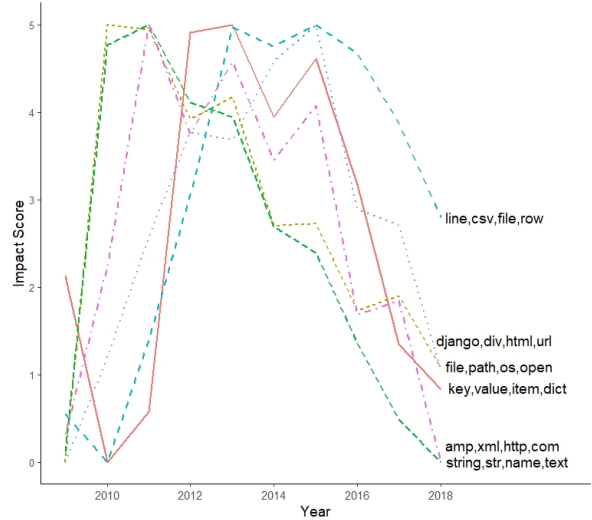| Language	Questions
JavaScript	1,723,695
Java	1,487,204
Python	1,068,646
C++	595,662
R	265,946

to subset the data for 5 programming languages: Python, JavaScript, Java, R, and C++. These languages are the most tagged on SO and comprise a large corpus of questions for our analyses. We are also using data beginning January 2009, to allow for uniform per-year stats.

We choose to use the text pertaining to the question asked instead of the question title. The question body provides greater detail and insight about the problem itself. We further remove any source code references from the question body as we are interested in discovering latent topics from the description of the programming language feature rather than from relevant code. From this, we are able to field more connections between words for LDA to utilize and get results which give a better understanding of the trends. For the community stats, we extract the first and accepted answer times for all questions. An accepted answer is selected by the user who posted the question, which they deem to be the best answer among all others. Table 1 describes the total number of questions, for each language, that were used in training the LDA model.

3. Method

We start by collecting all question posts for a programming language, and removing all stop-words, punctuation, and numbers from them. We extensively make use of the gensim [10] package in Python for topic modeling and natural language processing (NLP). Using built-in functions, we tokenize each question, and create a dictionary for each word and its occurrence count. We use spaCy [11], the NLP Python library, to lemmatize each token to reduce the word space to a common base form and discover more coherent topics. A bag-of-words model [12] is then generated from the dictionary. This step is crucial because we are not interested in the order of words for each question, but the word occurrences in all questions. This pre-processing step is done separately for all languages we run LDA on.

LDA is a probabilistic model for a collection of documents, where each document consists of a bag of words and is viewed as a mixture of different topics with varying prob-
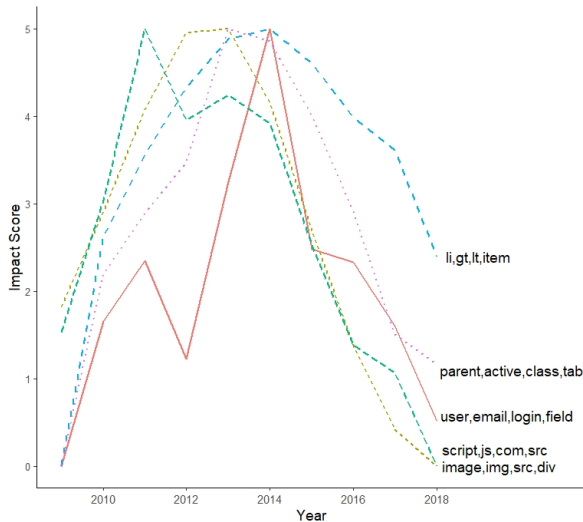

Figure 1. Trend of exhausted topics for Python

abilities. Using this explicit representation of documents in terms of topic probabilities, LDA identifies the topic that most represents each document. This is done by utilizing the mixture model we learn for each set of questions corresponding to a programming language. Given the vast vocabulary of words and the need to identify both high-level and low-level topics, we parameterize LDA to discover 30 topics for each programming language. This number also gave us the most coherent latent topics.
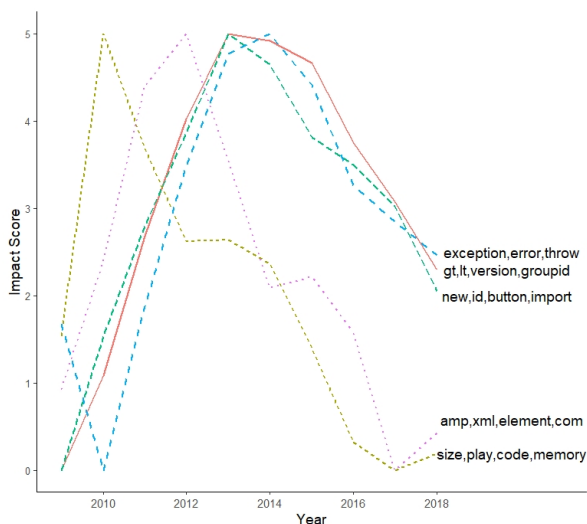
A topic coherence [13] for each LDA model is calculated which gives a measure of strength and consistency of all topics generated. Only models with a topic coherence greater than 0.5 are chosen. This is to ensure that, while 30 topics do cover all of the questions, the model has been successful in understanding the relationships in the corpus and the topics describe the data in the best possible way. The topics, identified by the model, can be joined back to the questions that they have the highest probability of belonging to. Next, we calculate the topic impact score, which is a rank of a topic's occurrence each year for a programming language in comparison to other topics utilized for the language within the same year. The impact score is scaled so that we are able to compare multiple exhausted topics at the same time and are able to observe common trends for them.

4. Results

For our results, we present the topics that have been exhausted, in terms of questions asked, across the years for the selected programming languages.By using a threshold of 30 topics for LDA, we were able to find not only top-
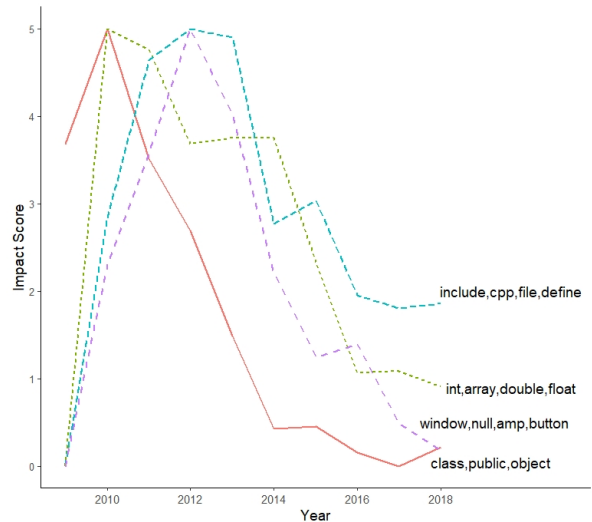
Figure 2. Trend of exhausted topics for JavaScript

Figure 4. Trend of exhausted topics for C++

[14][15][16]. But the focus has remained on what kind of topics are discovered, or which topics have been becoming more popular or are in steady decline. Our research focuses on exhausted topics, that are not likely to have an impact anymore on the community, but need to be identified to further study a topic's trend over the years. It could be the case that adoption of a new programming language or an update in the functionality that the developer is interested in asking about, heavily saturates such topics and we no longer see them making the same impact on SO. Through LDA, we found these exhausted topics that cover features like file I/O, front-end and back-end development, and basic language functionality.

We start our discussion by identifying exhausted topics for Python. Figure 1 shows the trend of such topics between 2009 and 2018, given their impact scores. We only report topics that had an increase in impact after the inception of SO, but are now in a decline, in terms of questions asked. We note that topics related to HTML in Python are completely saturated and developers are no longer asking a lot of questions about them. Interestingly, we also find some topics related to Python's built-in functionality in a decline too. Python has more or less used the same functions for file IO and data structures, so it is apparent that developers have run out of new questions to ask about them, and their queries have most likely already been answered by a previous post. The decline in HTML topics is uniform across all platforms. It is possible that adoption of newer web scripting languages has moved developers away from using Python's web features.
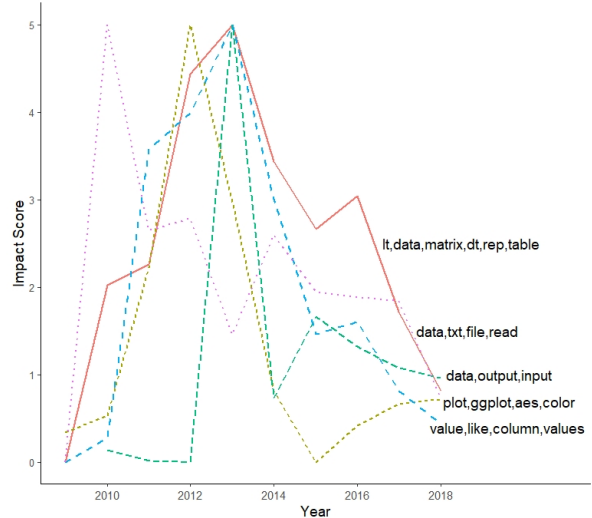
Figure 2 describes the trends of exhausted topics for JavaScript (JS) on SO. We, here too, note that topics related to web development and visual programming are the ones

Figure 3. Trend of exhausted topics for Java

ics that represented a small set of questions, but also multiple topics that were part of a bigger theme and described different aspects of it. Each graph shows the yearly trend of exhausted topics, with respect to their impact score, for a different programming language. A set of words is assigned to each topic, extracted from the questions, and is used in the graphs to describe the overall theme of the topic. These words have the highest probability of belonging to that topic, among all words. They present an understanding of what kind of questions are being asked.

LDA has been extensively used in finding inherent topics for a corpus of questions asked on StackOverflow

that have been most exhausted by the end of 2018. With no new features introduced to JavaScript related to these topics, developers have saturated the topics by asking all questions that could be asked, and are satisfied with the answers on questions already posted on SO, that are similar to their queries. Figure 3 shows the trend of exhausted Java topics across the years, with topics related to HTML again being the ones that have been exhausted. Since the introduction of Go, Rust, Kotlin, and Swift, the focus of developers interested in making visual and web based applications has switched over to these new programming languages, abandoning the functionality offered by the relatively older languages. This results in some topics for a language, that was popular a few years ago, no longer having the same impact as others. We see a similar trend for exhausted topics in C++ in Figure 4. Web application related questions are no longer being asked, and questions about the basic functionality of the language, which has not changed after many version updates, are also on a downward trend.

Another set of topics, we notice have been exhausted, are ones related to the basic functionality of a programming language. Since R is used for statistical modeling and has no applications in web development, all exhausted topics in R are related to the base functionality of the language. Questions about data I/O, plotting data points, and wrangling data of different types, are the main focus of the language. And all such topics have already been exhausted. Developers can independently develop packages for a programming language, to introduce new features, that core developers have not yet introduced, leaving base features to be the same across the years. Our analysis makes it apparent that questions for such features will not evolve over time and a developer looking to learn a language, will find answers to their questions already posted on the website. This points to the evolution of a programming language and the evolution of a developer using that language: how HTML based queries are on a decline and questions for big data are being asked more and more, or how people have a command over the basic features of Java and C++ and have now run out of questions to ask about them.

Modeling the trends of exhausted topics across different languages leads to more questions being asked about the role of the community in their decline. Given that no new functionality is added to a certain feature of a programming language, the community will eventually run out of new questions to ask and those seeking answers would get redirected to questions that have already been posted on SO. But if the number of such questions being asked declines, is the community also in decline in answering these questions? It could be possible that if the SO community takes a longer time to answer a question, or to give a suitable enough question that it is deemed 'accepted' by the original poster, that it deters developers from asking questions related to these

Figure 5. Trend of exhausted topics for R language on Stack Overflow

topics, and instead they try to find their solution in old posts.

To answer this question, we look at the average time taken, in hours, to post the first answer, and the accepted answer, for all questions asked for the exhausted topics. As the number of active users on SO increases over time, it is clear that the average wait time to get an answer will lower [17]. But only if the community is no longer interested in answering questions about exhausted topics, will we see an increase or even a plateau in the averages. Figures 6 and 7, respectively show the trend of average number of hours until a first answer is posted and an accepted answer is posted for the exhausted topics for all five programming languages between 2009 and 2018. With over a 100,000 questions being asked about these topics consistently since 2014, we observe a steady decline in the average wait time to get to a satisfactory answer. This shows that the community is is still active in answering questions, regardless of whether they belong to an exhausted topic or not.

We conclude that topic exhaustion of a programming language on SO is not due to the community's lack of interest in answering these questions, and instead, stems from the evolution of the language and its cohesion with other languages. Python still remains the programming language with the most questions asked. What has changed is that developers have found other, more efficient avenues to using some features of the language, via a different programming language. But with new functionality being added to languages, which leads to more questions being asked by developers, who want to learn about those functions, old topics become saturated and developers have exhausted the different type of questions they can ask about them.
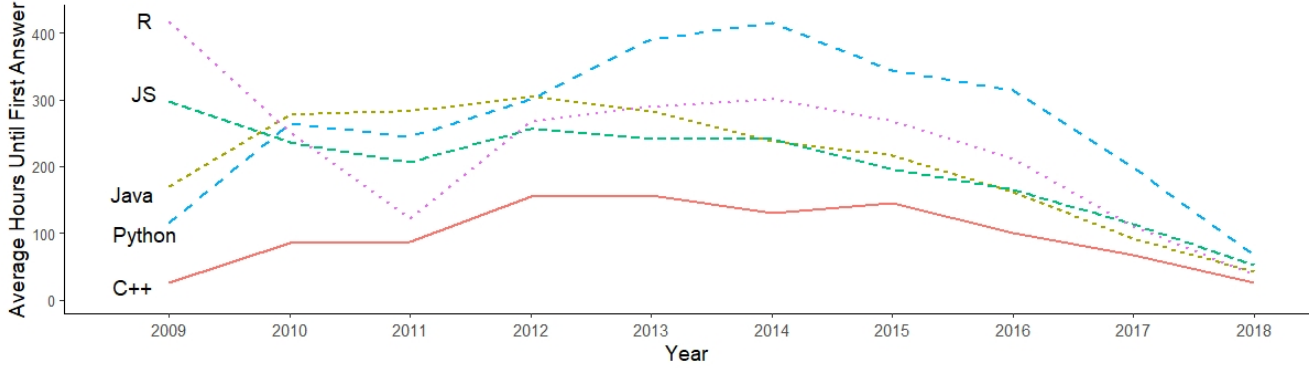
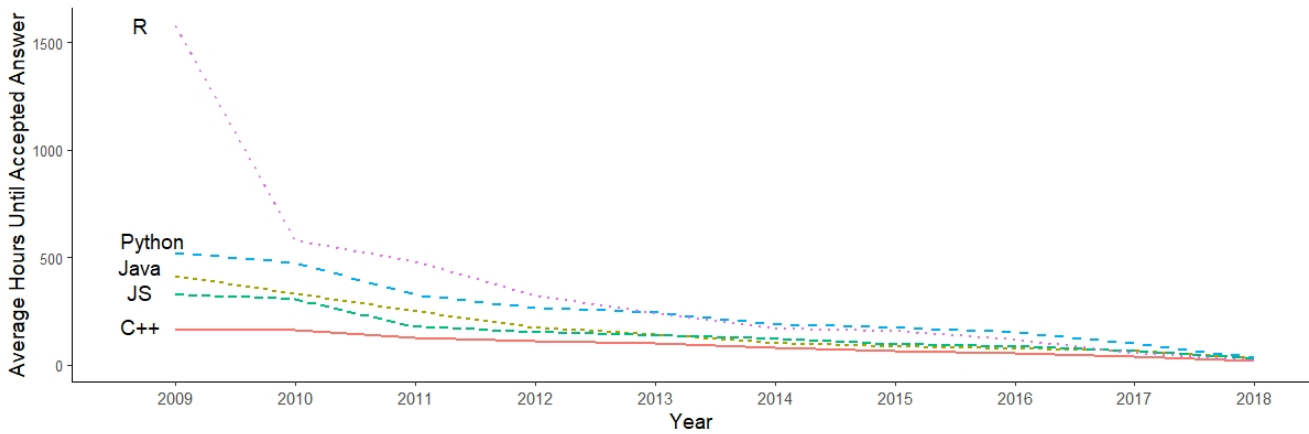**Figure 6. Average Hours until First Answer for Exhausted Topics**

Figure 7. Average Hours until Accepted Answer for Exhausted Topics

5. Related Work and Future Directions

Since first applied to software in 2007 [18], LDA has become a staple for textual analysis of software artifacts. The work in [6] extracted topics from questions asked on SO using LDA and compared them with Java code tokens to find that some topics generated were either text or code identifier only. Mentioned earlier, [2] performed topic modeling on SO questions and answers, posted between 2008 and 2010, and highlighted main discussion topics, scores comparison of answer topics, developer interest, and change in interest in technologies over time. In [16], the authors looked into how the community answers posts on SO and calculated user stats over topics generated by an LDA model. Our paper focuses on exhausted topics across 5 programming languages, over the course of 10 years, and provides an insight into why they are no longer a big focus of developers.

Our research also relates to [19], which explores the evolution of features of a programming language across version updates, using topic modeling. The topics were generated using source code for large open source Java projects, and the trends showed a stark comparison of feature usage between version updates. In this paper, we focus on the vocabulary of questions asked about these features in hope that we can provide trends regardless of version updates, over a long period of time, and view the exhaustion of questions to ask as a measure itself of depletion of newer ways to use a language feature. Exhaustion of questions asked about a topic does not mean that the feature is less popular, nor does it mean that the feature is now deprecated and requires replacement. We instead focus on a community driven vantage point, which views such features as something that programmers have mastered, and are now looking at other prospects that are more challenging.

In the future, topic modeling SO questions for viewing trends for older languages would highlight the most common challenges programmers face for a language and analyzing the answers for such questions would also give a way of solving them. Investigating why certain topics have been exhausted in terms of questions asked, from a language developer point of view, is also a challenge worth tackling, which could provide new insight into how the community

asks questions for a highly documented feature, or how a major update can increase the hype around it. This information, in turn, can be leveraged by language developers to prioritize the integration of new features or even improve old ones.

References

[1] S. Baltes, L. Dumani, C. Treude, and S. Diehl, "Sotorrent: reconstructing and analyzing the evolution of stack overflow posts," in *Proceedings of the 15th International Conference on Mining Software Repositories, MSR 2018, Gothenburg, Sweden, May 28-29, 2018*, 2018, pp. 319–330.

[2] A. Barua, S. W. Thomas, and A. E. Hassan, "What are developers talking about? an analysis of topics and trends in stack overflow," *Empirical Software Engineering*, vol. 19, no. 3, pp. 619–654, 2014.

[3] L. Mamykina, B. Manoim, M. Mittal, G. Hripcsak, and B. Hartmann, "Design lessons from the fastest q&a site in the west," in *Proceedings of the SIGCHI conference on Human factors in computing systems*. ACM, 2011, pp. 2857–2866.

[4] C. Rosen and E. Shihab, "What are mobile developers asking about? a large scale study using stack overflow," *Empirical Software Engineering*, vol. 21, no. 3, pp. 1192–1223, 2016.

[5] D. M. Blei, A. Y. Ng, and M. I. Jordan, "Latent dirichlet allocation," *Journal of machine Learning research*, vol. 3, no. Jan, pp. 993–1022, 2003.

[6] M. Allamanis and C. Sutton, "Why, when, and what: analyzing stack overflow questions by topic, type, and code," in *Proceedings of the 10th Working Conference on Mining Software Repositories*. IEEE Press, 2013, pp. 53–56.

[7] K. Bajaj, K. Pattabiraman, and A. Mesbah, "Mining questions asked by web developers," in *Proceedings of the 11th Working Conference on Mining Software Repositories*. ACM, 2014, pp. 112–121.

[8] K. Tian, M. Revelle, and D. Poshyvanyk, "Using latent dirichlet allocation for automatic categorization of software," in *Mining Software Repositories, 2009. MSR'09. 6th IEEE International Working Conference on*. IEEE, 2009, pp. 163–166.

[9] S. Baltes, C. Treude, and S. Diehl, "Sotorrent: Studying the origin, evolution, and usage of stack overflow code snippets," *CoRR*, vol. abs/1809.02814, 2018. [Online]. Available: http://arxiv.org/abs/1809.02814

[10] R. Řehůřek and P. Sojka, "Software Framework for Topic Modelling with Large Corpora," in *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*. Valletta, Malta: ELRA, May 2010, pp. 45–50, http://is.muni.cz/publication/884893/en.

[11] M. Honnibal and I. Montani, "spacy 2: Natural language understanding with bloom embeddings, convolutional neural networks and incremental parsing," *To appear*, 2017.

[12] Y. Zhang, R. Jin, and Z.-H. Zhou, "Understanding bag-of-words model: a statistical framework," *International Journal of Machine Learning and Cybernetics*, vol. 1, no. 1-4, pp. 43–52, 2010.

[13] K. Stevens, P. Kegelmeyer, D. Andrzejewski, and D. Buttler, "Exploring topic coherence over many models and many topics," in *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, 2012, pp. 952–961.

[14] M. Linares-Vásquez, B. Dit, and D. Poshyvanyk, "An exploratory analysis of mobile development issues using stack overflow," in *Mining Software Repositories (MSR), 2013 10th IEEE Working Conference on*. IEEE, 2013, pp. 93–96.

[15] X.-L. Yang, D. Lo, X. Xia, Z.-Y. Wan, and J.-L. Sun, "What security questions do developers ask? a large-scale study of stack overflow posts," *Journal of Computer Science and Technology*, vol. 31, no. 5, pp. 910–924, 2016.

[16] S. Wang, D. Lo, and L. Jiang, "An empirical study on developer interactions in stackoverflow," in *Proceedings of the 28th Annual ACM Symposium on Applied Computing*. ACM, 2013, pp. 1019–1024.

[17] G. Hewgill, *Meta Stack Overflow Statistics Graphs*, 2010 (accessed May 4, 2020). [Online]. Available: https://meta.stackexchange.com/questions/38297/meta-stack-overflow-statistics-graphs

[18] E. Linstead, P. Rigor, S. Bajracharya, C. Lopes, and P. Baldi, "Mining eclipse developer contributions via author-topic models," in *Proceedings of the Fourth International Workshop on Mining Software Repositories*. IEEE Computer Society, 2007, p. 30.

[19] E. Linstead, C. Lopes, and P. Baldi, "An application of latent dirichlet allocation to analyzing software evolution," in *Machine Learning and Applications, 2008. ICMLA'08. Seventh International Conference on*. IEEE, 2008, pp. 813–818.

An Efficient Application Searching Approach Based on User Review Knowledge Graph

Fang Li[1], Tong Li[2] *

1 Fan Gongxiu Honors College, 2 Faculty of Information Technology
Beijing University of Technology, Beijing, China
fraulifang@163.com, litong@bjut.edu.cn

Abstract—**Finding a software application that perfectly suits user needs is essential for improving user experiences, as well as contributing to the development of the application ecosystems. However, it is not an easy task regarding the huge number of existing applications that are available for use. In this paper, we propose to tackle this challenge by exploring valuable information from user reviews. In particular, we design a user review knowledge graph that consists of both functional information and user preferences in order to comprehensively and precisely characterize software applications. Based on such a review knowledge graph, our approach can support application search in an efficient and precise manner. To evaluate our proposal, we have collected a total of 4,370 applications and 4,396,950 pieces of reviews for constructing a comprehensive review knowledge graph and have illustrated how users and developers can efficiently retrieve applications and improve software functionality based on the knowledge graph.**

Index Terms—**NLP, Knowledge-graph, App searching**

I. Introduction

With the proliferation of types and quantities of mobile phone's application, it is increasingly difficult for users to find applications that perfectly meet their requirements through only the huge amount of application descriptions. There is an urgent need for efficient and accurate application search.

Many direct approaches to application searching have been proposed in the past decade, which are mainly based on application descriptions. Specifically, some researches mine application descriptions from App store and process data by extracting short textual application features or time series data to make application recommendation [1]–[3]. Another branch of research advocates on retrieving applications based on relevance or history of application usage [4]–[6]. Some other researchers further use a graph-based method to analyze the similarity between downloaded applications and search from the graph to improve the searching efficiency [7]. Although, the approaches mentioned above are considered as systematic and promising means to recommend useful application, they typically require a huge amount of textual data and time to obtain applications related to user needs. Moreover, the recommendations that are made based on application descriptions are not able to reflect the application's performance as they do not considering user feedback.

Application reviews contain a huge amount of customer's real opinions that are valuable for profiling users. Specifically, review analysis can obtain the user feedback regarding application functionality and qualities. We argue that the effectiveness of application search can be improved by considering information from both application descriptions and application reviews.

In this paper, we focused on extracting application features and corresponding user preference, as well as calculating the emotional distribution of users. Based on such analysis, we can meaningfully characterize applications to better support application recommendation. Specifically, we firstly retrieve application descriptions and user reviews via web crawling tool from Google Play Store[1]. By crawling the user reviews data and applying Natural Language Processing (NLP) methods [8], we obtain 4,370 applications and about 4,396,950 pieces of reviews in total. In order to promote the effectiveness of our approach, only applications that have a decent number of comments are considered in our study. We then extract the connections between user sentiment and app features based on a large scale of textual comments, and figure out the sentiment distribution accordingly. All the above information services as the foundation for construction a comprehensive Knowledge Graph (referred to as "KG" hereinafter), enabling efficient and effective application search.

Our work mainly leverages techniques of relation extraction and text classification. With the assistance of Sentiword Corpus [9] and an application features dictionary, we are able to extract the relationships between applications features and their corresponding user sentiment. Each relationship we mark a sentiment score to evaluate the tendency as well as the strength of emotions. Based on those relationships, we construct a KG containing user entity, application feature entity and sentiment relation for further clustering and searching. Using this Knowledge Graph, we can cluster applications via intelligent search, rendering multi-angle query results. In this way, we can deal with an enormous data set of App reviews and can search the application accurately.

This paper is organized as follows. Section II presents related work in which app searching has researched within software engineering. In section III we describe the methodology of this work, the collection and processing of data, and
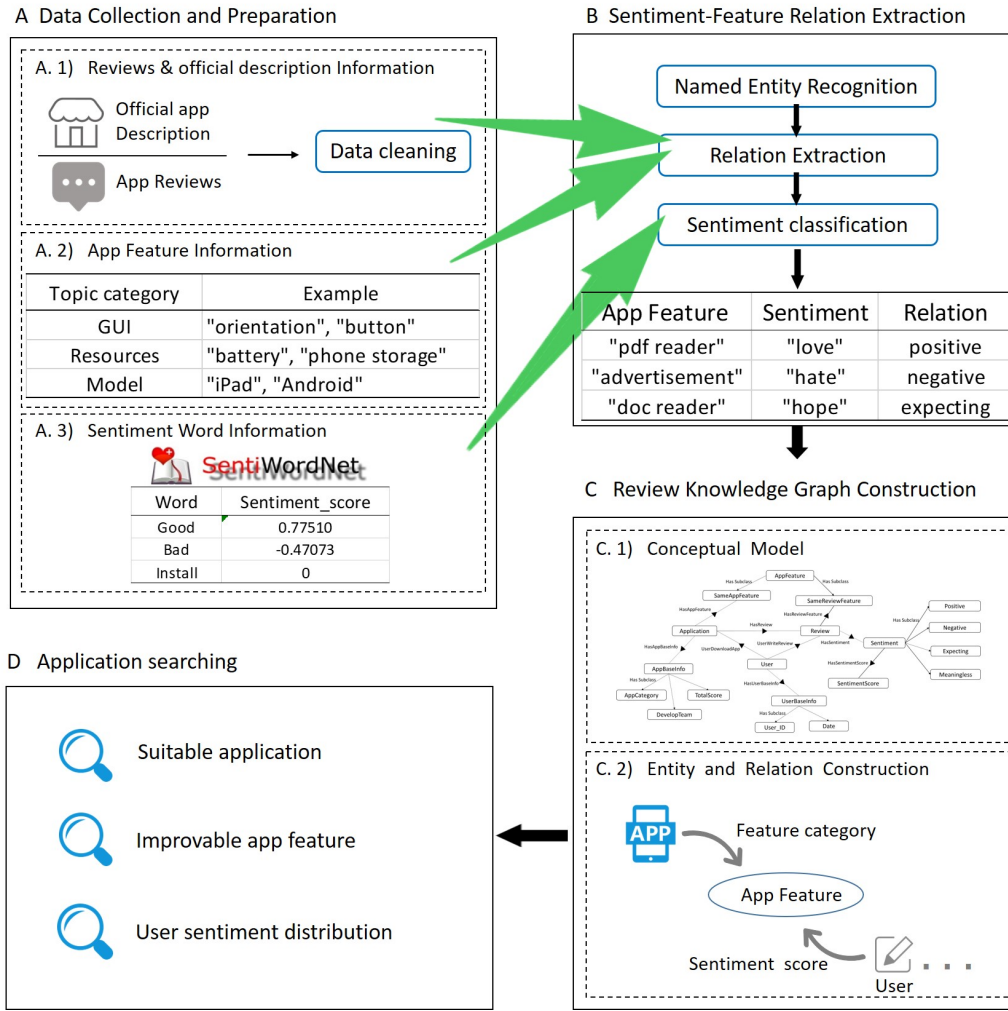
[1]https://play.google.com/store

Fig. 1. The Framework of our paper. The solid rectangle in the figure represents a working step, the dashed rectangle represents the sub-steps in each step, the blue rectangle with rounded corners represents the technical process. And the tables and sin the figure are examples of each step.

the Knowledge Graph construction phase. In section IV we provide evaluation of our work. In section V we provide some discussion. Section VI is our conclusion.

II. RELATED WORK

Instead of using a formal App description, App reviews are a key driver of application clustering. On the one hand, it can learn the true user's feedback for the App. On the other hand, it assists developers in finding the novel functions that can be improved. To date, the smartphone operating systems: Android and iOS have the whole worldwide smartphone shipment market share [10]. A large scale of application reviews is provided to researchers for review analysis. The history of application clustering and the use of KG in application research is discussed in this section.

Previous researches mainly solve the app searching problem by text mining and semantic awareness [11]. Jiang et al. [12] use a greedy algorithm to find the semantic awareness of the user's request and design an application order for user retrieval. Datta, Kajanan and Pervin [13] provide an independent

unbiased search machine for mobile apps with semantic search capabilities. Lavid Ben Lulu and Kuflik [3] use the Machine Learning method to automated analyze functional similarity on the application's description data. Al-Subaihin et al. [14] extracted App feature using information retrieval augmented with ontological analysis to characterize apps. User history and using experience can be used for app searching because user feelings are important for app searching. Costa-Montenegro, Barragáns-Martínez and Rey-López [15] used user history to select the application similar to the downloaded apps. And Krishna et al. [4], searched a similar app using the user's history and the app description information to achieve word import. Zhu et al. [16] combine the popularity of mobile Apps, personal preference, and mobile device constraints for app searching and recommendation. Such textual data is difficult to search and manage. Park et al. [17] leverage user reviews to find out important features of apps for app retrieval. Compared with using app descriptions, brief app reviews can indicate true feedback from users and comes to more detailed app features.

A more applicable method is to use both official descriptions and user reviews for information collection and select more app features to more in line with user needs.

Graph is a kind of data structure which models a set of objects (nodes) and their relationships (edges). Recently, researches of app searching with a graph representation have been receiving more and more attention. Jisha, Krishnan and Vikraman [18] construct a Knowledge Schema - a graphic model of interconnections of data that characterize any mobile app for app searching. Bae et al. [19] incorporates a graph-based technique for application recommendation. Such a method is difficult to find out relevant applications from a large app store. To concisely record the app features and achieve multi-angled searching, we construct a knowledge-based graph containing all the application's information and the user's feedback.

Based on the previous researches, we focus on the app searching based on formal descriptions and user reviews. By analyzing the content of reviews, we can extract not only the application's features but also true feedback from users. What's more, we design a Knowledge Graph-based on the extracted results for app searching. In this way, the large scale of review data can be managed well and can be searched efficiently.

III. METHODOLOGY

Our framework is shown in Fig.1. The sequence Numbers in the figure correspond to the III.Methodology part. We firstly design a web crawling tool to grasp the App reviews and official descriptions from the Google Play store. We also prepared the sentiment word information and application feature data for Named Entity Recognition. During the data preprocessing phase, we changed words in sentences to their lowercase and removed the stop words and special symbols, then we delete the irregular reviews. For example, reviews which only contain punctuation or non-English sentence. Based on these corpora, we extracted the sentences with co-occurrence of user sentiment and App feature. To figure out the sentiment of user, we classified the relation into different categories and use the sentiment score from SentimentWordNet to indicate emotional strength. At last, we build a Knowledge Graph using these relations for results cluster and retrieval. This step contains conceptual model and entity & relation construction. The conceptual model include the user and the application's basic information and the inclusion and emotional relation of them is linked in the graph.

A. Data Collection and Preparation

According to the introduction above, the relationships of user sentiment with app features are fundamental for constructing the Knowledge Graph and their information thus need to be collected and prepared beforehand.

1) App Reviews and official description Information: We choose the Google Play store as the app repositories for its large scale and high frequency of use. We design web crawling tools to catch the app reviews, which is built upon PhantomJS[2] and Selenium[3]. There are 20 app categories, including Art & Design, Augmented Reality, Auto & Vehicles, Beauty, Books & Reference, Business, Comics, Communication, Dating, Daydream, Education, Entertainment, Events, Finance, Food & Drink, Health & Fitness, House & Home, Libraries & Demo, Lifestyle, and Game.

We extract user reviews from App Store as records and save the content, user's ID, review date and other detailed information of each review. Before we extract the user sentiment information, the textual data need to be cleaned. We changed words in sentences to their lowercase and removed the stop words such as ". , !" and special symbols. Then we dismiss the irregular reviews which may only consist of punctuation, number or reviews with messy code.

2) Application feature Information: Fine-grained categorization of app features are considered in our research. Here we create a list of 13 different topics [20]. The topics were proposed by Di S. et al., which can be a great classification category for different review topics, for example, *"I like the GUI of Crazy Bird"* is a comment on application's GUI. Table I illustrates the definition of each review topic. These topic are concluded from user reviews and they indicate the categories that user comments most frequently. Nearly all mentioned application feature in user reviews can be classified into a specific feature category. The classify tools create a new category once the review does not belong to the existing categories, and then they summarize each category into a topic. This cluster has shown to achieve a classification accuracy of 0.76.

TABLE I
DEFINITIONS OF APPLICATION FEATURES

Topic	Definition
App	sentences related to the entire app, e.g., generic crash reports, ratings, or general feedback
GUI	sentences related to the Graphical User Interface or the look and feel of the app
Contents	sentences related to the content of the app
Pricing	sentences related to app pricing
Feature or Functionality	sentences related to specific features or functionality of the app
Improvement	sentences related to explicit enhancement requests
Updates/ Versions	sentences related to specific versions or the update process of the app
Resources	sentences dealing with device resources such as battery consumption, storage,etc.
Security	sentences related to the security of the app or to personal data privacy
Download	sentences containing feedback about the app download
Model	sentences reporting feedback about specific devices or OS versions
Company	sentences containing feedback related to the company/team which develops the app
Other	sentences not treating any of the previous topics

To extract the entity word from the review sentence, we construct an entity dictionary which contains all clusters words,

[2]http://phantomjs.org/
[3]http://www.seleniumhq.org/

e.g., "battery, phone storage" belong to cluster "Resources" and "Android, iOS" belong to the cluster "Model". We create a list of entity words manually which contained 134 original words identifying the topics. To make the dictionaries more exhaustive we used Word-Net [21] to generate synonyms for all the feature words. Synonyms word were then appended to the dictionary to extract the topic entity word from a different user, such as "price, expense and cost" all mean the amount of money for which something is sold.

3) Sentiment Word Information: For the sentiment information, we adopt the method of Gatti, Guerini and Turchi [9]. We adopt the sentiment corpus containing approximately 155,000 sentiment words called "SentiWords", which has both high precision and coverage. We use this sentiment lexicon as our emotional words dictionary to extract user's sentiment from reviews. Each sentiment word from lexicon has a sentiment score, ranging from -1 to +1. The absolute value of the sentiment score indicates the emotional tendency (like the application or dislike) and how strong the emotion is. When a score is a positive number, the higher the score is, the more positive the user comment is. On the contrary, the more negative it is. For example, positive word such as "Good" has the sentiment score 0.77510, negative word such as "Bad" has the score -0.47073 and the subjective word such as "install" has the sentiment score 0.

B. Sentiment-Feature Relation Extraction

In this phase, we mainly introduce how to extract the relationship of user sentiment with app features from review contents, which may have the complex grammar and unstructured sentence pattern. We utilize NLP methods to process the comments. During this process, Name Entity Recognition (NER) is significant for getting a high performance of relation extraction. We construct both user sentiment corpus and app feature dictionary. Then we use NLTK (Natural Language Toolkit)https://www.nltk.org/ to figure the sentences with co-occurrence of user sentiment and app features.

Using the sentiment information and app feature information, we dealt with the sentence relation extraction task as a classification problem. We divided the relation between the app feature and user sentiment into four main categories, positive (+), negative (-), expecting (1), and meaningless (0). More specifically, the relations are classified as "expecting" when sentiment word is in the expecting word list (which contains words expressing expected meaning, such as hope, wish, expect, etc.). Sentiment word with a score greater than 0 will be classified as "positive", and a score less than 0 is classified as "negative". If the above three conditions do not exist, it will be classified as "none". For instance, if the user likes the application, We labeled it as positive to the relationship of the entity with user sentiment. If the user dislikes the application, We labeled it as negative to the relationship. If a user expects an improvement of the app feature, we labeled it as expecting to the relationship. And if the user comments without real meaning, we labeled it as meaningless to it. The example of those four categories

is shown in Fig.1, which contains the user sentiment word, comment app feature word and the relation category.

Relation Extraction is a popular topic in NLP, many novel methods have been proposed with high performance. To simplify the model as well as avoid complex grammatical and semantic analysis, we here adopt the text CNN (Convolutional Neural Network) for sentence classification. In our study, we manually label part of the reviews and then utilize Text-CNN [22] to classify others automatically. The marked sentences are divided into two parts: training data and validation data. We then trained the CNN model on the marked corpus. In this way, we extract all sentences that contain co-occurrence of sentiment words and entity words. For each relationship, a sentiment score corresponds to the user's emotion.

C. Review Knowledge Graph Construction

1) Conceptual Model: Fig.2 shows the conceptual model of the review knowledge graph, including the concept of the application, user, and some attributes. Specifically, we defined the sentiment of reviews into four basic emotions: positive, negative, expecting, and meaningless. At the same time, we also defined app features that applications and reviews have. The detailed concept definition is illustrated as follows.

For each application we choose from App Store, we collect the basic information including the app's category, app's total score, and app's development team.

Each application has a lot of reviews. The user's ID and the comment date would be selected as basic information for uniquely identifying a comment. Specifically, each review containing user sentiment for app features. Here we particularly add sentiment score for each comment to transform the textual emotion to a digitized score, which is defined as sentiment score ranging from -1 to +1. This score can directly indicate the relationship including positive, negative, meaningless and expecting. In this way, the KG can display the distribution of the user's opinion about the app. The review is also linked with the app features that users comment on. The app feature entity belongs to a different perspective of the application, the detailed definition of app features is shown in Table I. Each category contains feature words with specific meaning. For example, "button" belongs to the topic "GUI". In the review KG, the app feature entity indicates comment objects around a specific app.

2) Entity and Relation construction: The relationships of user sentiment and app features were then embedded into the Knowledge Graph. Each relationship is displayed in the KG through the one-to-one correspondence of the concepts including user entity, app feature entity and the sentiment score. The attributes information such as app category and app feature topic are also linked to the app.

In this way, our Knowledge Graph can express the relations of users and the Apps. For each user entity and App feature entity, there exists a relation line with a sentiment score to indicate the meaning and the strength of the relation, such as "Good (0.77510)", "Bad (-0.47073)". We use Neo4j[4], an
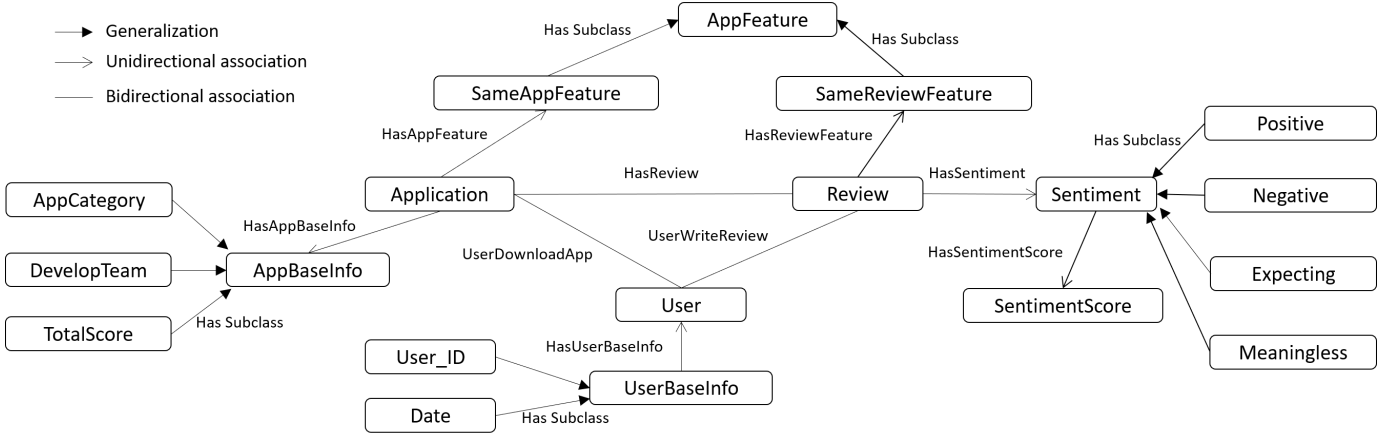
[4]https://neo4j.com/

Fig. 2. The Conceptual model of Review Knowledge Graph

online database, to construct the Knowledge Graph of the application's review.

D. Application searching based on Knowledge Graph

Based on the application's knowledge graph, application and its features can be searched accurately. In addition, the distribution of the user's emotions can be shown according to the sentiment score. Several searching problems as follows can be solved efficiently.

1) Suitable Application Retrieval: For a specific application searching, the knowledge graph performs a clustering algorithm to group the applications that have the same attributes, e.g. app category, app development team, app total score, etc.

2) Improvable Application Feature Retrieval: The development team can check the expecting relations in the application knowledge graph to know the most needed functions. In this way, the knowledge graph can achieve the application's multi-angled retrieval.

3) User Sentiment Distribution Retrieval: For selecting a specific function or standard of applications, the knowledge graph can give the results based on app features, which were linked with positive sentiment scores ranging from +1 to -1. It can also show the distribution of users' experience feelings for a certain app by ordering sentiment relations.

IV. EVALUATION

A. Experimental Data Set

We utilize crawling tools designed by ourselves to catch the application's description and user reviews from Google Play Store[5]. To increase the accuracy and reliability of review analysis, we only choose the application whose total score is bigger than 4.0/5.0 and the number of reviews is bigger than 1,000. For those categories that take less than 1% of the total apps we only choose the applications with score bigger than 3.5/5.0 and the number of reviews bigger than 100. We get a total of about 4,370 applications and about 4,396,950 pieces of reviews. The proportion of the reviews number from each

app category is shown in Fig. 4, in which all app categories and the sentiment distribution of reviews are illustrated. In this picture, each row represents an application category. For each application category, the figure identifies the total number of user comments in that category, the proportion of comments in the total number of experimental comments, and the review sentiment distribution in each category of comments. The proportion of different sentiment is distinguished by different color.

For each review, we save it as one record with review_id, review_app, review_content, review_date, and review_user to indicate the review details. The part of the database we crawled from the website is shared on another website[6].

B. Research Question

This research aims to use previous user comments to provide a more practical result for users. The most important evaluation standard of the results would be the user's satisfaction degree since there are much more fine-grained feature categories than the original searching method. So we have the following research questions to guide our evaluation.

- RQ1: Does this Knowledge Graph improve the algorithm generated searching results?
- RQ2: Does this Knowledge Graph deal with the application information in the same way for all app platforms?

C. Experimental Setting

The app reviews data is collected from the Google Play Store, which is a world-wide used platform for the mobile application's downloading. This app information has also been analyzed for app retrieval before. Since we only choose the app with a high total score and accept conspicuously rich searching terms than ever before, the searching efficiency would improve a lot. Which can explain RQ1. Using the sentiment corpus and app feature dictionary designed before, we manually mark 4000 sentences and achieved an agreement of 92%. If we hold a different opinion on the sentence, we

---

[5]https://play.google.com/store/apps

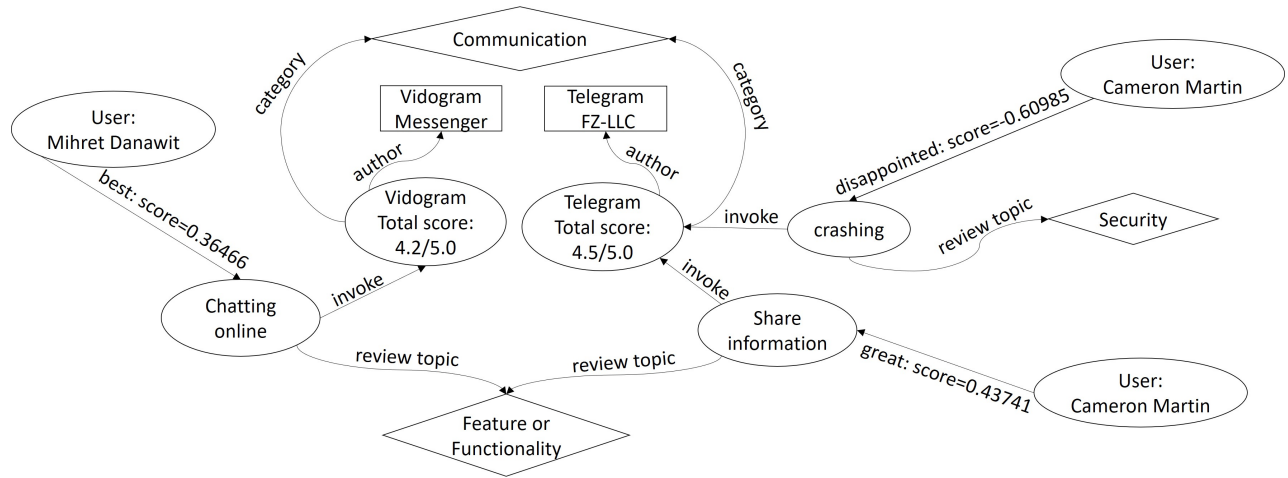[6]https://github.com/fraulifang/Google-app-reviews

410

Fig. 3. Example of Knowledge Graph

discussed and then labeled it. We collected 1802 positive relations, 1326 negative relations, 628 improve relations and 244 useless relations. Then we use the Text-CNN [22] model to automatically classify the other reviews. The parameters of the model are: filter size are 3,4 and 5, the number of filters is 100, and three convolution layers depending on three different window sizes. There are also three max-pooling layers, and an output layer with a softmax function to get the label. The performance of our model is presented in Table II.

D. Results and Analysis

For now, we get about 4,395,000 relation pairs of user and app features, including 1,923,038 positive relations, 1,174,434 negative relations, 874,480 improvement relations, and 423,048 meaningless relations. The more detailed distribution of user sentiment is shown in Fig.4. Then we construct the knowledge graph based on the relations and the application description information.

A simplified example of KG is shown in Fig.3, which contains user entity, app feature entity, relation score and entity attribute. In this figure, rectangles represent the Apps from the App Store, ovals represent the App features and rhombus denote the attributes such as App category and Topic category. In this figure, two apps, Telegram and Vidogram, with its total score from Google Play are shown in the graph. The lines with arrows denote the relationship between apps and their entities and users. The label of the solid line in the figure is the type of relationship between the entities.

Based on such a huge graph, application searching can be achieved. Apps have the same attributes that can be easily grouped and the category of the app is also shown clearly in the graph. Furthermore, users can simply search the KG to find the application they want and know the real experience feelings from others. For example, a user wants to find an app for communication. He or she can search the user sentiment distribution of the app and the result would be the app's sequential arrangement of sentiment score, e.g."Telegram-4.5, Vidogram-4.2" in Fig3. Or someone wants to find an

application that can read both .doc and .pdf format files, then they can search the app feature with those keywords. What's more, app developers can also check the expecting relations to see whether the application needs to be improved in some directions from the user's aspect. In this way, the multi-angled and user-friendly app searching can be achieved.

As for RQ2, we test our method on other application platforms- Uptodown[7], which contains Android apps in 6 categories, including Communication, Games, Lifestyle, Multimedia, Productivity, and Tools. We only choose the apps with a total score bigger than 4.5/5.0 and the number of reviews is bigger than 1,000. We get a total of about 700 applications. By dealing with the reviews as illustrated before, we get about 680,000 relationships. Then the Knowledge graph is expanded using this new application information. Since the app feature entities and user sentiment information are the same, this method can include more platforms application for more precise searching results.

TABLE II
PERFORMANCE OF OUR TEXT-CNN MODEL

category	Precision	Recall	F1 value
positive	0.75	0.71	0.70
negative	0.74	0.72	0.71
improve	0.76	0.76	0.73
useless	0.81	0.85	0.83
total	0.78	0.76	0.78

V. DISCUSSION

There are still many potential directions need to be explored. A meaningful direction is to refine the classification of the review category. For example, dividing the content of comments into version-related, platform-related and operation-related can help eliminate differences and improve the Knowledge Graph.
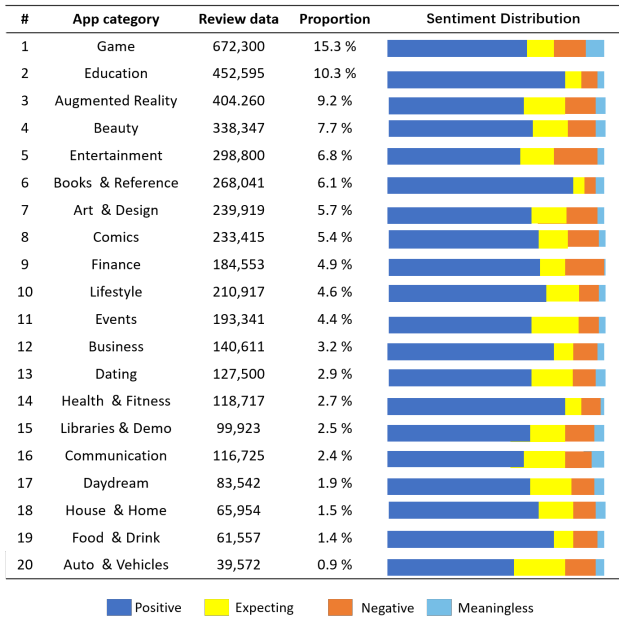
[7]https://en.uptodown.com/

411

#	App category	Review data	Proportion	Sentiment Distribution
1	Game	672,300	15.3 %	
2	Education	452,595	10.3 %	
3	Augmented Reality	404.260	9.2 %	
4	Beauty	338,347	7.7 %	
5	Entertainment	298,800	6.8 %	
6	Books & Reference	268,041	6.1 %	
7	Art & Design	239,919	5.7 %	
8	Comics	233,415	5.4 %	
9	Finance	184,553	4.9 %	
10	Lifestyle	210,917	4.6 %	
11	Events	193,341	4.4 %	
12	Business	140,611	3.2 %	
13	Dating	127,500	2.9 %	
14	Health & Fitness	118,717	2.7 %	
15	Libraries & Demo	99,923	2.5 %	
16	Communication	116,725	2.4 %	
17	Daydream	83,542	1.9 %	
18	House & Home	65,954	1.5 %	
19	Food & Drink	61,557	1.4 %	
20	Auto & Vehicles	39,572	0.9 %	

■ Positive   ■ Expecting   ■ Negative   ■ Meaningless

Fig. 4. The proportion of app category and the sentiment distribution

The size of the database is equally important and determines the accuracy and comprehensiveness of the knowledge map. We tend to continue the collection work and generate more knowledge on the data while achieving automatic real-time updates of some data sources. The resulting knowledge contributes to further development.

VI. Conclusion

In this work, we propose an efficient and effective application search approach based on a user review knowledge graph, which contains information of application features and their corresponding user sentiment. To this end, a total of 4,370 applications and 4,396,950 pieces of reviews were collected. We leverage advanced NLP techniques for extracting the app features and identifying their corresponding user's sentiment. The 4,396,000 structured relation pairs are then embedded into a Review Knowledge Graph. Based on such a KG, application users and developers can search specific applications, review the distribution of the user sentiment and application categories to better achieve their specific requirements.

Acknowledgement

References

[1] A. Gorla, I. Tavecchia, F. Gross, and A. Zeller, "Checking app behavior against app descriptions," *Proceedings of the 36th International Conference on Software Engineering - ICSE 2014*, 2014.

[2] N. Tignor, P. Wang, N. Genes, L. Rogers, S. G. Hershman, E. R. Scott, M. ZWweig, Y.-F. Yvonne Chan, and E. E. SchadtT, "Methods for clustering time series data acquired from mobile health apps," *Pacific Symposium on Biocomputing. Pacific Symposium on Biocomputing*, vol. 22, pp. 300–311, 02 2017.

[3] D. L. B. Lulu and T. Kuflik, "Functionality-based clustering using short textual description," *Proceedings of the 2013 international conference on Intelligent user interfaces - IUI '13*, 2013.

[4] S. Krishna, A. Bajaj, M. Rungta, V. Vala, and H. Tiwari, "Relemb: A relevance-based application embedding for mobile app retrieval and categorization," *Computación y Sistemas*, vol. 23, no. 3, 10 2019.

[5] N. Chen, S. C. Hoi, S. Li, and X. Xiao, "Simapp," *Proceedings of the Eighth ACM International Conference on Web Search and Data Mining - WSDM '15*, 2015.

[6] D. H. Park, Y. Fang, M. Liu, and C. Zhai, "Mobile app retrieval for social media users via inference of implicit intent in social media text," *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*, 10 2016.

[7] U. Bhandari, K. Sugiyama, A. Datta, and R. Jindal, "Serendipitous recommendation for mobile apps using item-item similarity graph," *Information Retrieval Technology*, pp. 440–451, 2013.

[8] P. D. Turney, "Thumbs up or thumbs down?," *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics - ACL '02*, 2001.

[9] L. Gatti, M. Guerini, and M. Turchi, "Sentiwords: Deriving a high precision and high coverage lexicon for sentiment analysis," *IEEE Transactions on Affective Computing*, vol. 6, no. 1, pp. 409–421, 2016.

[10] IDC, "Idc - smartphone market share - os." https://www.idc.com/promo/smartphone-market-share/os, 2018.

[11] A. Datta, K. Dutta, S. Kajanan, and N. Pervin, "Mobilewalla: A mobile application search engine," *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*, vol. 18, no. 1, pp. 172–187, 2012.

[12] D. Jiang, J. Vosecky, K. W. T. Leung, and W. Ng, "Panorama," *Proceedings of the 16th International Conference on Extending Database Technology - EDBT '13*, 2013.

[13] A. Datta, S. Kajanan, and N. Pervin, "A mobile app search engine," *Mobile Networks and Applications*, vol. 18, pp. 42–59, 10 2012.

[14] A. A. Al-Subaihin, F. Sarro, S. Black, L. Capra, M. Harman, Y. Jia, and Y. Zhang, "Clustering mobile apps based on mined textual features," *Proceedings of the 10th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement - ESEM '16*, 2016.

[15] E. Costa-Montenegro, A. B. Barragáns-Martínez, and M. Rey-López, "Which app? a recommender system of applications in markets: Implementation of the service for monitoring users' interaction," *Expert Systems with Applications*, vol. 39, no. 10, pp. 9367–9375, 08 2012.

[16] K. Zhu, Z. Liu, L. Zhang, and X. Gu, "A mobile application recommendation framework by exploiting personal preference with constraints," *Mobile Information Systems*, vol. 2017, pp. 1–9, 2017.

[17] D. H. Park, M. Liu, C. Zhai, and H. Wang, "Leveraging user reviews to improve accuracy for mobile app retrieval," *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval - SIGIR '15*, 2015.

[18] R. C. Jisha, R. Krishnan, and V. Vikraman, "Mobile applications recommendation based on user ratings and permissions," *2018 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, 09 2018.

[19] D. Bae, K. Han, J. Park, and M. Y. Yi, "Apptrends: A graph-based mobile app recommendation system using usage history," *2015 International Conference on Big Data and Smart Computing (BIGCOMP)*, 02 2015.

[20] A. Di Sorbo, S. Panichella, C. V. Alexandru, J. Shimagaki, C. A. Visaggio, G. Canfora, and H. C. Gall, "What would users change in my app? summarizing app reviews for recommending software changes," *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering - FSE 2016*, pp. 499–510, 2016.

[21] G. A. Miller, "Wordnet: a lexical database for english," *Communications of the ACM*, vol. 38, no. 11, pp. 39–41, 11 1995.

[22] Y. Kim, "Convolutional neural networks for sentence classification," *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2014.

SLK-NER: Exploiting Second-order Lexicon Knowledge for Chinese NER

Dou Hu* and Lingwei Wei † ‡

*National Computer System Engineering Research Institute of China, Beijing, China
†School of Cyber Security, University of Chinese Academy of Sciences, Beijing, China
‡Institute of Information Engineering, Chinese Academy of Sciences, Beijing, China
hudou18@mails.ucas.edu.cn, weilingwei@iie.ac.cn

Abstract—Although character-based models using lexicon have achieved promising results for Chinese named entity recognition (NER) task, some lexical words would introduce erroneous information due to wrongly matched words. Existing researches proposed many strategies to integrate lexicon knowledge. However, they performed with simple first-order lexicon knowledge, which provided insufficient word information and still faced the challenge of matched word boundary conflicts; or explored the lexicon knowledge with graph where higher-order information introducing negative words may disturb the identification.

To alleviate the above limitations, we present new insight into second-order lexicon knowledge (SLK) of each character in the sentence to provide more lexical word information including semantic and word boundary features. Based on these, we propose a SLK-based model with a novel strategy to integrate the above lexicon knowledge. The proposed model can exploit more discernible lexical words information with the help of global context. Experimental results on three public datasets demonstrate the validity of SLK. The proposed model achieves more excellent performance than the state-of-the-art comparison methods.

Index Terms—lexicon knowledge, attention mechanism, Chinese named entity recognition

I. INTRODUCTION

Named Entity Recognition (NER) aims to locate and classify named entities into predefined entity categories in the corpus, which is a fundamental task for various downstream applications [1–3]. Word boundaries in Chinese are ambiguities and word segmentation errors have a negative impact on identifying Name Entity (NE) [4], which would make Chinese NER more difficult to identify. Explicit discussions have approved that character-based taggers can outperform word-based counterparts [5].

Because entity boundaries usually coincide with some word boundaries, integrating external lexicon knowledge into character-based models has attracted research attention [5]. Although lexicon can be useful, in practice the lexical words may introduce erroneous information and suffer from word boundary conflicts, which easily lead to wrongly matched entities and limit system the performance [6]. To address the above issues, many sequence-based efforts have been devoted to incorporated lexicon knowledge into sentences [7, 8].
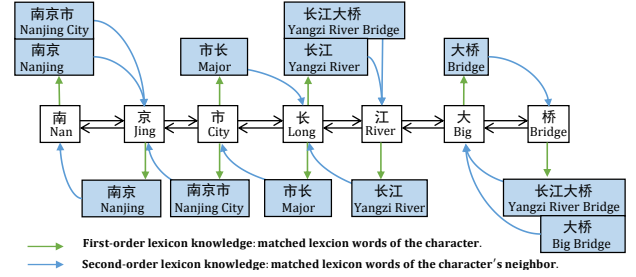
Fig. 1. An example of a word character lattice. The top is that predicting the label uses from left to right sequence and the bottom is using from right to left sequence. The green arrow line represents that the character (start) can match with the lexical word (end). The blue arrow line represents the lexical word (start) information would be integrated into the character (end).

However, these strategies explore simple *first-order lexicon knowledge*(FLK) of each character as shown in the green arrow line in Fig.1. FLK only contains the lexical features of the characters itself, which cannot offer adequate word information. For example, the character "京(Jing)" only introduces "南京(Nanjing)" based on FLK. The wrongly matched word information would misidentify as "南京(Nanjing)" instead of "南京市(Nanjing City)". As a result, they continue to suffer from boundary conflicts between potential words being incorporating in the lexicon. The conflict caused by this deficiency mainly comes from the middle of the named entity, such as "大(Big)" and "江(River)" in "长江大桥(Yangtze River Bridge)".

Recently, some models attempted to aggregate rich higher-order lexicon knowledge, such as the nearest lexical words [9] or graph structure [10–12]. This higher-order information probably introduces irrelevant words with the character, limiting the performance to some extent. In addition, the existence of shortcut paths may cause the model degeneration into a partially word-based model, which would suffer from segmentation errors.

To address the above issue, we introduce the *second-order lexicon knowledge* (SLK) to each character in the input sentence, that is the neighbor's lexicon knowledge of the character, as elaborated in Fig.1 with the blue arrow lines. The SLK of "京(Jing)" contains both "南京市(Nanjing City)" and "南京(Nanjing)" from its left neighbor "南(Nan)", and "南京市(Nanjing City)" from its right neighbor "市(City)".

With regard to global semantics of the sentence, "南京市(Nanjing City)" is more likely to be the named entity than "南京(Nanjing)" due to higher semantic similarity of "南京市(Nanjing City)". Similarly, the SLK of "江(River)" is the potential words "长江大桥(Yangtze River Bridge)" and "长江(Yangtze River)", and the SLK of "大(Big)" is "长江大桥(Yangtze River Bridge)" and "大桥(Big Bridge)". By synthesizing global considerations, these lexicon knowledge guides the character subsequence "长江大桥(Yangtze River Bridge)" to be recognized as the named entity.

To take advantage of this insight, we proposed a SLK-based model with a novel strategy named SLK-NER, to integrate more informative lexicon words into the character-based model. Specifically, we assign SLK to each character and ensure no shortcut paths between characters. Furthermore, we utilize global contextual information to fuse the lexicon knowledge via attention mechanism. The model enables capture more useful lexical word features automatically and relieves the word boundary conflicts problem for better Chinese NER performance.

The main contributions can be summarized as follows:

- **Insight.** We present a new insight about second-order lexicon knowledge (SLK) of the character. SLK can provide sufficient lexicon knowledge into characters in sentences and is capable of relieving the challenge of word boundary conflicts.
- **Method.** To properly leverage SLK, we propose a Chinese NER model named SLK-NER with a novel strategy to integrate lexicon knowledge into the character-based model. SLK-NER can enable to capture more beneficial word features with the help of global context information via attention mechanism.
- **Evaluation.** Experimental results demonstrate the efficiency of SLK and our model significantly outperforms pervious methods, achieving state-of-the-art over three public Chinese NER datasets. The source code and dataset are available[1].

II. RELATED WORKS

Early character-based methods for NER considered few word information in character sequence [4]. To tackle this limitation, many works generally use lexicon as extra word information for Chinese NER.

A. Sequence-based Methods

Zhang et al. [5] introduced a lattice LSTM to model all potential words matching a sentence to exploit explicit word information and achieved state-of-the-art results. Lattice LSTM enlightened various approaches for the useage of lexicon knowledge. Chain-structured LSTM [8] integrated word boundary features into input character vector via four strategies. Zhu et al. [9] investigated CNN-based model with local attention to capture adjacent characters and sentence contexts. Gui et al. [7] extended rethinking mechanism to relieve word boundary conflicts.
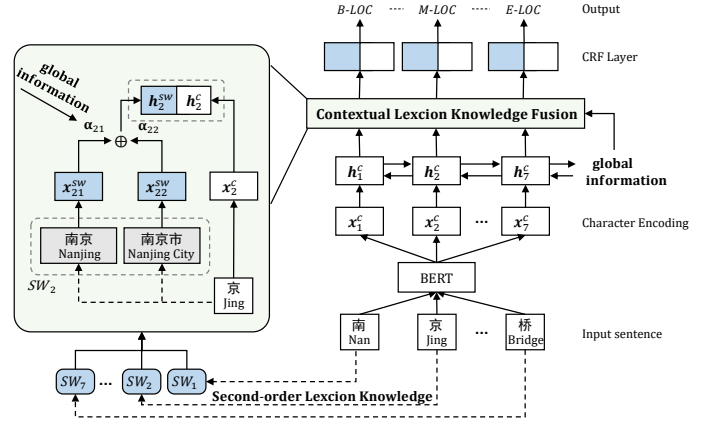
[1]https://github.com/zerohd4869/SLK-NER

Fig. 2. The whole architecture of SLK-NER. It is comprised of character encoding layer, lexicon knowledge encoding layer, contextual lexicon knowledge fusion and a CRF decoding layer.

B. Graph-based Methods

With the development of graph, there are some studies improved by graph neural networks. For instance, Gui et al. [10] proposed a GNN-based method to explore multiple graph-based interactions among characters, potential words, and the whole-sentence semantics and effectively alleviated the word ambiguity. Sui et al. [11] propose a collaborative graph network to assign both self-matched and the nearest contextual lexical words. Ding et al. [12] proposed a multi-digraph structure to learn the contextual information of the characters and the lexicon.

III. METHOD

A. Overview

The overall architecture of our proposed model is illustrated in Fig.2. First, we encode character-based sentences to explicitly capture the contextual features of the sentence via character encoding layer. Second, to integrate more lexicon knowledge, we construct the second-order lexicon knowledge (SLK) for each character. Third, a fusion layer with the global attention information is used for fusing different SLK to alleviate the impact of word boundary conflicts. Finally, a standard CRF model [13] is employed for decoding labels.

Formally, we denote an input sentence as $s = \{c_1, c_2..., c_n\}$, where c_i means the ith character. The lexicon $\mathcal{D}$ is the same as [5], which is built by using automatically segmented large raw text. For ith character, we use $\overrightarrow{\mathcal{FW}_i}$ to denote a set of words obtained by matching all possible forward subsequences in lexicon $\mathcal{D}$ [8]. Similarly, we use $\overleftarrow{\mathcal{FW}_i}$ to denote the words for ith character in backward process. The knowledge involved in these sets represents the FLK corresponding to the ith character, i.e., $\mathcal{FW}_i = \overrightarrow{\mathcal{FW}_i} \cup \overleftarrow{\mathcal{FW}_i}$. Based on FLK, SLK of ith character can be defined as:

$$\mathcal{SW}_i = \overrightarrow{\mathcal{FW}_{i-1}} \cup \overleftarrow{\mathcal{FW}_{i+1}}, i \in [1, n]. \tag{1}$$

As the example shows in Fig.2, SLK of the character "京(Jing)" is the word set including "南京(Nanjing)" and "南

京市(Nanjing City)". SLK can mitigate the negative impact of word boundary conflicts. Therefore, we utilize SLK in our proposed model.

B. Character Encoding Layer

Given the sentence s, a pre-trained model BERT [14] encodes each character c_i in the sentence to a vector.

$$\mathbf{x}_i^c = BERT(c_i). \tag{2}$$

To capture more contextual information, we apply bi-directional Gate Recurrent Unit:

$$\mathbf{h}_i^c = GRU(\mathbf{x}_i^c), i \in [1, n]. \tag{3}$$

The hidden state of last character contains the global features of the input sentence, i.e., $\mathbf{g} = \mathbf{h}_n^c$.

C. Lexicon Knowledge Encoding Layer

To represent the semantic information of SLK of ith character, we embed jth lexical word sw_{ij} in $\mathcal{SW}_i$ to distributional space as a semantic vector:

$$\mathbf{x}_{ij}^{sw} = \mathbf{e}^w(sw_{ij}), \tag{4}$$

where $\mathbf{e}^w$ is a pre-trained word embedding lookup table.

D. Contextual Lexicon Knowledge Fusion

Not all lexical words contribute equally to the representation of the character meaning. Hence, we introduce a global contextual information to extract such SLK that are important to the meaning of the character and aggregate them to refine a character vector. Specifically, for the jth word in the matching set $\mathcal{SW}_i$ of the ith character, we can obtain a hidden representation $\mathbf{u}_{ij}$ for word embedding $\mathbf{x}_{ij}^{sw}$:

$$\mathbf{u}_{ij} = \mathbf{W}_u \mathbf{x}_{ij}^{sw} + \mathbf{b}_u, \tag{5}$$

where $\mathbf{W}_u$ and $\mathbf{b}_u$ are update parameters. We measure the importance of lexical word as the similarity and get a normalized importance weight α_{ij}. Then, the SLK of ith character can be computed as a weighted sum of the word information.

$$\alpha_{ij} = \frac{exp(\mathbf{u}_{ij}^T \mathbf{g})}{\sum_j exp(\mathbf{u}_{ij}^T \mathbf{g})}, \tag{6}$$

$$\mathbf{h}_i^{sw} = \sum_j \alpha_{ij} \mathbf{x}_{ij}^{sw}. \tag{7}$$

Finally, the final representation of ith character is denoted as $\mathbf{r}_i = [\mathbf{h}_i^{sw}; \mathbf{h}_i^c]$.

E. Decoding and Training

To formulate the dependencies between successive labels, a standard CRF layer is used to make sequence tagging. We define matrix $\mathbf{O}$ to be scores calculated based on the final representations $\mathbf{R} = \{\mathbf{r}_1, ..., \mathbf{r}_n\}$:

$$\mathbf{O} = \mathbf{W}_o \mathbf{R} + \mathbf{b}_o, \tag{8}$$

where $\mathbf{W}_o$ and $\mathbf{b}_o$ are trainable parameters. Then, the probability of tag sequence $y = \{y_1, ..., y_n\}$ is:

$$p(y|s) = \frac{exp(\sum_i (\mathbf{O}_{i,y_i} + \mathbf{T}_{y_{i-1},y_i}))}{\sum_{\hat{y}} exp(\sum_i \mathbf{O}_{i,\hat{y}_i} + \mathbf{T}_{\hat{y}_{i-1},\hat{y}_i}))}, \tag{9}$$

where $\mathbf{T}$ is a transition score matrix, and $\hat{y}$ denotes all possible tag sequences. While decoding, we apply the Viterbi [15] algorithm to get label sequence with the highest score.

Given training examples $\{(s_j, y_j)\}|_{j=1}^N$, we optimize the model by minimizing the negative log-likelihood loss:

$$L = -\sum_j log(p(y_j|s_j)). \tag{10}$$

IV. EXPERIMENTS

A. Experimental Settings

1) Datasets: We evaluate our model on three datasets, **OntoNotes4**, **Weibo** and **Resume**. OntoNotes4 is a multi-lingual corpus in the news domain that contains four types of named entities. Weibo dataset consists of annotated NER messages drawn from Sina Weibo[2]. The corpus contains PER, ORG, GEP, and LOC for both named entity and nominal mention. Resume dataset is composed of resumes collected from Sina Finance[3]. It is annotated with 8 types of named entities. For OntoNotes4, we use the same training, validing and testing splits as [16]. Since other datasets have already been split, we don't change them. The statistics of three datasets are details in Table I.

TABLE I
THE STATISTICS OF THE DATASETS.

Dataset	Training	Validation	Testing
OntoNotes4	15724	4301	4346
Weibo	1350	270	270
Resume	3821	463	477

2) Comparisons: The methods evaluated are as follows. **BiLSTM-CRF** [17] was a sequence labeling model consisting of BiLSTM layer and CRF layer. **BERT** [14] was a pre-trained model with deep bidirectional transformer. **Lattice-LSTM** [5] encoded characters in a sequence and all potential words that match a lexicon. **LGN** [10] used lexicon to construct the graph and provide word-level features. The literature [12] applied a multi-digraph structure to incorporate gazetteer information, and we denote **MG-GNN** for convenience. **WC-LSTM** [8] was used to add word information into the start or the end character of the word. **LR-CNN** [7] extended the rethinking mechanism when using lexicon. **CAN** [9] investigated CNN-based model with local attention to capture adjacent characters and contexts.

3) Implementation Details: We use lexicon and word embeddings provided by [18], which is pretrained on Chinese Giga-Word using word2vec model. For character embeddings, we apply the bert-base Chinese model[4] (12-layer, 768-hidden,

[2]https://www.weibo.com
[3]https://finance.sina.com.cn/stock/
[4]https://github.com/google-research/bert

TABLE II
EXPERIMENTAL RESULTS(%) ON THREE DATASETS.

Models	OntoNotes4			Weibo			Resume		
	P	R	F1	P	R	F1	P	R	F1
BiLSTM-CRF[17]	72.0	75.1	73.5	60.8	52.9	56.6	93.7	93.3	93.5
BERT[14]	**78.0**	80.4	79.2	61.2	63.9	62.5	94.2	95.8	95.0
LGN[10]	76.1	73.7	74.9	-	-	60.2	95.3	95.5	95.4
MG-GNN[12]	74.3	76.2	75.2	**63.1**	56.3	59.5	-	-	-
LatticeLSTM[5]	76.4	71.6	73.9	53.0	62.3	58.8	94.8	94.1	94.1
WC-LSTM[8]	76.1	72.9	74.4	52.6	**67.4**	59.8	95.3	95.2	95.2
LR-CNN[7]	76.4	72.6	74.5	-	-	59.9	**95.4**	94.8	95.1
CAN[9]	75.1	72.3	73.6	55.4	63.0	59.3.	95.1	94.8	94.9
SLK-NER	77.9	**82.2**	**80.2**	61.8	66.3	**64.0**	95.2	**96.4**	**95.8**

Encoding Strategy	OntoNotes4			Weibo			Resume		
	P	R	F1	P	R	F1	P	R	F1
using SLK	**77.9**	82.2	**80.2**	**61.8**	66.3	**64.0**	**95.2**	**96.4**	**95.8**
using FLK	76.6	**82.9**	79.8	**61.8**	64.6	63.2	95.1	96.2	95.6
using SLK and FLK	76.4	82.7	79.6	60.6	63.6	62.1	94.9	96.2	95.5
no lexicon	77.7	81.3	79.6	56.7	**66.5**	61.2	94.2	96.1	95.1

Fusion Strategy	OntoNotes4			Weibo			Resume		
	P	R	F1	P	R	F1	P	R	F1
Global-Attention	77.9	**82.2**	**80.2**	**61.8**	**66.3**	**64.0**	**95.2**	**96.4**	**95.8**
Self-Attention	77.2	81.2	79.1	55.9	60.1	57.9	94.2	96.3	95.2
Shortest Word First	77.1	81.5	79.2	55.8	57.7	56.7	93.9	96.1	95.0
Longest Word First	77.1	81.6	79.3	57.6	56.9	57.3	94.7	96.1	95.4
Average	**78.6**	80.8	79.7	56.4	58.4	57.3	94.3	96.3	95.3

12-heads). For characters and words that do not appear in the pretrained embeddings, we initialize them with a uniform distribution[5]. When training, character embeddings and word embeddings are updated along with other parameters. For hyper-parameter configuration, we set max length of sentences to 250, word embedding size to 50, the dimensionality of Bi-GRU to 512, the number of Bi-GRU layer to 1, the dropout to 0.1, the batch size to 32. We use Adam to optimize all the trainable parameters with learning rate $5e - 5$. For evaluation, we use the Precision(P), Recall(R) and F1 score(F1) as metrics in our experiments.

B. Experimental Results

Firstly, we compare SLK-NER with two general sequence labeling model for NER. Both of them performed without any lexicon knowledge. The results in the first block in Table II, show that our proposed model achieves best F1 and R, which proves the efficiency of SLK-NER.

Next, the second block in Table II shows the performance of graph-based models. SLK-NER gives better F1 and R than both LGN and MG-GNN. Although these baselines explore lexicon knowledge via the graph structure, they performed without the consideration of contextual information. Hence,

---

[5]The range is $[-\sqrt{\frac{3}{dim}}, +\sqrt{\frac{3}{dim}}]$, where dim denotes the size of embedding.

---

we attribute the benefits to the efficiency of global context-aware in SLK-NER.

Furthermore, the third block in Table II shows results of state-of-the-art sequence-based models. We can observe that our proposed model achieves a remarkably improvement on F1 over three datasets. The results strongly verify the integrating SLK into character-based model enables to boost the performance. By leveraging the SLK properly, our model is capable of improving NER in various domains, such as social network, news and Chinese resume.

C. Strategies Analysis

In this part, we explore the effects of strategies about lexicon knowledge.

1) Lexicon Knowledge Types: We conduct comparative experiments on different kinds of lexicon knowledge. The results are illustrated in Table III. We can clearly see that the character-based model performs poorly without lexicon knowledge, demonstrating the usefulness of lexicon. Besides, adding FLK makes a small improvement on F1. While adding SLK outperforms significantly on F1 in all datasets. The fact demonstrates the efficiency of SLK, and reveals that leveraging second-order lexicon knowledge can indeed alleviate the word boundary conflicts Interestingly, when using both FLK and SLK, the F1 declines over three datasets. We conjecture

the reason is there may be some negative word conflicts simultaneously for a character which limit the performance.

2) Lexicon Knowledge Encoding: We analysze the difference between the strategy in our model (Global-Attention) with four strategies proposed by [8] for encoding word information, including Self-Attention, Shortest Word First, Longest Word First and Average. The results in Table IV show that global attention in our model achieves best performance on F1 score. This demonstrates that our model can combine more informative features to determine the word boundary and effectively alleviate the negative influence of word boundary conflicts.

D. Sentence Length Analysis

Fig.3 shows the F1 score of several baselines and SLK-NER against sentence length on OntoNotes4 dataset. BERT and SLK-NER outperform significantly than other baselines, which indicates the ability to capture long dependencies. However, BERT ignores the word information among the sentence. SLK-NER obtains a higher F1 over different sentence lengths compared to BERT, which proves the SLK and global context-aware can capture more useful contextual information.

Fig. 3. F1 against sentence length on OntoNotes4 dataset. We split samples into six parts according to the sentence length.

V. CONCLUSION

In this paper, we have investigated a lexicon-based model in Chinese NER task. We present a new insight about second-order lexicon knowledge to incorporate informative lexicon into character-based model. Based on this insight, SLK-NER is proposed to integrate more contextual word information into each character utilizing the global context. SLK-NER can effectively alleviate the impact of word boundary conflicts and word segmentation errors. Extensive experiments on three public datasets have demonstrated the superior performance of SLK-NER than state-of-the-art models.

REFERENCES

[1] Jiafeng Guo, Gu Xu, Xueqi Cheng, and Hang Li. Named entity recognition in query. In *SIGIR*, pages 267–274, 2009.

[2] Dennis Diefenbach, Vanessa Lopez, Kamal Singh, and Pierre Maret. Core techniques of question answering systems over knowledge bases: a survey. *Knowledge and Information systems*, 55(3):529–569, 2018.

[3] Sandipan Dandapat and Andy Way. Improved named entity recognition using machine translation-based cross-lingual information. *Computación y Sistemas*, 20(3):495–504, 2016.

[4] Nanyun Peng and Mark Dredze. Named entity recognition for chinese social media with jointly trained embeddings. In *EMNLP*, pages 548–554, 2015.

[5] Yue Zhang and Jie Yang. Chinese NER using lattice LSTM. In *ACL*, pages 1554–1564, 2018.

[6] Jason PC Chiu and Eric Nichols. Named entity recognition with bidirectional lstm-cnns. *TACL*, 4:357–370, 2016.

[7] Tao Gui, Ruotian Ma, Qi Zhang, Lujun Zhao, Yu-Gang Jiang, and Xuanjing Huang. Cnn-based chinese ner with lexicon rethinking. In *28th IJCAI*, pages 4982–4988. AAAI Press, 2019.

[8] Wei Liu, Tongge Xu, Qinghua Xu, Jiayu Song, and Yueran Zu. An encoding strategy based word-character lstm for chinese ner. In *NAACL*, pages 2379–2389, 2019.

[9] Yuying Zhu and Guoxin Wang. Can-ner: Convolutional attention network for chinese named entity recognition. In *NAACL*, pages 3384–3393, 2019.

[10] Tao Gui, Yicheng Zou, Qi Zhang, Minlong Peng, Jinlan Fu, Zhongyu Wei, and Xuan-Jing Huang. A lexicon-based graph neural network for chinese ner. In *EMNLP-IJCNLP*, pages 1039–1049, 2019.

[11] Dianbo Sui, Yubo Chen, Kang Liu, Jun Zhao, and Shengping Liu. Leverage lexical knowledge for chinese named entity recognition via collaborative graph network. In *EMNLP-IJCNLP*, pages 3821–3831, 2019.

[12] Ruixue Ding, Pengjun Xie, Xiaoyan Zhang, Wei Lu, Linlin Li, and Luo Si. A neural multi-digraph model for chinese ner with gazetteers. In *ACL*, pages 1462–1467, 2019.

[13] John Lafferty, Andrew McCallum, and Fernando CN Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. 2001.

[14] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *NAACL*, pages 4171–4186, Minneapolis, June 2019.

[15] Andrew Viterbi. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE transactions on Information Theory*, 13(2):260–269, 1967.

[16] Wanxiang Che, Mengqiu Wang, Christopher D. Manning, and Ting Liu. Named entity recognition with bilingual constraints. In *HLT-NAACL*, pages 52–62. ACL, 2013.

[17] Zhiheng Huang, Wei Xu, and Kai Yu. Bidirectional lstm-crf models for sequence tagging. *arXiv preprint arXiv:1508.01991*, 2015.

[18] Shen Li, Zhe Zhao, Renfen Hu, Wensi Li, Tao Liu, and Xiaoyong Du. Analogical reasoning on Chinese morphological and semantic relations. In *ACL*, pages 138–143, Australia, July 2018.

Sentiment Analysis over Collaborative Relationships in Open Source Software Projects

Lingjia Li‡, Jian Cao*‡ and David Lo§

‡Department of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai, 200240, China
§School of Information Systems, Singapore Management University, 178902, Singapore
Email: ‡{jessie_llj,cao-jian}@sjtu.edu.cn, §davidlo@smu.edu.sg

Abstract—Sentiments and collaboration efficiency are key factors in the success of the open source software (OSS) development process. However, in the software engineering domain, no studies have been conducted to analyze the effect between collaborators' sentiments, and the role of sentiment in collaborative relationships during the development process. In this study, we apply sentiment analysis and statistical analysis on collaboration artifacts over five projects on GitHub. We use sentiment consistency to quantify the relation between sentiments in collaborative relationships. It is found that sentiment consistency is positively correlated with the closeness of collaborative relationships and collaborators' overall sentiment states. We also perform the Granger causality test and network analysis to study the impact of sentiment consistency on a time series basis. It is found that positive consistent sentiments not only improve collaboration willingness to the utmost extent, followed by inconsistent and negative consistent sentiments, they also boost the closeness of the entire project community. These findings can be applied to develop better OSS project monitoring tools and improve project management by taking developers' sentiments during collaborations into consideration.

Index Terms—Sentiment analysis, Human factors, GitHub, Collaborative and social computing, Project management

I. INTRODUCTION

Software development is a highly collaborative activity where developers work on collaborative tasks and interact with shared artifacts to create and maintain a complex software system. The efficiency of collaboration is a distinguishing factor in the success or failure of many modest to large software development organizations [1]. Therefore, how to improve collaboration efficiency is an important issue in project management. Many factors affect collaboration efficiency, including ease of use of technology, trust between the teams and well-defined task structure [2], etc.

Basic emotions include *anger*, *disgust*, *fear*, *joy*, *sadness*, and *surprise*. Emotions can generally imply people's sentiments which are usually classified into *positive*, *negative* and *neutral*. Such sentiments in professional work can affect creativity, group rapport, user focus, and job satisfaction [3]. In software development, happy developers have higher debugging performance [4], self-assessed productivity and solve problems better [5]. Studies that perform sentiment analysis on software artifacts find that positive sentiments in

development activities increase the number of commit files [6] and decrease issue resolution time [7]. These findings highlight the role of sentiment in software development and suggest that understanding its key factors can help improve developers' perfformance.

Sharing feedback in the form of sentiments can positively affect online trust in inter-user collaborations among Wikipedia editors [8]. In the education domain, it is also found that emotions have an impact on forming successful collaborative relationships [9]. However, in the software engineering (SE) domain, there is no study to analyze the effect between collaborators' sentiments, and the role of sentiment in collaborative relationships. Therefore, in this study we focus on the sentiment over collaborative relationships during software development. Specifically, we are interested in the collaborative relationships in open source software (OSS) projects, where the work is volunteer-driven, hence developers' enjoyment plays a dominant role during the developments [10]. To identify the key factors behind collaborators' sentiment relations and understand how these relations interact with collaborative relationships is important for managing OSS project. On one hand, it can help develop better tools for monitoring sentiments to resolve potential risks. On the other hand, effective strategies can be adopted to coordinate the development process, for example, recommending compatible developers by taking their sentiment effects during collaboration into consideration.

We perform sentiment analysis on issue comments in GitHub and define sentiment consistency to measure the relationship between collaborators expressed by their sentiments. We intend to answer the following research questions:

RQ1: Are collaborators' sentiments more consistent than those between other developers?

We compare sentiment consistency over collaborative and non-collaborative relationships to find out whether developers' sentiments are affected by the collaborative relationship.

RQ2: What factors does sentiment consistency correlate with in a collaborative relationship?

We examine closeness of collaborative relationships, collaborators' overall sentiment states and position difference to understand how collaborative relationships affect sentiment consistency. From an organizational standpoint, this can provide guidelines to promote effective collaborative relationships.

RQ3: What is the impact of sentiment consistency on the formation of collaborative relationships?

We investigate whether sentiment consistency has an inverse impact on collaborations. This effort aims to help managers coordinate collaborators' sentiments targetedly.

II. RELATED WORK

A. Sentiment Analysis for OSS projects

Sentiment analysis [11] uses natural language processing, text analysis and computational techniques to automate the extraction or classification of sentiments from texts. There are a number of mature and publicly available tools like SentiStrength [12], Stanford NLP sentiment analyser [13] and Natural Language Text Processing (NLTK) [14]. Applying sentiment analysis to SE communities is a relative new research field. However, sentiment analysis tools trained or evaluated on non-technical datasets can generate unreliable results on SE datasets [15]. Therefore, some tools have been developed specifically for SE domain, like SentiStrength-SE [16], Senti4SD [17] and SentiCR [18].

Based on these tools, researchers are focusing their efforts on understanding how different factors interact with developers' sentiments. Some studies explore the factors that affect developers' sentiments. Pletea, Vasilescu, and Serebrenik [19] analyze commits and pull requests on GitHub and finds that more negative emotions are expressed in security-related discussions. Java projects are found to attract more negative comments while projects with more distributed teams attract more positive comments [20]. A study investigating commit logs on GitHub finds that Tuesday's comments have the most negative sentiments [6].

Other studies evaluate how developers' sentiments impact their performance to reveal what kinds of sentiments benefit the development process. An online survey [21] shows that *anger* can enhance developers' productivity, while *frustration* and *disgust* may bring risks. A study on the OSS project GENTOO shows that developers expressing strong emotions in issue trackers are more likely to become inactive in the projects they contribute to [22]. Ortu et al. [7] build a logistic regression model on 560k JIRA comments and find that the more positive the average sentiment, the faster an issue is fixed.

B. Factors influencing successful collaborative relationships

The success of software development largely depends on developers' collaboration efficiency and many factors influence the formation of successful collaborative relationships. The work by Kotlarsky and Oshri [23] suggests that human-related issues, such as rapport and transactive memory, are important for collaborative work. Joint intention, sharing of goals, plans and knowledge of the environment, awareness of the roles and responsibilities and team awareness are identified as the capabilities needed by an effective team [24]. Trust is another factor in forming successful collaborative relationships [25]. Perrault et al. [26] prove having learning as a purpose and sharing leadership to be success factors. Unfortunately, the role of sentiment in collaborative relationships has not been investigated yet.

TABLE I
DETAILS OF PROJECTS

Project	Language	#Issues	#Developers	Avg. Iss. per Dev.
Three.js	JavaScript	5465	2011	2.72
Pandas	Python	22854	5713	4.0
IPython	Python	10172	3413	2.98
gRPC	C++	14828	3142	4.72
OpenRA	C#	6026	682	8.84

III. PROPOSED METHODS

In this section, we describe our dataset and data processing methods for the subsequent analyses[1].

A. Dataset

GitHub is a popular code repository site for many well-known and active OSS projects. In GitHub, each project has its own repository and the history of the source code, commits, issues, and other related data are all publicly accessible. We obtain the dataset through GitHub REST API[2]. TABLE I lists the five mid-to-large scale projects we focus on. To ensure a high sample coverage [27] of the sampled data, the selection of programming languages is basically in line with the Top Languages[3] on GitHub. Besides, we take average number of issues each developer participates into account.

B. Data Extraction

1) Identification of Collaborative Relationships: Generally, collaborative relationships are formed when two people work together to accomplish common goals. In GitHub, issue reports are used by team members to ask for advice, and express and share opinions related to software maintenance and evolution [28]. In our study, collaboration between two developers is defined as the issue resolution process they both participate in. A collaborative relationship is identified when two developers both post comments under an issue.

2) Sentiment Analysis and Sentiment Consistency: Sentiments are commonly expressed in developers' issue comments [29]. We perform sentiment analysis using SentiCR and retrain the classifier by a gold standard [30] containing 3,000 manually labeled issue comments of ten OSS projects on GitHub.

To quantify the relation between sentiments expressed by collaborators in the software development process, we define sentiment consistency, which is identified through comparing collaborators' sentiment polarities. Two comments with the same sentiment polarity (both positive/negative/neutral) are considered to be sentiment consistent. Two comments with opposing sentiment polarities (one detected as positive while the other detected as negative) are considered to be sentiment inconsistent.

For a collaborative relationship involving two developers d_j and d_k, sentiment consistency in issue ι, denoted as $C_{\langle d_j, d_k \rangle}(\iota)$, is the number of sentiment-consistent comment pairs they post

---

[1] Code and data are released on http://doi.org/10.5281/zenodo.3608892
[2] https://developer.github.com/v3/
[3] https://octoverse.github.com/

in ι divided by the total number of comment pairs they post in ι. Formally,

$$C_{\langle d_j, d_k \rangle}(\iota) = \frac{\sum\limits_{p_i \in P} c_{d_j}^{p_i}(\iota) c_{d_k}^{p_i}(\iota)}{c_{d_j}(\iota) c_{d_k}(\iota)} \qquad (1)$$

where $P = \{positive, \ negative, \ neutral\}$. $c_{d_j}^{p_i}(\iota)$ and $c_{d_k}^{p_i}(\iota)$ represent the number of comments posted by d_j and d_k in issue ι with polarity p_i respectively. $c_{d_j}(\iota)$ and $c_{d_k}(\iota)$ represent the total number of comments in issue ι by d_j and d_k. Sentiment consistency over collaborative relationship $\langle d_j, d_k \rangle$ is then formulated as the mean of sentiment consistency in all the n issues co-participated by d_j and d_k:

$$C_{\langle d_j, d_k \rangle} = \frac{\sum\limits_{i=1}^{n} C_{\langle d_j, d_k \rangle}(\iota_i)}{n} \qquad (2)$$

C. Dynamic Collaboration Sentiment Network

Fig. 1. From left to right: Original, Positive and Negative Network of Three.js.

1) Network construction: A (static) collaboration network N^t is defined as a network of collaborative relationships in which each node is a developer, and two nodes are connected if there is a collaborative relationship between these two developers during period t. Each edge is associated with a weight corresponding to the times of collaborations.

Besides the original collaboration network, we construct a positive-consistent and a negative-consistent sentiment collaboration network from the extracted data. This is achieved through consistency filtering: we only keep the sentiment-consistent collaborations and remove the others. In the network view, this means that the weight of an edge is reduced by the times of collaborations in which the two collaborators (nodes) do not share common positive or negative sentiments. Figure 1 shows the three obtained networks of Three.js. These networks provide useful informaiton of collaborative relationships as well as corresponding sentiment effects inside the project, so that we can interpret how sentiment consistency impacts collaborative relationships from a structural point of view.

Moreover, we construct dynamic networks to analyze the evolution of each network structure. A dynamic collaboration network N is a sequence of collaboration networks corresponding to different periods of time.

$$N := (N^{t_1}, N^{t_2}, ..., N^{t_n}) \qquad (3)$$

where the periods $t_1, ..., t_n$ are obtained by dividing the overall development time into half-year intervals.

2) Network Analysis: We want to analyze the collaboration networks in terms of connectivity, community structure and betweenness to identify positive and negative sentiments'

	Collaborators mean	std	Non-collaborators mean	std	p for t-test
Three.js	0.471	0.176	0.4464	0.148	<0.0001
Pandas	0.517	0.189	0.483	0.161	<0.0001
IPython	0.475	0.162	0.448	0.139	<0.0001
gRPC	0.581	0.211	0.545	0.174	<0.0001
OpenRA	0.635	0.172	0.543	0.162	<0.0001

different impacts on collaborative relationships. We focus on three global measures: mean clustering coefficient, modularity and average betweenness centrality to mitigate the influence of network size.

a) Mean clustering coefficient: The clustering coefficient [31] of a node is defined as:

$$c_i = \frac{2n_i}{k_i(k_i - 1)} \qquad (4)$$

where n_i denotes the number of edges between the k_i neighbors of node i. The intuition is that $k_i(k_i - 1)/2$ edges can exist between k_i nodes, and the clustering coefficient reflects the fraction of existing edges between neighbors divided by the total number of possible edges. We employ the mean clustering coefficient to measure to what degree collaborators tend to cluster together in different networks.

b) Modularity: Modularity is the standard measure to quantify the strength of a community structure [32]. Networks with high modularity have dense connections between the nodes within modules but sparse connections between nodes in different modules. In our context, modularity indicates whether collaborators with consistent sentiment are divided into separate groups or integrated into a cohesive whole.

c) Average Betweenness Centrality: For a node v, betweenness centrality [33] is the sum of the fraction of all-pairs shortest paths that pass through v in the network:

$$c_B(v) = \sum_{s,t \in V} \frac{\sigma(s, t|v)}{\sigma(s, t)} \qquad (5)$$

where V is the set of nodes, $\sigma(s, t)$ is the number of shortest (s, t)-paths and $\sigma(s, t|v)$ is the number of those paths that contain node v in between. High betweenness centrality indicates that the person plays the role of gatekeeper in the social network, with the potential to disrupt connections between various end points.

IV. RESULTS AND FINDINGS

A. Collaborators vs. Non-collaborators

Sentiment consistency over non-collaborative relationships is the sentiment-consistent comment pairs divided by the total comment pairs of two developers in issues excluding the n co-participated ones. Formally,

$$C'_{\langle d_j, d_k \rangle}(\iota) = \frac{\sum\limits_{p_i \in P} c_{d_j}^{p_i} c_{d_k}^{p_i} - \sum\limits_{i=1}^{n} \sum\limits_{p_i \in P} c_{d_j}^{p_i}(\iota_i) c_{d_k}^{p_i}(\iota_i)}{c_{d_j} c_{d_k} - \sum\limits_{i=1}^{n} c_{d_j}(\iota_i) c_{d_k}(\iota_i)} \qquad (6)$$

TABLE III
SPEARMAN COEFFICIENTS BETWEEN THREE FACTORS AND
SENTIMENT CONSISTENCY

	#Common Issues		Sentiment		Position Difference	
Three.js	0.116	***	0.357	***	0.057	*
Pandas	0.062	**	0.328	***	0.001	0.942
IPython	0.122	***	0.273	***	-0.035	0.132
gRPC	0.27	***	0.107	***	-0.227	***
OpenRA	0.233	***	0.44	***	-0.076	0.107

*$p < 0.05$, **$p < 0.01$, ***$p < 0.001$

TABLE IV
SPEARMAN COEFFICIENTS BETWEEN COLLABORATORS' POSITION
DIFFERENCE AND TIMES OF COLLABORATIONS

Repository	Coefficient	p-value
Three.js	-0.085	<0.001
Pandas	-0.356	<0.0001
IPython	-0.304	<0.0001
gRPC	-0.629	<0.0001
OpenRA	-0.423	<0.0001

The means of sentiment consistency for collaborators and non-collaborators are compared through independent t-tests. It is estimated from the distribution plots that our data are normally distributed. It can be found in TABLE II that collaborators share more consistent sentiments than non-collaborators.

B. Factors Influencing Sentiment Consistency

We investigate the correlation of three factors with sentiment consistency over collaborative relationships, i.e., closeness of the collaborative relationship, the collaborators' overall sentiment state and the position difference between them. The model outputs are listed in TABLE III.

1) Closeness of Collaborative Relationship: We measure the closeness of a collaborative relationship through the number of issues each pair of collaborators co-participate in. Its Spearman correlation coefficient with collaborators' sentiment consistency is calculated. It is found that sentiment consistency is higher in collaborators with more co-participated issues.

2) Collaborators' Overall Sentiment State: A study on Twitter shows that the positive sentiment is contagious because community members increasingly share positive tweets more than negative ones over time [34]. We try to validate whether this effect can be applied to collaborations in OSS development. A developer's sentiment state is measured through the number of non-negative comments divided by the total number of his/her comments. We analyze the correlation between the average sentiment state of two collaborators and their sentiment consistency over the collaborative relationship. A positive correlation is found.

3) Position difference between collaborators: In this study, we want to identify whether different positions between collaborators can impact sentiment consistency in the OSS development process. We employ node degree of the basic collaboration network in Section 3.3.1 to represent a developer's position in the project. The normalized position difference of two collaborators is formulated as $\frac{|d_1 - d_2|}{max(d_1, d_2)}$, where d_1 and d_2 stand for the node degree. Its Spearman correlation coefficient with sentiment consistency is then measured.

A negative correlation between the sentiment consistency and collaborators' position difference is found in gRPC and OpenRA. We further investigate the correlations between collaborators' position difference and times of collaborations. The results are shown in TABLE IV.

It can be found that there are negative correlations between collaborators' position difference and times of collaborations in gRPC and OpenRA. Weak negative correlations appear in

Pandas and IPython while there is no correlation in Three.js. Furthermore, by comparing the attributes of these five projects, it can be found that the correlation between collaborators' position difference and times of collaborations is affected by the average number of issues that a developer participated in (See TABLE I). In projects that developers participate in quite a few issues (gRPC and OpenRA), the developers of similar positions tend to have more collaborations, which contribute to higher sentiment consistency. On the contrary, in projects that developers only participate in a small number of issues (Three.js), positions do not impact their collaborations. Actually, there are no big differences between developers' positions in these projects.

C. Impacts of Sentiment Consistency on Collaboration

We employ the Granger causality test [35] to determine whether sentiment consistency has a causal relationship with collaboration willingness. For two time series X and Y, if Y can be better predicted using the lagged values of both X and Y than using the lagged values of Y alone, then X is said to Granger cause Y. In this context, we investigate whether the occurrence of collaborations is Granger caused by the increase of sentiment consistency in a prior period.

$$Y_t = \mu + \sum_{i=1}^{L} \alpha_i Y_{t-i} + \sum_{i=1}^{L} \beta_i X_{t-i} \qquad (7)$$
$$L = max.\ no.\ of\ lags$$

For each pair of collaborators, we extract the frequency of sentiment consistency and the frequency of collaborations within a week. We run the adfuller test to select stationary time series, which is the precondition required by Granger causality tests. Then we run an independent Granger causality test on the two time series. The maximum time lags are 2, 4, 8 and 12 weeks respectively. The numbers of significant causal relationships are compared in TABLE V. It can be found that the effect of sentiment consistency on collaborations is more significant than sentiment inconsistency; the effect of positive consistency is more significant than negative consistency. We further investigate the evolution of network measures of the three constructed networks. Figure 2 illustrates the results. As can be seen, the mean clustering coefficients of negative collaboration networks are lower than the positive ones, implying that collaborators with positive-consistent sentiments are more clustered and interconnected. The modularity of negative collaboration networks is higher than positive ones, implying that developers with negative-consistent sentiments
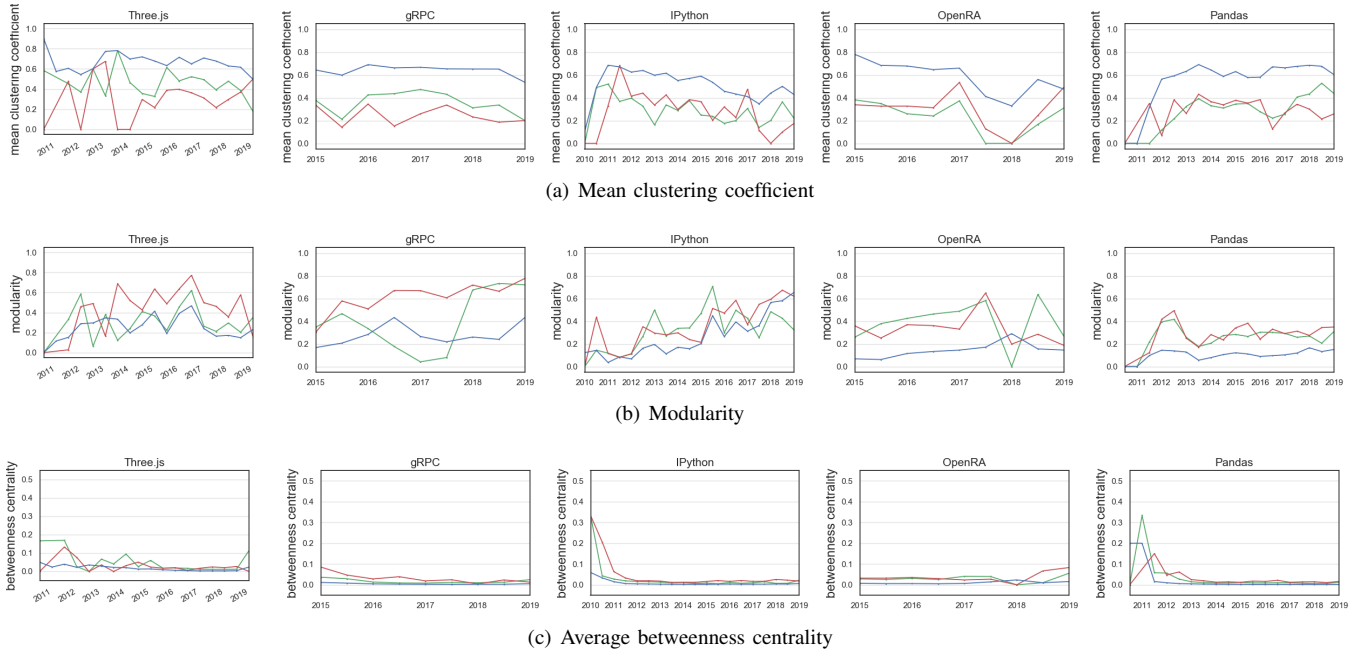
(a) Mean clustering coefficient

(b) Modularity

(c) Average betweenness centrality

Fig. 2. Evolution of Network Measures in Original *(blue)*, Positive *(green)* and Negative *(red)* Collaboration Networks in Five Projects.

TABLE V
TOTAL NUMBER OF COLLABORATIVE RELATIONSHIPS WITH SIGNIFICANT
CAUSAL EFFECTS OVER DIFFERENT SENTIMENT POLARITIES.

Polarity	#Collaborations	Total	*p*-value
consistent	1782 (47.9%)	3807	3.80e-35
inconsistent	1109 (27.5%)	4029	
positive consistent	1120 (28.2%)	3973	7.22e-71
negative consistent	686 (16.7%)	4100	

are more densely distributed inside small groups. The betweenness centrality coefficients are generally higher in negative collaboration networks, implying that negative developers play a more important role in sentiment propagation through collaborations than positive developers do.

V. IMPLICATIONS

Although how different factors interact with developers' sentiments during development has been studied, there is no study to analyze the role of sentiment in collaborative relationships. Our findings indicate that positive sentiment linkage can boost collaborators' closeness and should be encouraged in software development to foster a better collaboration ecology.

It is also suggested that negative sentiment effects are more likely to be reduced than augmenting positive ones through adjustments and the reassignment of collaborators based on the network features as well as factors influencing sentiment consistency, so as to maximize collaboration willingness and efficiency. The results also indicate that developers of different positions in the collaboration network tend to have fewer collaborations in practice, which may reduce sentiment consistency accordingly. This inspires us that we should promote the collaborations among developers of different positions in the collaboration network. More specifically, to encourage positive developers of high degree (i.e., of central position) to cooperate with negative developers of low degree (i.e., of peripheral position) can bring more gains to projects.

These findings tell us when we are going to coordinate the OSS projects, we should take the consistency of developers' sentiments into account in order to promote their collaborations in a task. As a measure of implementations, the consistency of developer' sentiments can be monitored so that we can take appropriate measures to regulate the organization of the development process. It also encourages us to incorporate these new features into the future monitoring tools.

VI. THREATS TO VALIDITY

In this preliminary study, we only study a sample of projects. We take into account different developer densities and programming languages to ensure the diversity of samples and the generalizability of our results.

We use SentiCR, a customized sentiment analysis tool for SE domain for sentiment analysis, and retrain it with a gold standard on GitHub issues. However, misclassifications may still exist and bring noise to our dataset.

Another threat is the definition of collaborative relationships. Collaborations on the same file are not taken into account because it is difficult to extract the sentiments during this kind of collaboration. Additionally, all the comments in an issue are considered in the calculation of sentiment consistency between two developers because it is difficult to determine whether they are in the same thread of conversation. However, from our observation, two developers may discuss irrelevant tasks in one issue, leading to a mis-detected collaborative relationship. Topic extraction or NLP techniques can be further adopted to address this issue.

VII. Conclusions

We use sentiment consistency to quantify the relation between sentiments in collaborative relationships. Our results show that collaborators share more consistent sentiments and sentiment consistency has a positive correlation with the closeness of collaborative relationships and collaborators' overall sentiment states. It has a negative correlation with position difference in projects that developers participate in quite a few issues. Results of the Granger causality test show that positive consistent sentiments have the most significant impact on collaboration willingness, followed by inconsistent and negative-consistent sentiments. Network analysis shows negative consistent collaborators are more alienated and distributed in small groups, while positive consistent collaborators tend to cluster into a cohesive whole. In a follow-up study, we plan to include more projects and refine our methods for collaborative relationship detection. We also plan to research the main causes of sentiment fluctuation over collaborative relationships, and develop specific strategies for monitoring such sentiment phenomena. We hope our results can motivate further research to help coordinate developers' collaborations and provide better tools for higher serenity and productivity in software development communities.

Acknowledgement

References

[1] Grady Booch and Alan W Brown. "Collaborative development environments". In: *Advances in computers* 59.1 (2003), pp. 1–27.

[2] H Keith Edwards and Varadharajan Sridhar. "Analysis of the effectiveness of global virtual teams in software engineering projects". In: *36th Annual Hawaii International Conference on System Sciences, 2003. Proceedings of the*. IEEE. 2003, 9–pp.

[3] Cynthia D Fisher and Neal M Ashkanasy. "The emerging role of emotions in work life: An introduction". In: *Journal of Organizational Behavior: The International Journal of Industrial, Occupational and Organizational Psychology and Behavior* 21.2 (2000), pp. 123–129.

[4] Iftikhar Ahmed Khan, Willem-Paul Brinkman, and Robert M Hierons. "Do moods affect programmers' debug performance?" In: *Cognition, Technology & Work* 13.4 (2011), pp. 245–258.

[5] Daniel Graziotin, Xiaofeng Wang, and Pekka Abrahamsson. "Happy software developers solve problems better: psychological measurements in empirical software engineering". In: *PeerJ* 2 (2014), e289.

[6] Vinayak Sinha, Alina Lazar, and Bonita Sharif. "Analyzing developer sentiment in commit logs". In: *Proceedings of the 13th International Conference on Mining Software Repositories*. ACM. 2016, pp. 520–523.

[7] Marco Ortu et al. "Are bullies more productive?: empirical study of affectiveness vs. issue fixing time". In: *Proceedings of the 12th Working Conference on Mining Software Repositories*. IEEE Press. 2015, pp. 303–313.

[8] Mihai Grigore and Christoph Rosenkranz. "Increasing the Willingness to Collaborate Online: an Analysis of Sentiment-Driven Interactions in Peer Content Production." In: vol. 4. Jan. 2011.

[9] Jane Conoley and Collie Conoley. "Why Does Collaboration Work? Linking Positive Psychology and Collaboration". In: *Journal of Educational and Psychological Consultation - J EDUC PSYCHOLOGICAL CONS* 20 (Feb. 2010), pp. 75–82.

[10] Karim R Lakhani and Eric Von Hippel. "How open source software works:"free" user-to-user assistance". In: *Produktentwicklung mit virtuellen Communities*. Springer, 2004, pp. 303–339.

[11] Bing Liu et al. "Sentiment Analysis and Subjectivity." In: *Handbook of natural language processing* 2.2010 (2010), pp. 627–666.

[12] Mike Thelwall et al. "Sentiment strength detection in short informal text". In: *Journal of the American Society for Information Science and Technology* 61.12 (2010), pp. 2544–2558.

[13] Richard Socher et al. "Recursive deep models for semantic compositionality over a sentiment treebank". In: *Proceedings of the 2013 Conference on Empirical mMthods in Natural Language Processing*. 2013, pp. 1631–1642.

[14] Edward Loper and Steven Bird. "NLTK: the natural language toolkit". In: *arXiv preprint cs/0205028* (2002).

[15] Robbert Jongeling et al. "On negative results when using sentiment analysis tools for software engineering research". In: *Empirical Software Engineering* 22.5 (2017), pp. 2543–2584.

[16] Md Rakibul Islam and Minhaz F Zibran. "Leveraging automated sentiment analysis in software engineering". In: *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*. IEEE. 2017, pp. 203–214.

[17] Fabio Calefato et al. "Sentiment polarity detection for software development". In: *Empirical Software Engineering* 23.3 (2018), pp. 1352–1382.

[18] Toufique Ahmed et al. "SentiCR: a customized sentiment analysis tool for code review interactions". In: *Proceedings of the 32nd IEEE/ACM International Conference on Automated Software Engineering*. IEEE Press. 2017, pp. 106–111.

[19] Daniel Pletea, Bogdan Vasilescu, and Alexander Serebrenik. "Security and emotion: sentiment analysis of security discussions on GitHub". In: *Proceedings of the 11th Working Conference on Mining Software Repositories*. ACM. 2014, pp. 348–351.

[20] Emitza Guzman, David Azócar, and Yang Li. "Sentiment analysis of commit comments in GitHub: An empirical study". In: *11th Working Conference on Mining Software Repositories, MSR 2014 - Proceedings* (May 2014). DOI: 10.1145/2597073.2597118.

[21] Michal R Wrobel. "Emotions in the software development process". In: *2013 6th International Conference on Human System Interactions (HSI)*. IEEE. 2013, pp. 518–523.

[22] David Garcia, Marcelo Serrano Zanetti, and Frank Schweitzer. "The role of emotions in contributors activity: A case study on the Gentoo community". In: *2013 International Conference on Cloud and Green Computing*. IEEE. 2013, pp. 410–417.

[23] Julia Kotlarsky and Ilan Oshri. "Social ties, knowledge sharing and successful collaboration in globally distributed system development projects". In: *European Journal of Information Systems* 14.1 (2005), pp. 37–48.

[24] Katia Sycara and Gita Sukthankar. "Literature review of teamwork models". In: *Robotics Institute, Carnegie Mellon University* 31 (2006), p. 31.

[25] Valerie Lynne Herzog. "2000 International Student Paper Award Winner: Trust Building on Corporate Collaborative Project Teams". In: *Project Management Journal* 32.1 (2001), pp. 28–37.

[26] Ellen Perrault et al. "Working together in collaborations: Successful process factors for community collaboration". In: *Administration in social work* 35.3 (2011), pp. 282–298.

[27] Meiyappan Nagappan, Thomas Zimmermann, and Christian Bird. "Diversity in software engineering research". In: *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering*. ACM. 2013, pp. 466–476.

[28] Alessandro Murgia et al. "Do developers feel emotions? an exploratory analysis of emotions in software artifacts". In: *Proceedings of the 11th Working Conference on Mining Software Repositories*. ACM. 2014, pp. 262–271.

[29] Francisco Jurado and Pilar Rodriguez. "Sentiment Analysis in monitoring software development processes: An exploratory case study on GitHub's project issues". In: *Journal of Systems and Software* 104 (2015), pp. 82–89.

[30] Jin Ding et al. "Entity-level sentiment analysis of issue comments". In: *Proceedings of the 3rd International Workshop on Emotion Awareness in Software Engineering*. ACM. 2018, pp. 7–13.

[31] Stefano Boccaletti et al. "Complex networks: Structure and dynamics". In: *Physics reports* 424.4-5 (2006), pp. 175–308.

[32] Mark EJ Newman. "Modularity and community structure in networks". In: *Proceedings of the National Academy of Sciences* 103.23 (2006), pp. 8577–8582.

[33] Linton C Freeman. "A set of measures of centrality based on betweenness". In: *Sociometry* (1977), pp. 35–41.

[34] Noha Alduaiji and Amitava Datta. "An Empirical Study on Sentiments in Twitter Communities". In: *2018 IEEE International Conference on Data Mining Workshops (ICDMW)*. IEEE. 2018, pp. 1166–1172.

[35] Clive WJ Granger. "Investigating causal relations by econometric models and cross-spectral methods". In: *Econometrica: Journal of the Econometric Society* (1969), pp. 424–438.

Searching, Examining, and Exploiting In-demand Technical (SEE IT) Skills using Web Data Mining

Taeghyun Kang
Department of Computer Science
University of Central Missouri
Warrensburg, MO, USA
tkang@ucmo.edu

Hyungbae Park
Department of Computer Science
University of Central Missouri
Warrensburg, MO, USA
park@ucmo.edu

Sunae Shin
Department of Computer Science
University of Central Missouri
Warrensburg, MO, USA
sshin@ucmo.edu

Abstract—There has always been a mismatch between academic education and industry demands in various industry fields. Especially, in the computer related jobs such as software developers or engineers, employers and job seekers are experiencing a large curriculum or skill gap compared to other industry fields due to its fast-changing nature. In order to minimize the gap, identifying the current job skill trends and offering/learning relevant skills are critically important for both educators and students in the fast-paced job market. However, due to the lack of relative information about the trends of in demand technical skills in certain regions or around the nation, it is difficult for educators to keep up with these changes and determine what needs to be conveyed to students so they can be ready for the job on day one. In this paper, in order to address these challenges, we focused on the analysis of in demand technical skills and their trends in software developer and engineer jobs around the nation. We collected and analyzed over 120,000 software developer or engineer job postings and summarized demanding and required skills from different regions throughout the nation.

Index Terms—Web Data Mining, Job Trend Analysis, Software Developer, Software Engineer, Programming Languages

I. INTRODUCTION

The academic program's job placement rate is one of important performance indicators that can be used to measure the success of a program. Programs, which demonstrate a strong job placement rate, frequently use this information for advertisement. It is critical for educators to convey current knowledge that is currently demanded or required by employers. However, due to the fast-changing nature in the requirements of computer related jobs, it is challenging that we minimize the gap between academic education and industry requirements. This is not the matter of the quality of the academic program or students being lazy and not studying hard enough to meet the job requirements. The rapid changes of technologies are a nature of industry caused by factors such as satisfying the customers' demands and achieving a company's revenue goal efficiently. As an educator, we need to keep asking ourselves, "Do our graduates possess the skills employers need?" and "How do we keep up with fast-changing industry requirements?" In order to clearly answer the questions, it is inevitable to make a systematic way that continuously monitors and analyzes the trends and changes in job markets. There are various job search engines available such as Glassdoor, Indeed, Handshake, etc. Even though they

provide advanced search filter so the job seekers can easily identify the jobs they are looking for, search features of these websites have limited capability and do not show the trends of jobs or specific skills. Many academic programs have their industry advisory board meeting to minimize the gap, but it is still difficult to see the overall trends of required skills between states and most in-demand skills in each state. Job seekers and students also proactively find available resources to identify job trends or requirements so that they can be prepared for a job and find their job effectively and efficiently. If failed, they may need to re-educate themselves.

In this paper, we focus on making a systematic tool to monitor and analyze in demand technical skills and their trends in software developer and engineer jobs around the nation rather than addressing programming languages expected to be offered in specific regions. We collected and analyzed over 120,000 software developer or engineer job postings around the nation over the five month period from October 2019 to February 2020 and summarized demanding and required skills from different regions throughout the nation. The results show the popularity of integrated development environments, project management tools, and programming languages and their correlations. The remainder of the paper is structured as follows: Section II describes the motivation of the paper and reviews background knowledge and literature references. Section III describes the proposed tool for job skills trend analysis and the data sets processed by the tool. Section IV shows the results of our analysis and Section V concludes the paper.

II. RELATED WORK

Web mining is the technique that collects and processes freely available data from web postings and web documents and discovers and extracts useful insight, knowledge, and information. Web mining can be categorized into three broad areas [14], [17]–[19] such as web usage mining, web content mining, and web structure mining. The enormous amount of web data can be counterproductive without proper processing procedures or automated and systematic methodologies. It will take a long time to collect and digest data and this tedious and un-automated process can mislead the viewers. There are various important techniques in order to make

data mining effective such as generalization, tracking patterns, classification, characterization, association, outlier detection, clustering, regression, and prediction [8], [9]. Many research work have used these web data mining techniques to find and extract hidden information from the ever-growing number of web postings and documents.

Milad Eftekhar et al. [15] proposed two models such as intrinsic burst model and social burst model, in order to find and extract knowledge and information from burst behaviors and neighbors' influences when identifying bursts on social network sites. They used two graphs called action graph and holistic graph to characterize and identify users' bursty behavior.

Nicola Barbieri et al. [16] proposed the Community–Cascade Network(CCN) model that utilizes information propagation and community formation in a social network can be explained according to the level of active involvement and the degree of passive involvement, which ultimately guide a user behavior within the network. They validated the proposed CCN model by applying it to real-world social networks.

Wilden et al. [10] presented a framework that suggests the effectiveness of a brand signal to potential employees. The employee-based brand equity influenced by employer brand clarity, consistency, brand investments and the credibility of brand signals. Furthermore, the paper present that the brand investment influences both attractiveness of a prospective employer and employee based brand equity.

Xu et al. [11] investigated the prediction of job change occasion based on career mobility and daily activity patterns at the individual level. In order to model the job change motivations and correlations between professional and daily life, the work experiences and check-in records of individuals are collected. They found that the job change occasions are predictable and shown on the experiment based on the large real-world data set.

The talent exchange prediction method is developed in [12] for predicting the possible companies for the potential employees. The proposed talent circle detection model extracts talent circles that includes the organizations with similar talent patterns. In addition, the semantic meaning is offered for detected circles are labeled with job description. With the proposed method, the organizations are able to seek the right talent during recruitment and job seekers can find appropriate jobs.

H. Li et al. [13] proposed a model that focuses on predicting the turnover and career progression of talents. The survival analysis approach shows survival status at a sequence of time intervals for turnover behaviors of employees. Moreover, the prediction of the relative occupational level is framed for modeling career progression.

Among various applications mentioned above, the applications related to job search and trends [10]–[13] are our interest in this paper. This analysis will help identify and reduce the gap between academia education and industry demands. L. Buth et al. [1] analyzed the needs of industry by interviewing employers and the situation of a university

responsible for educating students. They identified the gaps in 1) applied knowledge and problem solving skills, 2) communication skills, and 3) self-discipline and positive work attitude. Bracey [22] pointed particularly to the absence of teaching "soft skills" in academic degree programs. She mentioned that majority of employers are demanding soft skills as a pre-condition for their employment. McGill [24] particularly surveyed the hiring needs of game industry and compared them against game development curriculum at post-secondary institutions. Hynninen et al. [23] conducted the similar survey in the field of software testing and quality assurance. They identified the differences between what is considered important skills by the industry and those taught in academia. Cheng et al. [25] presented an analysis of job transition network for recruitment. The authors suggested a real-time system for mining job–related patterns collected from social media sources.

The references listed above indicate the need from field professionals to understand the employable skills in this dynamic job market. This is critical for employers and students because they seek qualified employees and these job opportunities, respectively. For employers, it will improve the chances of hiring the right person and for students it will improve the chances of getting hired.
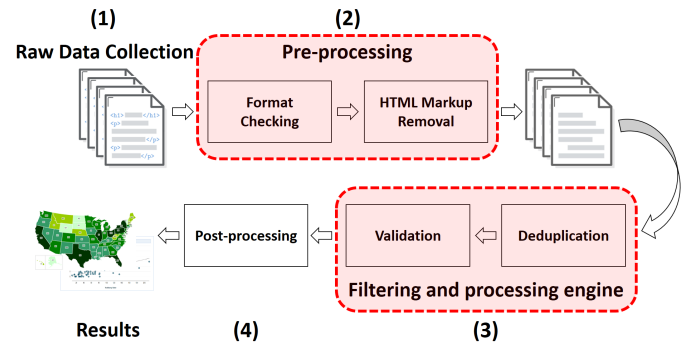

Fig. 1. The overview of data processing flow through multiple phases

III. Data Mining Tool and Data Sets

We developed a home-grown web data mining tool that crawls job posting web pages from job search websites (i.e., indeed), collects raw data of job postings, and extracts necessary information from each job posting in order to achieve the objectives of our analysis. In order to improve the efficiency of our tool and the accuracy of the data sets, we conducted black box and white box testings on our tool. Through this continuous improvement process, we were able to get more accurate data sets within a shorter amount of time.

The series of data handling processes can be divided into several phases. In the raw data collection phase (1) in Figure 1, we developed our own crawler that is customized for different job search websites. Each job search website has a distinct structure and it uses different web page management methods.

In order to efficiently discover and collect job postings from the web, we needed to customize our crawler for different job search websites and collected more than 120,000 job postings from the web. This may seem that we are having an enormous amount of opportunities. However the enormity of the data can be counterproductive in many ways. If we have to check out each job posting individually, then it would take a long time to tabulates and extract what educators, students, and job seekers need to focus and it may give misleading information to the viewers. In worst case, they may need to start over learning new programming languages. Therefore the collected raw data sets will be refined throughout the following three phases.

The next pre-processing phase (2) in Figure 1 utilizes the Beautiful Soup Python package [7] in order to efficiently extract necessary information from each job posting. The Beautiful Soup Python package makes it easy to parse HTML and XML documents and generates a parse tree. This parse tree can be used to extract specific data from HTML.

In the Filtering and processing phase (3) in Figure 1, we check the duplication of job postings and remove redundant job postings from each company. We noticed that there are many duplicated job postings for the same position from the same company. For each job posting, a unique value is assigned in the tag called "vjk". This is a unique identifier for each unique job description. We utilized the "vjk" ids and job descriptions to filter out unnecessary redundancies from the data sets. In order to achieve a highly accurate and quality collection of the data, the collected data was deduplicated and validated.

In the last phase (4) in Figure 1, we extract only information that corresponds to our analysis. After the collected raw data sets are processed through the four phases described in Figure 1, we were able to identify total 2,171 unique job descriptions out of over 120,000 job postings.

IV. Job Trend Analysis and Results

As described in the previous section, we have identified total 2,171 unique job descriptions out of over 120,000 job postings. This shows that there are on average 55 job postings for the same position with the exactly same job descriptions from the same company. Various reasons can cause the enormous amount of duplications; 1) a position has not been filled for a while, 2) a company wants to expose their job descriptions more frequently than others, 3) a company needs more employees for the same position with the same skill sets, etc. In this section, we will focus on those uniquely specified jobs for our analysis in order to minimize noise that may be caused by enormous duplications. The following subsections describe job requirement trends in several categories such as integrated development environments, software management tools, programming languages, etc.

A. Integrated Development Environments (IDEs)

IDEs have been used for decades and increase software developers' productivity. Therefore, this is definitely one of the required skills for any programmer positions around the world. However, the commonly used or required IDEs have not been specified in most of the job descriptions. The IDEs are specified by only a few companies around the nation. This is because each team even within a company uses multiple or different IDEs depending on programming languages they use to develop software or applications or to complete given tasks. For example, IntelliJ would be used for a project written in Java, Android studio for Android, RubyMine for Ruby, and etc. Therefore, it would be difficult for employers to specify any IDEs used in their job descriptions. This suggests that job seekers may want to expose themselves to multiple IDEs for different programming languages. Programmers need to be flexible and should be able to use any IDEs in their work environments. Based on the popularity of programming languages, the IDEs that support Java are placed top ranks (see Figure 4). Xcode is placed the fourth place and the popularity of Xcode is due to the popularity of Apple's iPhone around the world. We can see the strong correlation between the popularity of both IDEs and programming languages.
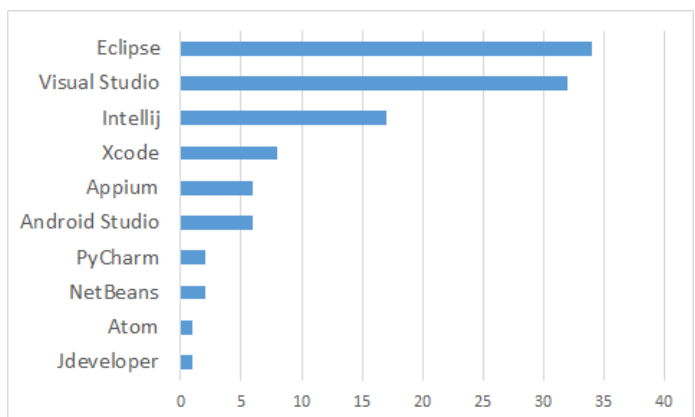

Fig. 2. Popularity of integrated development environments

B. Project Management Tools

We also conducted an analysis to identify which software development models or project management tools are commonly used in industry. Interestingly, the vast majority of companies is not specifying or asking a specific software development model or a project management tool in their job descriptions. However, companies in certain areas such as Washington, California, Arizona, Colorado, Missouri, Georgia, and New York showed a strong trend that these specifications are listed in their job description.

As shown in Figure 3, the Agile development is being specified most frequently compared to other agile project management processes such as Scrum, Lean, Kanban, etc. Traditional project management models such as the spiral or evolutionary models have not been specified by any of the job postings but only the waterfall model has been specified along with the agile process as a required background. These job descriptions required understanding of both agile and waterfall developments. This may indicate that the traditional software
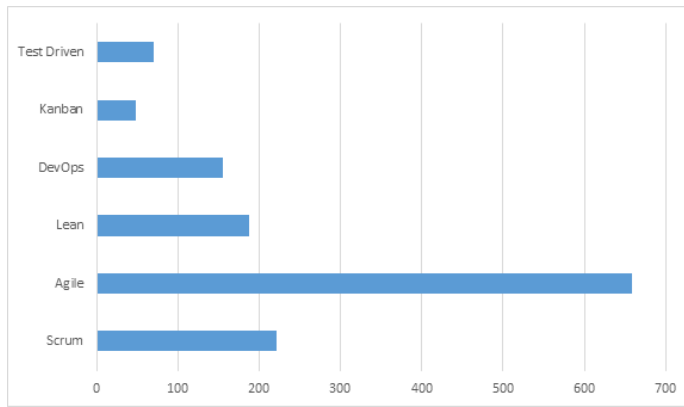
Fig. 3. Project management tools

development process has been converged to the most broadly used model among those traditional software development models. It also indicates that the trend of software development cycle has been shifted from the traditional project management models to the agile project management models. This does not mean that the traditional software development models are not being utilized in industry. There are many companies that use the combination of traditional software development and agile process.

C. Programming Languages

As we see in Figure 4, Java is the most popular programming languages around the nation. This is evident by the top ranked IDEs in Figure 2. All the top 4 IDEs in the list support Java. Compared to the popularity of Xcode shown in Figure 2, 'Swift' ranked lower in the category of popular programming languages. This is because Xcode support not just Swift, but also other programming languages such as C, C++, Objective-C, Objective-C++, Java, Python, and Ruby.

Fig. 4. Popularity of programming languages

D. Soft Skills

Teamwork, written and spoken communication skills, critical thinking, leadership, and work ethics are critical soft skills required and demanded by employers. Especially, some large corporations already have well-organized intensive programs for their new employees to teach hard skills and they even more emphasize the importance of soft skills as soft skills are hard to teach in a short period of time. It was evident by many surveys such as [22]–[24]. However, the vast majority of job descriptions may mislead job seekers, students, and educators as they are not specifying these critical skill sets. Even though the vast majority of companies wants to hire people who has excellent soft skills, they do not actively require or specify these soft skills in the job description in reality. It is understandable as soft skills are not only hard to teach but also hard to evaluate. But a clear job description with soft skills would help job seekers, students, and educators be aware of the importance of soft skills and prepare for it.

E. New Technologies

Along with the traditional programming languages and IDEs, we also conducted analysis for relatively new technologies such as blockchain, and TensorFlow. We have found 29 and 23 job postings specifying blockchain technologies and TensorFlow knowledge and background, respectively. These numbers are relatively lower than other traditional and popular programming languages. The adaptation of these new technologies will be collected and analyzed in our future work.

V. Conclusion

Offering and learning appropriate programming languages and demanding skills are critically important for both educators and students in the fast-paced job market. However, due to the lack of a systematic approach that analyzes job skills demands and trends, it is difficult to keep up with these fast changes. In this paper, we developed a job trend analysis tool, analyzed over 120,000 job postings, and summarized demanding and required skills from different regions throughout the nation.

As a continued effort toward this work, we will expand our analysis to other fast-paced and emerging disciplines such as cybersecurity and etc. This will also help both educators and students to identify in demand technical skills in the industry and frequently exploited vulnerabilities. In addition, we will analyze other countries' in demand technical skills to investigate the worldwide trend for a certain job. In addition, we are currently working on integrating machine learning features (e.g., TensorFlow [20]) so the tool can automatically generate reports that show the trends of job and programming languages without involving or requiring a manual analysis.

References

[1] L. Buth, V. Bhakar, N. Sihag, G. Posselt, S. Bohme, K.S. Sangwan, and C. Herrmann, "Bridging the qualification gap between academia and industry in India," Proceedings of the 7th conference on Learning Factories (CLF), Volume 9, pp.275–282, Elsevier, 2017.

[2] M.M. McGill, "Defining the expectation gap: a comparison of industry needs and existing game development curriculum," Proceedings of the 4th International Conference on Foundations of Digital Games, pp.129–136, April 2009.

[3] A.H. Harris, T.H. Greer, S.A. Morris and W.J. Clark, "Information systems job market late 1970'2-early 2010's," Journal of Computer Information Systems, pp.72–79, 2012.

[4] C. McLean, "A foot in the door: IT job-search strategies," Certification Magazine, Volume 8(4), pp.38–40, 2006.

[5] C. Litecky, A. Aken, A. Ahmad, and H.J. Nelson, "Mining for computing jobs." IEEE Software, Volume 27(1), pp.78–85, 2010.

[6] S. Zhong, "Information intelligent system based on web data mining," Proceedings of the International Symposium on Electronic Commerce and Security, pp.514–517, 2008.

[7] https://www.crummy.com/software/BeautifulSoup/#Download

[8] P. Berkhin, "Survey of Clustering Data Mining Techniques," Grouping Multidimensional Data, Springer, 2006.

[9] Shu-Hsien Liao, Pei-Hui Chu, Pei-Yuan Hsiao, "Data mining techniques and applications – A decade review from 2000 to 2011," Expert Systems with Applications, Volume 39, Issue 12, pp.11303–11311, Elsevier, 2012.

[10] R. Wilden, S. Gudergan, and I. Lings, "Employer branding: strategic implications for staff recruitment," Journal of Marketing Management, 26(1-2), pp.56–73, 2010.

[11] H. Xu, Z. Yu, H. Xiong, B. Guo, and H. Zhu, "Learning career mobility and human activity patterns for job change analysis," IEEE International Conference on Data Mining, pp.1057–1062, 2015.

[12] H. Xu, Z. Yu, J. Yang, H. Xiong, and H. Zhu, "Talent circle detection in job transition networks," Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp.655–664, 2016.

[13] H. Li, Y. Ge, H. Zhu, H. Xiong, and H. Zhao, "Prospecting the career development of talents: A survival analysis perspective," Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp.917–925, 2017.

[14] N. R. Satish, "A Study on Applications, Approaches and Issues of Web Content Mining," International Journal of Trend in Research and Development, Volume 4(6), ISSN: 2394-9333, 2017.

[15] Milad Eftekhar, Nick Koudas, and Yashar Ganjali, "Bursty Subgraphs in Social Networks," Proceedings of the 6th ACM international conference on Web search and data mining (WSDM), pp.213–222, Rome, Italy, 2013.

[16] Nicola Barbieri, Francesco Bonchi, Giuseppe Manco, "Cascade-based Community Detection," Proceedings of the 6th ACM international conference on Web search and data mining (WSDM), pp.33–42, Rome, Italy, 2013.

[17] Ujwala Manoj Patil, J.B. Patil, "Web Data Mining Trends and Techniques," International Conference on Advances in Computing, Communications and Informatics (ICACCI), pp.962–965, 2012.

[18] B. Liu, "Web Data Mining Exploring Hyperlinks, Contents, and Usages Data," 2nd Edition, Springer, 2007.

[19] Jai Prakash, Bankim Patel, Atul Patel, "Web Mining: Opinion and Feedback Analysis for Educational Institutions," IJCA, Volume 84(6), 2013.

[20] "TensorFlow: An end-to-end open source machine learning platform," url: https://www.tensorflow.org/

[21] D. Smith and A. Ali, "Analyzing Computer Programming Job Trend Using Web Data Mining," Issues in Informing Science and Information Technology, Volume 11, pp.203–214, 2014.

[22] P. Bracey, "Analyzing Internet Job Advertisements to Compare IT Employer Demands versus Undergraduate IT Program Curriculum in Texas," Proceedings of E-Learn: World Conference on E-Learning in Corporate, Government, Healthcare, and Higher Education, pp.37–42, ISBN 978-1-880094-90-7, 2011.

[23] T. Hynninen, J. Kasurinen, A. Knutas, and O. Taipale, "Guidelines for software testing education objectives from industry practices with a constructive alignment approach," pp.278–283. 10.1145/3197091.3197108, 2018.

[24] M. M. McGill, "Defining the expectation gap: a comparison of industry needs and existing game development curriculum," Proceedings of the 4th International Conference on Foundations of Digital GamesApril (FDG), pp.129-–136, 2009.

[25] Y. Cheng, Y. Xie, Z. Chen, A. Agrawal, A. Choudhary, and S. Guo, "Jobminer: A real-time system for mining job-related patterns from social media," Proceedings of ACM Special Interest Group on Knowledge Discovery and Data Mining (SIGKDD), 2013

Sentiment Analysis of Online Reviews with a Hierarchical Attention Network

Jingren Zhou[1], Peiquan Jin[1*], Jie Zhao[2*]

[1]School of Computer Science and Technology, University of Science and Technology of China, Hefei, China
[2]School of Business, Anhui University, Hefei, China
jpq@ustc.edu.cn

Abstract—Sentiment analysis of online reviews has been an important task in online shopping and electronic commerce to understand customers' opinions and behavior. Generally, customers can express their opinions by posting comments and uploading images, therefore online reviews can be regarded as the combination of a textual document and a few related images. Images may not contain obvious sentimental information, but the visual aspects of images can augment the sentiment information of textual comments. Thus, it is a better way to consider both textual comments and visual images in sentiment analysis of online reviews. Following this idea, in this paper, we propose a hierarchical attention network that combines visual aspect attention, sentence attention, and self-attention to provide effective sentiment analysis of online reviews. With this mechanism, we can model the interactions among words within one sentence as well as the interactions between texts and images. We conduct experiments on a dataset about online restaurant reviews from Yelp.com and compare our model with five existing models. The results suggest the superiority of our proposal.

Keywords-hierarchical attention network; online review; sentiment analysis

I. INTRODUCTION

The sentiment analysis of web data has been a widely-studied topic [1-5]. Sentiment analysis aims to detect the sentimental polarity, e.g., positive or negative, which is represented by the underlying data. Sentiment analysis is helpful for many web-based applications such as online reviews analysis, product recommendation, and personalized search. For example, in E-commerce platforms, vendors can know the public opinion toward a new product by analyzing the sentimental polarity of online reviews, which can be taken as the basis of market-promoting decisions.

Sentiment analysis used to be a research field in natural language processing, where the sentiment of a document is to be classified into different classes (e.g., positive, negative, and neutral), or a scaling factor (e.g., 1 to 5) is used to measure users' sentiment [2]. Researchers have proposed various ways to extract textual features, which are used as input for supervised learning. The recent works were mostly based on deep neural networks [3-5], which are proven to be highly effective for fitting nonlinear functions.

However, traditional sentiment analysis of online reviews mainly focused on the textual information and highly relied on natural language processing techniques. On the other hand,

Figure1. Example of the visual-textual connection in *Yelp* reviews.

many E-commerce or online shopping sites have allowed users to post images as part of reviews. Visual images can contain complementary information that is highly connected to the textual information in a review. Thus, in this paper, we aim to figure out the sentiment of online reviews that involve both textual comments and images. One challenge is that images may only contain some objects but no explicit sentiment words like "great" and "wonderful". For example, an image may only show some food on the table in a restaurant. Thus, using visual features directly for sentiment analysis is not effective for online reviews. We noted that in many cases, the attached images in a review usually reflect the main topic of the review. For example, when a person wrote a review about a restaurant, if he posted an image of his meal, it was likely that his review mainly expressed his opinions about food. On the other side, if the image was about the surrounding environment, then we could assign high weights to the restaurant environment related texts in the textual part, which was expected to achieve high accuracy of the sentiment analysis of online reviews.

Following the above idea, we propose to use visual images and the attention mechanism to enhance the performance of sentiment analysis on textual reviews. For image analysis, we propose to adopt visual aspect attention [6], which can help each sentence to find some "aspects" that appear in an image and align the semantic information in the text with visual information in the image. Figure 1 shows an example of *Yelp* restaurant review, two images on the right of the document describe drinks and tater tots separately, while two related sentences state that the drinks are not good and the tater tots are fine. The visual aspect attention can bridge the gap between the visual part in the images with the aspect words in the textual reviews.

On the other hand, there are some limitations for visual aspect attention. As Fig. 2 shows, the content of two images is curry, fried chicken, and the beef soup, which are referred as

Food: 1 star; dry chicken in both the red curry and the fried chicken. Curry, fried chicken and the beef soup was mediocre at best. The roti was the highlight of dinner.
Service: 5 stars; Amazing servers with impeccable service!

Figure 2. Example of deviation of images and texts.

"mediocre at best" in the first sentence, while the second sentence surrounded by a dashed green rectangle box indicated that the service was not good, but this was not directly reflected in the two images.

Briefly, this study aims to present a new method for the sentiment analysis of online reviews by combing visual aspect attention with sentence attention, yielding a hierarchical attention network model. The contributions of this study can be summarized as follows:

(1) We propose a hierarchical attention network for online-review-oriented sentiment analysis, which can both process images and texts in a review to generate effective sentiment polarity. Specially, we adopt self-attention as the base encoding layer to catch the interactions among words with long distances for information augment. We also use the visual aspect attention

(2) We present a dense layer to concatenate document representations generated by visual aspect attention and sentence attention to avoid the neglecting of some important sentences in sentiment analysis.

(3) We conduct experiments on a real dataset about online restaurant reviews from Yelp.com and compare our model with various baselines. The experimental results show that our method can improve the sentiment recognition accuracy of reviews, and can generalize to other scenarios where images reflex the main content of texts.

The remainder of the paper is structured as follows. Section II provides a brief literature review on recent research progress. Section III describes the proposed model. Section IV reports the experimental results, and finally, we conclude the entire paper in Section V.

II. RELATED WORK

Previous works on sentiment analysis have focused on text classification [1], where variants of general classifying techniques are applied and deep learning also brings advancement. In recent years, neural networks like recurrent neural networks have achieved success by incorporating attention into natural language process tasks, these models are usually hierarchical, e.g., word encoding and sentence encoding.

Dimension based sentiment analysis [2] means not to analyze the general sentiment of some document but to detect the sentiment of each dimension or aspect mentioned in the document. The state-of-the-art approach for aspect-level sentiment analysis is attention based deep learning systems. We focus on the sentiment of the whole document instead of producing prediction for relevant aspects in images.

Recent works show that sentiment analysis can use information from more than one modality, e.g., text, acoustic, image, which is referred to as multimodal sentiment analysis, while this paper tries to work out sentiment analysis of online reviews involving text and image. Katsurai et al. [3] proposed mapping textual, and sentiment views into the latent embedding space, then mining correlations among these features. The visual features can be learned from color histograms of images and this method achieved success on Flickr dataset and Instagram dataset. Zhang et al. [4] tried to solve sentiment analysis on microblogging by integrating text features and image features into multiple kernel learning. You et al. [5] proposed to extract visual features with CNN and extract textual features from an unsupervised language model by learning distributed representations for documents and paragraphs, then to fuse these two modalities.

Truong et al. [6] proposed to incorporate images as attention for review-based sentiment analysis. They adopted an architecture of word encoder and sentence encoder, and used visual aspect attention to decide the weight of each sentence. Karpathy et al. [7] proposed a combination of CNNs over image regions, bidirectional RNNs, and a structured objective to align language and visual data into a multimodal embedding. Peng et al. [14] proposed using a visual-textual bi-attention mechanism for visual-textual alignment, their model tries to learn multi-level visual-textual correlation for enhancing the matched pairs of different media types. Xu et al. [8] proposed using soft deterministic attention and hard stochastic attention for image captioning. Lu et al. [9] proposed a model for name tagging in multimodal social media based on visual attention that provides deeper visual understanding of the decisions of the model. Lu et al. [10] proposed a mechanism that jointly reasons visual attention and question attention for visual question answering.

Differing from existing studies, in this paper we propose a hierarchical attention network that combines visual aspect attention, sentence attention, and self-attention, which can model the inter-word correlations among texts as well as the interactions between texts and images.

III. ARCHITECTURE OF THE HIERARCHICAL ATTENTION NETWORK

Reviews are comprised of a collection of documents C. Each document is a sequence of L sentences, s_i, $i \in [1, L]$. Each sentence consists of K words $x_{i,k}$, $k \in [1, K]$. Each document has a set of N images $g_j \in \{g_1, g_2, \cdots, g_N\}$, the vector representation of each image is noted as e_j. The goal of our study is to train a classification function to predict sentiment labels for unseen documents.

Our model is a four-layered hierarchical architecture, as shown in Fig. 3. The bottom layer is the self-attention layer that tries to encode each word vector. The next layer is the word encoding layer with soft attention that encodes word vectors into sentence vectors. The third layer is the sentence encoding layer with visual aspect attention. The top layer is the classification layer for the sentiment label.

The main difference of our model from previous models is that we present a layered attention mechanism based on visual
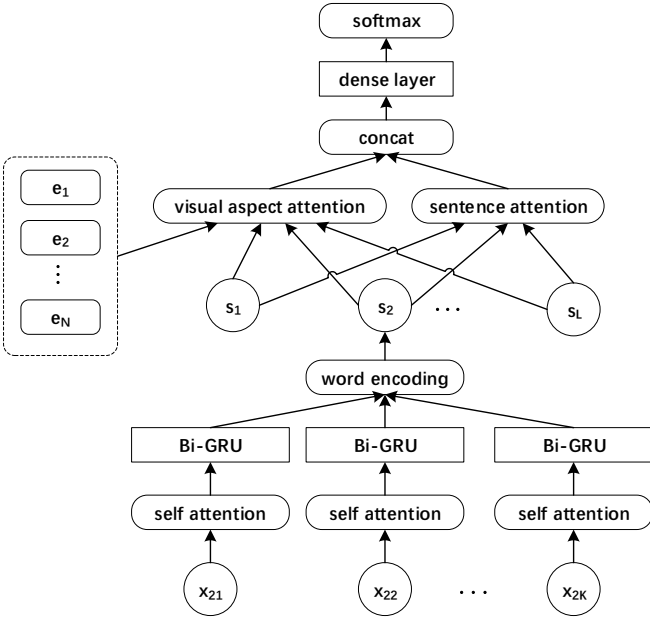
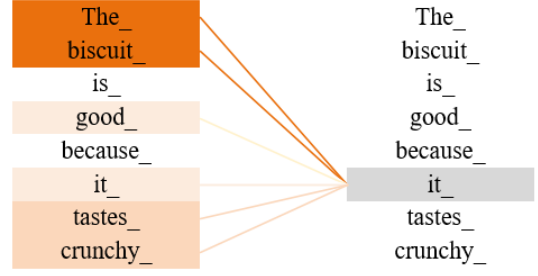Figure 3.   Architecture of the hierarchical attention model.

Figure 4.   An example of how self-attention works.

$$V = X_i W^V \qquad (4)$$

In the self-attention layer, the calculation process of the output Z includes dot product of *Q* and *K*, division for scaling, softmax for normalization and getting the weighted sum of *V*.

$$Z_i = softmax(\frac{QK^T}{\sqrt{d_k}})V \qquad (5)$$

The output vector of this layer, which will be used as the new word representation vector in the next layer, is the concatenation of original word vector $x_{i,k}$ and the vector $z_{i,k}$ generated by equation 5, as shown below.

$$y_{i,k} = [x_{i,k}, z_{i,k}]. \qquad (6)$$

B. Word Encoding

In this part of the architecture, we try to encode a sentence matrix of word embedding vectors into a sentence embedding vector. The input to this layer is the output of self-attention layer.

We choose bidirectional recurrent neural networks (RNN) [16] with GRU (Gated Recurrent Unit) cell to encode the word embedding sequence, whose output $h_{i,k}$ is the concatenation of $\vec{h}_{i,k}$ generated by the forward GRU and $\overleftarrow{h}_{i,k}$ generated by the backward GRU. This component is noted as Bi-GRU (Bidirectional GRU) in this paper. GRU is a variant of RNN that costs less computation price and solves the problem of long term dependency.

$$h_{i,k} = \text{Bi-GRU}(y_{i,k}) \qquad (7)$$

Words are not equally important, and attention helps to assign greater weight to more important words. We employ a soft attention mechanism [6] to distribute weights among words.

$$o_{i,k} = O^T \tanh (W_h h_{i,k} + b_h) \qquad (8)$$

$$\alpha_{i,k} = \frac{\exp (o_{i,k})}{\sum_k \exp (o_{i,k})} \qquad (9)$$

$$s_i = \sum_k \alpha_{i,k} h_{i,k} \qquad (10)$$

aspect attention, sentence attention, and self-attention, to integrate the texts and images to enhance the effectiveness of sentiment analysis of online reviews. With our design, both textual and imaginal information of online reviews can be reflected in the sentiment analysis process. By using the hierarchical attention network, especially the self-attention method, we can model the inter-word correlations among texts as well as the interactions between texts and images, resulting in the performance improvement of sentiment analysis of online reviews.

A. Self-Attention

First of all, words must be transferred into embedding vectors as input for the model. We use an embedding matrix W_e initialized from pre-trained word embedding models [17] to retrieve the embedding $x_{i,k}$ of each word $w_{i,k}$.

$$x_{i,k} = W_e w_{i,k}, \text{k} \in [1, \text{K}] \qquad (1)$$

Self-attention is an important concept brought in the model Transformer [15] that helped improve the performance of neural machine translation applications. As shown in Fig. 4, self-attention tries to encode representations of other relevant words into the current one being processed, while the relevance degree varies for different words. We use this method to put word interactions into word embedding vectors. For each word, we create a query vector *Q*, a key vector *K* and a value vector *V*. The input of this layer is a sentence matrix composed of word vectors. There are three parameter matrices W^Q, W^K, W^V which are initialized randomly and updated during the training process.

$$Q = X_i W^Q \qquad (2)$$

$$K = X_i W^K \qquad (3)$$

We use a tangent function to project $h_{i,k}$ into its representation in the attention space. O is a context vector randomly initialized and updated during training, it is used to multiply the projection for the relative importance $o_{i,k}$ of $h_{i,k}$. Then we use softmax as normalization to obtain attention weight $\alpha_{i,k}$ of $h_{i,k}$. The embedding of sentence s_i is represented as the weighted summation of all its word representations $h_{i,k}$ by attention weights $\alpha_{i,k}$.

C. Visual Aspect Attention and Sentence Attention

This layer transfers the output of the word encoding layer into document-level representations using visual aspect attention and sentence attention, assigning greater weight to more salient sentences. We still use Bi-GRU taking sentence s_i as input to generate forward hidden states vector $\vec{h}_i$ and backward hidden states vector $\overleftarrow{h}_i$ and concatenate them as the output vector h_i.

$$h_i = \text{Bi-GRU}(s_i) \qquad (11)$$

We take semantic connections between images and text as attention for sentences. A document is usually attached with several pictures, which are associated with sentences with varying degrees.

First, we need to encode visual images and VGG convolutional neural networks [18] have proven effective for learning image presentations in many similar situations. We employ VGG-16 to process image g_j and take the output of the last fully-connected layer (FC7) as the image representation e_j.

$$e_j = VGG(g_j) \qquad (12)$$

Visual Aspect Attention. We learn the attention weights $\gamma_{j,i}$'s for sentence representations h_i's with respect to each image representation e_j.

$$p_j = \tanh(W_p e_j + b_p) \qquad (13)$$

$$q_i = \tanh(W_q h_i + b_q) \qquad (14)$$

$$m_{j,i} = M^T(p_j \odot q_i + q_i) \qquad (15)$$

$$\gamma_{j,i} = \frac{\exp(m_{j,i})}{\sum_i \exp(m_{j,i})} \qquad (16)$$

We project image representation e_j and sentence representation h_i into an attention space followed by a non-linear activation function to obtain output p_j and q_i. Then we try to find interactions between p_j and q_i by element-wise multiplication and summation. The learned vector M is a global attention vector similar to O in word encoder. Then we use softmax to normalize each attention value $m_{j,i}$ in M as $\gamma_{j,i}$.

With the visual-aspect attention weight $\gamma_{j,i}$, we aggregate sentence representations into document representation d_j as follows.

$$d_j = \sum_i \gamma_{j,i} h_i \qquad (17)$$

Each document has a set of image-specific document representation d_j, $j \in [1, N]$. Attached images are not equally informative, thus we try to learn the importance weight τ_j of each document representation d_j. The visual attention-based document representation d' is the aggregation of image-specific document representation d_j.

$$a_j = A^T \tanh(W_a d_j + b_a) \qquad (18)$$

$$\tau_j = \frac{\exp(a_j)}{\sum_j \exp(a_j)} \qquad (19)$$

$$d' = \sum_j \tau_j d_j \qquad (20)$$

Sentence Attention. We use sentence attention to generate a context vector U and reward sentences that are clues to classify a document correctly.

$$u_i = U^T tanh(W_u h_i + b_u) \qquad (21)$$

$$\pi_i = \frac{exp(u_i)}{\sum_i exp(u_i)} \qquad (22)$$

$$d'' = \sum_i \pi_i h_i \qquad (23)$$

The concatenation of d' and d'' is fed into a dense layer, the output of which is the final document representation d.

$$d = \text{Dense}([d', d'']) \qquad (24)$$

D. Sentiment Classification

The top layer treats the document representation d with a softmax based sentiment classifier, generating the probabilities distribution μ of sentiment classes.

$$\mu = \text{softmax}(W_\mu d + b_\mu) \qquad (25)$$

The loss of this model is the cross-entropy error of sentiment classification:

$$\text{loss} = -\sum_d log\mu_{d,l}, \qquad (26)$$

where l is the ground-truth label of review d.

IV. PERFORMANCE EVALUATION

A. Settings

Dataset. We use a dataset of restaurant reviews on Yelp.com [6], covering five US cities including Boston (BO), Los Angeles (LA), Chicago (CH), New York (NY), and San Francisco (SF). The dataset contains more than 44 thousand reviews and 244 thousand images with each review having at least 3 images. We split 80% of the dataset for training, 5% for validation, and 15% for tests. There are five classes of sentiment labels in this dataset, ranging from very negative to very positive.

Training. In the training process, we use NLTK [13] for sentence and word tokenization. In addition, we use the pre-trained Glove word embedding with dimensionality D = 200. The GRU cells are 50-dimensional in the encoding process, thus

TABLE I.   ACCURACY COMPARISON OF SENTIMENT ANALYSIS (%)

Models	BO	CH	LA	NY	SF	Avg.	Improvement (compared with TFN-aVGG)
TFN-aVGG	46.35	43.69	43.91	43.79	42.81	43.89	-
TFN-mVGG	48.25	47.08	46.70	46.71	47.54	46.87	6.8%
HAN-aVGG	55.18	54.88	53.11	52.96	51.98	53.16	21.1%
HAN-mVGG	56.77	57.02	55.06	54.66	53.69	55.01	25.3%
VistaNet	63.81	65.74	62.01	61.08	60.14	61.88	41.0%
Our model	**66.67**	**68.31**	**60.83**	**61.10**	**61.05**	**63.59**	**44.9%**

the output of bidirectional cells is 100 dimensional. The model is implemented with Python 3.7 and TensorFlow 1.14. We select the Adam optimizer for gradient-based optimization and set the batch size to 32. The model is trained for 20 epochs and the result of the epoch with the least training loss is outputted as the final result.

B. Baselines

We compare our model with several baselines that use both textual and visual features, including TFN-aVGG, TFN-mVGG, HAN-aVGG, HAN-mVGG, and VistaNet. We focus on the accuracy of each model when evaluating the sentiment polarity of the online restaurant reviews.

(1) *HAN-aVGG* and *HAN-mVGG* [11]. HAN-aVGG and HAN-mVGG are composites of HAN-ATT for text and VGG for images. HAN-ATT uses a hierarchical architecture of word encoder and sentence encoder. HAN-aVGG and HAN-mVGG correspond to using average pooling and max pooling for image feature vectors respectively, which will be concatenated with textual feature vectors as the input vectors.

(2) *TFN-aVGG* and *TFN-mVGG* [12]. TFN-aVGG and TFN-mVGG are composites of Tensor Fusion Network. Textual features from HAN-ATT and visual features from VGG are combined using Tensor Fusion Layer and fed through Sentiment Inference Subnetwork for the final sentiment label. We use average pooling for TFN-aVGG and max pooling for TFN-mVGG.

(3) *VistaNet* [6]. VistaNet is a hierarchical architecture that adopts a soft-attention-based word encoding layer and a visual aspect attention based sentence encoding layer.

C. Results

Table I lists the comparative accuracy of our method with other baseline methods. The five columns, namely BO, CH, LA, NY, and SF represent the five cities, and the avg. column is the average accuracy of all the five cities.

As shown in Table I, our model outperforms these multimodal baselines in all five cities and average results. This result demonstrates that combing visual attention and sentence attention could effectively draw attention to more salient sentences of a review document. The second-best model VistaNet is ahead of other baselines, which proves that visual attention has significant effects in this experiment. Our model has a 1.71% accuracy improvement over VistaNet, showing that the self-attention layer is useful for encoding word vectors and a tradeoff between visual aspect attention and sentence attention could end up with better results.

We can also notice that TFN-aVGG and TFN-mVGG perform badly in this experiment even though TFN can provide rich interactions between textual features and visual features. In this experiment of online reviews, images seldom carry enough sentimental information, e.g., images of food cannot tell whether the customer likes the food or not. This is the reason why our model using visual features as attention for sentences can outperform models that use visual features as additional sentimental information.

D. Ablation Analysis

We conduct an ablation analysis to specifically analyze the contributions of each component of our architecture. We start from the most basic architecture and incrementally add a component until reaching the full architecture.

TABLE II.   ABLATION ANALYSIS

Components					CITY (%)					
Bi-GRU	Hierarchical Structure	Visual Aspect Attention	Self-Attention	Sentence Attention	BO	CH	LA	NY	SF	Avg.
√	x	x	x	x	57.70	60.01	56.74	56.59	55.84	56.83
√	√	x	x	x	60.39	64.39	59.08	59.58	59.18	59.54
√	√	√	x	x	63.81	65.74	62.01	61.08	60.14	61.88
√	√	√	√	x	63.17	62.77	62.12	61.40	62.63	62.42
√	√	√	√	√	66.67	68.31	60.83	61.10	61.05	63.59

We first carry out experiments with the base model Bi-GRU using only text. Then, we implement a hierarchical structure with a word encoding layer and max-pooling sentence representations, the accuracy is 59.54%. By applying the visual aspect attention upon sentence-level representations, this structure has achieved an average accuracy of 61.88%. When a self-attention encoding layer is added to the hierarchical structure, the average accuracy has increased to 62.42%. Finally, the sentence attention is combined with the visual attention through a dense layer. We can see that the model has been improved to an average accuracy of 63.59%.

All the results in Table II show that our model outperforms other models in the average accuracy. We can also see that every component contributes to our model.

V. Conclusions and Future Work

In this paper, we propose a novel model for text-and-image-based bi-modal sentiment analysis. Our model utilizes visual aspect attention, sentence attention, and self-attention to form a hierarchical attention network, which has been experimentally demonstrated that it works well for the sentiment analysis of online reviews. In particular, we use self-attention as the base encoding layer and combine visual aspect attention with sentence attention to present a better attention mechanism. Compared with existing studies, the four-layered hierarchical attention model can encode the interactions among words within a sentence as well as the interactions between texts and images. It adopts a hierarchical attention mechanism by aggregating word representations into sentence representations, aggregating sentence representations into document representations, and finally generating the sentiment label. Our model also employs images as alignment to select important sentences within a document and employs a soft attention mechanism for sentences that may have few interactions with images. We conduct experiments on a real dataset about online restaurant reviews in five US cities. The results show that our model outperforms the other five baselines, indicating the effectiveness of our proposal.

Our future work will concentrate on building a more elastic attention mechanism, e.g., assigning higher weights to most influential words in a document and introducing most recent models of natural language processing for a better understanding of document content. We will also consider to apply the hierarchical attention model to sentiment analysis of multimodal social media such as microblogs [19, 20].

References

[1] Pang, B. and Lee, L., Opinion mining and sentiment analysis. *Foundations and Trends® in Information Retrieval*, 2008, 2(1–2), pp.1-135.

[2] Zheng, L., Jin, P., Zhao, J., Yue, L. Multi-dimensional sentiment analysis for large-scale E-commerce reviews. In *Proceedings of the 25th International Conference on Database and Expert Systems Applications (DEXA)*, 2014, 449-463

[3] Katsurai, M. and Satoh, S.I., March. Image sentiment analysis using latent correlations among visual, textual, and sentiment views. In *Proceedings of the 2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2016, 2837-2841.

[4] Zhang, Y., Shang, L. and Jia, X., Sentiment analysis on microblogging by integrating text and image features. In *Proceedings of the Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD)*, 2015, 52-63.

[5] You, Q., Luo, J., Jin, H. and Yang, J., Joint visual-textual sentiment analysis with deep neural networks. In *Proceedings of the 23rd ACM International Conference on Multimedia (MM)*, 2015, 1071-1074.

[6] Truong, Q.T. and Lauw, H.W., VistaNet: Visual Aspect Attention Network for Multimodal Sentiment Analysis. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, 2019, 305-312.

[7] Karpathy, A. and Fei-Fei, L., Deep visual-semantic alignments for generating image descriptions. *In Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, 3128-3137.

[8] Xu, K., Ba, J., Kiros, R., Cho, K., Courville, A., Salakhudinov, R., Zemel, R. and Bengio, Y., Show, attend and tell: Neural image caption generation with visual attention. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2015, 2048-2057.

[9] Lu, D., Neves, L., Carvalho, V., Zhang, N. and Ji, H., Visual attention model for name tagging in multimodal social media. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (ACL)*, 2018, 1990-1999.

[10] Lu, J., Yang, J., Batra, D. and Parikh, D., Hierarchical question-image co-attention for visual question answering. In *Advances in Neural Information Processing Systems (NIPS)*, 2016, 289-297.

[11] Yang, Z., Yang, D., Dyer, C., He, X., Smola, A. and Hovy, E., Hierarchical attention networks for document classification. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 2016, 1480-1489.

[12] Zadeh, A., Chen, M., Poria, S., Cambria, E. and Morency, L.P., Tensor Fusion Network for Multimodal Sentiment Analysis. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2017, 1103-1114.

[13] Loper, E. and Bird, S., NLTK: The Natural Language Toolkit. In *Proceedings of the ACL-02 Workshop on Effective Tools and Methodologies for Teaching Natural Language Processing and Computational Linguistics*, 2002, 63-70.

[14] Peng, Y., Qi, J. and Zhuo, Y., MAVA: Multi-level Adaptive Visual-textual Alignment by Cross-media Bi-attention Mechanism. *IEEE Transactions on Image Processing*, 2020, 29: 2728-2741.

[15] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, Ł. and Polosukhin, I., Attention is all you need. *In Advances in Neural Information Processing Systems (NIPS)*, 2017, 5998-6008.

[16] Cho, K., van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H. and Bengio, Y., Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2014, 1724-1734.

[17] Pennington, J., Socher, R. and Manning, C.D., Glove: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2014, 1532-1543.

[18] Simonyan, K. and Zisserman, A., Very deep convolutional networks for large-scale image recognition. *arXiv preprint,* arXiv: 1409.1556, 2014

[19] Jin, P., Mu, L., Zheng, L., Zhao, J., Yue, L. News feature extraction for events on social network platforms. In *Proceedings of the 26th International World Wide Web Conference (WWW)*, 2017, 69-78

[20] Mu, L., Jin, P., Zheng, L., Chen, E., Yue, L., Lifecycle-based event detection from microblogs. In *Proceedings of the 27th International World Wide Web Conference (WWW),* 2018, 283-290

Cross-project Reopened Pull Request Prediction in GitHub

Abdillah Mohamed[†‡], Li Zhang[†], Jing Jiang[†*]

[†]State Key Laboratory of Software Development Environment, Beihang University, Beijing, China
[‡]University Institute of Technology, University of Comoros, Comoros
Email:{abdillah,lily,jiangjing}@buaa.edu.cn

Abstract—In GitHub, pull requests may get reopened again for further modification and code review. Prediction of within-project reopened pull requests works well if there is enough amount of training data to build the training model. However, for new projects that have a limited amount of pull requests, using training data from other projects can help to predict the reopened pull requests. Therefore, it is important to study cross-project reopened pull request prediction and help integrators in new projects.

In this paper, we propose a cross-project approach that consists of building a decision tree training model based on an external project as a source project to predict the reopened pull requests in another project. We evaluate the effectiveness of cross-project prediction on 7 open source projects containing 100,622 pull requests. Experiment results show that the cross-project prediction achieves accuracy from 78.76% to 96.52%, and F1-measure from 53.34% to 90.58% across 7 projects. We examine the feature importance using the decision tree predictor and find that the number of commits is the most important feature in the majority of projects.

Keywords—Reopened pull request prediction, Cross project, GitHub.

I. INTRODUCTION

GitHub is popular among a large number of software developers around the word [1].

To identify whether or not a pull request will be reopened, we proposed in our prior work a within-project predictor that consists of splitting the entire dataset of a project into a training set and a testing set to predict whether or not a closed pull request would be reopened [2]. Prediction of within-project reopened pull requests works well if there is enough amount of training data to build the training model.

However, for new projects that have a limited amount of pull requests, using training data from other projects can help to predict the reopened pull requests. It is important to study cross-project reopened pull request prediction, and help integrators in new projects. Several researchers studied the cross-project defect prediction [3]–[5]. To the best of our knowledge, the cross-project reopened pull request prediction has not been explored yet.

In this paper, we proposes a cross-project approach that consists of building a decision tree training model based on an external project as source project to predict the reopened pull requests in another project. This approach first extracts code features of modified changes, review features during

evaluation, and developer feature of contributors from a source project. Then it uses decision tree classifier to make prediction for pull requests in a target project.

In order to explore the performances of this approach, we collect datasets of 7 open-source projects and 100,622 pull requests. Results show that the cross-project reopened pull request prediction achieves accuracy of 78.76%, 95.11%, 94.12%, 89.95%, 93.06%, 96.52%, 94.87%, and F1-measure of 53.34%, 86.52%, 83.72%, 73.54%, 81.54%, 90.58%, 85.72% for the target projects *bootstrap, cocos2d-x, symfony, homebrew-cask, zendframework, rails*, and *angular.js* respectively. We explore feature importance, and find that in the majority of projects, number of commits is the most important in the prediction of reopened pull requests.

The main contributions of this paper are as follow:
- We build a cross-project approach based on a source project to predict the reopened pull requests in a target project. Results show that cross-project approach performs well in predicting reopened pull requests.
- We find that the number of commits is the most important feature in the cross-project reopened pull request prediction in most of the projects.

The remainder of the work is structured as follows. Section II presents the data collection. In Section III, we present the approach of the cross-project reopened pull requests. Section IV presents the experimental settings. Section V presents the experimental results of our approach. In section VI, we present threats to validity. Section VII presents the related work. Finally, section VIII presents summarise our findings.

II. DATA COLLECTION

We use the same dataset as our previous work [2]. We choose 7 popular projects such as rails, cocos2d-x, symfony, homebrew-cask, zendframework, angular.js, and bootstrap with more than 5,000 stars, because they receive many pull requests and provide datasets for our research.

Table I shows the basic statistics of 7 projects. The table represents the percentage of reopened pull requests. In the fifth column, the value before the slash is the number of reopened pull requests, and the value after the slash is its percentage. Reopened pull requests exist in all projects.

III. APPROACH

In this section, we describe the cross-project reopened pull request prediction.

Project owner	Repository	Language	#Pull requests	#Reopened pull requests	#Stars
rails	rails	Ruby	19,190	467/2.43%	36,253
cocos2d	cocos2d-x	C++	14,134	113/0.80%	10,514
symfony	symfony	PHP	14,569	220/1.37%	14,800
caskroom	homebrew-cask	Ruby	31,980	331/1.04%	11,229
zendframework	zendframework	PHP	5,631	213/3.78%	5,522
angular	angular.js	JavaScript	7,504	223/2.97%	56,359
twbs	bootstrap	JavaScript	7,614	136/1.79%	112,425

Fig. 1.   Overall framework of the cross-project predictor

A. Model-building phase

As shown in Figure 1, our framework takes as input instances (pull requests) from source project (step 1) with a known class (i.e., reopened or non-reopened). We collect code features, review features and developer feature. Next, it extracts various metrics from the source project to build the cross-project model (step 2). Then we use a weighted vector to represent each pull request, and each element in this vector We describe details of features as follow:

Code feature. We use code features in cross-project reopened pull requests prediction at the first close. We take in count four features to measure modified codes, including *number of commits*, *number of changed files*, *number of added lines* and *number of deleted lines* in a pull request.

Review feature. We consider review features, including *number of comments*, *evaluation time* and *closed status*. *Evaluation time* is the time difference between the pull request's submission and first close. *Closed status* assess whether a pull request is accepted or rejected at its first close.

Developer feature. We apply developer feature which quantifies the reputation of contributors who submit pull requests. For each pull request, we compute the number of accepted and rejected pull requests submitted by the same contributor before its creation time. Briefly, the reputation is the proportion of previous pull requests which are submitted by the contributor

and get accepted.

B. Prediction phase

In the prediction phase, the same cross-project prediction model built in step 2 is applied to predict whether a closed pull request would be reopened in the target project. For a pull request in a target project, we first extract code features, review features and developer feature as those extracted the model-building phase (step 3). We then input the values of these metrics into the cross-project model (Step 4). It outputs the pull request prediction result about whether it will be either reopened or non-reopened (Step 5).

IV. EXPERIMENTAL SETTINGS

The main goal of this work is twofold. (i) We build trained model based one source project to train a model and use it to predict the reopening of a pull request of another project. (ii) We study feature importance in predicting reopened pull requests.

A. Evaluation process and metrics

In evaluation, we use accuracy, precision, recall and f1-measure. The accuracy measures the number of correctly classified reopened pull requests (both non-reopened and re-opened) over the total number of pull requests. Precision is the ratio of correctly predicted reopened pull requests over all the pull requests predicted as reopened. Recall is the ratio of correctly predicted reopened pull requests over all actually reopened pull requests. F1-measure is the weighted harmonic mean of precision and recall.

B. Research Questions

We are interested to answer following research questions:

RQ1: How does the cross-project prediction perform?

Motivation. In this research question, we aim at building a cross-project predictor based on one project as a source project to predict the pull request reopening in a data of another project.

Approach. To solve this research question, we aim at building decision tree training models based on one projects as a source project by crossing the seven projects between them. For each of the 6 source projects used separately to predict the reopened pull requests in one and only target project, we select the results of the source project that achieves high f1-measure.

RQ2: Which features are important in cross-project reopened pull request prediction?

Motivation. Different features may have various weights in cross-project reopened pull request prediction. We wonder which features are more important than other.

Approach. In order to answer this question, we use decision tree classifier to compute feature importance in the prediction of reopened pull requests. Feature importance is calculated as the decrease in node impurity weighted by the probability of reaching that node. The node probability can be calculated by the number of reopened pull request that reach the node, divided by the total number of pull requests. The higher the value is, and the more important the feature is.

V. Experimental results

In this section, we study the results of our study aiming at answering above research questions.

A. RQ1: Performance of cross-project prediction

In order to answer RQ1, we study results based on different combination of source projects and target project. We first analyze the project *rails* as an example. Table II shows results when the project *rails* is the target project. In each row, we predict reopened pull requests in the project *rails* as target projects by crossing the projects *symfony, cocos2d-x, angular.js, zendframework, homebrew-cask* and *bootstrap* respectively as source projects. The best results are in bold. Results show that the combination *cocos2d-x =>rails* achieves the best performance by achieving an accuracy of 96.52% and f1-measure of 90.58%.

TABLE II
PREDICTING THE REOPENED PULL REQUEST BASED ON THE PROJECT
RAILS AS THE TARGET PROJECT

Source =>Target projects	Accuracy	Precision	Recall	F1-measure
symfony =>rails	96.47%	**98.07%**	83.92%	90.45%
cocos2d-x =>rails	**96.52%**	96.60%	85.20%	**90.58%**
angular.js =>rails	96.02%	95.29%	85.00%	89.85%
zendframework =>rails	96.42%	96.61%	84.75%	90.29%
homebrew-cask =>rails	92.24%	78.51%	83.92%	81.13%
bootstrap =>rails	94.83%	82.77%	**92.97%**	87.57%

Table III shows the performances of the cross-projects reopened pull requests prediction across 7 projects. The projects on top of the table are used as a target for single source cross-projects, while the projects on the left side of the table are used as source projects. We use the source project to train the decision tree model, and the target project is used as a class project to predict the reopened pull requests. Results in green color represent the highest performance predictions of the cross-project prediction of each target across 6 target projects. Results show that when predicting reopened pull requests in the target project *angular.js*, the source project *symfony* is more suitable comparing to the other source projects. In the same way, we compared the performances of the other source projects, and find the source project which achieves the highest F1-measure in predicting reopened pull requests for a specific target project.

The Table IV presents the combinations of the cross-project that carry out the best results across 42 combinations from the Table III. Each target project is used separately with each of the six remaining projects as source projects to predict the reopened pull requests and select the combination that achieves the best results. In the same way, we processed to select the best combination of crossed projects (sources and targets) that has good performances. Thus, we notice that the single source cross-project reopened pull requests prediction achieves good performances in most of the projects.

TABLE IV
PERFORMANCES OF CROSS-PROJECT REOPENED PULL REQUESTS
PREDICTOR

Source=>Target projects	Accuracy	Precision	Recall	F1-measure
homebrew-cask =>bootstrap	78.76%	48.12%	59.83%	53.34%
zendframework =>cocos2d-x	95.11%	97.36%	77.86%	86.52%
zendframework =>symfony	94.12%	93.72%	75.64%	83.72%
cocos2d-x =>homebrew-cask	89.95%	78.51%	69.16%	73.54%
rails =>zendframework	93.06%	90.18%	74.41%	81.54%
cocos2d-x =>rails	96.52%	96.60%	85.20%	90.58%
symfony =>angular.js	94.87%	97.91%	76.24%	85.72%

> **RQ1:** Across the 7 projects, the single source cross-project reopened pull requests prediction achieves good performances in most of the projects.

B. RQ2: Important features for predicting reopened pull requests.

Decision tree classifier also computes the importance of each feature in the prediction of reopened pull requests, and we plot the results in the Table V. Feature importance may be different in various projects. In the majority of projects, the number of commits is the most important in the prediction of reopened pull requests. Some pull requests have many commits, and they may be difficult for integrators to make a complete evaluation. Therefore, pull requests with many commits are likely to be reopened, and the number of commits is the most important feature.

> **RQ2:** In the majority of projects, the number of commits is the most important in the cross-project reopened pull request prediction.

VI. Threats to validity

In this section, we introduce threats to the validity of our study.

Threats to external validity relate to the generalization of our research. Firstly, our experimental results are limited to 7 projects in GitHub. In the future, we plan to use more projects to better generalize the results of our method. Secondly, we analyze open-source software projects in GitHub. In the future, we plan to study other platforms and compare their results with our findings in GitHub.

Threats to construct validity refer to the degree to which the construct being studied is affected by experiment settings. We use accuracy, precision, recall, and F1-measure. As a results, there is little threat to construct validity.

TABLE III
F1-MEASURE COMPARISON BETWEEN THE CROSS-PROJECTS REOPENED PULL REQUESTS PREDICTION

Source/Target	rails	angular.js	cocos2d-x	Symfony	homebrew-cask	zendframework	bootstrap
rails	/	83.61%	86.22%	82.55%	61.82%	81.54%	24.81%
angular.js	89.85%	/	84.06%	80.65%	59.58%	77.73%	24.25%
cocos2d-x	90.58%	84.26%	/	67.61%	73.54%	80.36%	35.74%
symfony	90.45%	85.72%	84.18%	/	61.79%	79.57%	20.59%
homebrew-cask	81.13%	81.62%	83.75%	66.15%	/	79.40%	53.34%
zendframework	90.29%	84.87%	86.52%	83.72%	67.34%	/	33.68%
bootstrap	87.57%	76.33%	84.68%	69.84%	73.24%	74.43%	/

TABLE V
FEATURE IMPORTANCE FOR CROSS-PROJECT REOPENED PULL REQUESTS PREDICTION

Features	homebrew-cask =>bootstrap	zendframework =>cocos2d-x	zendframework =>symfony	cocos2d-x =>homebrew-cask	rails =>zend-framework	cocos2d-x =>rails	symfony =>Angular.js	Average
Number of commits	0.327	0.275	0.275	0.611	0.476	0.611	0.463	0.434
Number of changed file	0.038	0.411	0.411	0.040	0.361	0.040	0.274	0.225
Number of added lines	0.128	0.000	0.000	0.000	0.045	0.000	0.000	0.025
Number of deleted lines	0.003	0.000	0.000	0.000	0.000	0.000	0.000	0.000
Number of comments	0.019	0.033	0.034	0.002	0.017	0.002	0.015	0.017
Evaluation time	0.079	0.169	0.169	0.083	0.041	0.084	0.116	0.106
Closed status	0.322	0.038	0.038	0.234	0.040	0.234	0.025	0.133
Reputation	0.084	0.074	0.073	0.029	0.021	0.029	0.107	0.060

VII. RELATED WORKS

In this section, we mainly discuss related works, including reopened pull requests and cross-project prediction.

A. Reopened pull requests

In GitHub, there are several works which are focusing on pull requests evaluation and prediction [2], [6]. We conducted a case study to understand reopened pull requests [6]. Previous work [2] designed a within-project reopened pull request prediction, while this paper explores the cross-project reopened pull request prediction.

B. Cross-project prediction

The cross-project prediction has been the main area of researches in different aspects by reusing training data from other projects to make a prediction in a new project. Several authors discussed the cross-project defect prediction [3]–[5]. Rahman et al. [3] compared the cross-project defect prediction with the prediction within a project, and they found that cross-project prediction performance was no worse than within-project performance and considerably better than random prediction.

Unlike the above researches, we address a different problem, namely cross-project reopened pull request prediction.

VIII. CONCLUSION

Cross-project reopened pull requests are important for the projects that do not have enough historical data to build prediction models. In this paper, we propose a cross-project approach for predicting reopened pull requests in GitHub. This study brings new insight into the performances of the cross-project using a decision tree classifier. Based on 100,622 pull requests from 7 open-source projects, experimental results show that the cross-project reopened pull request prediction achieves an f1-measure of 53.34%, 86.52%, 83.72%, 73.54%, 81.54%, 90.58%, and 85.72% for the target projects *bootstrap, cocos2d-x, symfony, homebrew-cask, zendframework, rails*, and *angular.js* respectively. We use decision tree to compute feature importance, and find that number of commits is the most important feature in the majority of projects.

ACKNOWLEDGMENT

REFERENCES

[1] A Lima, L Rossi, and M Musolesi. Coding together at scale: Github asa collaborative social network. In *Proceedings of 8th AAAI International Conference on Weblogs and Social Media*, 2014.

[2] Abdillah Mohamed, Li Zhang, Jing Jiang, and Ahmed Ktob. Predicting which pull requests will get reopened in github. In *2018 25th Asia-Pacific Software Engineering Conference (APSEC)*, pages 375–385. IEEE, 2018.

[3] Foyzur Rahman, Daryl Posnett, and Premkumar Devanbu. Recalling the" imprecision" of cross-project defect prediction. In *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering*, pages 1–11, 2012.

[4] Feng Zhang, Quan Zheng, Ying Zou, and Ahmed E Hassan. Cross-project defect prediction using a connectivity-based unsupervised classifier. In *2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE)*, pages 309–320. IEEE, 2016.

[5] Xin Xia, David Lo, Sinno Jialin Pan, Nachiappan Nagappan, and Xinyu Wang. Hydra: Massively compositional model for cross-project defect prediction. *IEEE Transactions on software Engineering*, 42(10):977–998, 2016.

[6] Jing Jiang, Abdillah Mohamed, and Li Zhang. What are the characteristics of reopened pull requests? a case study on open source projects in github. *IEEE Access*, 7:102751–102761, 2019.

Restaurant Failure Prediction Based on Multi-View Online Data

Xiaoxiao Sun, Ping Liang, DongjinYu*

School of Computer Science and Technology
Hangzhou Dianzi University
Hangzhou, China
sunxiaoxiao@hdu.edu.cn, liangpingprivate@gmail.com, yudj@hdu.edu.cn

Abstract—**Predicting future development trends of restaurants (especially failure judgement) helps entrepreneurs to identify potential downward trends in their business and supports potential investors' investment decisions. Review apps, such as Yelp, are generating massive restaurant-related online data every day, which provides a solid data source for the prediction through big data technology rather than applying commercial data with limited access and poor time efficiency. In this paper, we propose a novel multi-view restaurant failure prediction model named Semantic Business Cluster Effect Model (SBCM) based on online review data. Specifically, our model consists of three views: semantic view (we capture semantic features of reviews via a neural network and different reviews are assigned with different importance according to their reviewers' habits), business attribute view (we select the most influential business attributes from datasets), and business cluster effect view (we identify business clusters based on density and differentiate restaurants into different clusters). All attributes are then input into a LightGBM model to conduct the prediction. Experiments on public Yelp datasets of Toronto and Las Vegas from 2016 to 2017 demonstrate that SBCM averagely outperforms SVM and XGBoost by 14% and 3% respectively in terms of AUC. Furthermore, we find that credit card support, lunch support and noise level are the three most significant business attributes that influence the restaurant popularity online.**

Keywords: restaurant failure prediction; big data analysis; semantics extraction; cluster effect; LightGBM; Yelp

I. INTRODUCTION

Business failure prediction is a scientific field with long history, whose accurate results help entrepreneurs to identify potential downward trends in their business performance and give them timely warning to change business strategies in advance. Meanwhile, according to National Restaurant Association (NRA) [1], restaurant industry sales are projected to total $863 billion in 2019 and equal 4 percent of the U.S. gross domestic product. Meanwhile, restaurant workforce is about 10% of the overall U.S. workforce. Restaurants have played an essential role in the economy of a thriving society. Therefore, restaurant failure prediction is worthy of deep studying.

With the development of mobile Internet technology, restaurants are changing their traditional business patterns and starting to pay more attention to online advertisement. Apps such as Yelp provide platforms for restaurants to advertise their foods online and for customers to share their dining experiences. These reviews provide important references accordingly for other customers to select restaurants. Studies show that online data (i.e. reviews and check-ins) are related to restaurant performance and using them to predict restaurant failure is feasible [2,3,4,5]. However, few researchers have made in-depth studies on the relationship between the abundant semantics of reviews (The taste, environment, service, price, etc.) and business performance. Even some did, they ignored the fact that different customers have different preferences on giving ratings and reviews. For example, some customers like giving high ratings to almost all restaurants and some customers prefer to give low scores. In addition, some people tend to use personalized words to express their point of view (e.g., "good" is used to express satisfaction by some strict customers and express borderline by some lenient customers). Yelp also defines various attributes, such as credit card, Wi-Fi, parking etc., which can be used to predict the future performance of restaurants [3]. Moreover, the success and failure of restaurants are usually affected by their surrounding business districts, which is not considered in most studies.

To address the above problems, in this paper, we propose a novel prediction model named Semantic Business Cluster Effect Model (SBCM) based on the review semantics, business attributes and cluster effect to predict business failure of restaurants. In SBCM, we first design a neural network to capture semantic features from the newest and most popular reviews of each restaurant. Secondly, we design a review importance weight metric to match reviews with reviewing habits of different customers. Thirdly, we identify the importance of different attributes provided by Yelp and select the most important ones as the input of the prediction. Fourthly, we identify business clusters by Density-Based Spatial Clustering of Applications with Noise (DBSCAN) [6] and differentiate restaurants into different clusters. Finally, semantic features, business features and cluster effect features are integrated and input into LightGBM[7] to obtain the final prediction value. Experiments on public Yelp datasets demonstrate that SBCM averagely outperforms SVM and XGBoost by 14% and 3% respectively in terms of AUC.

The rest of the paper is structured as follows. After discussing related work in Section 2, we introduce some basic definitions and raise our problem in Section 3. Section 4 presents

our prediction model in detail. Experiments and visualization results are given in Section 5. Finally, Section 6 concludes the paper and introduces the future work.

II. RELATED WORKS

Since the late 1960s, business failure prediction has been widely investigated through statistical techniques and discriminant analysis [8]. Logit analysis [9], generalized extreme value [10], machine learning techniques [11], neural networks [12] etc., have also been applied in the prediction in recent years. The common ground of these business failure prediction methods is that they mostly rely on commercial data, such as stock prices, working capital and debt, which are more suitable for large enterprises rather than medium-scale businesses, especially not applicable for restaurants that we focus on in this paper. In addition, commercial data are usually statistical-based, which lack timeliness and sustainability.

In recent years, online review apps become popular with the proposal of concepts such as big data and smart city, which provide consistent and substantial data of small and medium-scale businesses, giving researchers a new idea to predict restaurant failure. For example, Zhang et al. found that the rating stars, sentiments, and photos of reviews are closely associated with restaurant survival [2]. Snow et al. [3] studied the influence of different business attributes and reviews on restaurant failure. Wang et al. [5] incorporated check-in data captured from location-based services to predict restaurant failure and obtained better results than using business characteristic variables only. The aforementioned works, however, neglect the fact that business performance of a restaurant is not only affected by its own factors, but also affected by its surrounding neighbors. Hu et al. [13] proved that applying the neighbors' business performance to predict the rating of a business is feasible, and they observed positive correlations between business individual's ratings and its neighbors' ratings.

In conclusion, restaurant failure is a complex problem affected by various factors. But until now, these factors have not been adequately considered simultaneously in previous researches. This paper is dedicated to proposing a prediction model that synthesize all influence factors, including review semantic features incorporated with the corresponding review habit of customers, influential business attributes and business cluster effect.

III. THE PREMIMINARY

In this section, we define several basic concepts, which are designed based on Yelp dataset, and are also applicable in other datasets.

Since Yelp data only shows whether one restaurant is still open and does not indicate the specific closure date, existing researches approximates the business status of restaurants by the time of reviews or check-in records [2,3,5]. Therefore, we use the similar method and define *Restaurant Failure* as follows:

Definition 1. Restaurant Failure. *The date of the first review submitted is regarded as the opening date of a restaurant, and the date of the last review submitted is regarded as the closure date of the restaurant.*

Generally, business entities such as restaurants choose to cluster together. Studies have shown that clusters significantly promote business booming [14,15], and the incentive effect is called cluster effect [16]. One good example of this phenomenon is that customers prefer to choose a venue with many restaurants rather than a place with one standalone McDonald's.

Definition 2. Restaurant Cluster. *The restaurants are mapped to the map by latitude and longitude. A certain number of restaurants gathering geographically forms a restaurant cluster.*

The problem to be solved in this paper is to predict whether a restaurant will fail in some time, which is defined as follows:

Problem Definition. *Build a prediction model $\mathcal{F}$, which contains the following structures:*

Input: (a)heterogeneous data (review, check-in and business attributes) of a target restaurant; and (b)the cluster information data (the latitude and longitude) of all restaurants in the same city.

Output: whether the target restaurant will fail in the year of $t + 1$.

IV. THE FRAMEWORK

The model that we propose in this paper mainly consists of three views: semantic view (we capture semantic features of reviews via a neural network and different reviews are assigned with different importance according to their reviewers' habits), business attribute view (we select the most influential business attributes from datasets), and business cluster effect view (we identify business clusters based on density and differentiate restaurants in different clusters). After capturing features of these views, we input the compositive feature vector into lightGBM to get the prediction result as shown in Fig.1.

A. Extractation of Influence Factors

1) Extracting Semantic Features: To capture the abundant semantics about different aspects of a restaurant, review texts are firstly converted to machine learnable sequences. Since one-hot encoding leads to too long vector, the output of word embedding tools such as word2vec[17] or GloVle[18] is still too long (e.g. 100 dimensions is a common length of word vector, but a review with only 10 words is converted to 10*100 dimensions). Convolutional neural network (CNN), which is famous for its ability of high-dimension information extraction, is applied to reduce the size of vectors. The intermediate vectors outputted from CNN represent the highly condensed semantic features and contain the same semantics with origin sentence. So, we design a deep learning sentiment classification model to obtain the review representation vector from CNN layers as shown in Fig.2. When customers submit reviews, they are required to attach rating stars at the same time, which contain the same emotion with reviews. Lots of studies use the rating stars as sentiment label to train sentiment classification model [19,20]. In this paper, we also employ the rating star as our sentiment label.
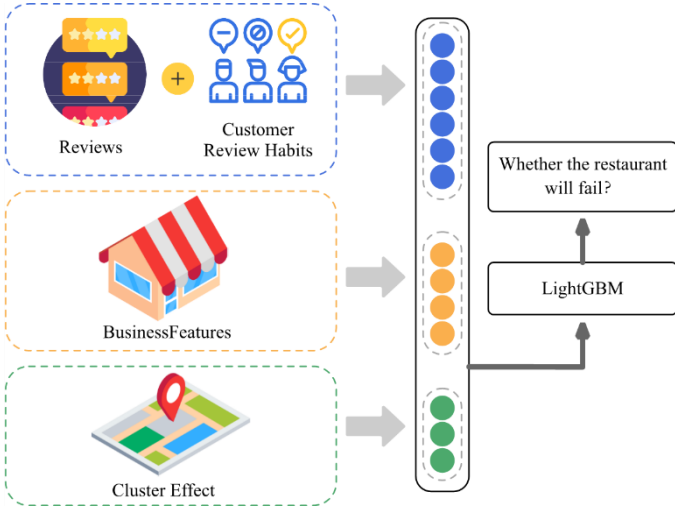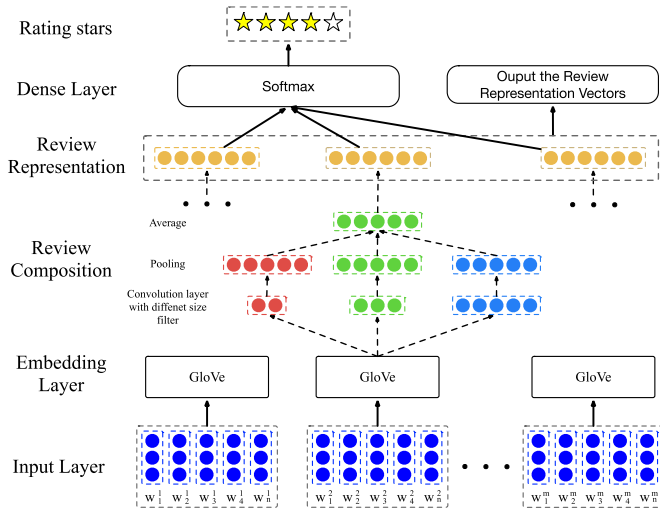
Figure 1. The Framework of SBCM.

Figure 2. The method to reduce dimensions of review representation vectors.

For each restaurant α, we choose the most popular and recent m reviews to input into our model to get the review representation vectors. The input matrix of input layer is created by concatenating the word vectors of a review. The neural network accepts fixed length vector as input, but the length of reviews is usually unfixed. To solve this mismatching problem, we cut out n words from a review as a review vector (If the length of some reviews is less than n, the reviews are padding with zero). After assembling the review vectors, we get a $m*n$ review matrix, which is defined as follows:

$$M_r = (w_1^m; w_2^m; \dots ; w_n^m) \qquad (1)$$

where w_n^m stands for the n-th word in the m-th review, $M_r \in \mathbb{R}^{m*n}$ is the review matrix.

In our method, GolVe is employed to generate embedding word vectors. All the word vectors in the GloVe are stacked in a word embedding matrix $M_w \in \mathbb{R}^{d \times |V|}$, where d is the dimension of word vector and $|V|$ is the vocabulary size. We

employ the pre-trained M_w from GloVe's official website[1] to ensure the efficiency of the word vector. In the embedding layer, every word in M_r is converted into a vector of floating number by finding every word vector in the M_w, which is defined as follows:

$$M_e = F_e(M_r, M_w) \qquad (2)$$

where F_e denotes the operation of embedding. The matrix M_e is a set of m reviews, in which every word is converted.

Then, CNN is applied to compute representation vectors of reviews and to reduce the length of vectors. Several convolutional filters of different widths are used in the convolution layer to capture different semantic of various granularities. For example, a convolutional filter with a width of 2 captures the semantics of phrases in a sentence and a width of 5 captures the semantics of short sentences in a sentence as Fig.2 shows. The process of the CNN convolution is defined as:

$$M_c = f(W_e * M_e + b_e)$$

where $*$ denotes the operation of convolution and f is an activation function. W_{rl} and b_{rl} are learnable parameters. We input M_c into a pooling layer to reduce the size. Then an average pooling layer is employed to capture the whole semantics of the review. So far, we get the review representation feature vector matrix $M_o \in \mathbb{R}^{m*k}$, in which k is the output length of the review representation vector.

2) Integrating Review Importance Weight: Reviews are given by different reviewers with respective reviewing habits. Yelp has a simple and intuitive weight metric, i.e. review vote, which is not able to reflect this difference. Inspired by this, we design a review importance weight metric, considering both review vote and reviewer attributes. In Yelp, there are three tree type of votes, i.e., useful, funny and cool, all of which are positive vote. We employ the number of all votes as the weight of a review, which is defined as follows:

$$W_r = \theta_u^r + \theta_f^r + \theta_c^r \qquad (3)$$

where θ_u, θ_f, and θ_c denote the number of useful, funny and cool votes. We think that the more votes a review have received the more important a review is. We then adopt the average received votes as the weight of a reviewer, which is defined as follows:

$$W_u = \frac{\zeta_u}{\delta_u} \qquad (4)$$

where δ_u denotes the number of reviews that a reviewer has written. ζ_u denotes the number of all kinds of votes that a reviewer have received. η_r denotes the rating star of a review. η_u denotes the average rating star submitted by the reviewer. $\eta = |\eta_r - \eta_u|$ stands for how different between the sentiment of review and the reviewer's ordinary habits. Intuitively, the bigger η is, the more influential a review is. Considering the above factors, the review weight metric is defined as follows:

$$I = ln\big((W_r + W_u) * \eta + 1\big) \qquad (5)$$

where I is in [0,1). By calculating every I of m reviews, we can get matrix $M_I \in \mathbb{R}^{m*1}$. To concatenate M_o with M_I, we get a matrix containing semantics and corresponding weights, which is denoted as $M_s \in \mathbb{R}^{m*(k+1)}$.

3) Screening Business Attributes:

As we mentioned above, commercial data are commercial secrets with limited access. Thanks to Yelp, we obtain business attributes of the restaurants instead, such as credit card, Wi-Fi, parking etc. We explore the importance of each attribute on restaurant failure by inputting all business attributes as a vector into our prediction component and output the weights.

By removing the zero-impact and low-impact attributessuch as music type and atmosphere, we select the most influential business attributes. Then we concatenate these attributes a_i into a business attribute vector $\Gamma_b = a_1 \oplus a_2 \oplus \ldots \oplus a_i$, where $\oplus$ represents vector connection. Finally, by flattening M_s into a vector and combining with Γ_b, we obtain a vector Γ_{sb}, which contains semantics and business attributes.

4) Capturing the Influence of Cluster Effect :

A restaurant cluster is a geographical location where enough resources and competences amass and reach a critical threshold, which is close to the density cluster. We employ DBSCAN to cluster the restaurants. DBSCAN is an algorithm to discover arbitrary-shaped clusters and to distinguish noise points simultaneously. In detail, DBSCAN accepts a radius value ε and a minimal value $MinPts$, which means that there are at least $MinPts$ points within the area of ε radius. Fig.3 shows the restaurant clusters of Toronto and Las Vegas in 2017 calculated by DBSCAN.

Figure 3. The restaurant cluster of (a) Toronto and (b) Las Vegas in 2017. In the figure, different colors denotes different restaurant clusters.

Researchers find that increasing the productivity of business clusters increases the competitive advantage of their individual [16]. Therefore, we employ the total number of review and check-in of restaurants in a cluster to reflect its competitive advantage, which is defined as follows:

$$E = \sum (review_\alpha + checkin_\alpha), \alpha \text{ in } C_\alpha \qquad (6)$$

where α denotes a restaurant and C_α denotes the restaurant cluster where α is located in.

By concatenating Γ_{sb} with E, we get a restaurant feature vector Γ_{sbc}, which contains semantic features, business features and cluster effect.

B. Prediction Component

LightGBM [7] is a fast, distributed, high-performance gradient boosting framework based on Gradient Boosting, which provides a good way to solve classification and regression problems by combining many tree models into a more accurate one. Compared to other Gradient Boosting Decision Tree (GBDT) algorithms using level-wise tree growth strategy, LightGBM produces more complex trees by following leaf-wise split approach, which is the main factor in achieving higher accuracy. In addition, it supports parallel and GPU learning and has compatibility in handling large-scale data. Therefore, LightGBM is adopted in this paper to predict whether restaurants will fail in the future, which is essentially a classification problem as succeed or fail.

We split the restaurant dataset into a training set and a testing set. After each restaurant in these two sets going through the process and forming a feature vector Γ_{sbc}, as shown in Fig.4, we get a training feature vector set ϕ_{train} and a testing feature vector set ϕ_{test}. Then we input ϕ_{train} into LightGBM to train a model $\mathcal{M}$. Finally, we input ϕ_{test} into model $\mathcal{M}$ and get the final prediction value set ψ, in which every prediction value $\hat{y}$ denotes whether the corresponding restaurant will fail.

Figure 4. The process to generate a combined feature vector Γ_{sbc}.

V. EXPERIMENTS

A. Experimental Setup

1) Datasets: In this paper, we select the open dataset from Yelp[2] for experiments, which includes more than 190,000 restaurants, more than 5 million review data and more than 1 minllion user data in various cities from October 2004 to November 2018. After analyzing the datasets, we find that Las Vegas and Toronto have the highest number of reviews from different countries, which indicates the popularity of yelp in these cities. Therefore, we select the latest data from 2016 to 2017 of Las Vegas and Toronto as our experiment dataset (the data of 2018 is incomplete). The detailed statistics of the datasets are shown in Table Ⅰ. In our experiment, restaurants that receive less than 10 reviews are filtered to ensure enough data for semantic extraction. We split 80% of the dataset as training data and 20% of the dataset as testing data.

TABLE I.   STATISTICS OF DATASETS

	Las Vegas		Toronto	
	2016	*2017*	*2016*	*2017*
#Restaurant	22515	24004	14286	14760
#Closed Restaurant	883	873	744	668

2)   Parameters Settings: In our experiment, the number of the most popular and recent reviews(i.e., m)is set as 10. The length of every review(i.e., n) is set as 20. We implement the method to get semantic feature vector by Keras, which is a fast experimentation neural networks API running on top of TensorFlow. Our experiments were run on a cluster with four NVIDIA 1080Ti GPUs.

3)   Evaluation Metrics: As Table Ⅰ shows, the numbers of the closed restaurants and surivial restauants are imbalance. Due to this, we employ ROC curve (receiver operating characteristic curve) and AUC (receiver operating characteristic's area under curve) as evaluation metrics. An ROC curve is defined by FPR (false positive rate) and TPR (true positive rate) as x and y axes, respectively, which depicts relative trade-offs between true positive (benefits) and false positive (costs). The area under the curve is defined as AUC.

4)   Comparison Methods: We compare our model with two other methods. As for the SBCM that we proposed, we also conduct multiple experiments without semantics and without cluster effect.

- SVM [21]: A supervised learning model that uses classification algorithms for two-group classification problems.

- XGBoost [22]: A gradient boosting tree model, which has gained widely popularity and attention recently after many winning teams of competitions using it.

- SBCM with no semantics: A variant model of SBCM, which does not contain semantic features.

- SBCM with no cluster effect: A variant model of SBCM, which does not contain cluster effect features.

B.   Experimental Results

1)   Performance Comparison:

The comparisons between SBCM and other methods is shown in Fig.5. As we can see, SBCM obtains the best performance on the datasets of Toronto and Las Vegas both in 2016 and 2017. On average, SBCM outperforms SVM and XGBoost by 14% and 3%, respectively in terms of AUC. Besides, the model performance on four datasets in terms of AUC also shows that our proposed model is most stable, and the range of SBCM in terms of AUC is 0.05. In general, if the AUC score of a model is above 0.7, it is regarded as a "fair model", and the average AUC of our proposed model is 0.78, which is above the standard.

In order to verify the validity of SBCM, we remove part of the structure in our model and conduct the same experiment. The results, shown in Fig.6, indicate the importance of semantics and cluster effect. Specifically, SBCM outperforms SBCM without semantics and SBCM without cluster effect by 8% and 2%,

respectively in terms of AUC, which demonstrates that semantics is more important than cluster effect in our model. In addition, we notice that results in 2017 is worse than 2016 both in Toronto and Las Vegas. The possible reason is that restaurants experienced bad periods of closures in 2016 [3], which reduces the training datasets for 2017 prediction and leads to the low performance.

| (a) ROC curve - 2016 Toronto | (b) ROC curve - 2017 Toronto |

Figure 5.   Performance comparisons of different methods.

Figure 6.   Performance comparisons of SBCM with different structures.

2)   Importance ranking of business attributes:

We also explore the importance of business attributes, as shown in Fig.7. We notice that the three most important attributes that affect the future performance of restaurants are credit card support, lunch support and noise level. The importance of credit card support is comprehensible as it is safer and more convenient than cash. We also infer from the results that whether a restaurant provides lunch is closely relevant to customer's choice. Moreover, customers attach importance to the dinning environment such as noise influence.
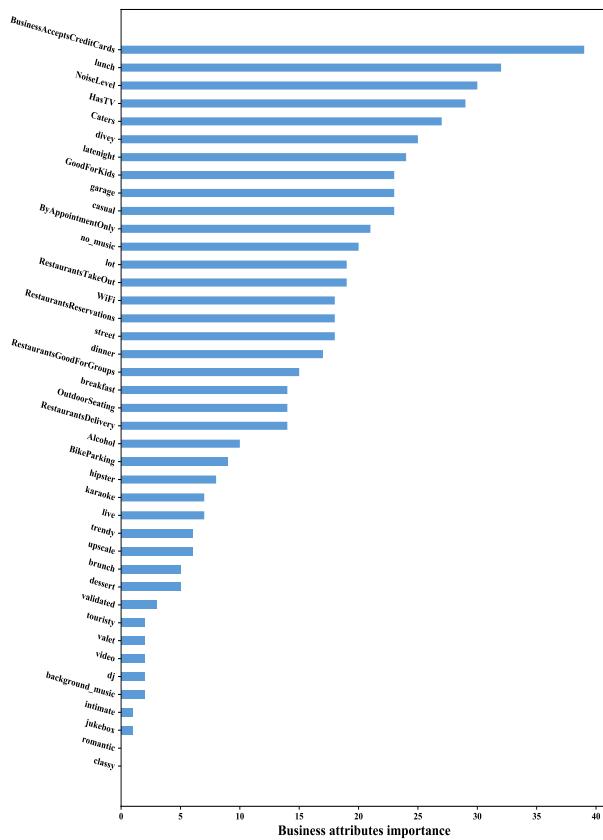
Figure 7. The importance ranking of different business attributes.

VI. CONCLUSION

In this paper, we propose a novel prediction model named SBCM based on review semantics, business attributes and cluster effect to predict business failure of restaurants. Specifically, our model consists of the following steps: 1) we capture semantic features of reviews via a neural network and different reviews are assigned with different importance according to their reviewers' habits; 2) we select the most influential business attributes from datasets; 3) we identify business clusters based on density and differentiate restaurants in different clusters; 4) The above semantic features, business features and cluster effect features are combined and input into LightGBM [7] to get the final prediction value. Experiments on public Yelp datasets of Toronto and Las Vegas from 2016 to 2017 demonstrate that SBCM averagely outperforms SVM and XGBoost by 14% and 3% respectively in terms of AUC.

In the future, we will further study the following issues: (a) explore the semantic influence of specific words that represent restaurant failure; (b) introduce more heterogeneous information to complete the model; and (c) improve the business cluster method to better simulate the actual restaurant clusters.

REFERENCES

[1] "National restaurant association," https://www.restaurant.org/Downloads/PDFs/Research/SOI/restaurant_industry_fact_sheet_2019.pdf, 2019.

[2] Zhang M, Luo L. Can user generated content predict restaurant survival: deep learning of yelp photos and reviews[J]. Available at SSRN 3108288, 2018.

[3] Snow D. Predicting Restaurant Facility Closures[J]. Available at SSRN 3420490, 2018.

[4] Hegde S, Satyappanavar S, Setty S. Restaurant setup business analysis using yelp dataset[C]//2017 International Conference on Advances in Computing, Communications and Informatics (ICACCI). IEEE, 2017: 2342-2348.

[5] Wang L, Gopal R, Shankar R, et al. On the brink: Predicting business failure with mobile location-based checkins[J]. Decision Support Systems, 2015, 76: 3-13.

[6] Ester M, Kriegel H P, Sander J, et al. A density-based algorithm for discovering clusters in large spatial databases with noise[C]//Kdd. 1996, 96(34): 226-231.

[7] Ke G, Meng Q, Finley T, et al. Lightgbm: A highly efficient gradient boosting decision tree[C]//Advances in neural information processing systems. 2017: 3146-3154.

[8] Altman E I. Financial ratios, discriminant analysis and the prediction of corporate bankruptcy[J]. The journal of finance, 1968, 23(4): 589-609.

[9] Jones S, Hensher D A. Predicting firm financial distress: A mixed logit model[J]. The accounting review, 2004, 79(4): 1011-1038.

[10] Andreeva G, Calabrese R, Osmetti S A. A comparative analysis of the UK and Italian small businesses using Generalised Extreme Value models[J]. European Journal of Operational Research, 2016, 249(2): 506-516.

[11] Pal R, Kupka K, Aneja A P, et al. Business health characterization: A hybrid regression and support vector machine analysis[J]. Expert Systems with Applications, 2016, 49: 48-59.

[12] Tkáč M, Verner R. Artificial neural networks in business: Two decades of research[J]. Applied Soft Computing, 2016, 38: 788-804.

[13] Hu L, Sun A, Liu Y. Your neighbors affect your ratings: on geographical neighborhood influence to rating prediction[C]//Proceedings of the 37th international ACM SIGIR conference on Research & development in information retrieval. 2014: 345-354.

[14] Canina L, Enz C A, Harrison J S. Agglomeration efects and strategic orientations: Evidence from the US lodging industry[J]. Academy of management journal, 2005, 48(4): 565-581.

[15] McClanahan B, Gokhale S S. Centrality and cluster analysis of yelp mutual customer business graph[C]//2016 IEEE 40th Annual Computer Software and Applications Conference (COMPSAC). IEEE, 2016, 1: 592-601.

[16] Porter M E. The competitive advantage of nations. New York: Free Press[J]. PorterThe Competitive Advantage of Nations1990, 1990.

[17] Mikolov T, Chen K, Corrado G, et al. Efficient estimation of word representations in vector space[J]. arXiv preprint arXiv:1301.3781, 2013.

[18] Pennington J, Socher R, Manning C D. Glove: Global vectors for word representation[C]//Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP). 2014: 1532-1543.

[19] Tang D, Qin B, Liu T. Learning semantic representations of users and products for document level sentiment classification[C]//Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers). 2015: 1014-1023.

[20] Wen S, Li J. Recurrent Convolutional Neural Network with Attention for Twitter and Yelp Sentiment Classification: ARC Model for Sentiment Classification[C]//Proceedings of the 2018 International Conference on Algorithms, Computing and Artificial Intelligence. 2018: 1-7.

[21] Cortes C, Vapnik V. Support-vector networks[J]. Machine learning, 1995, 20(3): 273-297.

[22] Chen T, Guestrin C. Xgboost: A scalable tree boosting system[C]//Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining. 2016: 785-79

Detecting Spammers from Hot Events on Microblog Platforms: An Experimental Study

Jialing Liang[1], Peiquan Jin[1*], Lin Mu[1], Jie Zhao[2*]

[1]School of Computer Science and Technology, University of Science and Technology of China, Hefei, China
[2]School of Business, Anhui University, Hefei, China
jpq@ustc.edu.cn

Abstract—**With the development of Web 2.0, social media such as Sina Weibo, Douban, and Zhihu have become an important platform for the dissemination and fermentation of hot events. At the same time, many spammers are hidden in the network, and they are driven by the interests to participate in the process of event dissemination, disseminating information with a propensity, and guiding public opinion through speculation, malicious comments, malicious attacks, etc. It interferes with the network order and the decision making based on social networks, and even affects social stability. Therefore, it is important for government and enterprises to accurately detect spammers from hot events on microblog platforms and further make sure whether a hot event is natural developing or driven by spammers. In this paper, we focus on the hot event list on Sina Weibo and collect relevant microblogs and involved users of each hot event. Then, we employ typical machine learning methods to conduct an experimental study on detecting spammers. Specially, we develop a new set of features based on three aspects, including user profile, user behavior, and user relationships, to reflect various factors affecting the detection of spammers. Finally, we conduct experiments on a real data set from the Sina Weibo, and compare three machine learning models including the Naive Bayes, the J48 Decision Tree, and the Logistic Regression model, concerning various metrics like precision, recall, F-measure and AUC. The results show that the Logistic Regression model achieves the best average F-measure in detecting both spammers and non-spammers.**

Keywords-microblog; spammer detection; user feature; classification model

I. INTRODUCTION

As online users migrate to mobile terminals and social media, social platforms such as Weibo that are instant and convenient have become important channels for netizens to interact, share, and disseminate with others. Weibo users can share a daily life by posting a short text on the web and mobile terminals and can browse the information posted by other users to get attention to current events and hot spots or participate in the discussion of popular topics and the spread of events through methods such as reposting comments. in. According to the CNNIC Report of China [1] released by the China Internet Network Information Center in February 2019, as of December 2018, the number of microblog users in China was 350 million, accounting for the total number of Internet users. The proportion reached 42.3%. In typical social applications, compared to the privacy of the WeChat circle of friends and QQ space, Weibo

has suddenly become a mainstream online media with many significant features such as large user scale, strong sociality, fast propagation speed, and fast response speed. For example, on November 17, 2018, the People's Daily published the Weibo topic "China is not a little bit". It has been reposted 1.259 million times in just half a day, received 118,000 comments and 943,000 likes. The topic of reading reached 8.94 billion.

The existence of Weibo provides a fast platform for the propagation of hot events [2-4]. But at the same time, due to the emergence and promotion of the online water army, Weibo has also become a target platform for spreading rumors and hype. Spammers are those that are driven by commercial interests to achieve improper purposes such as influencing public opinion and disrupting the network environment, thereby manipulating software robots or spam accounts and producing and disseminating false information and spam on the Internet. Generally, spammers may manipulate spam accounts to speculate bland blog posts or topics as hot events. The purpose is to gain fame and attention, fight against hostile forces, stir up public sentiment, or guide public opinion, Spammers can mislead Internet users' correct judgment of the situation of events, and even maliciously attack the government and affect social stability Due to its large scale, the high degree of coverage, and wide target range, the network spam makes it difficult to identify spam accounts from a great number of users solely by manual means.

Based on the above analysis, this article focuses on the detection of spammers during the spread of Sina Weibo hot events. Particularly, we use machine learning methods to identify spam accounts from users who participate in the process of promoting an event to become a hot spot, giving evidence of whether the event propagation process is naturally fermented or promoted by the spam, and finally help decision-makers to guide and control public opinion [2].

Briefly, we make the following contributions in this paper:

(1) We design a crawler platform to collect detailed personal information of an event-related Weibo user, making the data set more realistic. We also set up a Weibo user manual labeling platform and design the labeling process so that each Weibo user can judge the labeled results, reducing the errors caused by manual labeling. In addition, the web-based platform makes labeling work much convenient and reliable.

(2) A Weibo spammers detection method combining user attribute characteristics, user behavior characteristics, and user relationship characteristics is proposed. Compared with the existing methods, the method proposed in this paper

comprehensively considers three user characteristics, which is more suitable for the identification of spammers.

(3) Based on the defined feature set, we conduct experiments on a real dataset and compare the performance of three types of classification models, including Naive Bayes, J48 decision tree, and logistic regression model. The results show that the logistic regression model has the best detection effect.

The remainder of the paper is structured as follows. Section II provides a brief literature review on recent research progress. Section III describes the data crawling and cleaning process. Section IV presents feature selection. Section V reports the experimental results, and finally, we conclude the entire paper in Section VI.

II. RELATED WORK

Spammers first appeared in the e-mail field, and then quickly spread to the e-commerce and social fields. Existing detection methods of network spammers are mainly divided into detection based on content features, user features, environment features, and comprehensive features [5].

In the early network environment, the spam was mainly used to create many spam emails and false comments on e-commerce platforms. The content generated by the online spam included obvious characteristics, such as commercial advertisements, spam, duplicate comments, etc. Most of the network spammers' recognition is based on the detection of content features, involving text orientation analysis [6], sentiment analysis [7], and other methods in natural language processing. The filtering and detection of spam have been researched for a long time: the literature [8] analyzes the existing detection and evaluation work in the two fields of electronic spam and image spam; literature [9] has seven differences A comparative study of the version of the Naive Bayes classifier and the linear support vector machine for automatic filtering of e-mail spam was conducted. For fake reviewers in e-commerce platforms and forums, usually by analyzing the text 's propensity analysis to identify fake reviews that deviate from unspammed user reviews [10]; some researchers also look for different rules or groups of rules. Used to detect abnormal comment user behavior [11].

With the rise and development of social platforms such as Twitter, Facebook, and Sina Weibo, and the increase in the number of users on the social network, coupled with the enhancement of user identification, the Internet has continued to improve its concealment and deception strategies. For normal users, its published content no longer has obvious spam features. Therefore, the detection and recognition of the network spam have also gradually shifted from content-based features to user-based features. Benevenuto et al. [12] used tweets related to the three hot topics to manually construct a labeled dataset, determine 39 attribute features related to the content of the tweet and 23 attribute features related to the user, and then use the SVM method is used to classify and finally the analysis of experimental results is performed. Murmann et al. [13] used neighbor nodes with interactive relationships to detect the trust relationship between users in Twitter, and obtained a new relationship feature set, and used this feature set to rank the suspiciousness of all users, with the highest suspiciousness among them. That is judged as the network spam. Wang et al. [14] created a directed social graph to show the relationship between followers and fans. Based on the tweet content features and user relationship graph features, the Bayesian classifier was used for spam detection, achieving an accuracy of 89%. rate. Yang et al. [15] deeply analyzed the concealment and deception strategies of the Twitter network spam and proposed a method to detect the network spam in Twitter based on the characteristics of neighbor nodes. Han Cao et al. [16] constructed a recognition network by taking the user's attribute characteristics as the input variables of the learning model, the user's behavior characteristics as the observation variables, and the probability that the user is a spam force is the hidden variable between the input and the observed variables The spam's probability map model is used to calculate the probability that the user is spam. Bhat et al. [17] found that similar to ordinary users, the network sailors in the social field can also form a certain size network sailor community. To this end, they extracted user interaction diagrams from the behavior logs of Weibo users, found overlapping community maps formed therein, and after manually marking some of the network spam nodes, they calculated each node to be identified. Communicate with the community of marked nodes to classify unknown nodes. In addition, Azad, et al. [18] presented a rapid detection method for spammers through collaborative information sharing across multiple service providers, which showed that fusing muttiple information provided by various providers was helpful for spammer detection.

Compared with the existing work, this paper designs a new crawler algorithm, which takes the keywords of the hot event as seeds to crawl the event-related microblogs and the detailed personal information of the microblog users who participated in this hot event. We also construct a manual labeling platform to tag the dataset. In addition, we propose a new feature set based on user attribute characteristics, user behavior characteristics, and user relationship characteristics. Compared with the existing methods, the method proposed in this paper comprehensively considers three user characteristics, which are more suitable for the identification of spammers.

III. DATA CRAWLING AND LABELLING

A. Data Crawling

The data crawling part uses Python's crawler framework and configures the Google Chrome driver to simulate login to obtain cookie data. We use popular event keywords and link to the old search interface of Sina Weibo to form a seed URL, and crawl the Weibo details returned by the search page, including Weibo content, likes, retweets, comments, release time and personal information. The crawled data is stored in MongoDB, which maintains Weibo information tables and personal information tables.

B. Data Cleaning

The data cleaning of the original data crawled by the crawler is mainly divided into two steps. The first step is to filter the Weibo or missing important information generated by the dynamic webpage or Weibo anti-crawling caused by the 302 transfer when crawling data. The second step is the manual labeling phase. When the user information is abnormally absent, for example, the number of followers, followers, and tweets of a user is not 0, but the list of followers, followers, and tweets is empty, we cannot judge whether the user is a spam user based
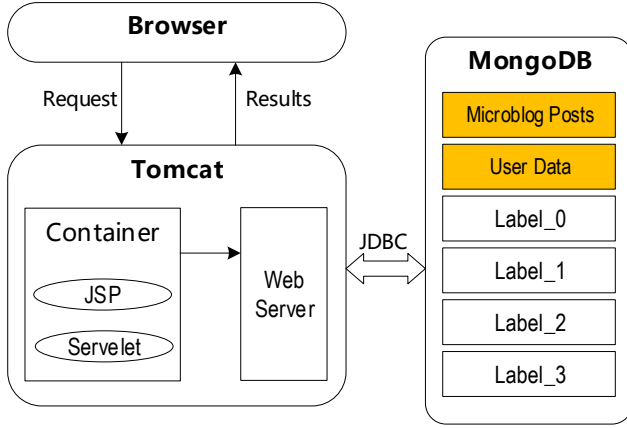
Figure1.   Architecture of the labeling platform.

TABLE I.   FEATURES OF MICROBLOG USERS

No.	Type	Feature
1	User-Profile Features	Num_Follows
2	User-Profile Features	Num_Fans
3	User-Profile Features	Num_Tweets
4	User-Profile Features	FAuthentication
5	User-Profile Features	FBriefIntroduction
6	User-Profile Features	FVIP
7	User-Behavior Features	Original_Ratio
8	User-Behavior Features	URLs_Ratio
9	User-Behavior Features	Mentions_Ratio
10	User-Behavior Features	Topics_Ratio
11	User-Behavior Features	Self-Similarity
12	User-Relationship Features	Fans_Follows_Ratio
13	User-Relationship Features	Aggregation_Coef

on the existing information. As a result, such users will be removed from the database at this time.

C. The Labeling Platform

The manual labelling platform is a tool we developed and deployed on a Tomcat server. The architecture is shown in Fig. 1.

There are four label tables denoted as Lable_0, Label_1, Lable_2, and Label_3 in Fig. 1. The Label_0 stores the initial unlabeled user ID. The Label_1 stores the user ID and label labeled by one tagger. The Label_2 stores the user ID and label labeled by two taggers, and the Label_3 stores the user ID and labeled mark.

The process of the labeling platform is as follows:

(1) Copy the IDs of all users from the personal information table to Label_0, and set the flag field to -1 to indicate no flag;

(2) The client sends a request, and the web container will perform the JSP conversion and the compiled file. When the Label_0 table is not empty, randomly obtain a user ID from Label_0, and obtain the detailed information of the ID from the personal information table. The result is returned to the browser. If Label_0 is empty, the ID is obtained from Label_1, and so on, until the Label_2 table is also empty, indicating that the mark has been completely completed;

(3) Submit the tag determined by the tagger to the web server and forward it to the servlet container. At this time, delete the ID from Label$_i$, change the value of the tag field, and add the user ID and the tag field to Label$_i$ + 1.

(4) The browser refreshes the current page after receiving a response, that is, continues to step (2) until all users have completed the labeling by three users.

IV.   FEATURES SELECTION

To effectively identify the spam users in the user group, in this paper we design the following features (see Table 1).

A. User-Profile Features

(1) Num_Follows
Unspammed users generally only pay attention to the people they are interested in, so the number of followers will be in a

relatively reasonable range. To achieve the effect of publicity and hype, spammers often follow a lot of bloggers. Users will have a higher number of followers than non-spammers.

(2) Num_Fans
Unspammed users will have a circle of friends on the Weibo platform, so there is a certain percentage of followers, and spammers are often fans of other people, but they rarely attract the interest and attention of others. Compared with unspammed users, the number of fans of the spam is very small.

(3) Num_Tweets
Unspammed users use Weibo normally. There will be a certain percentage of Weibo users, who are either new users only publishing or forwarding specific tweets or be active in the comments to promote hype and public opinions. On the other side, their own posts are few. Thus, non-spammer users generally post more than spammers.

(4) FAuthentication
On microblogging platforms, an authenticated account will generally be more credible and authoritative than an unauthenticated account. Therefore, authenticated users are more likely to be unspammed users, while unauthenticated users are more likely to be spammers.

(5) FBriefIntroduction
Spammers generally have relatively low completeness of the information. Few Spam users fill out the personal profile field. Therefore, Spammers are more likely to have no profile, and unspammed users are more likely to have a profile.

(6) FVIP
Generally speaking, spammers do not need to register a VIP because it is costly. However, many normal users will choose to

pay for VIPs to obtain more functions and benefits when using microblogging services. To this end, users who are VIPs are more inclined to be unspammed users, and users who have not registered as VIPs are more likely to be spammers.

B. User-Behavior Features

(1) Original_Ratio

$$Original\_Ratio = \frac{Original\_Tweets}{Num\_Tweets}$$

Spammers are usually controlled by machines or robots, so most spammers are more likely to repost and comment on a certain microblog, and rarely publish original microblogs. On the contrary, normal users will share their daily life around them and will publish a certain percentage of original tweets. Thus, the original ratio of tweets posted by spammers can be generally lower than that of normal users.

(2) URLs_Ratio

$$URLs\_Ratio = \frac{Num\_URLs}{Num\_Tweets}$$

Unspammed users are limited to 140 characters of tweet text for propaganda and hype. The microblogs posted or reposted may contain more URLs of web links than unspammed users, thus inducing users to click on the links to browse the page they want to display. Therefore, the utilization rate of URLs for spam users is generally higher than that of non-spam users.

(3) Mentions_Ratio

$$Mentions\_Ratio = \frac{Num\_Mentions}{Num\_Tweets}$$

The @ method is used to remind users who are @ to view the Weibo content in time. After being logged in by @ users, they can see the reminder information of the Weibo. Spammers will attract @ users' attention through @some unrelated users, to achieve rapid diffusion. Therefore, the @usage rate of spam users may be higher than that of unspammed users.

(4) Topics_Ratio

$$Topics\_Ratio = \frac{Num\_Topics}{2 \times Num\_Tweets}$$

When users participate in the discussion of a hot topic, a # sign is often included outside the topic. Spammers will use the hashtag # more to achieve the hype topic and promote the topic to become a popular purpose. Therefore, the # usage rate of spam users may be higher than that of unspammed users. Since # always appears in pairs, and Num_Topics only represents the number of occurrences of # in tweets, the denominator in the definition needs to be multiplied by 2.

(5) Self-Similarity

The self-similarity among historical tweets refers to the proportion of similar tweets in the total number of posts published by users. To achieve the purpose of publicity and marketing, spammers often use content templates to generate many similar microblogs. Therefore, the historical microblog self-similarity of spammers is generally higher than that of non-spammers.

To calculate the self-similarity, we use a hierarchical clustering method based on the cosine similarity to cluster the historical microblogs of a user to form clusters $S=(C\_1, C\_2, ..., C\_k)$. Here, k is the number of clustered classes. C_J is the jth class that contains N tweets, which can be defined as $C\_J=(T\_J1, T\_J2, ..., T\_JN)$. The N tweets are regarded as similar tweets. Then, we define the self-similarity of tweets as follows [15].

$$Similarity = \frac{\sum_{J=1}^{J=k} G\left(C_J\right)}{Num\_Tweets}, G(C_J) = \begin{cases} N & ,N \geq 2 \\ 0 & ,\text{otherwise} \end{cases}$$

C. User-Relationship Features

(1) Fans_Follows_Ratio

$$Fans\_Follows\_Ratio = \frac{Num\_Followees}{Num\_Followers}$$

Unspammed users have a normal social circle of friends with a similar number of followees or followers, or large V users have a great number of followees, and the ratio of followees to followers is large. However, spammers tend to follow many users but only a small number of followees; therefore, the follower ratio of spam users will be lower, and the follower ratio of unspammed users will be higher.

(2) $Aggregation\_Coef_i$

$$Aggregation\_Coef_i = \frac{\sum_{j,k=1}^{N} a_{ij}\, a_{jk} a_{ki}}{k_i(k_i - 1)}$$

We construct an undirected graph $G = (V, E)$ using the followee and follower list of all users crawled. The adjacency matrix of graph G is expressed as $A = (a_{ij})_{N \times N}$, and k_i is the degree of node i, $\frac{1}{2}\sum_{j,k=1}^{N} a_{ij}\, a_{jk} a_{ki}$ represents the number of neighbor pairs formed between node i and k_i neighbor nodes. $ggregation_{Coef_i}$ calculates the clustering coefficient of user i. In general, the clustering coefficient is used to evaluate the probability that a user's friends are also friends with each other: for unspammed users, they are closer to their neighbors, that is, the network of friends is closer, yielding a large coefficient. However, as more neighbors of spammers are independent points, their clustering coefficient will be relatively small.

V. Performance Evaluation

A. Settings

Dataset. The data set in this study was collected from the users who participated in comments and reposts under the popular tweets returned by the search keyword "Huawei sued the US government" in the old Sina Weibo search interface. There are 341 popular microblogs related to words and topics. After filtering out users who are restricted by Weibo anti-crawling restrictions and crawling incomplete information, 8149 users have been collected. After manual labeling, 312 of them are non-spammer users. There are 7,837 spammers. To avoid the

category imbalance caused by the large difference between the positive and negative examples, the data of a total of 800 users, including 312 spam users and 488 unspammed users are used for 10-fold cross-validation in the experiment.

Metrics. In the experiments, we use precision, recall, F-measure, and the AUC under the ROC curve as the evaluation indicators for spammer detection. Let TP be the number of spam users correctly classified by the classifier, FP be normal users incorrectly classified as spam users, and FN be spam users incorrectly classified as normal users. The accuracy, recall and F1 values are defined as follows:

$$Precision = \frac{TP}{TP + FP}$$

$$Recall = \frac{TP}{TP + FN}$$

$$F1 = \frac{2 \times P \times R}{P + R}$$

B. Classification Models

We select three classification models and compare their performance on detecting spammers.

(1) *Naïve Bayes.* Let D be the training set, A be the attribute set of users, we represent each user by an n-dimensional vector $X = (x_1, x_2, \ldots, x_n)$, and the results are labeled into $m=2$ classes $C = (C_1, C_2, \ldots, C_m)$.

(2) *J48 Decision Tree.* The J48 decision tree algorithm is a top-down, recursive divide-and-conquer strategy: selecting a certain attribute to place at the root node, generating a branch for each possible attribute value, dividing the instance into multiple subsets, each subset corresponding to a root Node branches, then repeat this process recursively on each branch. When all instances have the same classification, the algorithm stops. Let D be the training set, A be the attribute set of users, we represent each user by an n-dimensional vector $X = (x_1, x_2, \ldots, x_n)$, and the results are labeled into $m=2$ classes $C = (C_1, C_2, \ldots, C_m)$.

(3) *Logistic Regression.* Let D be the training set, A be the attribute set of users, and m be number of samples, we represent each user by an n-dimensional vector $X = (x_1, x_2, \ldots, x_n)$, and the results are labeled into $C = (C_1, C_2)$.

C. Results

We use Java-based machine learning library Weka for the classification experiments. Tables II and II list the confusion matrix and classification results of the dataset under the Naive Bayes algorithm. Tables IV and V list the confusion matrix and classification results of the dataset under the J48 decision tree algorithm. Tables VI and VII list the confusion matrix and classification results of the data set under the logistic regression algorithm.

We can see that the decision tree achieves the highest precision for detecting spammers, but its recall is not the highest among all the three models. The naïve Bayes model achieves the best recall, but its precision is the lowest among all compared models, which leads to the lowest F-measure in the experiments.

We also list the performance of non-spammer detection in the tables. Generally, the recognition of non-spammers is as important as the detection of spammers. Thus, we calculate the

TABLE II. CONFUSION MATRIX OF NAIVE BAYES

Type	Detected as spammers	Detected as non-spammers
Spammer	96.67%	3.33%
Non-spammer	41.04%	58.96%

TABLE III. CLASSIFICATION RESULTS OF NAÏVE BAYES

Type	Hit Ratio	Error Rate	Precision	Recall	F-Measure	AUC
Spammer	0.967	0.410	0.613	0.967	0.750	0.938
Non-Spammers	0.590	0.033	0.963	0.590	0.731	0.937
Avg.	0.741	0.185	0.822	0.741	**0.739**	0.938

TABLE IV. CONFUSION MATRIX OF DECISION TREE

Type	Detected as spammers	Detected as non-spammers
Spammer	90.00%	10.00%
Non-spammer	9.70%	92.30%

TABLE V. CLASSIFICATION RESULTS OF DECISION TREE

Type	Hit Ratio	Error Rate	Precision	Recall	F-Measure	AUC
Spammer	0.900	0.097	0.862	0.900	0.880	0.925
Non-Spammers	0.903	0.100	0.931	0.903	0.917	0.925
Avg.	0.902	0.099	0.903	0.902	**0.902**	0.925

TABLE VI. CONFUSION MATRIX OF LOGISTIC REGRESSION

Type	Detected as spammers	Detected as non-spammers
Spammer	93.33%	6.67%
Non-spammer	11.19%	88.81%

TABLE VII. CLASSIFICATION RESULTS OF LOGISTIC REGRESSION

Type	Hit Ratio	Error Rate	Precision	Recall	F-Measure	AUC
Spammer	0.933	0.112	0.848	0.933	0.889	0.956
Non-Spammers	0.888	0.067	0.952	0.888	0.919	0.956
Avg.	0.906	0.085	0.910	0.906	**0.907**	0.956

average precision, recall, and F-measure of both spammers and non-spammers detection for all three models. The average value is denoted as the "avg." column in all tables. We can see that in terms of the average F-measure, which can be regarded as a balanced metric of precision and recall, the naïve Bayes performs worst and the Logistic Regression model performs best. The decision tree model gets comparable performance with the Logistic Regressions, indicating that it can also be considered in the detection of spammers.

VI. Conclusions and Future Work

Spammers have severely disrupted network order and decision analysis based on social networks. For hot events on the Weibo platform, judging whether the event is a natural fermentation or a spam promotion is of great significance for the government and enterprises to correctly evaluate the event situation. This article uses Sina Weibo's popular event keywords as a starting point, crawls the details of the Weibo returned under this keyword, and crawls the personal details of the commenting and forwarding users under Weibo. Furthermore, we designed and built an artificial labeling platform for Weibo users of the spammers / unspammed army. By displaying the user's personal information, the tagger will make judgments based on personal information to reduce the error of manual judgment. Based on this, a spam identification method combining user attribute characteristics, behavior characteristics, and relationship characteristics is proposed. We combine the results of artificial labeling to build the input set of the classification model and use the naive Bayes, J48 decision tree, and logistic regression models in the classification model to experimentally verify the real data set. The experimental results show that the logistic regression algorithm has the best classification effect on microblog user spam detection.

In future research, we will investigate a few topics. First, we will carry out the detection and analysis of the spam to obtain the proportion of spammers involved in a hot event. Second, we will study the evolution of public sentiment and topics related to spam [19-21], which is an important indicator to reveal the dynamic feature of spam on social networks. Third, we will crawl real-time hotspot events on social networks and construct a prototype that can monitor real-time spammers on microblogging platforms.

References

[1] The CNNIC Report of China. http://www.cnnic.net.cn/hlwfzyj/hlwxzbg/hlwtjbg/201902/P0201903185 23029756345.pdf. Accessed on 1 March, 2020

[2] J. Zhao, X. Wang, P. Jin, Feature selection for event discovery in social media: A comparative study . *Computers in Human Behaviour,* 2015, 51(B): 903-909

[3] L. Zheng, P. Jin, J. Zhao, L. Yue: A fine-grained approach for extracting events on microblogs. *Proceedings of the 25th International Conference on Database and Expert Systems Applications (DEXA)*, 2014: 275-283

[4] P. Jin, L. Mu, L. Zheng, J. Zhao, L. Yue: News feature extraction for events on social network platforms. *Proceedings of the 26th International World Wide Web Conference (WWW)*, 2017: 69-78

[5] S. Rathore, et al. SpamSpotter: An efficient spammer detection framework based on intelligent decision support system on Facebook. *Applied Soft Computing*. 2018, 67: 920-932

[6] H. Do, et al. Deep learning for aspect-based sentiment analysis: A comparative review. *Expert Systems with Applications*. 2019, 118: 272-299

[7] D. Zimbra, et al. The state-of-the-art in Twitter sentiment analysis: A review and benchmark evaluation. *ACM Transactions on Management Information Systems.* 2018, 9(2): 5:1-5:29

[8] P. Hayati, et al. Evaluation of spam detection and prevention frameworks for email and image spam: a state of art, *Proceedings of the 10th International Conference on Information Integration and Web-based Applications & Services (iiWAS)*, 2008: 520-527.

[9] T. Almeida, et al. Content-based spam filtering, *Proceedings of The 2010 International Joint Conference on Neural Networks (IJCNN).* IEEE, 2010: 1-7.

[10] F. Li, et al. Learning to identify review spam, *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence (IJCAI).* 2011: 2488-2493.

[11] N. Jindal, et al. Finding unusual review patterns using unexpected rules, *Proceedings of the 19th ACM International Conference on Information and Knowledge Management (CIKM)*, 2010: 1549-1552.

[12] F. Benevenuto, et al. Detecting spammers on twitter, *Proceedings of the Seventh annual Collaboration, Electronic messaging, AntiAbuse and Spam Conference (CEAS)*, Vol. 6, 2010: 12.

[13] A. Murmann. Enhancing spammer detection in online social networks with trust-based metrics. San Jose State University, 2009.

[14] A. Wang. Don't follow me: Spam detection in twitter, *Proceedings of the 2010 International Conference on Security and Cryptography (SECRYPT)*, 2010: 1-10.

[15] C. Yang, et al. Die free or live hard? Empirical evaluation and new design for fighting evolving twitter spammers, *Proceedings of the International Workshop on Recent Advances in Intrusion Detection*. 2011: 318-337.

[16] J. Cao, et al. Collusion-aware detection of review spammers in location based social networks. *World Wide Web*, 2019, 22(6): 2921-2951

[17] S. Bhat, et al. Community-based features for identifying spammers in online social networks, *Proceedings of the 2013 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)*. 2013: 100-107.

[18] M. Azad, et al. Rapid detection of spammers through collaborative information sharing across multiple service providers. *Future Generation of Computer Systems*. 2019, 95: 841-854

[19] J. Liang, L. Mu, P. Jin. MGP: Extracting multi-granular phases for evolutional events on social network platforms. *Proceedings of the 14th International Conference on Semantics, Knowledge and Grids (SKG)*, 2018: 269-272

[20] L. Mu, P. Jin, L. Zheng, E. Chen, L. Yue. Lifecycle-based event detection from microblogs. *Proceedings of the 27th International World Wide Web Conference (WWW),* 2018: 283-290

[21] L. Mu, P. Jin, L. Zheng, E. Chen. EventSys: Tracking event evolution on microblogging platforms. *Proceedings of the 23rd International Conference on Database Systems for Advanced Applications (DASFAA),* 2018: 797-801

Automatic Identification of Architecture Smell Discussions from Stack Overflow

Fangchao Tian [1,2], Fan Lu [1], Peng Liang [1,*], Muhammad Ali Babar [2]

[1] School of Computer Science, Wuhan University, Wuhan, China

[2] School of Computer Science, The University of Adelaide, Adelaide, Australia

Abstract—**Architecture Smells (ASs), as one source of technical debt, indicate underlying problems at a high level of systems and negatively impact various system qualities, such as maintainability and evolvability. Detecting and refactoring ASs requires the relevant architectural knowledge and experience. Therefore, gathering the knowledge of ASs from various sources can facilitate ASs detecting and refactoring. However, manually identifying AS knowledge is time-consuming. Automatically and correctly identifying AS-related posts from Stack Overflow is a step toward utilizing the AS knowledge to help developers better maintain their systems. In this work, we propose an approach to automatically identify AS-related posts from Stack Overflow (SoF) by using machine learning algorithms. We evaluate the performance of 12 classifiers based on 3 feature extraction techniques and 4 classification algorithms with a created dataset of SoF posts (including 208 AS-related posts and 187 AS-unrelated posts). The results demonstrate that the SVM algorithm with Word2Vec achieved the best overall performance with an accuracy of 0.650, a precision of 0.613, a recall of 0.905, and an F1-score of 0.731. These results imply that the obtained model of the AS-related posts identification can be used to aid developers and researchers in collecting AS discussions from SoF.**

Keywords—Architecture Smell, Architecture Smell Discussion, Stack Overflow, Text Classification

I. INTRODUCTION

Architecture Smells (ASs) are proposed as frequently recurring architectural decisions that negatively impact system quality [1]. ASs, as the counterpart of code smells, occur at a higher granularity level of a system and can have system-wide impact on maintainability issues. Therefore, detecting and refactoring ASs require more effort compared to code smells [2]. Different researchers defined different categories of ASs with supported detection tools [3]. Fontana *et al.* defined three dependency-related ASs and proposed a tool, called ARCAN, to detect them by analyzing dependency graphs extracted from the packages of compiled Java projects [2]. Mo *et al.* proposed Hotspot Detector to detect five types of ASs, called Hotspot Patterns, defined at the package and file levels [4]. Le *et al.* presented ARCADE which can detect 11 types of ASs across 4 categories [5]. These ASs detection tools are metrics-based and apply some fixed threshold to judge whether a package or component is smelly or not. But it is challenging to manually choose the metrics and thresholds, which can induce false-

positive instances of ASs. Correctly detecting and refactoring ASs requires knowledge and experience of developers and researchers to remove false positive instances [2]. Therefore, smell detection and refactoring rely on the knowledge and experience of developers and researchers, and need to consider different aspects such as system domain, software context, and software engineering experience. Moreover, unlike ASs detection, ASs refactoring is less researched and reported in the literature. Empirical studies on investing the knowledge and experience of detecting and refactoring ASs are needed [6].

Stack Overflow (SoF), as a crowdsourced knowledge sharing platform, has been a popularly and widely used software and development Questions and Answers (Q&A) sites that contain more than 18 million questions across a wide variety of topics since 2008 [7]. The knowledge and experience of developers in SoF has been adopted by researchers to study various topics. Tahir *et al.* investigated the developers' perception of code smells and anti-patterns by mining and analyzing the discussions about these two concepts in SoF [8]. In our previous work, we manually collected and analyzed 207 AS-related posts to investigate the understanding of developers about ASs [3], such as the approaches and tools used to detect and refactor ASs. Therefore, the discussions in SoF posts can provide knowledge of ASs and refactoring suggestions that can be used to guide a developer or architect in understanding and addressing potential issues in the architecture of a software system. However, from these studies [3][8], we can find that searching smell-related posts via tags or search terms is ineffective and induces false-positive posts. Furthermore, manually identifying AS-related posts is a time-consuming and subjective process which requires the expertise and experience of ASs and can also lead to inaccurate or incomplete posts. To address this challenge, automatically mining and identifying AS-related posts is needed.

Machine Learning (ML) and Nature Language Process (NLP) techniques have been extensively used to automatically identify or mine meaningful information from SoF posts. For example, Ahasanuzzaman *et al.* built a technique, called CAPS, that can automatically classify SoF posts concerning API issues [9]. Borg *et al.* used active learning to train an SVM classifier for identifying SoF posts concerning the performance of software components [10]. To the best of our knowledge, there is currently no study that automatically mines SoF post discussing ASs. Our research aims at closing this gap by automatically identifying AS-related posts from SoF.

We developed an approach to automate the classification of AS-related posts. We created a dataset, consisting of labelled

208 AS-related posts and 187 AS-unrelated posts, for training classification models. Furthermore, we ran an experiment with 12 configurations by using three feature extraction techniques (i.e., BoW, TF-IDF, and Word2Vec) and four classification algorithms (i.e., LR, SVM, KNN, and RF). We then compared the performance of the models measured in terms of accuracy, precision, recall, and F1-score to determine the best configuration. In our experiment, the use of the SVM algorithm with Word2Vec performed best for automating the classification of AS-related posts.

Thus, our paper makes these contributions: (1) a manually labelled dataset consisting of 208 AS-related posts and 187 AS-unrelated posts; (2) an approach to automatically identify AS-related posts using 12 different configurations regarding 3 feature extraction techniques and 4 classification algorithms; (3) an evaluation of the performance of 12 different classifiers on the dataset.

The rest of this paper is organized as follows. Section II presents the related work. The experiment methodology is explained in Section III. The results of our experiment are reported and discussed in Section IV. Section V concludes this work with future directions.

II. RELATED WORK

In the last years, research and practice on ASs has gained significant attention [2]. In this section, we provide an overview of ASs and automatic techniques for mining textual information from Stack Overflow.

A. Architecture Smells

Several studies proposed the definitions of ASs with different subtypes. ASs was originally proposed by Lipper [11] to indicate the underlying problems that occur at the architecture level of a system. They also provided a catalogue of ASs at different levels: dependency graphs, inheritance hierarchies, packages, subsystems and layer. Some of these ASs were provided with refactoring measures. Garcia et al. considered ASs as instances of poor architecture decisions that can affect a system life cycle properties, such as understandability and testability [1]. Moreover, they described four types of ASs and each smell's impact on a system lifecycle properties. Fontana et al. presented an ASs Detector, called ARCAN, which can identify three different dependency based ASs: Unstable Dependency, HubLike Dependency and Cyclic Dependency [2]. They later developed a prototype tool, as an extension of the Arcan tool, which can provide refactoring suggestions to remove Cyclic Dependency smell [12]. In another study, Mo et al. formally defined five architecture hotspot patterns and presented a tool, called hotspot detector, to automatically detect and identify these smells at packages or files level [4]. Based on these works [1][4][6], Le *et al.* reviewed and integrated previously reported ASs. Finally, they described 11 ASs and classified them into four categories. More importantly, all these 11 ASs can be automatically detected by the proposed ARCAN and the corresponding detection algorithm [5].

As mentioned by Fontana *et al.* [2], even the detection tools can induce false-negative AS instances, which require additional effort and experience as well as a better understanding of the smells to avoid false-positives instances. As reported in our previous study [6], SoF, as an online community for sharing knowledge, can provide a rich knowledge and experience about AS understanding, detection, and refactoring.

B. Mining Information from Stack Overflow

Many studies have been performed to automatically mine SoF data from different perspectives using ML or NLP. Karthik *et al.* developed an automated mechanism using an unsupervised deep learning based method to identify three different types of compatibility relations between components from the unstructured text on Q&A site postings [13]. Beyer *et al.* built a classification model using ML algorithms (Random Forest and Support Vector Machines) to automatically classify SO posts into seven question categories [14]. In another study, Borg *et al.* made an attempt to use Active Learning and an SVM classifier for mining performance discussions on SoF posts with two alternating annotators [10]. Zhang *et al.* investigated an approach using NLP and sentiment analysis techniques to automatically extract problematic API features from SoF posts [15]. Furthermore, Ahasanuzzaman *et al.* presented a supervised learning approach using Conditional Random Field (CRF) to identify API issue-related sentences in an SoF post [9].

However, none of the works above focuses on the classification of AS posts in SoF. Inspired by the existing works, we plan to use ML and NLP techniques to automatically identify AS discussions on SoF posts.

III. RESEARCH DESIGN

In this section, we describe the goal and Research Questions (RQs), and the method used in the study design.

A. Research Questions

The objective of our work is to provide an approach to automatically mine and identify AS discussions from textual artefacts. To achieve this objective, we define the following three RQs and explain their rationale.

RQ1: Which technique (BoW vs. TF-IDF vs. Word2Vec) performs best in the feature extraction step when identifying AS-related posts from SoF?

Rationale: In text identification tasks, Text Data Vectorization is an essential process that converts text data into a set of real numbers (a vector). We use four well-performed vectorization methods for extracting textual features: BoW, TF-IDF, and Word2Vec. BoW (Bag of Words), as one of the most commonly used traditional vector representations, links each word or n-gram to a vector index that represents weather word occurs in a document or not. TF- IDF is a statistical measure used to evaluate the importance of a word to a document in a collection of documents or corpus. Word2Vec, introduced by Google, is a predictive embedding model to produce a distributed representation of words with word semantics [16]. There are two main models of Word2Vec - Continuous Bag of Words (CBOW) and Skip-Gram. Employing different vectorization techniques may affect the final performance of classifiers. Therefore, the aim of the question is to determine the vector representation which can achieve the best performance when identifying AS discussions from SoF post.

Step 1: Training data collection

Step 2: Preprocess Texts
(Remove useless characters and stopwords)

Step 3: Extract Features
(BoW, TF-IDF, and Word2Vec)

Step 4: Train Classifiers
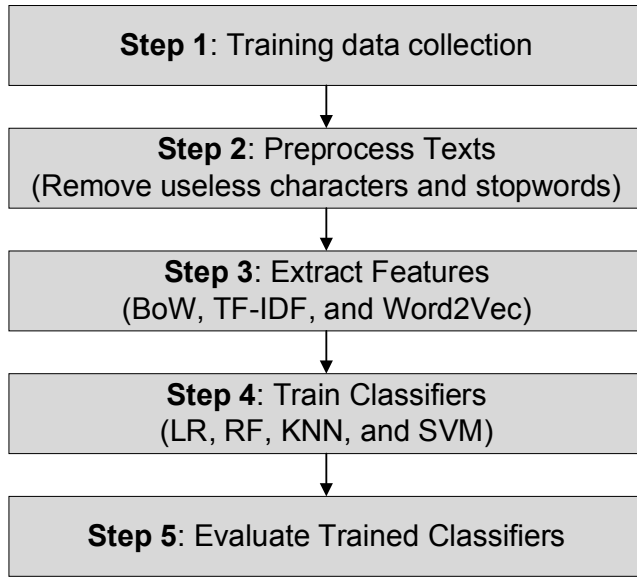(LR, RF, KNN, and SVM)

Step 5: Evaluate Trained Classifiers

Fig. 1. The overall process of classifying AS-related posts

RQ2: Which classification algorithm (i.e., LR, SVM, KNN, and RF) performs best when identifying AS-related posts from SoF?

Rationale: Various text classifiers have been employed in the literature based on ML techniques, probabilistic models, etc. Different classification methods may lead to differences in classification performance when coping with text identification tasks [18]. We use four commonly used classification algorithms (i.e., Random Forest (RF), KNN, SVM, and Logistic Regression (LR)) for classifying textual artifacts in software development (e.g., [17]) and compare their performance in our AS-related posts identification tasks. By answering this question, we can determine the kind of classification algorithm that can perform best in automatically identifying AS-related posts.

RQ3: What is the best configuration to automatically identify AS-related posts from SoF by combining different feature extraction techniques and classification algorithms?

Rationale: Different performances can be achieved by using different feature extraction techniques and classification algorithms. We used three feature extraction techniques and four classification algorithms, which results in 12 classifier configurations. The configuration with best overall performance may not be combined by the technique and algorithm which achieve the best performance in each separate step. Therefore, the aim of this RQ is to analyze the performance of classifiers with different configurations and determine the best configuration (that achieves the best performance) for identifying AS-related posts.

B. Study Design

In this section, we introduce how we performed an experiment to identify AS discussions from SoF posts using automatic techniques. As shown in Fig. 1, the study design consists of five steps. In the following, we describe the details of the tasks conducted in each step.

Step 1: Training data collection. The input of the classification process of AS-related posts in Fig. 1 is SoF posts. We used a set of inclusion and exclusion criteria (enlisted in TABLE I) [3] for manually selecting and labeling AS posts. If a post contained at least one sentence which met one criterion, we labeled this post as AS-related. Furthermore, we used the criterion C7 to label AS-unrelated posts. Each post was independently analyzed and manually labelled as AS-related or unrelated by two of the authors. To mitigate unconscious bias, any disagreements of the labelling results were discussed and resolved with the help of a third author.

We selected and labelled 395 posts (208 AS-related posts and 187 AS-unrelated posts) according to the above criteria. A few examples of AS-related and AS-unrelated posts are provided in TABLE II. This dataset was then split into two parts: (a) 90% of the posts as the training data set, and (b) 10% of the posts as the testing data set. To support preprocessing in the next step, we used a web crawler to collect and store the content of the labelled SO posts by using their URLs. The crawler works in four steps: (1) extract the URLs of the labelled posts, (2) remove useless URL information in the posts (e.g., *http://www.xxx.org/*), (3) parse and extract post information (i.e., titles, questions, and answers), and (4) save the post information into a CSV document. For the replicability of our experiment, the dataset of our experiment has been made available online[1].

TABLE I.   Criteria for Labelling AS-Related Sentences

	Criterion ID	Description
Criteria for labelling AS-related sentences	C1: Description of Ass	Descriptions of ASs by practitioners based on their understanding
	C2: Cause of ASs	Causes that lead to ASs
	C3: Approach for detecting and refactoring ASs	Methods used to detect/refactor specific ASs (e.g., machine learning based approaches)
	C4: Tool for detecting and refactoring ASs	Tools for detecting and refactoring specific ASs (e.g., source code analysis tools to identify dependency cycles)
	C5: Impact of Ass	Impact of ASs on software development (e.g., understandability, testability, extensibility, and reusability)
	C6: Challenge of detecting and refactoring ASs	Challenges identified in detecting and refactoring ASs
Criterion for labelling AS-unrelated sentences	C7: Not AS-related topic	The sentence does not describe ASs or the sentence topic is not about AS, for example sentences do not describe AS but only other types of smells (e.g., code smells).

Type	Example
AS-related Sentence	"*Is using a root persistent class or base persistable object an architecture smell?*"
	"*I think my architecture has kind of a smell to it: The webservice is acting as a proxy, collecting information from different sources.*"
	"*Message Bus and Message Based Architecture With Winforms/Desktop Application and Strategies/Policies for View/UI Logic*"
	"*What would be a nice architecture so I can pass information of eventual problems to a higher layer?*"
	"*Using a command architecture is a good idea, since this moves all business logic out of the controller, and allows you to add cross-cutting concerns without changes to the code.*"
AS-unrelated Sentence	"*There's a distinct smell of burned out circuits coming from my head, so forgive my ignorance.*"
	"*For some derived classes, I want to ensure that one of two overloaded abstract methods get overridden, but not both. Is this possible?*"
	"*Judging by the quality of the pixels that have been restored properly, the network architecture seems to be fine for this task.*"
	"*I came across the Open Test Architecture API and was wondering if there are any good Python or java examples for the same that I could see.*"

Step 2: Data Preprocessing. Data preprocessing eliminates the terms or characters in the training posts that are unnecessary to train classifiers for identifying AS discussions, which is composed of 2 steps: (1) **Removing useless characters**. Since the posts were crawled from Stack Overflow website which is formed in HTML 5, most posts contain some useless punctuations like "…" and escape characters like "/n" or "/r". Those characters provide invalid information in semantic parsing, so we removed those useless characters. (2) **Processing stop words**. Referring to the original idea of TF-IDF, daily language interaction like Q&A posts from Stack Overflow can be filled with common words such as auxiliary verbs, conjunctions and articles. Removing stop words can reduce the noise in natural language, because these words also lack the distinguishing feature for training classifiers. To remove those meaningless words, we apply the default stop words list in Natural Language Toolkit (NLTK) package.

Note that, in our study, we did not apply stemming and lemmatizing but stop words removing to preprocess the training data, because stemming and lemmatizing may change the meaning of the text. Moreover, prior research shows that the text preprocessing method with No Stemming and Lemmatization performs best when preprocessing posts to identify decisions from textual artifacts in software development [17].

Step 3: Feature extraction. The aim of feature extraction is to transform preprocessed documents into numerical vector representations for classifiers regardless of the vectorization method. In our work, we applied three feature extraction techniques, i.e., BoW, TF-IDF, and Word2Vec to calculate the feature value of each post.

BoW is a commonly used traditional feature extraction technique. A corpus is created consisting of every unique word across the documents. Then each word is converted into the corresponding vector by counting the occurrence of a word in a document. While BoW is simple to understand and implement, it lacks the ordering of words, which leads to loss of contextual information and word meaning in the document (semantics).

TF-IDF is a basic vectorization method which considers frequencies of words in one document and words relationships among documents. However, TF-IDF is also unable to capture the word meaning. TF-IDF produces vectors based on frequencies of words in one document (TF) and the weight of rare words across all documents (IDF). TF-IDF used in this study is defined as in Formula (1):

$$w_{i,j} = tf_{i,j} \times \log(\frac{N}{df_i}) \tag{1}$$

$tf_{i,j}$ represent the count of a term "I" in a document "j", N represents the number of total documents in the corpus, and df_i represents the number of total documents containing the term i.

Word2Vec is a word embedding method which produces dimensional numerical representations of words and more syntactic information than BoW and TF-IDF. These two models of Word2Vec: Continuous Bag of Words (CBOW) and Skip-Gram. The CBOW model obtains word representations by predicting the current word based on its context (surrounding words). Contrary to the CBOW model, The Skip-Gram learns the embedding by predicting the surrounding words (context) given a current word. CBOW is several times faster than Skip-Gram, while Skip-Gram performs better for even rare words or phrases than CBOW.

Step 4: Classifier Training. After transforming the collected SoF into numerical vectors, we used the extracted features to train four algorithms, i.e., LR, SVM, RF, and KNN, to automatically identify AS-related posts. RF, KNN, and SVM are three non-parametric classifiers. In contrast, LR is a parametric classifier and faster and simpler classification method than the other three. We used the implementation of these four classifiers in the scikit-learn Python package.

Step 5: Performance Evaluation. To evaluate the performance of three feature extracting methods and four classifiers, we used four common measures: accuracy, precision, recall, and F1-score. We used Formula (2), (3), (4), and (5) to calculate accuracy, precision, recall, and F1-score, respectively.

$$Precision = \frac{TP}{TP+FP} \tag{2}$$

$$Accuracy = \frac{TP+TN}{TP+TN+FP+FN} \tag{3}$$

$$Recall = \frac{TP}{TP+FN} \tag{4}$$

$$F1-score = \frac{2*Precision*Recall}{Precision+Recall} \tag{5}$$

Among these formulas, True Positive (TP) denotes the number of correctly identified AS-related posts by classifiers. False Positive (FP) represents the number of AS-unrelated posts that are incorrectly identified as AS-related posts by classifiers. False Negative (FN) indicates the number of AS-related posts that are incorrectly labelled as AS-unrelated posts by classifiers. True Negative (FN) shows the number of correctly identified AS-unrelated posts by classifiers.

Therefore, in our context, precision is used to measure the exactness of prediction set and represented as the ratio of posts correctly identified as AS-related posts to all posts identified as AS-related posts. Recall is the fraction of all AS-related posts correctly demarcated. F1-score is a combination of precision and recall. We consider precision and recall equally important. F1-score is calculated by the harmonic mean of precision and recall. Accuracy is the ratio of correctly identified posts, including AS-related posts (TP) and AS-unrelated posts (TN) to all identified posts.

IV. Results and Analysis

As described in Section III, we conducted an experiment with 3 (three feature extraction techniques) × 4 (four classification algorithms) = 12 configurations. We calculated these metrics (i.e., precision, recall, F1-score, and accuracy) to evaluate the performance of each configuration for identifying AS-related posts in SoF. In this section, we present and analyze the results to answer the research questions.

A. Results and Analysis of RQ1

To answer RQ1, we calculated the average of the results obtained by three feature extraction techniques to observe the impact of different techniques on the performance of classifiers. TABLE III shows the performance of ASs classifiers by employing three feature extraction techniques (i.e., TF-IDF, BoW, Word2vec). The highest accuracy, precision, recall, and F1-score values of AS discussion classification across all techniques are highlighted in boldface. we found that TF-IDF technique can achieve the highest average precision (0.620), accuracy (0.619), and F1-score (0.714), and the BoW technique can achieve the best Recall (0.798). One important observation is that the recall value for the TF-IDF is very poor. It can also be found that the performance of the Word2Vec model is poor, which is consistent with the result observed by Li *et al.* [17]. The results indicate that the TF-IDF technique can perform better than the other techniques (i.e., Bow, Word2Vec) and extract meaningful AS discussions from SoF posts.

TABLE III. Average Results of Different Feature Extraction Techniques for Identifying Architecture Smell Discussions

Technique	Accuracy	Precision	Recall	F1-score
TF-IDF	**0.619**	**0.620**	0.698	**0.714**
BoW	0.516	0.528	**0.798**	0.629
Word2vec	0.556	0.563	0.631	0.589

B. Results and Analysis of RQ2

To answer RQ2, we calculated the average of the results obtained by the four classification algorithms to observe the impact of different algorithms on the performance of classifiers (i.e., the results of AS discussions classification). TABLE IV

presents the performance of ASs classifiers by employing the four classification algorithms (i.e., LR, SVM, RF, KNN). The highest precision, recall and F1-score of AS identification across all classification algorithms are highlighted in boldface. We observed that the highest average accuracy (0.600), recall (0.921) and F1-score (0.709) are achieved by the SVM-based classifier, and the highest average precision (0.589) are achieved by KNN-based algorithm. Moreover, SVM can achieve the second-highest average precision (0.579). The performance of KNN-based classifier is slightly worse than SVM, with an accuracy of (0.592), a recall of (0.762), and an F1-score of (0.620). DT-based classifier got the lowest average precision (0.583), recall (0.577), and F1-score (0.577). These results indicate that the SVM-based classifier performs the best in term of the overall performance and can be used as the most suitable classification algorithm in the classifier training step when automatically identifying ASs from SoF compared to other three classification algorithms. This conclusion is consistent with the findings reported in [17].

TABLE IV. Average Results of Different Classification Algorithms for Identifying Architecture Smell Discussions

Algorithm	Accuracy	Precision	Recall	F1-score
LR	0.542	0.560	0.571	0.566
SVM	**0.600**	0.579	**0.921**	**0.709**
RF	0.525	0.553	0.603	0.570
KNN	0.592	**0.589**	0.762	0.620

C. Results and Analysis of RQ3

To answer RQ3, we calculated and compared the precision, recall, and F1-score of each of 12 configurations to analyze how accurately we can automatically identify AS-related posts from SoF. TABLE V presents the performance of 12 ASs classifiers by employing four classification algorithms (i.e., LR, SVM, RF, KNN) across three feature extraction techniques. The highest precision, recall, and F1-score values for ASs classifiers across all configurations are highlighted in boldface. It can be found that the highest accuracy (0.650), precision (0.636), recall (1.000), and F1-score (0.731) can be achieved with the combination of three feature extraction techniques and four classification algorithms. We can also find that there are two configurations with an F1-score more than 0.7 and accuracy greater than 0.625. It is also worth noting that these two configurations employ SVM. The highest precision (0.650) and F1-score (0.731) are achieved by the combination of Word2Vec with SVM, which confirms the result of RQ2 that SVM-based classifier performs the best for automatic classification of AS-related posts. One important observation is that although the TF-IDF technique can achieve the best performance in the step of text feature extraction when identifying AS-related posts, the combination of Word2vec and SVM can achieve the best overall performance. These results indicate that each configuration presents a diverse performance. Moreover, the configuration with the best performance is not simply combined by the technique and algorithm which achieves the best performance in each separate step, respectively. Overall, with an accuracy of 0.650, a precision of 0.613, a recall of 0.905, and an F1-score of 0.731, the configuration with a combination of Word2Vec and SVM can achieve the best performance for

identifying AS-related posts, and can be used by developers and researchers to collect AS discussions from SoF.

TABLE V. Evaluation Results of Combining Different Feature Extraction Techniques and Classification Algorithms for Identifying Architecture Smell Discussions

Technique	Algorithm	Accuracy	Precision	Recall	F1-score
TF-IDF	LR	0.625	**0.636**	0.667	0.651
	SVM	**0.625**	0.600	0.857	**0.706**
	RF	0.600	0.619	0.619	0.619
	KNN	0.625	0.625	0.714	0.619
BoW	LR	0.525	0.545	0.571	0.558
	SVM	0.525	0.525	**1.000**	0.689
	RF	0.450	0.484	0.714	0.577
	KNN	0.575	0.559	0.905	0.691
Word2Vec	LR	0.475	0.500	0.476	0.488
	SVM	**0.650**	0.613	0.905	**0.731**
	RF	0.525	0.556	0.476	0.513
	KNN	0.575	0.583	0.667	0.622

V. Conclusions and Future Work

Architecture Smells (ASs), as a type of technical debt, have been attaining importance in recent years since they may significantly depreciate software quality. Detecting and refactoring ASs correctly require appropriate architectural knowledge. The online communities, such as Stack Overflow (SoF), contain a wealth of latent information expressed in natural language and become a valuable source of information for sharing knowledge of ASs. But manually gathering AS discussions from the online communities is a time-consuming and labor-intensive task. To address this problem, we have proposed a solution to automatically identify AS related posts from SoF using 12 different configurations of feature extraction techniques and classification algorithms.

We first created a dataset from SoF consisting of 208 AS-related posts and 187 AS-unrelated posts and manually labelled AS-related sentences for training classification models across the combinations of the three feature extraction techniques and four classification algorithms. We evaluated and discussed the performance of the 12 combinations by calculating four metrics: accuracy, precision, recall, and F1-score. The results from our experiment show that: (1) the TF-IDF technique performs best when extracting text features to identify ASs; (2) the SVM based classifier achieves the best overall performance regarding accuracy and F1-score of automatic AS discussions classification; and (3) the SVM algorithm with Word2Vec outperforms the other combinations when automatically identifying AS-related posts.

These results provide several implications for our future research including: (1) validating the best configurations in other online sources of textual information such as developer mailing lists and issue tracking systems, (2) to identify AS-related discussions at the sentence level, which provides more focused content about AS discussions, (3) employing multi-classification algorithms (e.g., Softmax) to automatically classify different types of AS-related discussions from textual artefacts, and (4) automatically extracting AS information to enrich practioners' knowledge and experience of detecting and refactoring ASs, and evaluating whether and how the AS information can improve detecting and refactoring ASs.

References

[1] Garcia, J., Popescu, D., Edwards, G. and Medvidovic, N., 2009. Identifying architectural bad smells. In: Proceedings of the 13th European Conference on Software Maintenance and Reengineering (CSMR). Kaiserslautern, Germany. pp. 255-258.

[2] Fontana, F.A., Pigazzini, I., Roveda, R. and Zanoni, M., 2016. Automatic detection of instability architectural smells. In: Proceedings of the 32nd IEEE International Conference on Software Maintenance and Evolution (ICSME). Raleigh, NC, USA, pp. 433-437.

[3] Tian, F., Liang, P. and Babar, M. A., 2019. How developers discuss architecture smells? an exploratory study on Stack Overflow. In: Proceedings of the 16th International Conference on Software Architecture (ICSA). Hamburg, Germany, pp. 91-100.

[4] Mo, R., Cai, Y., Kazman, R. and Xiao, L., 2015. Hotspot patterns: the formal definition and automatic detection of architecture smells. In: Proceedings of the 12th Working IEEE/IFIP Conference on Software Architecture (WICSA), Montreal, QC, Canada, pp. 51-60.

[5] Le, D. M., Link, D., Shahbazian, A., & Medvidovic, N. 2018. An empirical study of architectural decay in open-source software. In: Proceedings of the 15th IEEE International Conference on Software Architecture (ICSA), Seattle, WA, USA, pp. 176-185.

[6] Samarthyam, G., Suryanarayana, G. and Sharma, T., 2016. Refactoring for software architecture smells. In: Proceedings of the 1st International Workshop on Software Refactoring (IWoR). Singapore, Singapore, pp. 1-4.

[7] Grant, S. and Betts, B., 2013. Encouraging user behaviour with achievements: an empirical study. In: Proceedings of the 10th Working Conference on Mining Software Repositories (MSR). San Francisco, CA, USA, pp. 65-68.

[8] Tahir, A., Yamashita, A., Licorish, S., Dietrich, J. and Counsell, S., 2018. Can you tell me if it smells?: a study on how developers discuss code smells and anti-patterns in Stack Overflow. In: Proceedings of the 22nd International Conference on Evaluation and Assessment in Software Engineering (EASE), Christchurch, New Zealand, pp. 68-78.

[9] Ahasanuzzaman, M., Asaduzzaman, M., Roy, C.K. and Schneider, K.A., 2020. CAPS: a supervised technique for classifying Stack Overflow posts concerning API issues. Empirical Software Engineering, 25: 1493-1532.

[10] Borg, M., Lennerstad, I., Ros, R. and Bjarnason, E., 2017. On using active learning and self-training when mining performance discussions on Stack Overflow. In: Proceedings of the 21st International Conference on Evaluation and Assessment in Software Engineering (EASE), Karlskrona Sweden, pp. 308-313.

[11] Lippert, M. and Roock, S., 2006. Refactoring in large software projects: performing complex restructurings successfully. John Wiley & Sons.

[12] Rizzi, L., Fontana, F.A. and Roveda, R., 2018. Support for architectural smell refactoring. In: Proceedings of the 2nd International Workshop on Refactoring (IwoR), Montpellier, France, pp. 7-10.

[13] Karthik, S. and Medvidovic, N., 2019. Automatic detection of latent software component relationships from online Q&A sites. In: Proceedings of the 7th International Workshop on Realizing Artificial Intelligence Synergies in Software Engineering (RAISE), Montreal, QC, Canada, pp. 15-21.

[14] Beyer, S., Macho, C., Di Penta, M. and Pinzger, M., 2018. Automatically classifying posts into question categories on Stack Overflow. In: Proceedings of the 26th International Conference on Program Comprehension (ICPC), Gothenburg, Sweden, pp. 211-221.

[15] Zhang, Y. and Hou, D., 2013. Extracting problematic API features from forum discussions. In: Proceedings of the 21st International Conference on Program Comprehension (ICPC), San Francisco, CA, USA, pp. 142-151.

[16] Mikolov, T., Chen, K., Corrado, G. and Dean, J., 2013. Efficient estimation of word representations in vector space. arXiv preprint arXiv:1301.3781.

[17] Li, X., Liang, P. and Li, Z., 2020. Automatic identification of decisions from the Hibernate developer mailing list. In: Proceedings of the 24st International Conference on Evaluation and Assessment in Software Engineering (EASE), Trondheim, Norway, pp. 51-60.

[18] Ikonomakis, M., Kotsiantis, S. and Tampakas, V., 2005. Text classification using machine learning techniques. WSEAS Transactions on Computers, 4(8): 966-974.

Exploring CQA User Contributions and Their Influence on Answer Distribution

Yi Yang[1, 2, 3], Xinjun Mao,[1, 3*], Zixi Xu[1, 3], Yao Lu[1, 3]

[1]College of Computer, National University of Defense Technology, Changsha, China
[2]College of Information Science and Engineering, Hunan Womens' University, Changsha, China
[3]Key Laboratory of Complex Systems Software Engineering, National University of Defense Technology, Changsha, China

Abstract—**In community question answering (CQA) sites like Stack Overflow (SO), users provide contributions driven by community incentives and self-motivation, which gives rise to micro activities of individual user and macro emergence of community. According to our empirical research, the emergence of answer distribution in SO is of concern, about 90% questions have no more than two answers and almost 40% of questions have no accepted answer due to lack of alternative acceptable answers. In this paper, we explore CQA user contributions by considering both the external incentives of community and internal motivations of users, and study why CQA users contribute and how they influence the macro emergence of answer distribution. We present CQA community model based on normative multi-agent system approach, in which the users are modeled as agents and the community incentives as norms. The internal motivations are studied based on self-determination theory. The paper further analyzes how the internal and external factors together influence the activities of agents and ultimately the answer distribution of the whole community. We conduct experiments based on a simulation system and the SO dataset to validate the effectiveness of our proposed model. The results show that our model can reproduce the emergence that well matches up with the observation on real community.**

Keywords-NorMAS; BDI; User Contribution; Emergence; Incentive Mechanism; Self-Determination Theory

I. INTRODUCTION

Community Question Answering (CQA) sites such as *Stack Overflow* (SO), *Yahoo!Answers*, and *Quora* are a type of knowledge sharing communities, in which users contribute their knowledge in term of various activities such as asking, answering and voting [1]. These activities give rise to various macro phenomena, i.e. emergence [2]. For example, based on 2,509,027 SO questions from January 2018 to September 2019, we can observe the emergence of answer distribution to questions: about 90% SO questions have less than two answers and almost 40% of questions have no accepted answer. Such observation on the emergence is of great concern, as they may affect the prosperity of the community. It is significant for managers of CQA sites to understand why the macro emergence occurs and how to improve the emergence.

To address the issue, we need to explore the following three problems of CQA: user contribution motivation, user contribution decision, and the influence of user contribution on the emergence of CQA community. User contribution motivation refers to why users in CQA contribute their knowledge. Numerous scholars have conducted research on CQA user contribution motivations. For example, Lou [3], Jin [4], and Chen [5] used online surveys and statistical analysis to study the influence degree of various motivations to CQA user contributions. However, these researches lack of study and analysis on the factors that govern and drive users to contribute.

User contribution decision refers to how users select and take actions to participate in contributions in CQA community. Many of existing methods are equation-based to model user behavior. For example, Anderson [6], Gao [7] applied a game-theoretic model to analyze user behavior decision in CQA community. In essence, the equation-based methods are difficult to capture the autonomy feature of users [8][9] and the community incentives that govern users' decisions on their contributions.

For the influence of user contributions on the emergence of community, most of existing methods are normally based on the statistical analysis to explore what gives rise to the emergence. For example, Srba [10] applied statistical analysis based on the dataset of CQA community to conclude that low-quality user contributions lead to user churn. However, these studies can not naturally reveal the community emergence that results from the user contributions and their interactions.

In this paper we present an approach based on Normative Multi-Agent System (NorMAS) and Self-Determination Theory (SDT) to model CQA communities and examine how individual users in communities provide contributions and influence the answer distribution. The remaining sections are organized as following. The next section discusses the related works. Section III describes the NorMAS-based model of SO community. Thereafter, Section IV details contribution mechanism analysis of SO users based on BDI and SDT theory. Section V describes our experiment and result analysis. Finally, Section VI concludes the contributions of this paper and points out the future research direction.

II. RELATED WORK

NorMAS has been widely used to model complex social systems [11]. For example, Mao et al. [12] presented an adaptive casteship mechanism to model and design adaptive multi-agent systems. Mastio [13] et al. applied multi-agent system approach to simulate and tackle traffic management. As a classical architecture, BDI (belief-desire-intention) [14] is

often applied to model and simulate the rule-based reasoning scenario. For instance, Yang et al. [15] considered a robot software as a multi-agent system and employed multiple interacting agents with different roles cooperate to achieve software functionalities. Yan et al. [16] proposed a BDI agent-based method to simulate suppliers' belief, reasoning processes, deception intention and their behavior. Moreover, they provided buyers with inspection suggestions to detect suppliers' falsified test results. In this work, we apply BDI model to describe CQA users and their reasoning.

III. NorMAS model of CQA communities

In this section, we present the normative MAS model of CQA communities, and illustrate the model with the sample of SO community. We define the NorMAS model of SO community as a 3-tuple

$$CQA\_MAS =< MAS, UGC, norm > . \quad (1)$$

- *MAS* represents a set of the agents.

- $UGC = UGC_q \bigcup UGC_a \bigcup UGCv$ represents the user generated contents (UGC), i.e. questions, answers, and votes. UGC_q , UGC_a , and UGC_v represent the sets of questions, answers, and votes, respectively. There exists three kinds of relations :

  (1) $R_1 = \{ < q,a > | q \in UGC_q \wedge a \in UGC_a \}$ represents the relation between questions and answers;
  (2) $R_2 = \{ < q,v > | q \in UGC_q \wedge v \in UGC_v \}$ represents the relation between questions and votes;
  (3) $R_3 = \{ < a,v > | a \in UGC_a \wedge v \in UGC_v \}$ represents the relation between answers and votes.

- *norm* represents the reputation incentive mechanism in the community.

Figure 1. The NorMAS model of SO community

Fig. 1 depicts the NorMAS model of SO community. Governed by the norms in *norm*, agents in *MAS* autonomously ask, answer, or vote based on the current state of their contributed knowledge *UGC*, i.e. the number of questions, answers, votes, and their relations. In turn, their contribution behavior change the state of the contributed knowledge *UGC*.

A. The model of CQA user

In this paper, we employ BDI model to represent agents in *MAS*. Hence, we define agent as a 5-tuple

$$agent =< A, REP, BEL, DES, INT > . \quad (2)$$

- *A={ask, answer, upvote, downvote}* defines the action set of agents.

- *REP* is an agent's reputation point represented as integer.

- *BEL, DES, INT* are an agent's belief, desire, and intention, respectively.

B. The model of community incentives

To achieve system goals, CQA communities generally design and adopt incentive mechanisms to stimulate and govern users' behavior. For example, the reputation incentive mechanism in SO community (see Table I) describes how user can obtain their reputation points in SO. For example, the third row of Table I means that when an answer is upvoted, the answer contributor will be rewarded with 10 reputation points. In line with the incentive mechanism of SO, we do not take into account the rule "Upvote an answer".

TABLE I. Reputation rules in SO

Rule	Action	Reputation change
1	Question is upvoted	+5 (to asker)
2	Question is downvoted	-2 (to voter)
3	Answer is upvoted	+10 (to answerer)
4	Answer is downvoted	-2 (to answerer)
5	Downvote an answer	-1 (to voter)

We employ the norms of NorMAS to represent the reputation incentive mechanism of SO. Here, we define $UGC_o = UGC_q \bigcup UGC_a$ as the set of questions and answers. We define $A_v = \{ upvote，downvote \}$ to represent *agents'* vote actions. The reputation incentive mechanism of SO can be defined as

$$norm: UGC_o \times A_v \rightarrow I \times I. \quad (3)$$

Here, *I* represents the reputation points. The function *norm* represents that when an agent votes another agent's question or answer, the two agents will be gave a certain reputation points, respectively.

IV. Contribution Mechanism Analysis of SO users

In this section, we analyze SO user contribution mechanism from two aspects: user contribution motivations and user contribution decision.

A. Contribution motivation analysis of SO users

Why do users contribute to CQA community? To answer this question, we adopt a combination of self-determination theory and online survey to analyze user contribution motivations. Self-determination theory [17] divides users

behavioral motivations into five types of regulations: external regulation, introjected regulation, identified regulation, integrated regulation, and intrinsic regulation. Moreover, to obtain the contribution motivations of SO users, we investigated some SO users in January 2018. A total of 656 valid feedback samples were obtained. The questionnaire is in the form of five-level Likert scale. The respondents can choose one of the five answers: strongly disagree, disagree, indifferent, agree and strongly agree. The weights of the answers are 5, 4, 3, 2, 1, respectively. Based on the characteristics of the five motivations summarized by Ryan et al. [18] and our survey result, the motivations of SO users are classified into four types:

- Gaining reputation is an introjected regulation. When SO users ask or answer questions, they have a chance to gain their reputation points through received votes.

- Gaining privilege is an identified regulation desire for personal importance in the community. SO users are gave some privilege to manage the affairs of the community based on their reputation points. Privileges have an incentive effect on users' behavior.

- Returning favor is an integrated regulation to be competent or synthesis with self, which is a motivation not related to external reward. For SO users, if their questions have been answered by other users, they may have the motivation of giving back to community.

- Helping others is an intrinsic regulation for inner satisfaction. For SO users, when the questions of other users are not answered, they have the motivation to help others.

The first two desires are from the external incentives of community and the last two are internal motivations of individual users. Here, we do not consider the external regulation and the desire related to money and material.

B. Contribution decision of SO users

Here, we apply BDI model to analyze the contribution decision of SO users. The contribution decision process consists of four steps: belief update, desire generation, intention filter, and intention selection. Agent first updates its belief according to current belief and contributed knowledge information. Then agent generates its desire based on current belief related to current contributed knowledge and intention. Thereafter, agent filters intention based on current belief and desire. Finally, it selects the intention with the maximum of intensity to execute.

1)   The update of agent's belief

In the context of CQA, an agent's belief is the cognition of self, the state of contributed knowledge, and norms. Combining the above analysis of user contribution motivation, we consider four types of cognition and define the belief of agent as

$$BEL =< RREP, PRI, RF, HO > . \qquad (4)$$

- *RREP* is agents' cognition of probably obtained reputation points.

- *PRI* is agents' cognition of privilege corresponding to reputation points. There is a simple correspondence between privilege and reputation points.

- *RF* is agents' cognition of giving back to community.

- *HO* is agents' cognition of helping others.

Based on the perception of community information and their current belief, SO users update their belief. The belief update function can be defined as

$$brf： BEL \times UGC \rightarrow BEL'. \qquad (5)$$

Here, *UGC* is the information of the contributed knowledge. The updated rules for each belief component are as follows.

- *RREP* can be updated by the probably received upvotes *VOTE* and the action *A*.

$$fr : VOTE \times A \rightarrow RREP. \qquad (6)$$

We divide SO users into four groups based on their reputation points: newcomer (points<10), normal (10≤points≤999), established (1000≤points≤19999), and trusted (points≥20000). SO users with more reputation points usually have a stronger capability to gain upvotes.

- PRI represents the probably obtained privilege. Because privilege are described by some words, we employ a reputation point transform function γ to get the intensity of privileges.

$$\gamma： REP \rightarrow PRI. \qquad (7)$$

- RF can be updated by the equation

$$RF = \begin{cases} 1, & iff \ na = true. \\ 0, & iff \ na = false. \end{cases} \qquad (8)$$

Here, *na* indicates whether an agent's questions have been answered by a community. If answered, it will give back to the community.

- HO can be updated by the equation

$$HO = \begin{cases} 0, iff \ ad = true. \\ 1, iff \ ad = false. \end{cases} \qquad (9)$$

Here, *ad* indicates if a question of the community has been answered. If answered, agents will generate the belief that they don't need to help others.

In conclusion, the update rule of the intensity of the belief of SO users can be defined as an vector:

$$\varphi_{BEL} =< RREP, PRI, RF, HO > . \qquad (10)$$

2)   The generation of agent's contribution desire

According to the analysis of user contribution motivation, we believe that agents also have four types of desire: gaining reputation, gaining privilege, giving back to community, and helping others. Agent may be affected by all the four kinds of desires at the same time. According to our survey, the intensity

of each desire for different types of users are different. Therefore agents need to update their desire intensity as their reputation points change. The update rule for the overall desire intensity of agent is defined as

$$\sigma_{Des} = \psi \times \sum_{i=1}^{4}(\mu_i \sigma_{Des_i}). \qquad (11)$$

Here, ψ is the overall ability of agent desire to drive agent behavior ($\psi \in [0,1]$); μ represents the weight of each desire of agents; σ_{Des_i} is the intensity of the i-th desire ($i \in [1,4]$). The values of μ and σ_{Des} are from statistics of our questionnaires.

3) The filter of agent's intention

Intention represents the behavior decision made by agents based on their own belief to achieve their desire. Agents possess four candidate actions: ask, answer, upvote, and downvote. Agents can filter their intention based on their current belief, desire, and intention. For example, if an agent wants to answer questions to gain reputation, it will filter some questions whose answers are difficult to obtain upvotes. Thus the intention filter function can be defined as

$$filter: BEL \times DES \times INT \to INT'. \qquad (12)$$

More specifically, an agent's choice of an action depends on the intensity of its intention. And the probability of choosing an intention is determined by the intensity of the desire and belief related to the intention. We define the filter rule of agents' intention as

$$p(a) = \sigma_{Des} \times \varphi_{Bel}. \qquad (13)$$

Here, $p(a)$ represents the probability of the candidate action a ($a \in \{ask, answer, upvote, downvote\}$).

4) The selection of agent intention

Agents select the intention with the maximum of $p(a)$ to perform. The selection rule of agent intention is defined as

$$a = \{int \mid max(P(int)), int \in INT\}. \qquad (14)$$

V. EXPERIMENT AND RESULT ANALYSIS

In order to investigate the influence of CQA user contributions on the emergence of answer distribution, we develop a simulation system to simulate massive users' contribution behavior to reproduce the emergence. If our simulation system can reproduce the emergence of answer distribution well, it will show that the proposed approach can well explain CQA user contributions and their influence on the emergence of answer distribution.

To achieve this goal, we first design a set of criteria to evaluate the recurrence of community emergence. Then, we collect the data of SO and investigate user motivations to initialize SO users. Finally, we run the simulation system on the dataset to reproduce the emergence of answer distribution. We compare the reproduced emergence on our simulation system with the observation of SO community based on the proposed criteria. The results will show whether the proposed model can effectively explain the CQA user contributions and their influence on the emergence of answer distribution.

A. Evaluation criteria

In this work, we adopt *PCC* (Pearson Correlation Coefficient) [19] and *MRE* (Mean Relative Error) [20] to evaluate the emergence recurrence effect of the simulation system. Given X and Y represent a real data and a simulation data, respectively. $\overline{X}$ and $\overline{Y}$ are the averaged value of X and Y, respectively. n is the number of samples. The *PCC* of X and Y is defined as

$$\rho_{X,Y} = \frac{\sum_{i=1}^{n}(X_i - \overline{X})(Y_i - \overline{Y})}{\sqrt{\sum_{i=1}^{n}(X_i - \overline{X})^2}\sqrt{\sum_{i=1}^{n}(Y_i - \overline{Y})^2}}. \qquad (15)$$

And, *MRE* is defined as

$$MRE = \frac{\left|\overline{Y} - \overline{X}\right|}{\overline{X}}. \qquad (16)$$

The greater the value of *PCC*, the better the emergence trend consistency between real SO community (X) and the simulation system (Y). The smaller the value of *MRE*, the smaller the value deviation between X and Y. According to the experience from statistics, when *PCC* is greater than 0.5, there exists a strong positive correlation between subjects.

Each comparison of the emergence between SO and the simulation system is comprised of multiple parts. For example, answer distribution consists of four types of answer count: 0, 1, 2, and more than 2. Therefore, it is necessary to modify the above equations as

$$\rho_{overall} = \frac{\sum_{i=1}^{m} w_i \rho_i}{m}. \qquad (17)$$

and

$$MRE_{overall} = \frac{\sum_{i=1}^{m} w_i MRE_i}{m}. \qquad (18)$$

Here, m is the component number of an emergence. we represents the weight of the i-th component. In this way, we give an overall score for the comparison of a certain emergence.

B. Data collection

We downloaded three versions of Stack Overflow datasets between 2017 and 2019. Based on the difference of the reputation points, the question number, the answer number, and the vote number of the users in different periods, we compute out the capability of gaining upvotes of different types of users. In addition, as described in Section IV, we investigated some SO users in January 2018 to obtain the contribution motivations of SO users and their motivation intensity distributions.

C. Result Analysis

We analyze 2,509,027 SO questions from January 2018 to September 2019 and observe the emergence of answer distribution. As shown in Fig. 2, about 90% questions in SO have no more than two answers.

Figure 2. The emergence of answer distribution.

Why are users reluctant to provide more answers for a question? To explain the phenomenon, we run the simulation system on So dataset from January 2018 to September 2019. We first count the distribution of answers of 0, 1, 2 and more than 2. Then we compare the captured emergence on the simulation system with the observation of SO community.

TABLE II.   THE SIMULATION EVALUATION OF ANSWER DISTRIBUTION

Index	PCC	MRE
0 answer	0.7653	0.0022
1 answer	0.9210	0.0013
2 answers	0.6901	0.0123
more than 2 answers	-0.9632	0.0956
overall	0.7729	0.0314

Table II presents our simulation performance of answer distribution. Except when the number of answers is more than 2, other simulation results are in line with our expectations. Other *PCC* of emergence simulation are greater than 0.65 and the overall *PCC* is 0.7729, which indicates that the simulation system successfully reflects the trend of the emergence of the real SO community. In addition, the deviations of the simulation is very small. The small deviation of the simulation indicates that the simulation system can accurately represent the real community's data. Moreover, Fig. 3 depicts the evolution of the trends of answer distribution in 12 months, which more intuitively illustrates a very good match between our simulation and observation of real SO community.

Figure 3. The evolution of the answer distribution in 12 months

To analyze the influence of external incentives of community and internal motivations of individuals on user contributions, we performed the influence analysis on answer distribution without the corresponding motivations. Fig. 4 depicts the evolution of answer distribution without external incentives in a year. We can observe that in the absence of external incentives from the community, the proportion of unanswered questions decreases, the proportion of questions with one answer does not almost change, and the proportions of questions with more than one answer increase. The result shows that in the absence of external incentives, users answer questions more based on the belief of helping others and giving back to the community. It leads to a decrease of the ratio of unanswered questions. Without the external incentives, users don't care about the influence of existing answer count to a question on their probably received upvotes. Hence, they will be willing to contribute more answers to the answered questions. This is why the proportions of answered questions increase.

Figure 4. The evolution of the answer distribution without external incentives of community in a year

Fig. 5 depicts answer distribution without internal motivations of users. We can observe that the proportion of unanswered questions continues to increase, whereas the other proportions of questions continue to decrease. The result shows that in the absence of internal motivations, users lose willingness to give back to the community and help other users, which makes most of them reluctant to answer any questions that have an answer or not.

Figure 5. The evolution of the answer distribution without internal motivations of users in a year

In conclusion, the answer provision of SO users for community questions are motivated by external incentives of community and internal motivations of individuals. On the one

hand, SO users behavior are affected by the community incentive mechanism. SO users answer questions based on the probably received upvotes. When a question has answers, SO users believe that their capability of gaining upvotes in answer will decrease. Thus they will be reluctant to provide more answers for a question that has answers. On the other hand, SO user behavior are affected by their internal motivations. If a question has been answered, SO users will not generate the desire to help others to answer the question. Both aspects may make users not to provide more answers for community questions. The similar behavior pattern of massive SO users eventually leads to the emergence of answer distribution. The result confirms that our approach can effectively explain CQA user contributions and their influence on answer distribution.

VI. CONCLUSIONS AND FUTURE WORKS

In this paper, we propose a NorMAS-based approach to explore CQA user contribution and their influence on answer distribution. The contributions of the paper are three-fold: (1) We put forward a NorMAS-based approach to model CQA communities under the governance of incentive mechanisms. The constructed model provides a natural description of CQA communities. (2) We apply self-determination theory and online survey to classify user contribution motivations. And we employ BDI theory to detail user contribution decision driven by community incentives and self-motivation to explain why users answer a question or not. (3) We develop a simulation system to simulate CQA communities and reproduce their emergence. Moreover, we design a set of criteria to examine the recurrence of the emergence in SO community. The results show that the proposed approach can effectively explain CQA user contributions and their influence on answer distribution.

The validity of our study may be threatened by the following aspects. First, we differentiate users' capability and desire based on the types of users grouped by their reputation points. The evaluation is not very accurate to some active users whose reputation points greatly fluctuate. Second, we only consider the individual behavior such as asking, answering, voting, and do not consider other behavior such as accepting an answer. Third, there is no consideration of the changes of SO reputation mechanism. Hence, the latter two may lead to a certain deviation in individual behavior analysis. In the future study, we will collect more data from CQA communities to improve the performance of our approach. More importantly, we will further consider how to use the proposed approach to improve community management and promote long-term prosperity of CQA communities.

ACKNOWLEDGMENT

REFERENCES

[1] S. Y. Lee, H. X. Rui, A. B. Whinston, "Is best answer really the best answer? the politeness bias," MIS Quarterly, vol. 43, 2019, pp. 579-600.

[2] John H. Holland, "Emergence: From chaos to order", Oxford University Press, 1998.

[3] J. Lou, Y.Fang, K. Lim, J. Peng, "Contributing high quantity and quality knowledge to online Q&A communities," Journal of the American society for information science and technology. vol. 64, 2013, pp. 356–371.

[4] X. Jin, Z. Zhou, M. Lee, C. Cheung, "Why users keep answering questions in online question answering communities: a theoretical and empirical investigation," International journal of information management. vol. 33, 2013, pp. 93–104.

[5] Chia Shen Chen, S. F. Chang, C. H. Liu, "Understanding Knowledge-Sharing Motivation, Incentive Mechanisms, and Satisfaction in Virtual Communities," Social Behavior and Personality: an international journal, vol. 40, 2012, pp. 639-647.

[6] A. Anderson, D. Huttenlocher, J. Kleinberg, J. Leskovec, "Steering user behavior with badges," in Proc. 22nd Int. Conf. World Wide Web, 2013, pp. 95–106.

[7] Y. Gao, Y. Chen, K. J. Ray Liu, "Understanding sequential user behavior in social computing: To answer or to vote?," IEEE Transactions on Network Science and Engineering, vol. 2, 2015, pp. 112-126.

[8] E. Bonabeau, "Agent-Based Modeling: Methods And Techniques for Simulating Human Systems," Proceedings of the National Academy of Sciences, vol. 99, 2002, pp. 7280-7287.

[9] R. Conte, M. Paolucci, "On Agent-Based Modelling and Computational Social Science," Frontiers in Psychology, vol. 5, 2014, pp. 1-21.

[10] I. Srba and M. Bielikova, "Why Stack Overflow Fails? Preservation of Sustainability in Community Question Answering," IEEE Software, vol. 33, 2016, pp. 1-14.

[11] N. Fornara, T. Balke. "Modeling organizations and institutions in MAS," Journal of Applied Logics-IFcolog Journal of Logics and their Applications, vol. 5, 2018, pp. 565-590.

[12] X. J. Mao, L. J. Shan, H. Z, J. W., "An adaptive casteship mechanism for developing multi-agent systems," International Journal of Computer Applications in Technology, vol. 31, 2008, pp. 17-34.

[13] M. Mastio, M. Zargayouna, G. Scemama, O. Rana. "Distributed Agent-Based Traffic Simulations," IEEE Intelligent Transportation Systems Magazine, vol. 10, 2018, pp. 145-156.

[14] V. D. Gabriel, A. R. Panisson, R. H. Bordini, D. F. Adamatti, C. Z. Billa, "Reasoning in BDI agents using Toulmin's argumentation model," Theoretical Computer Science, vol. 805, 2020, pp. 76-91.

[15] S. Yang, X. J. Mao, S. Yang, "Towards a hybrid software architecture and multi-agent approach for autonomous robot software," International Journal of Advanced Robotic Systems, vol. 14, 2017, pp. 1-15.

[16] J. Q. Yan, X. Li, X. Sun, Y. N. Shi, H. Q. Wang, "A BDI Modeling Approach for Decision Support in Supply Chain Quality Inspection," IEEE Transactions on Systems, Man, and Cybernetics: Systems, 2017, pp. 1-15.

[17] E. L. Deci, R. M. Ryan, "Intrinsic Motivation and Self-Determination in Human Behavior," Encyclopedia of Applied Psychology, vol. 3, 2004, pp. 437-448.

[18] R. M. Ryan, E. L. Deci, "Self-determination theory and the facilitation of intrinsic motivation," Social development, and well-being. American psychologist, vol. 55, 2000, pp. 68-78.

[19] R. W. Emerson, "Causation and Pearson's correlation coefficient," Journal of Visual Impairment Blindness, vol. 109, 2015, pp. 242-244.

[20] S. Jain, P. K. Mishra, V. V. Thakare, J. Mishra, "Design of microstrip moisture sensor for determination of moisture content in rice with improved mean relative error," Microwave and Optical Technology Letters, vol. 61, 2019, pp. 1764-1768.

Copy and Paste Behavior: A Systematic Mapping Study

Luqi Guan
Dep. Ing. Informática, Escuela Politécnica Superior
Universidad Autónoma de Madrid
Madrid, Spain
luqi.guan@estudiante.uam.es

John W. Castro[*]
Dep. Ing. Informática y Ciencias de la Computación
Universidad de Atacama
Copiapó, Chile
john.castro@uda.cl

Xavier Ferre
Dep. Lenguajes y Sistemas Informáticos e Ingeniería de
Software, E.T.S. de Ingenieros Informáticos
Universidad Politécnica de Madrid
Boadilla del Monte, Spain
xavier.ferre@upm.es

Silvia T. Acuña
Dep. Ing. Informática, Escuela Politécnica Superior
Universidad Autónoma de Madrid
Madrid, Spain
silvia.acunna@uam.es

Abstract—**Both novice and experienced developers rely more and more in external sources of code to include into their programs by copying and pasting code snippets. This behavior differs from the traditional software design approach where cohesion was achieved via a conscious design effort. Due to this fact, it is essential to know how this copy and paste programming practices are actually carried out, so that IDEs and code recommenders can be designed to fit with developer expectations and habits. Our objective is to identify the role of copy and paste programming or code clone in current development practices. A Systematic Mapping Study (SMS) has been conducted, searching the main scientific databases. The search retrieved 1271 citations and 39 articles were retained as primary studies. The primary studies were categorized according to eight areas: General information of clone usage, developer behavior, techniques and tools for clone detection, techniques and tools for clone reuse, patterns of cloning, clone evolution, effects of code cloning in software maintenance and development, and tools for clone visualization. The areas, techniques and tools of clone detection and developer behavior are strongly represented in the sample. The areas that have been least studied in the literature found in the SMS are tools of clone visualization and patterns of cloning.**

Keywords— Copy and Paste; Systematic Mapping Study

I. Introduction

The huge amount of source code available online has changed coding practices. Both novice and experienced developers rely more and more in external sources of code to include into their programs by copying and pasting code snippets [1][2], which is basically a term used in system engineering. To copy the code and reuse the code, either by doing some modifications or without doing any modification in the existing code, are common activities in software development [3]. Copy and paste is often done by inexperienced or student programmers, who find the act of writing code from scratch difficult or irritating and prefer to search for a pre-written solution or partial solution they can use as a basis for their own

problem solving [1]. Copy and paste is also done by experienced programmers, who often have their own libraries of well tested, ready-to-use code snippets and generic algorithms that are easily adapted to specific tasks [2]. This behavior differs from the traditional software design approach, where cohesion was achieved via a conscious design effort [4]. It also differs from the code reuse attained through the usage of re-use repositories built for such specific purpose. We need to know how this copy and paste programming practices are actually carried out, so that IDEs and code recommenders can be designed to fit with developer expectations and habits. The research work aims to identify the role of copy and paste programming or code clone in current development practices, by identifying through a Systematic Mapping Study [12] the current knowledge about this topic in the existing literature.

Paper organization. In Sec. 2, we present related work. In Sec. 3, we describe the research method of the SMS. Sec. 4 presents the results of the SMS. In Sec. 5, we discuss the results and threats to validity, and finally Sec. 6 concludes.

II. Related Works

We found six systematic reviews related to copy and paste [5]-[10]. The literature review by [5] presents various methods that researchers have used to study clone evolution and summarizes the advantages and disadvantages of relevant research on clone evolution. The literature review by [6] has studied code cloning and various techniques to detect code clones. The SMS by [7] focuses on metric-based clone detection techniques and various tools used in previous studies. The literature review by [8] puts a light on all the types of clones and various techniques for the detection of clones. The systematic review by [9] analyzes how code clones can be detected and which techniques and tools are used for this purpose. The literature review by [10] presented comparative review of various clone detection techniques. Most of these literature reviews are related to code clone detection and code clone

evolution, they do not refer to developer behavior, techniques and tools of clone reuse, patterns of cloning, tools for clone visualization and effects of code cloning in software maintenance and development. After analyzing papers that refer to those areas mentioned above, we can confirm that there is no SMS on these areas of code cloning, Therefore, we identify a lack of systematic approaches to identify the state of the art in these areas of code cloning.

III. RESEARCH METHOD

We aim to answer the following research questions: (**RQ1**) What is the state of the art of copy and paste? and (**RQ2**) How do developers use copy and paste? To answer both questions, we have carried out an SMS.

A. Define the Search Strategy

For the definition of the search string, we need to perform the following steps: Conformation of the control group (CG), identification and selection of the keywords, conformation of the search strings, and specification of the inclusion and exclusion criteria. To form the CG, we conducted a traditional search to identify papers directly related to our research. As a result of this search, we found a total of 10 papers: [3][13]-[21]. In the papers of the CG the words that appear most frequently must be identified. The keywords were obtained from a table with the frequency of all the words that appear in the articles of the CG. Once the keywords were identified, several options were built for the search string. Finally, we opted for the following search string: ("copy and paste code" OR "source code reuse" OR "code reuse" OR "code snippets reuse" OR "code clone" OR "code cloning" OR "software clones") AND (analysis OR design OR approach OR behavior OR habits OR intent OR research OR patterns OR "usage patterns" OR method OR techniques OR tools) AND ("software system" OR development OR developer OR system OR programming). The criteria used to retrieve the fundamental studies are summarized below. These criteria were applied by 3 of the authors of the paper.

a) Inclusion criteria: The paper is related to copy and paste behavior; OR the paper discusses aspects related to copy and paste patterns; OR the paper is related to code clones; OR the paper is about finding duplicated code.

b) Exclusion criteria: The paper is about traditional code reuse; OR the paper discusses about creating repository for future reuse; OR the paper is about programing for reuse; OR the paper is about managing duplicated code; OR the paper is a review; OR the paper is written in a language other than English.

B. Select the Studies

The search for studies was carried out in the following digital databases: Scopus, ACM Digital Library, and IEEE Xplorer. Once the list of *Retrieved Papers* is obtained (1271), it is necessary to eliminate duplicates between the databases and as a result of this first debug the *Non-Duplicate Candidate Papers* are obtained. Then, a first filter must be made applying the inclusion and exclusion criteria on the title, summary and keywords of each of the *Candidate Papers* (163). Studies

obtained from the first filter were evaluated again in a second filter. In this second filter, each researcher applied the inclusion and exclusion criteria to the full text of each of the studies. As a result, the group of *Primary Studies* was obtained (39). The search was conducted in November 2019.

C. Extract the Data and Perform Data Synthesis

Once the *primary studies* are obtained, the relevant information is extracted to answer the research questions. Figure 1 provides an overview of the primary studies retrieved by the SMS. It is made of three categories, determined by the year of publication, type of paper and research areas.

Figure 1. *Mapping showing the primary study distribution*

The left-hand side is composed of two scatter (XY) charts with bubbles at the intersections of each category. The size of each bubble is determined by the number of primary studies that have been classified as belonging to the respective category at the bubble coordinates. The right-hand side of the figure shows the number of primary studies by publication year. We can observe that publications started to grow from 2016 and many papers have been published since then, confirming the raising interest in this research area.

IV. RESULTS

After analyzing the primary studies (see Figure 1) and papers belonging to the CG, we identified eight different research areas: General information of clone usage, developer behavior, techniques and tools for clone detection, techniques and tools for clone reuse, patterns of cloning, clone evolution, effects of code cloning in software maintenance and development, and tools for clone visualization. Next, we will describe each of these areas.

General Information of Clone Usage. This area deals with clone types and high-level uses of clone information, as well clone usage patterns [3][15][18]-[23].

Developer Behavior. This area is about how developers face the use of clones (how they search, how they embed them in their code, etc.) [13]-[16][19][20][24]-[30].

Techniques and Tools for Clone Detection. This area studies the techniques and tools for clone detection, analysis and management and the use of clone-aware tools [3][11][14][31]-[43].

Techniques and Tools for Clone Reuse. This area studies the techniques and tools for clone reuse. Such as the interactive approach for recommending where and how to modify the pasted code, the approach to merge similar pieces of code by creating suitable abstractions, etc. [44]-[48].

Patterns of Cloning. This area describes several patterns of cloning, such as forking, templating and customization; the pros and cons of cloning; and methods for managing code clones [17][49].

Clone Evolution. In this area the clone community focuses on how cloned code evolves over time [15][24][50]-[54]. As this code changes, it exhibits various patterns and characteristics.

Effects of Code Cloning in Software Maintenance and Development. This area studies the effects of code cloning. It deals with the maintenance problems that clone codes can cause, as well as the clone display tools and clone patterns and refactoring recommendations to solve such problems [18][55]-[57].

Tools for Clone Visualization. This area studies tools for code clone visualization. These code clone visualization tools are used for checking code and analyzing code clones [58][59].

V. DISCUSSION AND VALIDITY THREATS

The analysis reveals that clone detection areas, techniques and tools, and the related developer behavior are strongly represented in the sample. Whereas techniques and tools for clone detection are represented by 14 publications (35.9% of the total), developer behavior is the second largest group of primary studies, with a total of 8 publications, that is, 20.5% of all of the primary studies retrieved in the SMS (39). The areas that have been least studied in the literature found in the SMS are tools for clone visualization and patterns of cloning. Judging by the increase in the number of publications since 2016, the practice of copy and paste is of notable interest.

We identify as possible threats to validity: (i) coverage of research questions (RQs), (ii) bias towards certain publications, (iii) quality of the evaluation, and (iv) lack of knowledge of the area. It is probable that the proposed RQs could partially cover the study theme, which we try to mitigate by defining a work objective and raising several RQs in consensus, with the purpose of making the objective attainable. It is possible that in an SMS the process is directed towards a specific group of studies, which we avoid by forming a literature CG and by consensus building a search chain with explicit terms obtained from the CG. It is likely that the quality of the evaluation of the studies was not adequate due to lack of expertise in the research area, which we mitigate by including in the team an investigator with experience in the subject of code clone.

VI. CONCLUSIONS

This paper describes the SMS conducted to answer the following research questions. In this section, we have considered the 39 primary studies plus the 10 papers of the control group where one of them has been obtained in the set of primary studies, making a total of 48 papers analyzed.

RQ1. The research on copy and paste or code clone deals with eight areas: *General information of clone usage*, *developer behavior*, *techniques and tools for clone detection*, *techniques and tools for clone reuse*, *patterns of cloning*, *clone evolution*, *effects of code cloning in software maintenance and development*, and *tools for clone visualization*. Most primary studies and papers belonging to the CG (33.3%) focus on techniques and tools for clone detection, followed by the ones about developer behavior (27.1%) and the studies dealing with general information of clone usage (18.8%).

RQ2. Several patterns for using copy and paste have been defined: Elementary patterns (between, within, within and between, external paste) and complex patterns (repeat, distribution, relay, unknown). On the one hand, the elementary patterns are composed of a single copy and paste interaction involving one or more files. On the other hand, complex patterns are composed of two or more copy and paste incidents involving more than two files [13].

REFERENCES

[1] G. Yarmish, and D. Kopec, "Revisiting novice programmer errors", ACM SIGCSE Bulletin, vol. 39(2), pp.131-137, 2007.

[2] R. Pittenger, "Building ASP.NET web pages dynamically in the code-behind", 2019, https://www.codeproject.com/Articles/25573/Building-ASP-NET-Web-Pages-Dynamically-in-the-Code.

[3] A. Vashisht, A. Sukhija, A. Verma, and P. Jain, "A detailed study of software code cloning", IIOAB J.-Special Issue: Comp. Science, vol. 9(2), pp. 20-32, 2018.

[4] R.N. Taylor, N. Medvidovic, and E. Dashofy, "Software architecture: Foundations, theory, and practice", John Wiley & Sons, First Ed., 2009.

[5] K. Wang, L. Zhang, and S. Yann, "A study on code clone evolution Analysis", in Proc. ICSESS'17. Beijing, China, pp. 340-345, 2017.

[6] K. Solanki, and S. Kumari, "Comparative study of software clone detection techniques", in Proc. MITicon'16. Bang-San, Thailand, pp. 152-156, 2016.

[7] D. Rattan, and J. Kaur, "Systematic mapping study of metrics based clone detection techniques", in Proc. AICTC'16. XBikaner, India, art. 76, pp. 1-7, 2016.

[8] G. Chatley, S. Kaur, and B. Sohal, "Software clone detection: A review", Int. J. Cont Theory and Applic., vol. 9(41), pp. 555-563, 2016.

[9] Q.U. Ain, W.H. Butt, M.W. Anwar, F. Azam, and B. Maqbool, "A systematic review on code clone detection", IEEE Access, vol. 7, pp. 86121-86144, 2019.

[10] N. Saini, S. Singh, and Suman, "Code clones: Detection and management", Procedia Computer Science, vol. 132, pp. 718-727, 2018.

[11] V. Saini, H. Sajnani, J. Kim, and C. Lopes, "SourcererCC and SourcererCC-I: Tools to detect clones in batch mode and during software development", in Proc. ICSE-C'16. Austin, TX, USA, pp. 597-600, 2016.

[12] B. Kitchenham, and S. Charters, "Guidelines for performing systematic literature reviews in software engineering", Tech. rep., Keele University and Department of Computer Science University of Durham, 2007.

[13] T.M. Ahmed, W. Shang, and A. E. Hassan, "An empirical study of the copy and paste behavior during development", in Proc. 12th Working Conf. on Mining Soft. Repositories. Florence, Italy, 2015, pp. 99-110.

[14] M. Balint, R. Marinescu, and T. Girba, "How developers copy", in Proc. ICPC'06. Athens, Greece, pp. 1-10, 2006.

[15] D. Chatterji, J. C. Carver, and N.A. Kraft, "Claims and beliefs about code clones: Do we agree as a community? A survey", in Proc. IWSC'12. Zurich, Switzerland, pp. 15-21, 2012.

[16] D. Chatterji, J.C. Carver, and N.A. Kraf, "Cloning: The need to understand developer intent", in Proc. IWSC'13. San Francisco, CA, USA, pp. 14-15, 2013.

[17] C. Kapser, and M.W. Godfrey, "Cloning considered harmful considered harmful", in Proc. WCRE'06. Benevento, Italy, pp. 645-692, 2006.

[18] M. Kim, L. Berman, T. Lau, and D. Notkin, "An ethnographic study of copy and paste programming practices in OOPL", in Proc. ISESE'04. Redondo, Beach, USA, pp. 83-92, 2004.

[19] T.D. LaToza, G. Venolia, and R. DeLine, "Maintaining mental models: A study of developer work habits", in Proc. ICSE'06. Shanghai, China, pp. 492-501, 2006.

[20] K.T. Stolee, S. Elbaum, and G. Rothermel, "Revealing the copy and paste habits of end users", in Proc. VL/HCC'09. Corvallis, OR, USA, pp. 59-66, 2009.

[21] G. Zhang, X. Peng, Z. Xing, and W. Zhao, "Cloning practices: Why developers clone and what can be changed", in Proc. ICSM'12. Trento, Italy, pp. 285-294, 2012.

[22] A. Khan, H.A. Basit, S.M. Sarwar, and M.M. Yousaf, "Cloning in popular server side technologies using agile development: An empirical study", Pakistan J. Eng. and Applied Sciences, Vol. 22, pp. 1-13, 2018.

[23] J.F. Islam, M. Mondal, and C.K. Roy, "Bug replication in code clones: An empirical study", in Proc. SANER'16. Suita, Japan, pp. 68-78, 2016.

[24] S. Bharti, and H. Singh, "An industrial study on developers' prevalent copy and paste activities", in Proc. ICNGCIS'17. Jammu, India, pp. 147-152, 2017.

[25] D. Chatterji, J.C. Carver, and N.A. Kraft, "Code clones and developer behavior: Results of two surveys of the clone research community", Emp. Soft. Eng., vol. 21(4), pp. 1476-1508, 2016.

[26] A. Ciborowska, N.A. Kraft, and K. Damevski, "Detecting and characterizing developer behavior following opportunistic reuse of code snippets from the web", in Proc. MSR'18. Gothenburg, Sweden, pp. 94-97, 2018.

[27] L. Müller, M.S. Silveira, and C.S. de Souza, "Do I know what my code is saying?: A study on novice programmers' perceptions of what reused source code may mean", in Proc. IHC'18. Belém, Brazil, pp. 1-10, 2018.

[28] T. Ohta, H. Murakami, H. Igaki, Y. Higo, and S. Kusumoto, "Source code reuse evaluation by using real/potential copy and paste", in Proc. IWSC'15. Montreal, pp. 33-39, 2015.

[29] B. Van Bladel, A. Murgia, and S. Demeyer, "An empirical study of clone density evolution and developer cloning tendency", in Proc. SANER'17. Klagenfurt, Austria, pp. 551-552, 2017.

[30] B. Xu, L. An, F. Thung, F. Khomh, and D. Lo, "Why reinventing the wheels? An empirical study on library reuse and re-implementation", Empirical Software Engineering, pp. 1-35, 2019.

[31] M.S. Aktas, and M. Kapdan, "Structural code clone detection methodology using software metrics", IJSEKE, vol. 26(2), pp. 307-332, 2016.

[32] M. Gharehyazie, B. Ray, M. Keshani, M.S. Zavosht, A. Heydarnoori, and V. Filkov, "Cross-project code clones in gitHub", Empirical Software Engineering, vol. 24, pp. 1538-1573, 2019.

[33] T.A.D. Henderson, and A. Podgurski, "Rethinking dependence clones", in Proc. IWSC'17. Klagenfurt, Austria, pp. 66-74, 2017.

[34] B. Joshi, P. Budhathoki, W.L. Woon, and D. Svetinovic, "Software clone detection using clustering approach", in: Arik S., Huang T., Lai W., Liu Q. (eds). Neural Information Processing. ICONIP 2015 (pp. 520-527). Lecture Notes in Computer Science, vol 9490. Springer, 2015.

[35] T. Kamiya, "An execution-semantic and content-and-context-based code-clone detection and analysis", in Proc. IWSC'15. Montreal, Canada, pp. 1-7, 2015.

[36] K. Kim, D. Kim, T.F. Bissyandé, E. Choi, L. Li, J. Klein, and Traon, "FaCoY: A code-to-code search engine", in Proc. ICSE'18. Gothenburg, Sweden, pp. 1-12, 2018.

[37] M. Mondal, C.K. Roy, and K.A. Schneider, "SPCP-Miner: A tool for mining code clones that are important for refactoring or tracking", in Proc. SANER'15. Montreal, Canada, pp. 484-488, 2015.

[38] A.-F. Mubarak-Ali, S. Sulaiman, S.M. Syed-Mohamad, and Z. Xing, "Code clone detection and analysis in open source applications", Comp. Syst. Softw. Eng.: Conc., Meth., Tools, and Appl., pp. 1112-1127, 2018.

[39] B. Priyambadha, and S. Rochimah, "Behavioral analysis for detecting code clones", Telkomnika, vol. 16(3), pp. 1264-1275, 2018.

[40] S. Reddivari, and M.S. Khan, "A topic modeling approach for code clone detection", in Proc. SEKE'18. San Francisco Bay, USA, pp. 486-491, 2018.

[41] M. Sudhamani, and L. Rangarajan, "Code similarity detection through control statement and program features", Expert Systems with Applications, vol. 132, pp. 63-75, 2019.

[42] J. Svajlenko, and C.K. Roy, "Fast and flexible large-scale clone detection with cloneworks", in Proc. ICSE-C'17. Buenos Aires, Argentina, pp. 27-30, 2017.

[43] C. Wijesiriwardana, and P. Wimalaratne, "Component-based experimental testbed to facilitate code clone detection research", in Proc. ICSESS'17. Beijing, China, pp. 165-168, 2017.

[44] S. Abid, S. Javed, M. Naseem, S. Shahid, H.A. Basit, and Y. Higo, "CodeEase: Harnessing method clone structures for reuse", in Proc. IWSC'17. Klagenfurt, Austria, pp. 24-30, 2017.

[45] Y. Lin, X. Peng, Z. Xing, D. Zheng, and W. Zhao, "Clone-based and interactive recommendation for modifying pasted code", in Proc. ESEC/FSE'15. Bergamo, Italy, pp. 520-531, 2015.

[46] K. Narasimhan, C. Reichenbach, and J. Lawall, "Cleaning up copy–paste clones with interactive merging", Automated Software Engineering, vol. 25, pp. 627-673, 2018.

[47] A. Ohtani, Y. Higo, T. Ishihara, and S. Kusumoto, "On the level of code suggestion for reuse", in Proc. IWSC'15. Montreal, Canada, pp. 26-32, 2015.

[48] T. Zhang, and M. Kim, "Poster: Grafter: Transplantation and differential testing for clones", in Proc. ICSE-Companion'18. Gothenburg, Sweden, pp. 422-423, 2018.

[49] J. Kanwal, K. Inoue, and O. Maqbool, "Refactoring patterns study in code clones during software evolution", in Proc. IWSC'17. Klagenfurt, Austria, pp. 45-46, 2017.

[50] J. Kanwal, H.A. Basit, and Maqbool, "Structural clones: An evolution perspective", in Proc. IWSC'18. Campobasso, Italy, pp. 9-15, 2018.

[51] M. Mondal, C.K. Roy, and K.A. Schneider, "Bug-proneness and late propagation tendency of code clones: A Comparative study on different clone types", J. of Systems and Softw., vol. 144, pp. 41-59, 2018.

[52] T.L. Nguyen, A. Fish, and M. Song, "An empirical study on similar changes in evolving software", in Proc. EIT'18. Rochester, USA, pp. 560-563, 2018.

[53] J.R. Pate, R. Tairas, and N.A. Kraft, "Clone evolution: A systematic review", J. Softw.: Evol. Proc., vol. 25(3), pp. 261-283, 2013.

[54] F. Zhang, X. Su, W. Zhao, and T. Wang, "An empirical study of code clone clustering based on clone evolution", J. of Harbin Institute of Technology (New Series), vol. 24(2), pp. 10-18, 2017.

[55] A. Lerina, and L. Nardi, "Investigating on the impact of software clones on technical debt", in Proc. TechDebt'19. Montreal, Canada, pp. 108-112, 2019.

[56] M. Mondal, C.K. Roy, and K.A. Schneider, "Does cloned code increase maintenance effort?", in Proc. IWSC'17. Klagenfurt, Austria, pp. 1-7, 2017.

[57] S. Wagner, A. Abdulkhaleq, K. Kaya, and A. Para, "On the relationship of inconsistent software clones and faults: An empirical study", in Proc. SANER'16. Suita, Japan, pp. 79-89, 2016.

[58] D. Mondal, M. Mondal, C.K. Roy, K.A. Schneider, S. Wang, and Y. Li "Towards visualizing large scale evolving clones", in Proc. ICSE-Companion'19. Montreal, Canada, pp. 302-303, 2019.

[59] H. Murakami, Y. Higo, and S. Kusumoto, "ClonePacker: A tool for clone set visualization", in Proc. SANER'15. Montreal, Canada, pp. 33-39, 2015.

Generating Luck from Weak Ties in Social Networks

Iaakov Exman, Omer Ganon and Asaf Yosef
Software Engineering Department
The Jerusalem College of Engineering – JCE - Azrieli
Jerusalem, Israel
iaakov@jce.ac.il,   ganon136@gmail.com,   asafjo23@gmail.com

Abstract— **One often assumes that for online Social Networks of related people, relations with strong ties better characterize the person one is looking for. However, a paradox already stated by Granovetter is the opposite assumption that weak ties to other people may be the more significant in certain contexts. This paper investigates this latter contrarian hypothesis as a novel tool to extract knowledge and systematically generate luck in the given contexts. Similarly to interestingness, luck is modeled relative to the context, by combining two functions – Relevance and Surprise. The Surprise expresses the importance of weak ties. A Luck-Generator software tool has been developed as an experimental testbed to interact with any social network. Its inputs, chosen by the Luck-Generator customer, are a context, a social network, and the customer's network page. The hypothesis is validated by results showing that relevance alone is not enough to actually generate all the potential luck: the weak ties' surprise contribute essentially to optimize success in the context task. Preliminary results are illustrated by 'Getting a Job' case study.**

Keywords: Luck calculation; Luck-Generator; Weak ties; Context; Social Network; Interestingness; Relevance; Surprise; Software Architecture; Knowledge Discovery; Knowledge Extraction.

I.   INTRODUCTION

Social Networks, besides being a huge source of searchable information, have the potential to significantly enhance performance of a variety of tasks, not necessarily related to the explicitly declared purpose of any particular network.

Concerning information search, we have previously defined and demonstrated the usefulness of an Interestingness [10] measure – composed of *Relevance* and *Surprise* functions – focusing search outcomes, beyond the capabilities of neutral search engines provided by social networks.

Regarding enhancing task performance, this work proposes a new kind of **knowledge extraction**, by means of **Luck measurement**, where by luck we mean systematically reaching goals with low apparent probability. Similarly to Interestingness, Luck is also obtained by a couple of functions, but now calculated upon different input types, with the special *Surprise* role, to overcome the low apparent probability.

A.   Systematic Generation of Luck

Our working hypothesis is the assumption stated long ago by Granovetter [16] that weak ties to other members of a – real or virtual – social network may be surprisingly more significant than strong ties in certain circumstances. Given a certain context, defining a task to be performed, one computes a Luck measure for *relevant* social network members, with a *Surprise* function quantitatively expressing the weak ties of network members. These were inspired by the generic definition of interestingness:

$$Interestingness = Relevance * Surprise \qquad (1)$$

The operator $*$ in this equation in its most generality, when not commutative, is a kind of composition. In the simpler commutative cases it is just plain multiplication (see [10]).

The rationale, actual functionalities in the analogous equation for calculating Luck, the input variables and additional motivation are formulated in the more theoretical section III of this paper.

B.   Weak Ties in Social Networks

A natural representation of a social network is a graph in which vertices stand for network members and edges represent their ties to other network members. The tie strength – or rather tie weakness – can be a function of a few different variables, e.g. distance in terms of counting weighted graph edges, content similarity and communication frequency.

The goal of this paper is to validate the working hypothesis by evaluating the calculated Luck with respect to the contribution of surprisingly weak ties and its effective results for the context task.

C.   Paper Organization

The remaining of the paper is organized as follows. Section II concisely reviews Related Work. Section III formulates the Luck generation underlying theory. Section IV describes the Luck-Generator software tool architecture and implementation. Section V illustrates the Luck generation task by means of a case study. Section VI concludes the paper with a discussion.

II. RELATED WORK

We concisely review the literature related to Luck characterization, Interestingness concepts, and practical applications of weak ties within social networks.

A. Luck Characterization

We refer to *Luck* in a positive context of systematic generation, in order to succeed in concrete tasks performance, in contrast to random uncontrollable situations, in which sometimes one achieves "by chance" desirable outcomes. An interesting extended example of the latter negative meaning is the book by Clayton Christensen and co-authors [6] entitled "Competing against Luck". It advocates causality as opposed to the frustration of hit-and-miss innovation, viz. leaving your fate to luck.

Dowding [9] deals mostly with moral aspects of luck; he also suggests a simplistic measure of luck as a relationship between expected value of outcome (EV) and the actual outcome (AV), thus $Luck = AV - EV$, where in a serial of trials one would expect that AV approaches EV.

Liechti et al. [23] use a more sophisticated definition of luck as the unexpected component of performance. It is a sum of three terms: a- the actual deviation from expected performance; b- an overconfidence bias; c- a look back bias (a difference of subjective expectation at a certain time *t* and at a previous time). This definition is closer to our own definition, which involves a surprise (or unexpectedness) factor.

B. Interestingness Concepts and Applications

It is worthwhile to be acquainted with the literature on Interestingness, as the calculation of this quantity shown in equation (1), inspired the proposed calculation of *Luck*, in particular the *Surprise* factor, as explained in section III.

Overviews of Interestingness measures for typical purposes, such as Data Mining and Knowledge Discovery are found in Geng et al. [14] and McGarry [25]. For instance, criteria on how to determine interesting rules/patterns generated in data mining are described by Lenca et al. [22].

There are several differing approaches to interestingness as described e.g. in the Klosgen and Zytkow Handbook [20], especially by Tuzhilin [28]. Exman, defined Interestingness as a product of relevance and surprise in 2009 [10]. This definition has been implemented with successful Web search results, in software tools such as the one described in [11].

C. Social Networks Weak Ties and Applications

Granovetter [16], [17] is the pioneer of asserting significance to Weak Ties in social networks. He also was one of the first researchers that actually made concrete application of the theory in his book [18] originally published in 1974, in the context of "Getting a Job". A generic analysis of networks from an historical viewpoint is the book by Ferguson [12], which includes chapter 6, explicitly dealing with weak ties.

The importance of weak ties in social networks triggered a variety of studies. Many of them supported the theory – such as Brown and Konrad [4], DeMeo et al. [7]. In contrast, some of them rather emphasized the importance of strong ties – such as

Gee et al. [10], Krackhardt and co-authors [17]. Others, extended the theory to different applications, – such as Baer [2], Centola [5] – or provided general appraisals e.g. Sinan [25].

Specifically concerning the "Getting a Job" context, besides Granovetter, one finds Gee et al. [13] and the paper by Tassier on "Labor Market implications of Weak Ties" [27]. Of significance for this work is the statement by Tassier that weak ties in a person's social network grows with network distance exponentially faster than strong ties, which is reasonable.

Finally, the technical issue of measuring the strength of a tie is dealt with e.g. in the paper by Marsden and Campbell [24].

III. LUCK IN CONTEXT

This section's goal is to formulate the theoretical basis for Luck calculation for any given context data set. It is the result of Luck mathematical modeling, based upon assumptions following experimental results, ours and in the literature on social network ties' strength. It starts from an abstract scheme reflecting actual experiments with (non-virtual) networks.

A. The Abstract Scheme

Our idea, on how to generate Luck, avoids the controversy on the relative importance of strong ties vs. weak ties, in a straightforward way by involving both strong and weak ties.

Our abstract scheme, in the next text-box has two actions, not necessarily in a fixed order, which may occur concurrently.

Abstract Scheme: Luck Generation

1st action: a relevant ***strong tie*** – determines the task to be performed, within the chosen context;

2nd action: a surprising ***weak tie*** – obtains a pointer to the desired outcome.

This abstract scheme is illustrated by 3 stories that actually occurred in human (not virtual) networks.

The first story task was to "*find a scientific collaborator*". The relevant ***strong tie*** was to participate in a conference whose main topic fits the researcher's scientific interests. The Conference was held in China. The ***weak tie*** was to find among the many conference participants a Spanish researcher with whom a vivid conversation of mutual interest developed. The surprising aspect was to travel a long distance to China to find a Spanish researcher.

The second story task was to "*find a job in the profession*". The ***strong tie*** was to be an active member in relevant professional interest groups in the internet. This story referred to a Java programming language interest group. The ***weak tie*** was, in response to an inquiry, to get an answer from an old acquaintance in the past, but disconnected for several years. The acquaintance enabled a successful information exchange, leading to a concrete job, which was actually taken.

The third story task was to "*find a candidate for a vacant position*" in our institution. In this story, the ***weak tie*** occurred first. A certain candidate presented himself to the candidates' recruiter, to show his credentials, and by the way mentioned members of his family. The ***strong tie*** was that the candidate's brother learnt years ago in the same class and was well-known to the recruiter, being a strong implicit recommendation.

B. Modeling Luck Calculation

Given the literature on social network ties' strength and the previous abstract scheme, we make the following assumptions:

1. **_Complementary Exponential decay of ties_** – strong ties decay exponentially with the network distance, while weak ties increase exponentially and vice-versa (see e.g. Tassier [27]);
2. **_People Matching with strong ties_** – strong ties bond similar people to each other (see e.g. Granovetter [16], and Krackhardt [21]) and vice-versa mismatching for weak ties;
3. **_Time Commutativity of strong/weak ties_** – sometimes the strong tie action precedes the weak tie action, other times the order is reversed (as illustrated by the above stories of the abstract scheme).

We now formulate the necessary equations to model Luck calculation, based upon the above assumptions. In terms of notation we define two functions that calculate the contribution of strong and weak ties as follows:

- **_Relevance_** – calculates the strong ties' contribution;
- **_Surprise_** – calculates the weak ties' contribution.

By the 1st assumption on "Complementary Exponential decay" each of these functions is an exponential, with complementary signs. By the 2nd assumption on "People Matching" with strong ties and "People Mismatching" with weak ties one has:

$$\mathrm{Re}\,levance = \exp(Match) \qquad (2)$$

$$Surprise = \exp(Mismatch - Match) \qquad (3)$$

By the 3rd assumption on "Time Commutativity" one has:

$$Luck = \mathrm{Re}\,levance + Surprise \qquad (4)$$

The "plus" operator is certainly commutative. A "multiplier" operator in this equation is obviously unsuitable, as the exponential nature of these terms would cause undesirable exponents addition.

Finally, substituting equations (2) and (3) into equation (4) one obtains:

$$Luck = \exp(Match) + \exp(Mismatch - Match) \quad (5)$$

In practice, to use this equation in calculations, one still needs to make adjustments to normalize the expressions of Match and Mismatch, in order to eliminate dependence on set sizes.

C. Luck Calculation with Keyword Sets

In this paper we restrict Luck calculation due to the representation of social network members by their respective Keyword Sets.

First, additional notations are introduced:

- **_Context_** – is the keyword set defining a task, e.g. "_find a job in a specific profession_", such as software engineering;
- **_Customer_** = **_C_** – is the person, member of a social network, who demands the performance of the Context task; it also designates the keyword set of this person;
- **_Follower_** = **_F_** – is another member of the same Customer's social network, which is a follower (in the social network sense) of the Customer; it also designates the keyword set of the Follower; **_F_** is generalizable to a Follower of a Follower of the Customer, or to any distance from the Customer.

The keyword set of the Context is determined before any computation starts. The keyword set of the Customer and of each Follower are sub-sets of the Context keyword set. These are determined by extracting sets from the person pages in the Social Network, and finding the intersection of the extracted sets with the Context keyword set.

In this work **_Match_** and **_Mismatch_**, are keyword set operations necessary to obtain respectively the Relevance and Surprise functions, by comparing keyword sets for each Customer **_C_** with the keyword set for a Follower **_F_**. _Match_ calculates a similarity measure of the input sets, i.e. keywords appearing in the intersection $\bigcap$ of these sets:

$$Match = C \bigcap F \qquad (6)$$

The output is the number of intersection elements of **_C_** and **_F_**.

Mismatch calculates the sets' dissimilarity, viz. a symmetric difference Δ between **_C_** and **_F_**. It is the union $\bigcup$ of the relative complements of these sets:

$$Mismatch = C\Delta E = (C - F) \bigcup (F - C) \qquad (7)$$

Match and Mismatch diagrams are seen in Fig. 1.

Figure 1. Schematic Match and Mismatch diagram – **_C_** represents the Customer keyword set (yellow). **_F_** represents the Follower keyword set (hatched green). Match is the intersection $C \bigcap F$. Mismatch is the union between the relative complements **_C-F_** and **_F-C_**.

In this section we describe the Luck-Generator software tool software architecture and implementation. The tool enabled testing of the Luck calculations and the Case study in section V. Its output is a list of candidates: a number of Customer followers with the highest calculated Luck values.

A. Luck-Generator Architecture

The Luck-Generator software architecture has the following roles, as shown in Fig. 2:

- *Front-End* – for input and output;
- *APIs* – for interaction with any chosen social networks;
- *Keyword handler* – to extract and collect keyword sets;
- *Local Storage* – to avoid repeated networks access;
- *Inquirer* – to retrieve necessary data from storage;
- *Calculators* – of Tie Strength and Luck;
- *Analysis tool* – for system maintenance.

The Luck-Generator architecture was designed to be generic, and not fitting any particular Social Network. One only needs to insert the needed specific   API.

Figure 2. Luck-Generator Software Architecture – The Front-End (yellow) in the figure upper part inputs data and outputs the resulting candidate list. The upper-right modules (in blue) get the inputs (customer, social network, context) and obtain keyword sets and followers to be stored locally. The Inquirer (orange) retrieves data to calculators (green) for Tie Strength and Luck. The Analysis tool obtains diagnostic graphs for system maintenance.

B. Luck-Generator: Implementation

As far as possible the system is composed of existing software modules. For instance, extraction of keywords to characterize the context, the Customer and followers is done with the help of Datamuse – a semantic network with a word-finding query engine for system developers – through its API.

Similarly, access to Social Networks is done by specific available APIs.

The chosen Context task for our case study is "*find a job in the profession*". The context was defined, data from a Social Network extracted and calculations performed, as reported here.

A. Context Definition: Getting a Job

The chosen profession was "Software Engineering". The Context diverse keyword set is in the next text-box: it has 'word pairs' and even keywords not exactly belonging to software.

Context Keyword Set

Software, engineering, developer, DevOps, computers, algorithm, TechOps, python, programmer, java, 'computer science', 'data science', 'data analyze', C++, web, framework, embedded 'alpha version', API, api, app, application, beta, version, bios, QA, automation, agile, scrum, demo, development, device, emulator, freeware, 'open source', interface, 'operating systems', workflow 'machine learning', 'deep learning', startup, innovation, internet, IoT, VR, code, coding.

The social network was dictated by an available API. We started testing with a couple of initial "customers" searching for the job. According to their extracted keywords characterization these have been involved previously with software, to assure that testing is realistic.

Normalization of both the Match and Mismatch functions in equation (5) was done by a sum of the intersection of the Context and Costumer keyword sets with the intersection of the Context and each Follower keyword sets.

B. Calculation Results: Relevance vs. Surprise

Calculation results were obtained with input data extracted from the social network, for each Customer, and a small number of followers and all the available followers of followers.

The next fig. 3 shows an inverse exponential relation between Relevance and Surprise for the data-set of a certain Customer.

Figure 3. Graph of Calculated Relevance against Surprise for Customer C.D. – There is an inverse relation between these two quantities: when Relevance – expressing Strong Ties – decreases exponentially, Surprise – expressing Weak ties – increases and vice-versa, as predicted by our Model.

C. Empirical Validation

We have used several validation techniques to increase confidence on the obtained results. These included:

- Results Consistency – Variation of customers;
- Robustness – Variation of Context keyword sets;
- Comparison with results shown in the literature.

As an example of Results Consistency, Figure 4 shows the same calculation of Luck vs. Surprise for four Customers (L.M., C.D., M.M. and S.C.) Although the Customers and their followers' data sets are totally independent, the functional behavior is very similar.

Figure 4. Graphs of Calculated Luck against Surprise for diverse Customers – As Surprise – expressing Weak Ties – increases, our Model predicts that calculated Luck also increases. A smaller increase of Luck at the left-hand-side of the graph, corresponds to a Relevance increase – expressing Strong Ties. Dots show results calculated for actual data from the Social Network. Trend-lines are very good polynomial fittings. All four graphs have the same scales.

VI. DISCUSSION

We deal here with open issues triggered by the preliminary results of this work.

A. The Functionality of Luck Calculation

The functionality of Luck calculation in this paper is based upon empirical assumptions. These have been validated to be reasonable and self-consistent.

Nonetheless, it would be desirable to formulate a more rigorous derivation of the equations we have used.

A few possible starting points are as follows:

- *Maximum entropy approach* – it is well-known that such an approach, i.e. maximum entropy under constraints, obtains probability distributions with exponential functionality, where the exponent is a negative quantity. This would be suitable to explain the exponential expressions in equation (5) of this paper;
- *Hyperbolic Modeling of Probability Distributions*- for example, one may perceive that the functionality of Luck calculation in the same equation (5) has an obvious similarity to a Hyperbolic Cosine. An example of probability modeling involving hyperbolic functions is found in the work of Hanaki et al. from Tsukuba University [19].

B. Systematic Luck vs. Irrationality

From the very beginning of this work we adopted a positive constructive view of Luck, in other words "Systematic Look". This paper is a contribution in this direction. This is not an esoteric point of view. There is a non-negligible number of works with this approach.

We mention here Dowding [8] which argues for the utility of ideas of luck and "systematic luck". Hanaki et al. [19] suggest that people learning from experience leads them to make choices with "luckier" outcomes than others. Contrast these with Adaval [1].

C. Other Variables for Tie Strength Measure

Besides keyword sets, we are aware of other important variables to characterize Tie Strength, which were not considered in this work. These include among others, topology measures such as relationships among edges and vertices in the social network and communication intensity between members of the social network, such as frequency and the nature of the communication, either generic such as 'like' or more personal contents.

We are currently working to integrate these other variables in the same generic equations of our model – described in section III *B*.

D. Future Work

In addition to the interpretation issues and the number of variables to characterize Tie Strength, important directions for further investigation are:

- Extensive application to a variety of Customers and their followers;
- Application to different contexts, besides "*finding a job*" that has already been intensively researched in the literature;
- Usage of different functions to calculate Relevance and Surprise, such as Tf*Idf, and compare their results with those of match and mismatch;

E. Main Contributions of this Paper

The main contributions of this paper are: 1- the idea of systematic generation of Luck in a constructive sense, within contexts of practical tasks, exploring social networks; 2- to model the significant contribution of Weak Ties for Luck generation in terms of a mathematical expression of Surprise.

REFERENCES

[1] R. Adaval, "Culture and Cognition: the Case of Irrational Beliefs about Luck", *Advances in Consumer Research*, Vol. 33, pp. 623-628 (2006).

[2] M. Baer, "The Strength-of-Weak-Ties Perspective on Creativity: A Comprehensive Examination and Extension", J.App.Psychology, 95, pp-592-601, (2010).

[3] M. Bertrand and S. Mullainathan, "Are CEOs Rewarded for Luck? The Ones with Principals are", The Quarterly Journal of Economics, Vol. 116, pp. 901-932 (Aug. 2001). http://www.jstor.org/stable/2696421

[4] D.W. Brown and A.M. Konrad, "Granovetter Was Right: The Importance of Weak Ties to a Contemporary Job Search", Group & Organization Management, 26, 434-462, (2001).

[5] D. Centola, "Complex Contagions and the Weakness of Long Ties", Am.J.Sociology, 113, pp. 702-734, (2007). DOI: **https://doi.org/10.1086/521848**

[6] C.M. Christensen, T. Hall, K. Dillon and D.S. Duncan, Competing Against Luck – The Story of Innovation and Customer Choice, Harper Business, New York, NY, USA, October 2016.

[7] P. DeMeo, E. Ferrara, G. Giumara and A. Provetti, "On Facebook, Most Ties are Weak", Comm. ACM, 157, pp. 78-84 (October 2014) https://doi.org/10.1145/2629438; See also: arXiv:1203.0535, 2012.

[8] K. Dowding, "Resources, Power and Systematic Luck: A Response to Barry", *Politics, Philosophy & Economics*, pp. 305-322 (October 2003) DOI: https://doi.org/10.1177/1470594X030023002

[9] K. Dowding, "Luck and Responsibility", in M. Matravers and L. Meyer, (eds.) Democracy, Equality and Justice, Routledge, London, UK, 2008.

[10] I. Exman, "Interestingness – A Unifying Paradigm – Bipolar Function Composition", in Proc. KDIR Int. Conf. on Knowledge Discovery and Information Retrieval, pp. 196-201, 2009.

[11] I. Exman, G. Amar and R. Shaltiel, R., "The Interestingness Tool for Search in the Web", in Proc. SKY'2012 Int. Workshop on Software Knowledge, pp. 54-63, 2012.

[12] N. Ferguson, *The Square and the Tower* – Networks, Hierarchies and the Struggle for Global Power", Penguin Books, UK, (2018).

[13] L.K. Gee, J. Jones and M. Burke, "Social Networks and Labor Markets: How Strong Ties Relate to Job Finding On Facebook's Social Network", J.Labor Economics, 35, pp. 485-518 (April 2017). DOI: https://doi.org/10.1086/686225

[14] L. Geng and H.J. Hamilton, "Interestingness Measures for Data Mining: A Survey", ACM Computing Surveys, Vol. 38, (3), Article 9, 2006.

[15] Z. Gilani, R. Farahbakhsh, G. Tyson, L. Wang and J. Crowcroft, "An in-depth characterization of Bots and Humans on Twitter", 2017.

[16] M.S. Granovetter, "The Strength of Weak Ties", Am.J.Sociology, 78, pp. 1360-1380, (May 1973).

[17] M.S. Granovetter, "The Strength of Weak Ties: A Network Theory Revisited", Sociological Theory, 1, pp. 201-233, (1983).

[18] M.S. Granovetter, *Getting a Job: A Study of Contacts and Careers*, 2nd ed. University of Chicago Press, Chicago , IL, USA, (1995).

[19] N. Hanaki, A. Kirman and M. Marsili, "Born Under a Lucky Star?", Tsukuba Economics Working Papers, No. 2009-003, (March 2009), Tsukuba, Japan.

[20] W. Klosgen and J.M. Zytkow, (eds.), *Handbook of Data Mining and Knowledge Discovery*, Oxford University Press, Oxford, UK, 2002.

[21] D. Krackhardt, N. Nohria and B. Eccles, "The Strength of Strong Ties: The Importance of Philos in Organizations", in Networks in the Knowledge Economy , (2003).

[22] P. Lenca, P. Meyer, B. Vaillant and S. Lallich, "On selecting interestingness measures for association rules: user oriented description and multiple criteria decision aid", European J. Operational Res., Vol. 183, pp. 610-626, 2008.

[23] D. Liechti, C. Loderer and U. Peyer, "Luck and Entrepreneurial Success", Political Science, Psychology, Economics, 2012. DOI:10.2139/ssrn.1954560

[24] P.V. Marsden and K.E. Campbell, "Measuring Tie Strength", Ann. Meeting Am. Sociological Assoc., pp. 482-499, (1984).

[25] K. McGarry, "A survey of interestingness measures for knowledge discovery", Knowledge Engineering Review J., 20 (1), 39-61, 2005.

[26] A. Sinan, "The Future of Weak Ties", Am.J.Sociology, 121. pp. 1931-1939, (2016). DOI: http://dx.doi.org/10.1086/686293

[27] T. Tassier, "Labor Market Implications of Weak Ties", Southern Econ.J., 72, pp. 704-719, (2006). DOI: 10.2307/20111842.

[28] A. Tuzhilin, "Usefulness, Novelty, and Integration of Interestingness Measures", chapter 19.2.2 in ref. [16], pp. 496-508, 2002.

Personalized Video Recommendation
Based on Latent Community

Han Yan, Ye Tian, Shunyao Wang, Xiangyang Gong, Xirong Que and Wendong Wang

State Lab of Networking and Switching Technologies

Beijing University of Posts and Telecommunications, China

{hanyan, yetian, wangshunyao, xygong, rongqx, wdwang}@bupt.edu.cn

Abstract—Facing with information overload, recommender system has been employed in many fields, from news, e-commerce to videos and musics. However, the traditional recommendation method that focuses on single individual may not has good performance because of the data sparsity and the curse of large dimensionality. Although group recommendation has been raised recently to utilize users' social information, many of them just simply aggregate users' rating information without analyzing the latent relations among users. In this paper, we proposed a latent community based video recommendation model (LCB-Rec). This method does not need explicit user preference information, it discovers the latent topics of each video with Latent Dirichlet Allocation (LDA) and finds latent user relations with Personalized PageRank. Then, latent community's profile is generated by cluster method and merge strategy. LCB-Rec focuses on giving recommendation to latent community rather than single user. We make comparative experiments with Matrix Factorization (MF) and Random Walk with Restart (RWR) based on the real-world datasets. The experiment results demonstrate that our proposed method has a better performance.

Keywords—video recommendation, latent community detection, topic model

I. INTRODUCTION

With the development of network transmission and data processing, people can spend more time interacting with the Internet. Recently, watching online video has become a popular entertainment among people. For instance, at YouTube, the world's most popular online video website, millions of users will request millions of videos in a single day. Besides, users will upload videos continuously to the YouTube with the speed of more than 24 hours of video per minute [5]. With such a tremendous video repositories, offering videos that match users' interest is a critical problem to be solved. This is why so many video websites adopt recommender system.

In general, the method for recommendation could be classified into three categories: collaborative filtering, content based method and hybrid recommendation [9]. These traditional recommendation methods focus on providing services for single user. Evidences that support this kind of recommendation are users' feedback to items, characteristic of each item and users' profile. However, facing large quantities of unregistered users and cold-start problem, the recommendation methods mentioned above may not have a good performance.

Since it is hard to recommend for single user, how about offering recommendation for a group of similar users to decrease dimensionality. In fact, it is reasonable to offer recommendation to a group of users. Since in real life, people with similar interest tend to like the same things [1]. However, in most circumstances [10], raw data does not contain enough information about users' social relations. Hence, it is necessary to find out the "latent social network" of users.

Recently, bullet comments are very popular in many video websites. As a form of socialized application, bullet comments give a real-time interaction between video contents and users' inside idea. Bullet comments can reflect topics of video, feelings of users, and what users may be interested in. Such vast amounts of information would be helpful to find the "latent community", and with these latent communities we can give a better recommendation for community users.

The main contributions and solved problems of this paper are as follows:

- We treat bullet comments as a "corpus" and build topic model with this "corpus" and LDA method, which returns the topic distribution for each video.
- We build a directed tripartite graph and apply Personalized PageRank to find similar users. we employ cluster method and merge strategy to generate the topic distribution of latent community.
- We compute the pearson correlations between new videos and each community and rank these videos with this correlations. The top k videos will be recommended.

II. RELATED WORK

A. Traditional Recommendation

Generally speaking, traditional recommendation methods may be divided into three classes: collaborative filtering, content based and hybrid of the two [9]. Content based method provides recommendation by analyzing item's similarity or user's preference [6]. Collaborative Filtering (CF) recommendation is based on users' past behaviors. It assumes that users with similar behavior history tend to have same interests. It uses the past item-rating matrix to build a model for the purpose of measuring similarity between users.

B. Group Recommendation

To utilize users' social information, latent information based recommendation has been proposed. Christakopoulou et al. [4]

Fig. 1: Framework of the recommendation model

proposed SVD based model to learn latent user relations from rating patterns. What's more, some recommendation method tends to detect latent community. Cao et al. [2] proposed an improved CF algorithm, which predicts rating scores based on communities. Cheng et al. [3] proposed a method to detect overlapping community in complex network. Lin et al. [7] proposed a recommendation model with implicit communities from user ratings and social connections.

However, methods in [2][4][7] need explicit rating information such as "like" or "dislike", our proposed method focuses on offering recommendation by mining data without explicit ratings. Our method utilizes users' watching records and their bullet comments to discover latent factors.

III. FRAMEWORK OF MODEL

As Figure.1 shows, our recommendation model consists of four parts: video's topic identification, user relationship discovery, latent community discovery and recommendation for community.

A. Video's Topic Identification

We focus on analyzing the similarity between community's topic distribution and video's topic distribution. Thus, it is critical to dig out latent topics from video's bullet comments.

In this part, we discover the latent probabilistic topic distribution from each video. Topic models such as LDA is employed to extract the abstract topics from documents as a form of document-topic distribution and topic-word distribution. Although, LDA cannot be applied to videos directly, the bullet comments of the videos contain detailed information of each video. Therefore, our proposed method heuristically treats each video as a document, and all bullet comments will be regard as the contents of document sets.

B. User relationship discovery

In this part, we utilize the existing user-video relations in the raw data and latent video-topic distribution found with LDA to identify latent user relations. We build a directed tripartite graph indicating relations among users, videos and topics. The weight η of each edge in graph is defined as follows:

1) Weight $\eta_{i,j}$ of the edge pointing from user u_i to video v_j : $\eta_{i,j} = \frac{C_{i,j}}{C_i}$, where $C_{i,j}$ is the number of bullet comments that user u_i made in video v_j, C_i is the number of all bullet comments made by user u_i.

2) Weight $\eta_{i,j}$ of the edge pointing from video v_i to topic k_j: $\eta_{i,j} = \theta_{i,j}$, where $\theta_{i,j}$ denotes the probability that video v_i belongs to topic k_j.

3) For the edge pointing from user u_i to topic k_j: $\eta_{i,j} = \frac{1}{C_i} \sum_{v=1}^{V} \sum_{c \in v} \theta_{v,j}^{\langle c \rangle}$, where C_i is the number of bullet comments made by user u_i, V is the number of videos. v identifies a single video. c indicates a single bullet comment. $\theta_{v,j}^{\langle c \rangle}$ is the probability that video v belongs to topic k_j and comment c is from user u_i to video v.

4) For the edges of other directions, the weight is calculated as below: $\eta_{i,j} = \frac{1}{|out(i)|}$, where the $|out(i)|$ is the out degree of the node i in the graph.

In this part, we make a matrix implementation of Personalized PageRank and describe the graph in a form of transition matrix M. The final matrix R that describe the degree of user similarity could be calculated as:

$$R = (E - dM^T)^{-1}(1 - d) \tag{1}$$

where E is diagonal matrix, d is the damping factor.

C. Latent community discovery

The essence of latent community discovery is to explore users' relationships and gather similar users. Affinity Propagation is a cluster method taking similarity matrix of sample nodes as input, which make it suitable for this problem.

The latent community profile is produced from user's profile. Three merge strategies (average strategy, least misery strategy and most pleasure strategy) are employed to generate community profile $\vec{g}_c$ for community c.

Average merging strategy (AMS) is a synthesize consideration of all users, $\vec{g}_c(k) = \sum_{x \in c} \frac{\vec{u}_x(k)}{N}$.

| (a) Perplexity comparison with different K | (b) Silhouette Coefficient comparison | (c) Calinski-Harabasz index comparison |

Fig. 2: Metrics comparison to select appropriate parameter K and preference

Least misery strategy (LeMS) represents lower bound of all users, $\vec{g}_c(k) = \min_{x \in c} \vec{u}_x(k)$

Most pleasure strategy (MoPS) represents upper bound of all users, $\vec{g}_c(k) = \max_{x \in c} \vec{u}_x(k)$.

$\vec{g}_c(k)$ is a vector denoting community c's preference for topic k, $\vec{u}_x(k)$ is a vector denoting user x's preference for topic k, N is the number of users in community c.

D. Video Recommendation

The recommendation is executed by analyzing correlations between latent community's topic distribution and video's topic distribution. As there is no information about new coming video's topic distribution, for each new coming video, we load the topic-word matrix Φ trained with LDA, transpose and normalize the matrix to get word-topic matrix Φ^T. For each word w_i in the new video's bullet comments, we sample word's topic k_j with probability Φ_{ij}^T. After that, we get a statistics vector $\vec{n} = \{n_1, n_2, \cdots, n_K\}$. Each element n_j denotes the number of words in each topic k_j. Finally, we calculate the topic distribution $\vec{r}_v = \{r_1, r_2, \cdots, r_K\}$ of video v, where $r_j = \frac{n_j}{\sum_{i=1}^{K} n_i}$.

With the topic distribution $\vec{r}_v = \{r_1, r_2, \cdots, r_K\}$ of video v and topic distribution $\vec{g}_c = \{g_1, g_2, \cdots, g_K\}$ of community c, we can use the Pearson correlation coefficient to measure the similarity between latent community c and video v:

$$corr(c,v) = \frac{\sum_{k=1}^{K} (r_k - \overline{r}) \times (g_k - \overline{g})}{\sqrt{\sum_{k=1}^{K}(r_k - \overline{r})^2} \times \sqrt{\sum_{k=1}^{K}(g_k - \overline{g})^2}} \quad (2)$$

where $\overline{r} = \frac{1}{K}\sum_{k=1}^{K} r_k$, $\overline{g} = \frac{1}{K}\sum_{k=1}^{K} g_k$.

We rank videos according to the Pearson correlation coefficient and select the top-k videos for recommending.

IV. EXPERIMENT

In this section, we perform experiments to answer the following questions: (1) What is the proper parameter K (the number of topics) to be set in the Latent Dirichlet Allocation. (2) What is the proper parameter $preference$ to be set in the Affinity Propagation cluster procedure. (3) Does our proposed method (LCB-Rec) have a better performance than the other recommendation methods.

A. Dataset

We obtained 3,847 video's bullet comments from video web sites (https://www.bilibili.com). To make sure that LDA have enough training data, we select top 120 videos and each video contains at least 8000 comments.

B. Comparative Methods

To evaluate the performance of LCB-Rec, Matrix Factorization (MF) and Random Walk with Restart (RWR) [8] are used for comparative experiments. LCB-Rec with different merging strategies and individual recommendation (Indi-Rec) without community profile are also comparative experiment.

C. Evaluation Metrics

1) Perplexity: Perplexity measures how well a probability distribution predicts a sample. Model with lower perplexity owns better performance. The definition of perplexity is:

$$perplexity = exp\{-\frac{\sum_{v=1}^{V} log(p(w_v))}{\sum_{v=1}^{V} N_v}\} \quad (3)$$

where V denotes the number of videos, N_v indicates the number of words without repetition in video v. $p(w_v)$ indicates word w's distribution in the video v's comments.

2) Cluster Performance Metrics: As the latent community is generated without the ground truth labels, it is indeed to use some metrics for evaluation.

We use Silhouette Coefficient (SC) and Calinski-Harabasz index (CH) to evaluate cluster performance. The definitions about the two metrics could be found within scikit-learn. A higher value of SC or CH relates to a better model.

3) Top-k Metrics: We measure $precision@k$, $recall@k$ and $f_1 score@k$ of each method to evaluate the performance. Definitions of these metrics are:

$$precision@k = \frac{|Actual(k) \cap Predicted(k)|}{Predicted(k)} \quad (4)$$

$$recall@k = \frac{|Actual(k) \cap Predicted(k)|}{Actual(k)} \quad (5)$$

$$f_1 score@k = \frac{2 \times precision@k \times recall@k}{precision@k + recall@k} \quad (6)$$

where $Actual(k)$ is the top k actual videos' set, $Predicted(k)$ is the top k predicted videos' set.

(a) $precision@k$ comparison   (b) $recall@k$ comparison   (c) $f_1score@k$ comparison

Fig. 3: Performance comparison with MF, RWR, Indi-Rec

D. Experiments Results and Analysis

Figure.2a shows the variation of perplexity with 10 different K. With a synthetic consideration of the training set's size and the variation of perplexity, we choose the value of K equals 60, since the perplexity decrease drastically with value of K from 10 to 60, and decrease slowly from 60 to 100.

To determine the proper value of $preference$ in Affinity Propagation, we measure Silhouette Coefficient (SC) and Calinski-Harabasz index (CH) of each cluster result. We select 9 values of $preference$ from -0.0009 to -0.0001 with a step of 0.0001 and use the default $preference$ value as the baseline (Shown as a horizontal green line in the figure).

As Figure.2b and 2c shows, both metrics (SC and CH) indicate that the best cluster result is when the $preference$ equals -0.0002. This cluster result ($preference$ equals -0.0002) will be used to generate latent community.

We compare the performance of LCB-Rec, MF, RWR and Indi-Rec. As Figure.3 shows, LCB-Rec method with "MoPS" or "AMS" strategy and Indi-Rec method has a better performance than MF and RWR methods. The performance improvement owes to following reasons: First, LCB-Rec method and Indi-Rec method both discover latent topic distribution with users' bullet comments, while MF and RWR simply use users' rating information. Second, LCB-Rec method generates latent community, which reduces the dimensions of latent relation matrix. While Indi-Rec method lacks information about similar users. Thus, LCB-Rec method with "MoPS" strategy has better impact than Indi-Rec.

V. CONCLUSION

In this paper, we focus on recommending videos to a latent community rather than a single user. Despite simply aggregating users' rating information, we try to discover the latent information among users and build a latent community with the help of topic model and cluster method. Compared to other recommending method like MF and RWR, our proposed method shows better performance.

Discovering latent information from user generated content (UGC), like bullet comments, does help to boost recommendation performance. In the future, more latent information could be excavated from UGC data to strengthen algorithm.

REFERENCES

[1] Irfan Ali and Sang-Wook Kim. Group recommendations: approaches and evaluation. In *Proceedings of the 9th International Conference on Ubiquitous Information Management and Communication*, page 105. ACM, 2015.

[2] Cen Cao, Qingjian Ni, and Yuqing Zhai. An improved collaborative filtering recommendation algorithm based on community detection in social networks. In *Proceedings of the 2015 annual conference on genetic and evolutionary computation*, pages 1–8. ACM, 2015.

[3] J. Cheng, X. Wu, M. Zhou, S. Gao, Z. Huang, and C. Liu. A novel method for detecting new overlapping community in complex evolving networks. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 49(9):1832–1844, 2019.

[4] Evangelia Christakopoulou and George Karypis. Local latent space models for top-n recommendation. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1235–1243. ACM, 2018.

[5] James Davidson, Benjamin Liebald, Junning Liu, Palash Nandy, Taylor Van Vleet, Ullas Gargi, Sujoy Gupta, Yu He, Mike Lambert, Blake Livingston, et al. The youtube video recommendation system. In *Proceedings of the fourth ACM conference on Recommender systems*, pages 293–296. ACM, 2010.

[6] Shanshan Feng, Huaxiang Zhang, Jian Cao, and Yan Yao. Merging user social network into the random walk model for better group recommendation. *Applied Intelligence*, 49(6):2046–2058, 2019.

[7] Xiao Lin, Min Zhang, Yiqun Liu, and Shaoping Ma. Enhancing personalized recommendation by implicit preference communities modeling. *ACM Transactions on Information Systems (TOIS)*, 37(4):48, 2019.

[8] Haekyu Park, Jinhong Jung, and U Kang. A comparative study of matrix factorization and random walk with restart in recommender systems. In *2017 IEEE International Conference on Big Data (Big Data)*, pages 756–765. IEEE, 2017.

[9] Lalita Sharma and Anju Gera. A survey of recommendation system: Research challenges. *International Journal of Engineering Trends and Technology (IJETT)*, 4(5):1989–1992, 2013.

[10] Jing Shi, Bin Wu, and Xiuqin Lin. A latent group model for group recommendation. In *2015 IEEE International conference on mobile services*, pages 233–238. IEEE, 2015.

Mining and Predicting Micro-Process Patterns of Issue Resolution for Open Source Software Projects

Yiran Wang[‡], Jian Cao*[‡] and David Lo[§]

[‡]Department of Computer Science and Engineering, Shanghai Jiaotong University, Shanghai, 200240, China
[§]School of Information Systems, Singapore Management University, 178902, Singapore
Email: [‡]{wangyiran33,cao-jian}@sjtu.edu.cn, [§]davidlo@smu.edu.sg

Abstract—**Addressing issue reports is an integral part of open source software (OSS) projects. Although several studies have attempted to discover the factors that affect issue resolution, few pay attention to the underlying micro-process patterns of resolution processes. Discovering these micro-patterns will help us understand the dynamics of issue resolution processes so that we can manage and improve them in better ways. Of the various types of issues, those relating to corrective maintenance account for nearly half hence resolving these issues efficiently is critical for the success of OSS projects. Therefore, we apply process mining techniques to discover the micro-patterns of resolution processes for issues relating to corrective maintenance. Four and five typical patterns are found for the identification stage and solving stage of the resolution processes respectively. Furthermore, it is shown that the consequent patterns can be predicted with a certain degree of accuracy by selecting the appropriate features and models. Furthermore, we make use of the pattern information predicted to forecast the issue lifetime and the results show that this information can also improve the accuracy in the earlier observation points. At the same time, pattern predictions provide good interpretability to the forecast of issue lifetime.**

Index Terms—**issue resolution, micro-pattern, process mining, issue pattern prediction, issue lifetime prediction**

I. INTRODUCTION

Issue registering, tracking and resolution are very important in open source software (OSS) projects [1]. Previous studies show that problems of issue overstocking may arise over time [2] and a considerable portion of issues in GitHub are pending for months or even over a year [3]. Therefore, effective strategies should be implemented to improve the issue resolution process such as prioritizing issues to be resolved and allocating resources for each issue clearly in practice and more importantly, we should understand the influential factors behind successful issue resolution processes.

As a collaboration process, issue resolution consists of multiple actions that occur in order and involves several persons. It is beneficial for us to understand issue resolution from the process view. For example, bottlenecks, the critical paths and the most frequent paths can be identified and studied when issue resolution is modeled as a process. Predicting the resolution time is not sufficient to understand and manage the dynamic issue resolution process that is full of uncertainties. Therefore, we aim to provide a richer prediction on the ongoing process.

Issue resolution is essentially a problem-solving process and follows a typical problem-solving procedure that includes steps consisting of defining the problem, brainstorming the ideas, deciding on a solution, implementing a solution and reviewing the results. However, these steps can only roughly describe the issue resolution process and in practice, there are many details for each step. The detailed typical process models for different steps or stages are called ***micro-process patterns*** or ***micro-patterns*** in this paper. On open source hosting platforms such as GitHub, different types of events, including development events and interaction events, are recorded in event logs. We can mine micro-patterns for issue resolution processes from these event logs by applying process mining techniques [4].

The value of process mining in OSS projects, as been described in [5], is that it not only reveals the variety of processes followed by open source communities, it also helps standardize or improve core activities. Unfortunately, no research has been conducted on the topic of process pattern mining and prediction on issue resolution processes yet. Knowing typical micro-process patterns help developers, managers and stakeholders gain a deeper understanding on the processes of issue resolution. It would be extremely helpful to risk assessment, work prioritization and resource allocation. For example, some patterns cost more time than others so that OSS project members can take actions to guide an issue resolution process to follow more efficient patterns to intentionally speed up the resolution process. At the same time, micro-process pattern information can also be a useful feature for the resolution time prediction model and provides better interpretability to the prediction results.

There are different types of issues and they may have very different micro-process patterns. In this paper, we focus on the issues relating to the maintenance activities. Moreover, the ISO/IEC 14764 standard [6] defines four types of maintenance activities spanning the different motivations that software engineers have while undertaking changes to an existing software system. Issues can be mapped to these maintenance activities through a classification model [7]. In this paper, we only focus on issues relating to corrective maintenance, which corresponds to bugs and accounts for nearly half of all issues.

Therefore, in this paper, we aim at answering the following research questions:

- *RQ1: What are the frequent micro-process patterns of resolution processes of issues relating to corrective main-*

- RQ2: Is it possible to predict which micro-process pattern will appear during the issue's lifetime?
- RQ3: Is pattern information useful for issue lifetime prediction?

In order to answer these questions, firstly, we mine frequent process patterns in different stages of issue resolution using process mining techniques. The distributions and characteristics of these patterns in various projects are analyzed. Then, we construct models for pattern prediction during issue lifetime using dynamic and static features. Finally, we try to utilize the pattern probability predicted as an additional feature for issue lifetime predictions.

II. Related Work

Influencing factors for issue resolution time [2], [7]–[9], bug-fixing time prediction and issue lifetime prediction [10]–[19] have received significant attention in recent years. Weiss et al. [11] predict the time spent on fixing an issue based on the average time of its similar issues. Al-Zubaidi et al. [19] use multi-objective search-based approach to estimate issue resolution time which makes estimation models accurate and simple simultaneously. Giger et al. [12] and Rees-Jones et al. [14] present decision-tree-based models to predict bug fix time while Panjer et al. [13] utilize logistic regression models, Zhang [15] utilize kNN-based model. Kikas [10] predict issue resolution time based on random forest models using dynamic and contextual features. In this work, we also construct issue lifetime prediction models but our emphasis is to show an improvement in the predictors' performance when calculating the probability of micro-process patterns predicted into features.

Process mining is now considered to be in a mature phase allowing its application to extract knowledge from event logs to a variety of sectors. Applying process mining in open-source software communities has seldom been studied [20]–[23]. [21]–[23] uses the characteristics of process mining to perform conformance checks to study the differences between the actual bug life cycle and the standard process on the guide. However, these studies are confined to extracting the overall process model and none of them mine internal micro-process patterns further. In contrast, we try to discover the micro-process patterns in the issue resolution process.

III. Mining Frequent Micro-Patterns of the Issue Resolution Process

In this section, we answer RQ1. In order to mine the micro-process patterns of issue resolution, we need to observe the micro-processes of issues. The events extracted through GitHub APIs record what has happened in an issue resolution process. We perform pre-processing on the raw data and filter some helpful and common events which are shown in I to make an event log made up of 38978 records and apply process mining algorithms on it.

TABLE I
EVENTS OF ISSUES USED IN THIS STUDY

Event name	Description
Created	The issue was created by the actor.
Assigned	The issue was assigned to the actor.
Labeled	A label was added to the issue.
Mentioned	The actor was @mentioned in an issue body.
Referenced	The issue was referenced from a commit message.
Renamed	The issue title was changed.
Reopened	The issue was reopened by the actor.
Closed	The issue was closed by the actor.

A. Dataset

In this study, we collect issue data using its public APIs[1] from GitHub and only closed issues updated at or after January 1, 2017 are included. The ten popular projects used in this study vary in domain, scale and programming language and they all provide dynamic platforms for bug reporting, discussing and fixing.

Since we focus on issues relating to corrective maintenance, issue reports must be classified first. We rely on labels applied to each issue report in GitHub to identify their maintenance type. Labels used by developers in GitHub are reliable since they are applied by the persons who actually perform the maintenance activity [7]. Finally, 4863 issues relating to corrective maintenance are collected and included in the dataset. We perform a preliminary analysis on the duration distribution for the dataset and find that the majority of issues are closed in a short period while few issues are pending for a long time, which appears to be a typical long-tail distribution.

B. Approach

The main approach we use to discover micro-processes is process mining. This technique has been successfully applied to distill a structured process description from a set of real executions in practice [24]. Its main objective is to discover processes, do conformance checking and process improvement. Many algorithms can be applied to generate process models like α–algorithm, heuristic miner, genetic algorithm. In accordance with these methods, process mining automatically discovers fact-based process models out of the raw data. Therefore, we use process mining to discover micro-process patterns in issue resolution processes. Celonis[2] as a mature process mining tool is used in this work.

C. Findings

The original extracted model is very complex and difficult to read and understand, so we remove a few activities and connections with low frequency to improve the readability of the model. Figure 1 shows the process model with 97.3% activities and 87.2% connections covered. Dashed arrows indicate connections from the Process Start or to the Process End. It can be easily found that the most common process path is :

[1] https://developer.github.com/v3/
[2] https://academiccloud.celonis.com

Fig. 1. The process model with 97.3% activities and 87.2% connections covered

(a) The Identification Process Model

(b) The Solving Stage Process Model

Fig. 2. The two-stage process models

created − > labeled:bug − > closed

This indicates that most issue resolution processes start with 'created' and end with 'closed', via 'labeled:bug' except for those which still have activities after the issue is closed. For this reason, the whole process model can be divided into two stages by the 'labeled:bug' activity:

1) *The First Stage (Identification Stage): 'created' to 'labeled:bug':* In this stage, project contributors inspect the issue in order to know the environment and details with or without further conversations with the issue authors. If they judge the issue is a bug rather than a misuse, they will assign the bug label to the issue for further fixing.

2) *The Second Stage (Solving Stage): 'labeled:bug' to 'closed':* In this stage, when an issue is confirmed as a bug, the actors try to fix it. It may finish directly without any activities, or a contributor or a team may be @mentioned or assigned to fix the found bug, and commits may be made to fix the bug.

Compared with the general problem solving process model, the first stage roughly corresponds to the step of defining the problem while the second stage corresponds to brainstorming the ideas, deciding on a solution, implementing a solution and reviewing the results. For issue resolution processes in OSS projects, it is very difficult if not impossible to divide the second stage into different steps since these steps are interwoven. Figure 2 shows the process models for the two stages, respectively.

Table II presents the most frequent micro-process model variances which exceed 5% of all for each stage. In general, Stage 1 costs much less time than Stage 2, which indicates that it is easier to confirm a bug than to fix it. The bottleneck period usually occurs in Stage 2.

In each stage, the durations differ clearly between patterns. It should be noted that activities with the circle arrow(↺) mean the resolution process goes through the activity twice or more times. The reason why we don't merge it into the process model that goes through the activity only once is that their durations of them differ greatly (See Table II). For example, the third pattern of the 1st stage spends 34.9 more

TABLE II
FREQUENT MICRO PROCESS PATTERNS IN TWO STAGES

Pattern No.	Micro Process Patterns	Freq.	Duration Median	Standard Deviation
The 1st Stage: Identification Stage				
1	created − > labeled:bug	41%	14.2d	69.8d
2	created − > assigned − > labeled:bug	12%	11.4d	52.0d
3	created − > mentioned↺ − >labeled:bug	9%	50.8d	134.0d
4	created − > mentioned − > labeled:bug	8%	15.9d	56.4d
others		30%	57.5d	127.5d
The 2nd Stage: Solving Stage				
1	labeled:bug − > closed	26%	114.7d	226.0d
2	labeled:bug − > mentioned − > closed	10%	134.8d	237.3d
3	labeled:bug − > mentioned ↺ − > closed	17%	189.5d	279.3d
4	labeled:bug − > assigned ↺ − > closed	7%	29.8d	72.0d
5	labeled:bug − > referenced ↺ − > closed	8%	61.5d	142.4d
others		32%	79.5d	232.6d

days(68.7%) than the fourth pattern on average, and the only difference between these two patterns is that the former goes through 'mentioned' activity twice or more while the latter goes through it only once. Going through the 'mentioned' activity twice or more may mean a several rounds of discussion or @mentioning a team and @mentioning a team often costs more time than @mentioning a person when judging a bug. It must be noted that spending more time doesn't necessarily imply inefficiency. The issue lifetime depends on many factors, one being the complexity of an issue. For a complex issue, the project manager may @mentioning a team so that more time is needed to label this issue and this is often the right way.

Another finding is that processes going through the 'assigned' activity in stage 1 or 2 and processes going through the 'referenced' activity in stage 2 will reduce the durations greatly. This conforms to our common sense that explicitly assigning the task to people can increase efficiency and it is extremely likely that committing to a pull request marks the end of issue resolution. It may also mean that when an issue is easy, the manager clearly knows who should be responsible for it and can directly assign it to him or a developer can directly commit a pull request to fix it.

IV. Pattern Prediction

Predicting which pattern occurs next in advance during issue lifetime can provide with a richer and more interpretable forecast result and in this section, we construct models to predict patterns to answer RQ2. The steps include feature selection, model training and evaluation.

A. Feature Selection

The performance of prediction models relies on features that are properly selected. Obviously, the patterns to be followed are affected by many factors. Our feature engineering is based on the work of [10] and [14]. Furthermore, we also add the following new features to the features we chose based on previous work: *CodeIncluded* for whether the body of an issue includes code or not, *CleanedTitleLength* for the number of words in the issue body with markdown parsed and tags removed and *CreatorAuthority* for whether the creator has authoritative identity in the project. It should be noted that since we would like to predict emerging patterns with time, the dynamic features proposed in [10] are used.

The selected features can be divided into three classes, i.e., **Issue features**, **Issue creator features** and **Project features**. Issue features describe the contents of an issue and its related events. Issue creator features reflect the characteristics of the author of an issue, which relates to issue contents and quality. The resolution processes of issues are also be affected by their projects and project features reflects their issue resolution statuses and activity levels. Features are not listed for lack of space.

B. Model Training

The target of pattern prediction is to select the most possible pattern type from a limited number of pattern types. This can be regarded as a classification problem. Since we will predict patterns with time, we trained different classification models at different observation points. For example, the observation point of 1 day means that we make a pattern prediction for an issue that has been opened for 1 day.

The observation points are chosen to match calendric periods, which leads to six observation points (1, 7, 14, 30, 90, and 180 days) . For each observation point, we train two classifiers to predict the pattern for the 1st stage and the 2nd stage respectively. Finally, we train 12 models in total.

TABLE III
THE MACRO-F1(MACF1) AND MICRO-F1(MICF1) SCORES AT DIFFERENT STAGES

metrics		Observation Point					
		1d	7d	14d	30d	90d	180d
Stage 1	macF1	0.617	0.679	0.653	0.591	0.604	0.664
	micF1	0.706	0.741	0.746	0.719	0.756	0.869
Stage 2	macF1	0.469	0.646	0.667	0.674	0.708	0.590
	micF1	0.508	0.626	0.661	0.682	0.713	0.702

C. Pattern Prediction Performance

We trained multiple classifiers including *MultiLayer Perceptron*, *Linear Discriminant Analysis*, *Gaussian Naive Bayes*, *Multinomial Naive Bayes*, *Bernoulli Naive Bayes*, *Logistic Regression*, *Decision Tree* and *Random Forest*. Random forest classifiers [25] perform best for most of the time and we utilize them in the following section. Table III shows the macro-averaged F1-score and micro-averaged F1-score [26] for Random forest classifiers at different observation point.

We calculate yhe Top-10 ranking of feature importance for models at different observation points. The importance distributions of features of different stages at the same observation point are quite similar. However, we can find a huge difference between importance distributions of features in different observation points of the same stage. In early periods, static features seem to play a greater role while at late observation points dynamic features play a major role. It is also shown that 'nMentionedByT' which denotes 'Number of times actors was mentioned in the issue body before T' is always of great importance.

To summarize the findings with respect to RQ2, it can be concluded that we can predict which pattern has the highest probability of appearing during issue lifetime with a certain degree of accuracy by selecting appropriate features and models.

V. Issue lifetime Prediction with Micro-Process Pattern Information

In order to answer RQ3, we construct models with micro-process pattern information in contrast to models without pattern information to show that predicting patterns with time is beneficial to lifetime prediction.

A. Feature Selection

The feature selection is similar to last section. In addition, predicted pattern information is used as the extra inputs to the models. Rather than providing the concrete predicted pattern information, here we provide the appearance probabilities of patterns as inputs, for example, [0.12, 0.21, 0.03, 0.64] for the 1st stage. The reason is the consequent pattern is essentially nondeterministic and can be changed with time due to several factors. This is also the reason why pattern prediction accuracy is not so high. Therefore, providing the name of a most possible pattern is too risky. If the predicted result is wrong, it will completely mislead the lifetime prediction model.

Pred.	macro-F1			micro-F1			AUC		
horizon	*init*	*prob*	*fore*	*init*	*prob*	*fore*	*init*	*prob*	*fore*
Observation at 1 day after issue is opened									
7d	0.524	**0.576**	0.554	0.740	**0.785**	0.759	0.531	**0.571**	0.552
14d	0.575	**0.623**	0.611	0.664	**0.704**	0.686	0.574	**0.618**	0.607
30d	0.661	**0.680**	0.671	0.662	**0.682**	0.680	0.658	**0.678**	0.672
90d	**0.696**	0.695	0.696	0.720	**0.726**	0.722	0.691	0.690	**0.692**
180d	**0.741**	0.728	0.737	0.803	**0.804**	0.803	**0.724**	0.708	0.720
365d	0.774	0.772	**0.775**	0.892	**0.894**	0.892	0.740	0.734	**0.741**
Observation at 7 days after issue is opened									
14d	0.508	**0.535**	0.512	0.860	**0.865**	0.854	0.517	**0.538**	0.517
30d	0.647	**0.669**	0.673	0.738	**0.761**	0.758	0.640	**0.667**	0.661
90d	0.673	**0.693**	0.680	0.676	**0.698**	0.680	0.678	**0.690**	0.680
180d	0.739	0.736	**0.746**	0.762	**0.765**	0.770	0.732	0.729	**0.740**
365d	0.778	0.756	**0.783**	0.864	0.852	**0.866**	0.750	0.727	**0.754**
Observation at 14 days after issue is opened									
30d	0.609	**0.637**	0.633	0.796	**0.820**	0.811	0.596	**0.618**	0.614
90d	0.660	**0.667**	0.661	0.667	**0.677**	0.670	0.657	**0.661**	0.661
180d	**0.744**	0.737	0.740	0.750	0.748	**0.751**	0.737	0.734	**0.737**
365d	0.751	0.750	**0.765**	0.839	0.836	**0.845**	0.730	0.725	**0.739**
Observation at 30 days after issue is opened									
90d	0.624	**0.626**	0.613	0.741	**0.743**	0.733	0.617	**0.620**	0.609
180d	0.712	**0.730**	0.716	0.716	**0.732**	0.718	0.713	**0.733**	0.717
365d	**0.749**	0.735	0.743	**0.790**	0.784	0.787	**0.738**	0.722	0.732
Observation at 90 days after issue is opened									
180d	**0.590**	0.584	0.554	0.724	**0.729**	0.721	0.578	**0.580**	0.566
365d	**0.703**	0.702	0.694	**0.716**	0.713	0.710	**0.704**	0.702	0.695
Observation at 180 days after issue is opened									
365d	0.678	0.683	**0.691**	0.692	0.708	**0.725**	0.679	0.689	**0.691**

(a) macro-F1

(b) micro-F1

Red ■■■ denotes that a model with pattern probabilities performs better than initial model and blue ■■■ denotes the contrary. A dark color ■■ denotes the gap is more than 6%, the medium ■■ denotes the gap is between 3%-6% and a light color ■■ denotes the gap is less than 3%.

B. Model Training

As in [10], our lifetime prediction tries to answer the question as to whether the issue can be closed before the given time or not. Therefore, it is a classification problem and we train prediction models at different observation points. At each observation point, we make predictions for whether

it will be closed with different prediction horizons. Naturally, the prediction horizon should end after the observation point. For each combination of an observation point and a prediction horizon, we should train one model. For example, we make a prediction for an issue to answer the question as to whether it will be closed within 30 days after it has been opened for 14 days.

C. Prediction Performance

Although the prediction task is a binary classification problem in our study, we still utilize the macro-averaged F1-score and micro-averaged F1-score to evaluate the classifiers because correctly predicting the fact that an issue can be closed in a given prediction horizon or not is equally important so that traditional metrics for binary-classification are not sufficient. As in [27], we also use random forest as the model for this task because it has been proven that random forest is better than other conventional models on this task.

We analyze model performance for different observation points and compare initial models ($init$), models with pattern probability predicted ($prob$) and models with exact pattern predicted ($fore$). Table IV shows the obtained macro-averaged and micro-averaged F1-scores of various models.

Accordingly, Table V shows the performance comparison in the form of a heat map. We find that models with pattern probability predicted achieve better performance when doing short-term prediction at earlier observation points.

Firstly, pattern probability is similar to prior probability. At an early period when other features can barely provide information, adding prior probability is of great help for classifiers. As time progresses, other features are of more value and begin to modify the 'prior probability' and even break away from it. So, the value of pattern probability become less and less at later observation points.

Secondly, we find that after adding pattern probability features, the classifiers tend to predict that a certain issue can be closed within a given period. Without pattern probability, issue lifetime predicted may range widely. But with pattern probability, the resolution time distribution predicted may shrink due to the pattern restrictions. In this case, the capability of the model to distinguish the unconventional ultra-long period issue becomes weak, so prediction performance deteriorates after adding pattern features.

Another interesting finding is that models with pattern probability predicted even perform better than models with the foresight of exact patterns in many cases, especially at earlier observation points. We explain the phenomenon in this way: The model precision is low at an early period, adding an exact pattern may result in over-fitting to some extent. Nevertheless, the probability of patterns means various possibilities, which improves the ability of generalization.

VI. THREATS TO VALIDITY

Threats to internal validity In this study, we assume that properties of issues in each project are uniformly distributed while the heterogeneity of issues in each project cannot

be avoided. In addition, there are may several relationships between some issues in a project, i.e., they are not independent. These will inevitably affect our results to some extent. Another issue is we apply 17 features to predict possible patterns while the most important factor, the complexity of the issue itself, is not included since it is difficult to measure directly. We have tried to remedy this by putting some of the features that are closely related to issue complexity into the model. Also, issue misclassification is reported to occurs in [28], which may have impact on issue pattern prediction and issue lifetime prediction in our study.

Threats to external validity In this study, we use issues from 10 projects that are representatives of the open source domain which have different backgrounds, development practices and goals. To improve generality, we propose extending our study to more representative projects.

VII. Conclusions

In this paper we try to mine micro-process patterns of the resolution process of issues of a corrective maintenance type. Based on the issues extracted from 10 distinctive and representative projects in the open source domain, we apply process mining techniques to extract process patterns from them. We divide the whole issue solving process into two stages, i.e., the identifying stage and the solving stage. Four and five patterns are discovered for the first stage and second stage, respectively. Then we analyze their properties and get some interesting findings. After this, we construct models for pattern prediction during issue lifetime using static and dynamic features and our model shows an improved performance with time. Then we construct models for issue lifetime prediction in GitHub projects for different calendric periods with the probability of patterns predicted in order to demonstrate value within pattern information. The results show that models with predicted pattern information achieve better accuracy for issue lifetime prediction at earlier observation points.

References

[1] D. Bertram, A. Voida, S. Greenberg, and R. Walker, "Communication, collaboration, and bugs: the social nature of issue tracking in small, collocated teams," in *Proceedings of the 2010 ACM conference on Computer supported cooperative work*. ACM, 2010, pp. 291–300.

[2] Z. Liao, D. He, Z. Chen, X. Fan, Y. Zhang, and S. Liu, "Exploring the characteristics of issue-related behaviors in github using visualization techniques," *IEEE Access*, vol. 6, pp. 24 003–24 015, 2018.

[3] R. Kikas, M. Dumas, and D. Pfahl, "Issue dynamics in github projects," in *International Conference on Product-Focused Software Process Improvement*. Springer, 2015, pp. 295–310.

[4] W. Van Der Aalst, *Process mining: discovery, conformance and enhancement of business processes*. Springer, 2011, vol. 2.

[5] E. Kouzari and I. Stamelos, "Process mining in software events of open source software projects," in *2nd International Symposium & 24th National Conference on Operational Research, HELORS*, 2013, pp. 25–27.

[6] ISO/IEC, "International standard-iso/iec 14764 ieee std 14764-2006 software engineering; software life cycle processes &; maintenance," 2006.

[7] A. Murgia, G. Concas, R. Tonelli, M. Ortu, S. Demeyer, and M. Marchesi, "On the influence of maintenance activity types on the issue resolution time," in *Proceedings of the 10th international conference on predictive models in software engineering*. ACM, 2014, pp. 12–21.

[8] G. Destefanis, M. Ortu, S. Counsell, S. Swift, M. Marchesi, and R. Tonelli, "Software development: do good manners matter?" *PeerJ Computer Science*, vol. 2, p. e73, 2016.

[9] F. Zhang, F. Khomh, Y. Zou, and A. E. Hassan, "An empirical study on factors impacting bug fixing time," in *2012 19th Working Conference on Reverse Engineering*. IEEE, 2012, pp. 225–234.

[10] R. Kikas, M. Dumas, and D. Pfahl, "Using dynamic and contextual features to predict issue lifetime in github projects," in *Proceedings of the 13th International Conference on Mining Software Repositories*. ACM, 2016, pp. 291–302.

[11] C. Weiss, R. Premraj, T. Zimmermann, and A. Zeller, "How long will it take to fix this bug?" in *Fourth International Workshop on Mining Software Repositories (MSR'07: ICSE Workshops 2007)*. IEEE, 2007, pp. 1–1.

[12] E. Giger, M. Pinzger, and H. Gall, "Predicting the fix time of bugs," in *Proceedings of the 2nd International Workshop on Recommendation Systems for Software Engineering*. ACM, 2010, pp. 52–56.

[13] L. D. Panjer, "Predicting eclipse bug lifetimes," in *Proceedings of the Fourth International Workshop on mining software repositories*. IEEE Computer Society, 2007, p. 29.

[14] M. Rees-Jones, M. Martin, and T. Menzies, "Better predictors for issue lifetime," *arXiv preprint arXiv:1702.07735*, 2017.

[15] H. Zhang, L. Gong, and S. Versteeg, "Predicting bug-fixing time: an empirical study of commercial software projects," in *Proceedings of the 2013 international conference on software engineering*. IEEE Press, 2013, pp. 1042–1051.

[16] P. Bhattacharya and I. Neamtiu, "Bug-fix time prediction models: can we do better?" in *Proceedings of the 8th Working Conference on Mining Software Repositories*. ACM, 2011, pp. 207–210.

[17] P. Anbalagan and M. Vouk, "On predicting the time taken to correct bug reports in open source projects," in *2009 IEEE International Conference on Software Maintenance*. IEEE, 2009, pp. 523–526.

[18] G. Canfora, M. Ceccarelli, L. Cerulo, and M. Di Penta, "How long does a bug survive? an empirical study," in *2011 18th Working Conference on Reverse Engineering*. IEEE, 2011, pp. 191–200.

[19] W. H. A. Al-Zubaidi, H. K. Dam, A. Ghose, and X. Li, "Multi-objective search-based approach to estimate issue resolution time," in *Proceedings of the 13th International Conference on Predictive Models and Data Analytics in Software Engineering*. ACM, 2017, pp. 53–62.

[20] E. Kouzari, L. Sotiriadis, and I. Stamelos, "Process mining for process conformance checking in an oss project: An empirical research," in *IFIP International Conference on Open Source Systems*. Springer, 2018, pp. 79–89.

[21] E. Kouzari and I. Stamelos, "Process mining in software events of open source software projects," in *2nd International Symposium & 24th National Conference on Operational Research, HELORS*, 2013, pp. 25–27.

[22] W. Poncin, A. Serebrenik, and M. Van Den Brand, "Process mining software repositories," in *2011 15th European Conference on Software Maintenance and Reengineering*. IEEE, 2011, pp. 5–14.

[23] M. Gupta, "Nirikshan: process mining software repositories to identify inefficiencies, imperfections, and enhance existing process capabilities," in *Companion Proceedings of the 36th International Conference on Software Engineering*, 2014, pp. 658–661.

[24] W. M. van der Aalst, H. A. Reijers, A. J. Weijters, B. F. van Dongen, A. A. De Medeiros, M. Song, and H. Verbeek, "Business process mining: An industrial application," *Information Systems*, vol. 32, no. 5, pp. 713–732, 2007.

[25] A. Liaw, M. Wiener *et al.*, "Classification and regression by randomforest," *R news*, vol. 2, no. 3, pp. 18–22, 2002.

[26] V. Van Asch, "Macro-and micro-averaged evaluation measures [[basic draft]]," *Belgium: CLiPS*, pp. 1–27, 2013.

[27] M. Fernández-Delgado, E. Cernadas, S. Barro, and D. Amorim, "Do we need hundreds of classifiers to solve real world classification problems?" *The Journal of Machine Learning Research*, vol. 15, no. 1, pp. 3133–3181, 2014.

[28] P. S. Kochhar, T.-D. B. Le, and D. Lo, "It's not a bug, it's a feature: does misclassification affect bug localization?" in *Proceedings of the 11th Working Conference on Mining Software Repositories*. ACM, 2014, pp. 296–299.

Deep Graph Attention Neural Network for Click-Through Rate Prediction

Wen Fang[1] and Lu Lu[1,2*]

[1]School of Computer Science and Engineering, South China University of Technology, Guangzhou, China
[2]Meizhou Modern Industrial Technology Research Institute, South China University of Technology, Meizhou, China
*Corresponding author email: lul@scut.edu.cn

Abstract—Click-through rate (CTR) prediction aims to estimate the probability of a user clicking on the item, which has critical importance both in advertising system and recommender system. Recently, deep learning-based methods have emerged due to its strong feature extraction ability. Learning user/item representations (aka. embeddings) is the core of these methods. However, these existing efforts pay little attention to encoding the relations among users and items in the embedding process, which limits the embedding effectiveness. In this paper, we propose a novel graph neural network framework for CTR prediction, namely the deep graph attention neural network (DGAN). Specifically, DGAN treats user-item interactions as a bipartite graph, which can naturally integrate node information and topological structure for modeling the relations. The key component of DGAN is attentive embedding propagation that recursively propagates embeddings from a node's neighbors to refine the node's embedding, and exploits graph attention mechanism to determine which neighbors to focus on. Comprehensive experiments are conducted on three public datasets and empirical results demonstrate DGAN achieves substantial gains compared with the mainstream models for CTR prediction.

Keywords—Click-Through Rate Prediction, Recommender System, Deep Learning, Graph Neural Network, Embedding Propagation

I. INTRODUCTION

Online advertising was born in the last century, which has become the major profit means of most internet companies. With the explosion of information on the internet, it is becoming increasingly important to explore the way to accurately and efficiently predict user behaviors with limited network resources. Click-through rate (CTR) refers to the ratio of ad clicks to ad impressions, which reflects user behaviors and serves as a key indicator to measure the advertising effectiveness. As such, predicting CTR becomes a critical task, which is beneficial to optimize marketing content, improve advertising effectiveness and ensure the quality of user experience.

Considering the superiority of deep learning, such as automatic high-order feature extraction, and inspired by its immense success in computer vision [1], speech recognition [2] and natural language processing [3], deep learning-based methods have been proposed to conduct CTR prediction task [4]–[6]. Compared with many previous works [7], [8], these deep learning-based methods can avoid a lot of manual

Fig. 1: An example of the user-item interaction graph.

feature engineering jobs and improve the model performance substantially.

Despite the great success achieved by these existing efforts, we argue that they are insufficient to learn effective embeddings for users and items. The key reason is that they treat each user-item interaction independently and overlook the latent relations in the interactions. For example, as shown in Figure 1, the path Mary $\rightarrow$ Mirage $\rightarrow$ Mike reveals the similar behavior between Mary and Mike, as they all have interacted with Mirage; the longer path Mary $\rightarrow$ Mirage $\rightarrow$ Mike $\rightarrow$ Blue Sky implies that Mary may be interested in Blue Sky, since her similar user Mike has ever watched Blue Sky. Furthermore, from the holistic view of a path length[1] of 3, Tom is more likely to have an interest in Mirage than Blue Sky, because there are two paths connecting Tom and Mirage, while only one path connects Tom and Blue Sky.

Being aware of aforementioned challenges and inspired by the wide success of leveraging graph neural networks [9]–[11], in this paper, we propose a novel graph neural network framework for CTR prediction, namely the deep graph attention neural network (DGAN). DGAN is a item-based model, which takes user behavior sequence and a candidate item as input, and outputs the probability of a user clicking the item. Specifically, we first represent user-item interactions as a bipartite graph, and then refine representation of each input item by recursively propagating the embeddings from its neighbors. Distinct from existing work [12], which treats

---

[1]In this paper, the length of a path refers to the number of edges it contains. For example, the length of path Mary $\rightarrow$ Mirage $\rightarrow$ Mike is 2 and the length of path Mary $\rightarrow$ Mirage $\rightarrow$ Mike $\rightarrow$ Blue Sky is 3.

neighbors equally, here we employ a graph attention mechanism to discriminate the importance of different neighbors. By doing so, latent relations among users and items are encoded in the process of embedding propagation.

After several embedding propagation layers, we obtain a refined representation vector for each item. To yield an adaptive representation of user interests with respect to current candidate item, we use an attention network to dynamically calculate the correlation between candidate item and historical behaviors, and take weighted sum to aggregate the user behavior sequence. User interests embedding and candidate item's embedding are finally fed into an interaction layer for CTR prediction.

The key contributions of this work are summarized as following:

- We emphasize the critical importance of encoding relations among users and items in the embedding process of deep learning-based models.
- We propose DGAN, a novel end-to-end CTR prediction framework based on graph attention network, which explicitly models relations among users and items by conducting embedding propagation and employs an attention module to capture the diversity characteristic of user interests.
- To validate the efficacy of DGAN, we conduct empirical studies on three real-world public datasets. The experimental results show our DGAN outperforms other mainstream models in the CTR prediction task.

The remaining parts of this paper are organized as follows. Section II reviews some related works. Section III describes proposed model DGAN in detail. Section IV presents the experiments. Section V summarizes this work and discusses future works.

II. RELATED WORK

In this section, we mainly present recent studies of the CTR prediction and graph neural networks.

A. Click-Through Rate Prediction

Compared with traditional shallow learning, deep learning shows great potential on feature representation and combination. As such, more and more researchers apply deep learning to CTR prediction. Qu et al. [13] propose a product-based neural network (PNN) that combines factorization machine with multilayer perceptron (MLP). Cheng et al. [5] propose a novel structure Wide&Deep that cleverly fuses the linear model and the deep neural network. Guo et al. [14] propose a factorization-machine based neural network (DeepFM), which employs factorization machines and deep neural network to model low-order and high-order feature interactions. Shan et al. [15] propose a Deep Crossing model composed of an embedding layer, a stacking layer and a cascade of residual units for CTR prediction. Zhu et al. [16] propose a Deep Embedding Forest (DEF) based on Deep Crossing, which replaces the residual units in Deep Crossing by a forest layer

and improves prediction efficiency by pre-training. Zhou et al. [17] propose a Deep Interest Network (DIN), which considers the lack of modeling of user behavior diversity and local activation in most CTR prediction studies. Zhou et al. [18] propose Deep Interest Evolution Network (DIEN) to model users' sequential behaviors, which enriches the representation of users and improves the prediction accuracy significantly. Feng et al. [19] propose a novel CTR model named Deep Session Interest Network (DSIN) that leverages users' multiple historical sessions in their behavior sequences. Despite the substantial gains achieved by these efforts, little attention has been paid to encoding relations among users and items in the embedding process, which degrades the model performance. In this paper, we employ graph neural network to fill this gap.

B. Graph Neural Networks

Graph Neural Network (GNN) is an extension of convolutional neural network to process underlying data with irregular structure (such as graphs). Kipf et al. [20] propose a convolutional architecture for semi-supervised learning on graph-structured. Velickovic et al. [21] propose an attention-based GNN architecture for node classification. Recently, researchers also apply GNN to recommender systems. Berg et al. [22] propose GC-MC, which employs GNN to learn representations of users/items on user-item graph, but only first-order neighbors are considered. Wang et al. [12] propose NGCF, which incorporates GNN into collaborative filtering and recursively performs propagation on user-item graph to capture the collaborative signal in high-order connectivity. However, neither GC-MC nor NGCF discriminate the importance of different neighbors. In this paper, we exploit the idea of graph attention network [21] to tackle this problem.

III. METHODOLOGY

In this section, we illustrate our proposed Deep Graph Attention Neural Network (DGAN) framework in detail, whose overall structure is shown in the Figure 2. We discuss the three main components: 1) embedding layer; 2) attentive embedding propagation layers; 3) attentive user interest extraction layer, respectively.

A. Embedding Layer

Embedding is a widely used technique to transform large scale sparse features into low-dimensional dense vectors, which has been used in many mainstream recommender models [12], [17]. With embedding, users can be represented as $E_u = [e_{u_1}, \cdots, e_{u_m}] \in R^{m \times d}$, where m is the number of users, d is the embedding size. Analogously, we can represent items as $E_i = [e_{i_1}, \cdots, e_{i_n}] \in R^{n \times d}$, where n is the number of items.

B. Attentive Embedding Propagation Layers

We take inspiration from the recent advance of GNNs [21], [23]. First, we introduce single layer propagation, and then exploit the idea of [12] to perform multiple layers propagation.

Fig. 2: An illustration of the DGAN model framework.

1) Single-layer Propagation: Obviously, histories of user-item interactions not only directly reflect a user's interests, but also represent the features of item to a certain extent [24]. We perform embedding propagation between user/item and its neighbors to encode these relations. For user-item interaction pair (u, i), embedding propagated from u to i is defined as:

$$m_{i \leftarrow u} = attn(e_i, e_u)e_u, \tag{1}$$

where $attn(\cdot)$ is the graph attention function that takes embeddings e_i and e_u as input and outputs the attention coefficients α_{iu}, reflecting the importance of local neighbors on e_i. Distinct from traditional graph attention network [21], which simply concatenates e_i, e_u and feeds it into a single-layer feedforward neural network to compute coefficients α_{iu}, we additionally model user-item interaction via $e_i \odot e_u$, where $\odot$ denotes the Hadamard product. Inspired by DKN [25] multi-channel mechanism, we treat user embedding e_u , item embedding e_i , user-item interaction embedding $e_i \odot e_u$ as three channels, and concatenate the three embedding matrices as

$$T_{iu} = [W_1 e_i || W_2 (e_i \odot e_u) || W_3 e_u], \tag{2}$$

where $W_1, W_2, W_3 \in R^{d' \times d}$ are the trainable weight matrices. After getting the multi-channel input T_{iu}, to increase the learning ability, we feed it in double-layer feedforward neural network $\mathcal{D}$:

$$H_{iu} = \mathcal{D}(T_{iu}). \tag{3}$$

To make coefficients easily comparable across different nodes, we normalize them by adopting the softmax function:

$$\beta_{iu} = softmax(\alpha_{iu}) = \frac{exp(H_{iu})}{\sum_{k \in N_i} exp(H_{ik})}, \tag{4}$$

where N_i denotes the neighbors of item i. By assigning different neighbors with a different weight, the attention mechanism is capable of discriminating the importance of different neighbors, so as to precisely capture latent relations among users and items. Therefore, embedding propagated from user u to item i is implemented as:

$$m_{i \leftarrow u} = \beta_{iu} e_u. \tag{5}$$

Given the embeddings propagated from neighbors, we take weighted sum to aggregate them. Specifically, the aggregation function is expressed as:

$$e_i^{(1)} = LeakyReLU(m_{i \leftarrow i} + \sum_{u \in N_i} m_{i \leftarrow u}), \tag{6}$$

$$m_{i \leftarrow i} = W_1 e_i, \tag{7}$$

where $e_i^{(1)}$ denotes the refined embedding of item i after the first propagation layer. Note that we additionally aggregate the original embedding of item i via $W_1 e_i$ (W_1 is the trainable weight matrix defined in Equation 2), whose purpose is to retain the original features of item i. Analogously, we can obtain the refined embedding $e_u^{(1)}$ for user u.

2) Multi-layer Propagation: We argue that the first-order neighbors are not sufficient to encode relations among users and items. Therefore, based on the single-layer propagation, we follow a similar paradigm [12] to gather the embeddings propagated from the longer path neighbors. Specifically, the high-order propagation strategy is represented as follow:

$$e_i^{(l)} = LeakyReLU(m_{i \leftarrow i}^{(l)} + \sum_{u \in N_i} m_{i \leftarrow u}^{(l)}), \quad (8)$$

where l is path length. Similar to Equation 1 and 7, terms in Equation 8 can be expressed as:

$$m_{i \leftarrow u}^{(l)} = attn(e_i^{(l-1)}, e_u^{(l-1)})e_u^{(l-1)}, \quad (9)$$

$$m_{i \leftarrow i}^{(l)} = W_1^{(l)} e_i^{(l-1)}, \quad (10)$$

where $W_1^{(l)} \in R^{d_l \times d_{l-1}}$ is the trainable transformation matrix, d_l is the dimension; $e_i^{(l-1)}$ is the representation of item i yielded from previous $(l-1)$ propagation layers, which further contributes to the representation of item i at layer l. Analogously, the representation for user u at the layer l can be obtained. Hereafter, high-order relation like path Mary $\rightarrow$ Mirage $\rightarrow$ Mike $\rightarrow$ Blue Sky can be extracted in the process of embedding propagation. Such high-order relation is crucial to encode the user's preference.

After propagating with l layers, we obtain a set of embeddings for item i, namely $\{e_i^{(0)}, e_i^{(1)}, \cdots, e_i^{(l)}\}$. Obviously, embeddings obtained from different propagation layers play a different role in representing item i. Towards this end, by exploiting the mechanism of layer-aggregation [23], we concatenate them to constitute the final embedding for item i:

$$e(i) = e_i^{(0)} || e_i^{(1)} || \cdots || e_i^{(l)}, \quad (11)$$

where $||$ is the concatenation operation. In this way, we encode relations among users and items in the embedding process, which not only enrich the initial embeddings, but also endow our model with powerful expressive capability.

C. Attentive User Interest Extraction

Given user u with behavior sequence $\{c_1^u, c_2^u, \cdots, c_{N_u}^u\}$, the embeddings of his clicked items can be expressed as $e(c_1^u), e(c_2^u), \cdots, e(c_{N_u}^u)$. To obtain a representation vector of user interests with respect to current candidate item, a general way [4], [5] is to process the list of embedding vectors via a pooling layer:

$$e(u) = pooling(e(c_1^u), e(c_2^u), \cdots, e(c_{N_u}^u)). \quad (12)$$

Average pooling seems a good choice to achieve above goal, since it simply conducts element-wise average operations of the list of embedding vectors. But the user interests embedding obtained this way remains the same for a specific user, no matter what candidate items are given. However, interests of user with rich behaviors are diverse, and user u's behaviors ought to have different effects on the candidate item t_j when predicting whether user u will click t_j. We model this process by using an attention network [17]. The attention network is illustrated in the left upper part of Figure 2. Specifically, for

each item c_i^u clicked by user u and the candidate item t_j, we first conduct Hadamard product of their representation vectors, then feed it into a feed-forward network $\mathcal{G}$ and use softmax function to normalize the outputs:

$$w_i = softmax(\mathcal{G}(e(c_i^u) \odot e(t_j))). \quad (13)$$

The attention network takes the embedding of candidate item and a clicked item as input and outputs the impact weight. Then, we can obtain the final user interests embedding with respect to the candidate item by calculating the weighted sum of his clicked items embeddings:

$$e(u) = \sum_{i=1}^{N_u} w_i e(c_i^u). \quad (14)$$

Finally, given a user interests embedding $e(u)$ and candidate item embedding $e(t_j)$, we perform inner product to calculate the probability of a user u clicking the candidate item t_j:

$$p_{u,t_j} = e(u)^\mathsf{T} e(t_j). \quad (15)$$

D. Model Optimization

In view of the good performance of binary cross-entropy loss (aka. log loss) in deep recommender models, in this paper, we adopt it as the objective function to optimize model parameters, as follows:

$$Loss = \frac{-1}{N} \sum_{(u,t_j) \in S} (y_{u,t_j} \log p_{u,t_j} + (1 - y_{u,t_j}) \log(1 - p_{u,t_j})), \quad (16)$$

Where S is training set, N denotes the number of samples in S, p_{u,t_j} to be in (0, 1) represents prediction probability of a user u clicking the candidate item t_j, which is calculated by the current model parameters. Target value y_{u,t_j} is a binarized 1 or 0, which denotes whether u has interacted with t_j or not.

IV. EXPERIMENTS

In this section, we present our experiments in detail. We start by introducing experimental datasets, baselines, evaluation metrics, parameter settings, then compare the proposed model with the baselines and analyze the results.

A. Dataset Description

Amazon(Electronics)[2]. Amazon Dataset is a widely used benchmark dataset in E-commerce, which consists of product reviews and metadata from Amazon. We use a subset called Electronics to conduct experiments.

Amazon(Video Games)[3]. Video Games dataset is also a subset of Amazon, which contains rich user behaviors.

Yelp2018[4]. This dataset is adopted from the 2018 edition of the Yelp challenge. Here local businesses like restaurants and cinemas are treated as items. Specially, we take the subset where the timestamp is from Jun, 2017 to Jun, 2018.

---

[2]http://jmcauley.ucsd.edu/data/amazon
[3]http://jmcauley.ucsd.edu/data/amazon
[4]https://www.yelp.com/dataset/challenge

| (a) Electronics-AUC | (b) Video Games-AUC | (c) Yelp2018-AUC |

Fig. 3: Testing performance of DGAN in each iteration.

To ensure the quality of datasets, we filter the original data which keeps each user with at least 10 interactions and generate negative samples which are of equal size with the positive ones. Samples in each dataset are split into 4:1 for training, test. The detailed characteristics of the three datasets are summarized in table I.

TABLE I: Statistics of datasets used in this paper.

Dataset	Users	Items	Samples
Amazon(Electronics)	45,225	61,918	1,547,004
Amazon(Video Games)	24,303	10,673	414,954
Yelp2018	25,937	92,249	796,994

B. Baselines

- **Wide&Deep [5]**. Wide&Deep is widely used in industrial applications, combining a (wide) linear part with a (deep) non-linear part.
- **PNN [13]**. PNN uses a product layer to capture interactive patterns between inter-field categories.
- **DeepFM [14]**. DeepFM is also a general deep model for recommendation, which employs factorization machines and deep neural network to model low-order and high-order feature interactions.
- **DIN [17]**. DIN fully considers user behavior diversity. By exploiting the idea of attention mechanism, it can learn the different representation of users' historical behaviors with respect to the candidate item.

C. Evaluation Metrics

In the experiment, we adopt AUC (Area Under ROC Curve) [26] to evaluate the performance of our framework and baselines. AUC is a widely used metric in CTR prediction field. It reflects the ranking ability of the model, defined as follows:

$$AUC = \frac{1}{m^+ m^-} \sum_{x^+ \in D^+} \sum_{x^- \in D^-} (I(p(x^+) > p(x^-))), \quad (17)$$

where D^+ is the set of all positive samples, D^- is the set of all negative samples, function $p(\cdot)$ outputs the predicted probability of the sample x, $I(\cdot)$ is the indicator function, m^+ is the size of D^+ and m^- is the size of D^-.

D. Parameter Settings

We implement our DGAN model in Tensorflow. The dimension of both user embeddings and item embeddings are set as 32. The number of propagation layers is set as 3. The learning rate is set as 0.01. The batch size is set as 512. We use Adam [27] to train DGAN by optimizing the log loss. The key parameter settings for baselines are as follows. The embedding size is fixed to 128 for all baselines, which results in better performance. Hyperparameters in the baselines are set the same as DGAN. Each experiment is repeated five times, and we report the average performance as results.

TABLE II: Results (AUC) on three public datasets.

Model	Electronics	Video Games	Yelp2018
Wide&Deep	0.7675	0.8137	0.8541
PNN	0.7681	0.8152	0.8597
DeepFM	0.7697	0.8173	0.8684
DIN	0.7712	0.8215	0.8692
DGAN	**0.7782**	**0.8302**	**0.8832**

E. Results

Table II reports the results of comparison of different models. Figure 3 shows the testing performance curve of DGAN. The major findings are summarized as below:

- Wide&Deep performs comparably poorly than other baselines. This indicates that manually designed features are insufficient to extract the representation of items.
- Compared with Wide&Deep, the performance of PNN verifies that automative high-order feature interactions can improve the representation learning ability.
- DeepFM generally performs better than PNN. Such improvement might be attributed to the combination of powerful factorization machines and a specially designed neural network.

- DIN generally achieves remarkable improvements since it uses attention mechanism to model user's diverse behaviors.
- DGAN consistently performs best in the three datasets. Specially, DGAN outperforms the strongest baselines with respect to AUC by 0.91%, 1.06%, and 1.61% in Electronics, Video Games, and Yelp2018, respectively. This is mainly because it considers the modeling of relations among users and items, which is overlooked in most click-through rate prediction studies.

V. Conclusion and Future Work

In this work, we propose a novel framework DGAN, which incorporates graph neural network into recommendation. DGAN addresses two major challenges on the CTR prediction task: 1) Distinct from traditional methods that encode user or item independently, DGAN fully takes relations among users and items into consideration by recursively propagating embeddings on user-item graph structure. 2) With respect to different candidate items, DGAN exploits an attention network to obtain an adaptive embedding vector of user interests. Extensive experiments are conducted on three datasets from Electronics, Video Games, and Yelp2018. The results demonstrate the rationality and efficacy of DGAN over several strong baselines.

In future, we plan to integrate knowledge graph and social networks into recommendation. This side information will be beneficial to understand user behaviors and improve recommendation interpretability. Moreover, with the great success of the Transformer for machine translation task in natural language processing, we will apply self-attention mechanism to investigate the sequential recommendation.

References

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[2] A. Graves, A.-r. Mohamed, and G. Hinton, "Speech recognition with deep recurrent neural networks," in *2013 IEEE international conference on acoustics, speech and signal processing*. IEEE, 2013, pp. 6645–6649.

[3] Y. Shen, X. He, J. Gao, L. Deng, and G. Mesnil, "A latent semantic model with convolutional-pooling structure for information retrieval," in *Proceedings of the 23rd ACM international conference on conference on information and knowledge management*, 2014, pp. 101–110.

[4] P. Covington, J. Adams, and E. Sargin, "Deep neural networks for youtube recommendations," in *Proceedings of the 10th ACM conference on recommender systems*, 2016, pp. 191–198.

[5] H.-T. Cheng, L. Koc, J. Harmsen, T. Shaked, T. Chandra, H. Aradhye, G. Anderson, G. Corrado, W. Chai, M. Ispir *et al.*, "Wide & deep learning for recommender systems," in *Proceedings of the 1st workshop on deep learning for recommender systems*, 2016, pp. 7–10.

[6] R. Wang, B. Fu, G. Fu, and M. Wang, "Deep & cross network for ad click predictions," in *Proceedings of the ADKDD'17*, 2017, pp. 1–7.

[7] T. Graepel, J. Q. Candela, T. Borchert, and R. Herbrich, "Web-scale bayesian click-through rate prediction for sponsored search advertising in microsoft's bing search engine." Omnipress, 2010.

[8] H. B. McMahan, G. Holt, D. Sculley, M. Young, D. Ebner, J. Grady, L. Nie, T. Phillips, E. Davydov, D. Golovin *et al.*, "Ad click prediction: a view from the trenches," in *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2013, pp. 1222–1230.

[9] M. Schlichtkrull, T. N. Kipf, P. Bloem, R. Van Den Berg, I. Titov, and M. Welling, "Modeling relational data with graph convolutional networks," in *European Semantic Web Conference*. Springer, 2018, pp. 593–607.

[10] R. Ying, R. He, K. Chen, P. Eksombatchai, W. L. Hamilton, and J. Leskovec, "Graph convolutional neural networks for web-scale recommender systems," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2018, pp. 974–983.

[11] H. Wang, F. Zhang, M. Zhang, J. Leskovec, M. Zhao, W. Li, and Z. Wang, "Knowledge-aware graph neural networks with label smoothness regularization for recommender systems," in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2019, pp. 968–977.

[12] X. Wang, X. He, M. Wang, F. Feng, and T.-S. Chua, "Neural graph collaborative filtering," in *Proceedings of the 42nd international ACM SIGIR conference on Research and development in Information Retrieval*, 2019, pp. 165–174.

[13] Y. Qu, H. Cai, K. Ren, W. Zhang, Y. Yu, Y. Wen, and J. Wang, "Product-based neural networks for user response prediction," in *2016 IEEE 16th International Conference on Data Mining (ICDM)*. IEEE, 2016, pp. 1149–1154.

[14] H. Guo, R. Tang, Y. Ye, Z. Li, and X. He, "Deepfm: a factorization-machine based neural network for ctr prediction," *arXiv preprint arXiv:1703.04247*, 2017.

[15] Y. Shan, T. R. Hoens, J. Jiao, H. Wang, D. Yu, and J. Mao, "Deep crossing: Web-scale modeling without manually crafted combinatorial features," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2016, pp. 255–262.

[16] J. Zhu, Y. Shan, J. Mao, D. Yu, H. Rahmanian, and Y. Zhang, "Deep embedding forest: Forest-based serving with deep embedding features," in *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2017, pp. 1703–1711.

[17] G. Zhou, X. Zhu, C. Song, Y. Fan, H. Zhu, X. Ma, Y. Yan, J. Jin, H. Li, and K. Gai, "Deep interest network for click-through rate prediction," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2018, pp. 1059–1068.

[18] G. Zhou, N. Mou, Y. Fan, Q. Pi, W. Bian, C. Zhou, X. Zhu, and K. Gai, "Deep interest evolution network for click-through rate prediction," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, 2019, pp. 5941–5948.

[19] Y. Feng, F. Lv, W. Shen, M. Wang, F. Sun, Y. Zhu, and K. Yang, "Deep session interest network for click-through rate prediction," *arXiv preprint arXiv:1905.06482*, 2019.

[20] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *5th International Conference on Learning Representations*, 2017.

[21] P. Velickovic, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph attention networks," in *6th International Conference on Learning Representations*, 2018.

[22] R. v. d. Berg, T. N. Kipf, and M. Welling, "Graph convolutional matrix completion," *arXiv preprint arXiv:1706.02263*, 2017.

[23] K. Xu, C. Li, Y. Tian, T. Sonobe, K.-i. Kawarabayashi, and S. Jegelka, "Representation learning on graphs with jumping knowledge networks," *arXiv preprint arXiv:1806.03536*, 2018.

[24] F. Xue, X. He, X. Wang, J. Xu, K. Liu, and R. Hong, "Deep item-based collaborative filtering for top-n recommendation," *ACM Transactions on Information Systems (TOIS)*, vol. 37, no. 3, pp. 1–25, 2019.

[25] H. Wang, F. Zhang, X. Xie, and M. Guo, "Dkn: Deep knowledge-aware network for news recommendation," in *Proceedings of the 2018 world wide web conference*, 2018, pp. 1835–1844.

[26] T. Fawcett, "An introduction to roc analysis," *Pattern recognition letters*, vol. 27, no. 8, pp. 861–874, 2006.

[27] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

A Session-based Job Recommendation System Combining Area Knowledge and Interest Graph Neural Networks

Yusen Wang[a], Kaize Shi[a], Zhendong Niu[a,b, *]

[a] School of Computer Science and Technology, Beijing Institute of Technology, Beijing 100081, China
[b] School of Computing and Information, University of Pittsburgh, Pittsburgh, PA 15260, USA
wangyusen@bit.edu.cn, kzshi@bit.edu.cn, zniu@bit.edu.cn

Abstract—**Online job boards become one of the central components of the modern recruitment industry. Existing systems are mainly focused on content analysis of resumes and job descriptions, so they heavily rely on the accuracy of semantic analysis and the coverage of content modeling, in which case they usually suffer from rigidity and the lack of implicit semantic relations. In recent years, session recommendation has attracted the attention of many researchers, as it can judge the user's interest preferences and recommend items based on the user's historical clicks. Most existing session-based recommendation systems are insufficient to obtain accurate user vectors in sessions and neglect complex transitions of items. We propose a novel method, Area Knowledge and Interest Graph Neural Networks(AIGNN). We add job area knowledge to job session recommendations, in which session sequences are modeled as graph-structured data, then GNN can capture complex transitions of items. Moreover, the attention mechanism is introduced to represent the user's interest. Experiments on real-world data set prove that the model we proposed better than other algorithms.**

Keywords-component; recommender system; session-based recommendation; GNN

I. INTRODUCTION

Now more and more candidates are looking for suitable jobs through the Internet. The traditional method of searching for keywords is inefficient, and users can only perform self-assessment based on recruitment information to determine whether they meet the post. This method cannot quickly find relevant recruitment data that matches candidates[1]. The recommendation system is a highly automated system that can efficiently recommend the items people need. The recommendation system has been applied in e-commerce[2], personalized advertising recommendation[3], e-learning recommendation[4]. In recent years, job recommendation systems have received increasing attention from researchers[5]. Job recommendation systems can quickly recommend suitable positions for job seekers.

Traditional job recommendation methods are mainly based on collaborative filtering[6], content-based approaches[7], the hybrid method[8]. Now deep learning method is also applied in job recommendation[9]. The core of session-based

recommendation methods is to recommend new items to users based on the changing relationship among items. In job recommendation, people need to consider not only the position but also the workplace. Specific knowledge shows its advantages in multiple tasks[10][11]. In this paper, we introduce area knowledge and consider the regional characteristics in job recommendations to further improve the performance of the model. In real life, the user's behavior can be expressed as a sequence. The user's recent behavior sequence can be regarded as the user's current preferences, and the user's early historical session information implies the user's previous long-term interest preferences. Since the user's long-term preferences will change with time, so the current preference of the user cannot correctly reflect the real situation of the user to a certain extent. The previous job recommendation has weakened the influence of the difference between long-term and short-term interests on job seekers. This paper realizes more accurate job recommendations for users by considering the weight difference among long-term, short-term and global interests of job seekers. In this paper, we propose a job session recommendation model that combines area knowledge and interest graph neural networks(AIGNN).

II. RELATED WORK

A. Conventional Recommendation Methods

The matrix factorization is a common method in recommendation system. The basic objective is to factorize into two low-rank matrices according to a user-item rating matrix, and each matrix represents a latent factor of users or items[12]. This method is not suitable for the session-based recommendation system because the user preference is only provided by some positive clicks. The item-based neighborhood methods[13], this method is difficult to consider the order relationship among items. Markov decision processes[14], mainly learns the probability of state transition. The problem is that as items increase, it is very tough to model all possible click sequences.

B. Deep-learning-based Methods

Hidasi et al.[15] used RNN to form a deep neural network to predict the probability of the next clicked item in the session.

Tan et al.[16] improved the recurrent network model by using appropriate data augmentation techniques and taking into account temporal changes in user behavior. Tuan et al.[17] proposed using a 3D convolutional neural network to better model user-item interaction data and content features in sequence recommendation. Li et al.[18] pointed out that the previous method only considered the user's sequence performance, but the primary purpose of the user was not clearly emphasized. Hence, he proposed to adopt an attention mechanism on RNN to capture the sequential behavior characteristics and the main purpose of the user. Liu et al.[19] proposed a session recommendation model using MLP networks and current attention.

C. Graph Neural Network

Nowadays, graph neural networks can generate graph structure data representation. GNN [20] can represent the dependency among graph nodes. In recent years, GNN has some variants such as Gated graph neural network (GGNN)[21] and Graph Attention Network (GAT) [22]. Wu et al.[23] proposed using the GNN method to extract complex changes among items in session recommendation and achieved good results.

III. METHODS

In this section, we introduce the proposed AIGNN, in which the model is shown in Fig 1. We describe the AIGNN method thoroughly.

A. Notations

The goal of the session recommendation is to predict the most likely clicked item for the user's next step based on the user's previous session order. Here we define the letters that appear in AIGNN.

In session-based recommendation, let $J = \{j_1, j_2, j_3, ..., j_m\}$ represents a set consisting of all unique items involved in all sessions. Each item contains the job knowledge clicked by the user and the area knowledge of the position. $B = \{b_1, b_2, b_3, ..., b_m\}$ represents knowledge for each job. $A = \{a_1, a_2, a_3, ..., a_m\}$ represents the area knowledge of each position. Fuse the information of A and B to form J, as shown in Eq. (1). An anonymous session sequence s can sort by timestamp to get list $s = [j_{s,1}, j_{s,2}, ..., j_{s,n}]$, Where $j_{s,i} \in J$ represents the user's clicked item in session s. The goal of the model is to predict the next project $j_{s,n+1}$ based on the previous clicked item of the user's. In each session, we calculate the probability $\hat{y}$ corresponding to each item and output the top-20 items as recommended items.

$$J = \begin{bmatrix} j_1 \\ j_2 \\ j_3 \\ ... \\ j_m \end{bmatrix} = \begin{bmatrix} a_1 \ b_1 \\ a_2 \ b_2 \\ a_3 \ b_3 \\ ... \\ a_m \ b_m \end{bmatrix} \qquad (1)$$

B. Session Graph

Since each item is a natural sequence, we construct an ordered session according to the click order of each item, and the ordered session sequence is converted into a session graph so that the GNN can process each session. We build each session s into a directed graph $\mathcal{G}_s = (j_s, \varepsilon_s)$. Each node in the session graph is an item $j_{s,i} \in J$. ε_s stands for all directed edge sets. $(j_{s,i-1}, j_{s,i}) \in \varepsilon_s$ $j_{s,i}$ represents the item clicked after clicking $j_{s,i-1}$. The weight of each edge is based on the occurrence of the edge divided by the outdegree of that edge's start node. We construct the vector of each session s according to the vector of each node item.

Figure 1.   The overview of the proposed model.

C. Item Embedding

GNN can learn the complex relationship transformation of each node in the graph structure[20]. Gated graph neural network (GGNN) is a model based on GRU's spatial domain message passing[21]. It uses a similar principle of RNN to realize the transfer of information in a graph. In the field of job recommendation, in addition to considering the changing relationship among jobs, it is also necessary to consider the change of workplace. There are regional differences in job areas. Different regions focus on different types of jobs. Therefore, we have added the consideration of the regional factors to general session recommendations. Besides, we also consider the current, long-term, and global preferences of users to better analyze their behavior characteristics.

In this paper, we use GGNN to generate each item containing area knowledge. A connection matrix A_s is constructed to determine how each node communicates with each other, $A_s \in \mathbb{R}^{n \times 2n}$. Since each node of the calculation considers the bidirectional transfer relationship of the nodes, the connection matrix is $n \times 2n$ dimension as shown in Fig 2. $A_{s,i} \in \mathbb{R}^{1 \times 2n}$ represents the connection matrix corresponding to each node in A_s. H is the weight, $z_{s,i}^t$ and $r_{s,i}^t$ are reset and update gates, $\sigma(\cdot)$ is the sigmoid function. The specific formula is as follows:

$$a_{s,i}^t = A_{s,i} : [j_1^{t-1}, ... j_n^{t-1}]H + b \qquad (2)$$

$$z_{s,i}^t = \sigma(W_z a_{s,i}^t + U_z j_i^{t-1}) \qquad (3)$$

$$r_{s,i}^t = \sigma(W_r a_{s,i}^t + U_r j_i^{t-1}) \tag{4}$$

$$\widetilde{j_i^t} = \tanh(W_o a_{s,i}^t + U_o(r_{s,i}^t \odot j_i^{t-1})) \tag{5}$$

$$j_i^t = (1 - z_{s,i}^t) \odot j_i^{t-1} + z_{s,i}^t \odot \widetilde{j_i^t} \tag{6}$$

The gated graph neural network processes $\mathcal{G}_s$ for each session, and GGNN extracts the latent vectors of neighborhoods into the neural network. The reset gate is used to control the degree of ignoring the state information of the previous moment. The update gate is used to control the degree to which the state information of the previous moment is brought into the current state. Then we calculate the newly generated messages $\widetilde{j_i^t}$ based on the status of the previous, current, and reset gates. $(1 - z_{s,i}^t)$ is to select the forgotten information, j_i^t is the final updated node state.

D. Session Embedding

For a user's session representation, we consider the user's global representation, long-term interest representation, and current interest representation. The global preference takes the user's every click item. The long-term interest is obtained by averaging all the user's click information. The current interest is the user's last clicked item. The formula is as follows:

$$j_a = \frac{1}{m} \sum_{i=1}^m j_i \tag{7}$$

$$\beta_i = q^T \sigma(W_1 j_n + W_2 j_i + W_3 j_a + C) \tag{8}$$

$$j_o = \sum_{i=1}^m \beta_i j_i \tag{9}$$

$$J_g = W_4[j_n; j_a; j_o] \tag{10}$$

j_a is the average of the sum of users' long-term interest preferences according to their click items. $q \in \mathbb{R}^d$ and j_n is the user's last click as the user's current interest. j_o is the global embedding of the session graph $\mathcal{G}_s$. We use the attention mechanism to obtain the over-all preferences of users better.

Finally, we get the user's global preference, long-term preference, and current preference to obtain J_g through a linear change.

Figure 2.   Connection matrix As representation

E. Recommendation

We calculate the integrated J_m and each item j_i to get the score $\widehat{s_i}$ of each candidate item and then send the candidate

score of each candidate item to softmax to calculate $\hat{y}$, which is the probability output of the next click in the click session. The formula is as follows:

$$\widehat{s_i} = J_g^T j_i \tag{11}$$
$$\hat{y} = softmax(\hat{s}) \tag{12}$$

F. Training

For each session, we use cross-entropy as the training loss function. Calculate loss by prediction and the ground truth, as shown below:

$$\mathcal{L}(\hat{y}) = -\sum_{i=1}^n y_i \log(\widehat{y_i}) + (1 - y_i)\log(1 - \widehat{y_i}) \tag{13}$$

IV.   EXPERIMENTS

A. Datasets

We verify the true validity of the AIGNN model recommendation in the job area. We use the real datasets CareerBuilder [1]. This is a real recruitment dataset, which contains user information, job information, etc. We selected 389,708 users, 10,913 job places, and a total of 315,105 jobs. We use 1023456 training sessions for training and 57858 test session data.

B. Baseline Methods

To evaluate the performance of the proposed method, we compare it with the following representative baselines: (1) POP: This is a simple baseline that ranks items according to their popularity measured by the number of interactions. (2) GRU4REC[15]: Session recommendation via RNN. (3) NARM[18]: The RNN with attention mechanism is used to capture the primary purpose and subsequent behavior of the user. (4) SR-GNN[23]: Session recommendation using gated graph neural network and attention mechanism.

C. Evaluation Metrics

Following metrics are used to evaluate compared methods. **Recall@20**: It is widely used as a measure of prediction accuracy. It represents the proportion of correctly recommended items in the top-20 items.
MRR@20: It is the average of reciprocal ranks of the correctly-recommended items. The reciprocal rank is set to 0 when the rank exceeds 20.

V.   RESULTS AND ANALYSIS

The results with Recall@20 and MRR@20 on the recommendation performance are presented in Table I. We set the dimensionality of latent vectors d = 100, the learning rate is set to 0.001 and the L2 penalty is set to 10^{-5}.

TABLE I.   PERFORMANCE COMPARISONS OF DIFFERENT METHODS ON THE SEQUENTIAL RECOMMENDATION TASK IN CAREERBUILDER

Method	Recall@20	MRR@20
POP	4.143	3.702

---

1 https://www.kaggle.com/c/job-recommendation/data

GRU4REC[15]	52.815	13.684
NARM[18]	63.181	17.572
SR-GNN[23]	68.324	19.729
AIGNN	**77.595**	**21.832**

For traditional algorithms such as POP, the performance is relatively poor. This simple model makes recommendations based on repeated co-occurring items or consecutive items. In NARM and GRU4REC, each user is explicitly modeled and represented by a separate sequence, ignoring the possible interaction among projects. SR-GNN considers the changing relationship among projects, without taking personalization modeling for users into account. We use the AIGNN, which combines the area knowledge with the job knowledge, and use the GNN method to extract the complex change relationship among projects more effectively. Besides, considering the current, long-term, and global preferences of users, the mechanism of attention can be more prepared to reflect the behavior characteristics of users, so it can achieve better results.

VI. Conclusion and Future Work

In this paper, we propose a Graph Neural Networks job session recommendation model based on area knowledge and interest. Considering the importance of job area knowledge for job recommendation system, The AIGNN method combines knowledge and job area knowledge and takes into account the user's long-term and current and global preferences. The changing relationship of items is extracted through GNN and calculated using softmax. Probability distribution selects the most suitable item. Comparison with other baseline methods on public datasets proves the effectiveness of our method.

Due to the session-based recommendation requires a large amount of user history to make recommendations, there is a problem of cold-start. In the future, it may be a research direction to add the mechanism of alleviating cold start into session-based job recommendation.

References

[1] K. Bradley and B. Smyth, "Personalized information ordering: a case study in online recruitment," in Research and Development in Intelligent Systems XIX. Springer, 2003, pp. 279–292.

[2] O. Hinz and J. Eckert, "The impact of search and recommendation systems on sales in electronic commerce," Business & Information Systems Engineering, vol. 2, no. 2, pp. 67–77, 2010.

[3] X. Liu and Y. Zhang, "A kind of personalized advertising recommendation method based on user-interest-behavior model," in 2019 8th International Symposium on Next Generation Electronics (ISNE).IEEE, 2019, pp. 1–4.

[4] S. Wan and Z. Niu, "A hybrid e-learning recommendation approach based on learners' influence propagation,"IEEE Transactions on Knowledge and Data Engineering, vol. 32,no. 5,pp. 827-840, 2020.

[5] M. Jiang, Y. Fang, H. Xie, J. Chong, and M. Meng, "User click prediction for personalized job recommendation,"World Wide Web,vol. 22, no. 1, pp. 325–345, 2019.

[6] M. Polato and F. Aiolli, "A preliminary study on a recommender system for the job recommendation challenge," in Proceedings of the Recommender Systems Challenge, 2016, pp. 1–4.

[7] M. Bianchi, F. Cesaro, F. Ciceri, M. Dagrada, A. Gasparin, D. Grattarola, I. Inajjar, A. M. Metelli, and L. Cella, "Content-based approaches for cold-start job recommendations," in Proceedings of the Recommender Systems Challenge 2017, 2017, pp. 1–5.

[8] Y. Lu, S. El Helou, and D. Gillet, "A recommender system for job seeking and recruiting website," in Proceedings of the 22nd International Conference on World Wide Web, 2013, pp. 963–966.

[9] S. Benabderrahmane, N. Mellouli, and M. Lamolle, "On the predictive analysis of behavioral massive job data using embedded clustering and deep recurrent neural networks,"Knowledge-Based Systems,vol. 151, pp. 95–113, 2018.

[10] K. Shi, H. Lu, Y. Zhu, and Z. Niu, "Automatic generation of meteorological briefing by event knowledge guided summarization model,"Knowledge-Based Systems, vol. 192, p. 105379, 2020.

[11] K. Shi, C. Gong, H. Lu, Y. Zhu, and Z. Niu, "Wide-grained capsulenetwork with sentence-level feature to detect meteorological event insocial network,"Future Generation Computer Systems, vol. 102, pp.323–332, 2020.

[12] A. Mnih and R. R. Salakhutdinov, "Probabilistic matrix factorization," in Advances in neural information processing systems, 2008,pp. 1257–1264.

[13] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl, "Item-based collaborative filtering recommendation algorithms," in Proceedings of the10th international conference on World Wide Web, 2001, pp. 285–295.

[14] G. Shani, D. Heckerman, and R. I. Brafman, "An MDP-based recommender system,"Journal of Machine Learning Research, vol. 6, no.Sep, pp. 1265–1295, 2005.

[15] B. Hidasi, A. Karatzoglou, L. Baltrunas, and D. Tikk, "Session-based recommendations with recurrent neural networks,"arXiv preprintarXiv:1511.06939, 2015.

[16] Y. K. Tan, X. Xu, and Y. Liu, "Improved recurrent neural networks for session-based recommendations," in Proceedings of the 1st Work-shop on Deep Learning for Recommender Systems, 2016, pp. 17–22.

[17] T. X. Tuan and T. M. Phuong, "3D convolutional networks for session-based recommendation with content features," in Proceeings of the Eleventh ACM Conference on Recommender Systems,2017, pp. 138–146.

[18] J. Li, P. Ren, Z. Chen, Z. Ren, T. Lian, and J. Ma, "Neural attentive session-based recommendation," in Proceedings of the 2017 ACM on Conference on Information and Knowledge Management, 2017, pp.1419–1428.

[19] Q. Liu, Y. Zeng, R. Mokhosi, and H. Zhang, "Stamp: short-term attention/memory priority model for session-based recommendation,"in Proceedings of the 24th ACM SIGKDD International Conferenceon Knowledge Discovery & Data Mining, 2018, pp. 1831–183.

[20] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, "The graph neural network model,"IEEE Transactions on Neural Networks, vol. 20, no. 1, pp. 61–80, 2008.

[21] Y. Li, D. Tarlow, M. Brockschmidt, and R. Zemel, "Gated graph sequence neural networks,"arXiv preprint arXiv:1511.05493, 2015.

[22] P. Veličković, G. Cucurull, A. Casanova, A. Romero , P. Lio, and Y. Bengio, "Graph attention networks," arXiv preprint arXiv:1710.10903,2017.

[23] S. Wu, Y. Tang, Y. Zhu, L. Wang, X. Xie, and T. Tan, "Session-based recommendation with graph neural networks," in Proceedings of the AAAI Conference on Artificial Intelligence, vol. 33, 2019, pp. 346–353.

Modeling and Selecting Frameworks in terms of Patterns, Tactics, and System Qualities

Hind Milhem[*], Michael Weiss, Stephane S. Somé[*]

[*]School of Electrical Engineering and Computer Science (EECS), Department of Systems and Computer Engineering
[*]University of Ottawa, Carleton University
Ottawa, Canada
hbani043@uottawa.ca, michael_weiss@carleton.ca, ssome@eecs.uottawa.ca

Abstract—**Selecting frameworks and documenting the rationale for the choice is an essential task for system architects. Different framework selection approaches have been proposed. However, none of these connects frameworks to qualities based on their implemented patterns and tactics. In this paper, we propose a way to compare automatically the quality attributes of frameworks by extracting the patterns and tactics from a framework's source code and documenting them to connect frameworks to requirements upon which a selection can be made. We use a tool called Archie (a tool used to extract tactics from a Java-based system's code) to extract the patterns/tactics from the implementation code of frameworks. We then document and model these patterns/tactics and their impact on qualities using the Goal-oriented Requirements Language (GRL). The satisfaction level of the quality requirements integrated with other criteria such as the preferences of an architect provide architects with a tool for comparing different frameworks and documenting their rationale for choosing a framework. As a validation of the approach, we apply it to realistic case studies with promising results.**

Keywords-Framework Selection; Architectural Tactic; Architectural Pattern; Non-Functional Requirement (NFR); Framework Modeling; Tactic/Pattern Extraction

I. INTRODUCTION

A framework is a highly reusable design for an application or part of an application in a given domain. With the increasing complexity of developing software systems and shorter delivery times, it is essential to reuse existing designs in the form of frameworks as much as possible. Many candidate frameworks are usually available for a given application. Therefore, selecting frameworks and documenting the underlying rationale for the choice, become an important task for system architects. Various previous work [1][2][3] have addressed the selection of frameworks based on various characteristics and criteria such as the features of the frameworks, the deployability and the interoperability of the frameworks, and how they perform (testing). However, none of these connects frameworks to qualities to compare frameworks based on their exhibited quality attributes expressed as Non-Functional Requirements (NFRs) [4].

Architectural patterns and tactics [4] are reusable building blocks for software development (including frameworks). They are characterized in terms of factors that affect the various quality attributes so that architecture can be understood in terms of those quality attributes. Our main assumption is that the implementation of a framework inherits from the quality attributes associated to the patterns and tactics used in its implementation.

In a previous work [5], we proposed an approach to select frameworks based on their quality attributes. We associate frameworks with quality attributes based on the architectural patterns used in their implementation. Since the implementation of frameworks is not always adequately documented, we use a source code analysis tool (Archie [6][7][8]) to determine which architectural patterns are used in the implementation of the frameworks. We then model the relation between the frameworks, patterns and quality a Goal model using the Goal-oriented Requirements Language (GRL) [9].

This work builds on our prior work [5] by considering two additional characteristics to select a framework in addition to architectural patterns. This is because selecting a framework based only on its patterns might not be sufficient. The two additional characteristics are the architectural tactics and the importance values of quality requirements (preferences of architects). We use the Archie tool to find the tactics implemented in frameworks. We then add the tactics and their impact on quality attributes to the GRL model. We calculate the importance values of the NFRs using the AHP method [10]. Adding the implemented tactics and the importance values of the NFRs as other criteria to the model, in addition to the patterns, would provide more details about how the implemented patterns/tactics, considering the architects' preferences in a framework, can together push or pull the framework toward or away from given NFRs in an informed way.

The remainder of this paper is organized as follows; Section 2 provides an overview of the related work. We present our proposed approach in Section 3. In Sections 4, we present a case study as an example to apply the approach to. Section 5 shows the preliminary validation of the approach. In Section 6, we

present threats to the validity of this work. In Section 7, we draw initial conclusions and describes plans for future work.

II. Related work

Cervantes et al. [1] extract patterns and tactics from a framework by applying a mapping process between the patterns and tactics in a framework and those patterns and tactics, which are employed in architecture design. They also mention that patterns can be extracted from the provided services of a framework and that framework selection is based on architecture drivers (such as the team's level of knowledge of a framework, or the framework's maturity). In comparison to our work, patterns are identified manually. This approach does not consider the patterns and tactics implemented in a framework as a selection criterion. It also does not provide any details on how to represent frameworks in terms of the patterns and tactics they implement.

Mirakhorli and Huang [6][7][8] present an approach that relies primarily on information retrieval and machine learning techniques for discovering tactics in code. This is done by training a classifier to recognize specific terms that occur commonly across implemented tactics. The probabilities of these terms (the probability that a particular term identifies a class associated with a tactic) are determined using specific mathematical equations. The resulting tool is called Archie and is used in our work to identify architectural patterns and tactics from source code. In comparison to our work, their work only focuses on the tactics. Their work does not focus on the selection method and modeling of architectures in terms of their implemented patterns and tactics.

Sena et al. [12] analyze studies reporting on software architectures of big data systems, to identify architectural patterns, quality attributes, as well as problems and liabilities of those patterns. They determined that various architectural patterns, such as the Layered pattern, the Pipe and Filter pattern, the Broker pattern, and the Shared Repository pattern have significant impacts on the qualities and characteristics of big data systems. We use the results of this work to determine the main quality requirements and the determined patterns, as discussed in the technical report [11]. In comparison to our work, this work does not focus on the modeling of architectures in terms of their implemented patterns. The extraction of the patterns is done manually by analyzing studies reporting on software architectures of big data systems.

Additional related work includes: Johnson [13], Aguiar and David [14], Beck and Johnson [25], Ryoo et al. [29], and Meusel et al. [30]. A key difference between our work and these is that their work does not connect frameworks to quality attributes based on both patterns and tactics. Their work also does not focus on the selection method and modeling of architectures in terms of their implemented patterns and tactics.

III. Proposed approach

The proposed approach includes three general steps: First, determining the patterns and tactics a framework implements. Second, modeling the frameworks in terms of their patterns and tactics. Third, choosing a framework.

In the following, we present the general steps (process) of the approach.

A. Determining Patterns and Tactics Implemented in a Framework

To determine the implemented patterns and tactics of a framework, we follow the following sub-steps:

1) Determine the Context/Domain

The objective of this sub-step is to restrict the scope of the search. This allows a more focused determination of the candidate frameworks, the patterns, and the tactics according to a specific context.

2) Choose the Patterns and Tactics that Need to be Checked for A Framework in the Determined Context/Domain

The set of patterns and tactics applied to a problem is typically restricted by the domain and context of that problem. These patterns and tactics are the ones known to contribute to solving aspects of the problem. Our approach searches for the patterns and tactics relevant to the context of a problem in frameworks used as part of the solution to this problem. Therefore, a knowledge of these patterns and tactics is needed as input.

In this work, we conduct a literature review to find the most relevant and common patterns and tactics of a framework in the determined context. The resulting list of tactics and patterns is however reusable in the same domain.

3) Determine the Tool to be Used to Extract the Patterns and Tactics of A Framework

Although a manual search of the implemented patterns and tactics in a framework is possible, it is not practical for large frameworks. Different alternative methods have been used in the literature such as Archie [6][7][8], Matching methods between the provided services of a framework and its patterns/tactics [13], Pattern instantiation (assigning the roles defined in a pattern to concrete classes, responsibilities, methods, and attributes of a practical design) [14], and Matching methods between the problem statement of an architecture and the applied patterns [15].

In this work, we use Archie [6][7][8] to extract the patterns and tactics from the frameworks' source code. We chose Archie because it is the only automated tool among the alternatives. So, it makes the extraction process faster by decreasing time and effort spent searching the patterns and tactics and their related terms in the documentation, websites, and source codes of frameworks. It is extensible so we can add or remove patterns and tactics.

4) Apply the Tool on a Framework to Extract the Patterns and Tactics it Implements

In this sub-step, we apply Archie on the candidate frameworks and get a set of candidate patterns and tactics for each framework. The interested reader may find more details about this step in [11].

5) Validate the Candidate Set of Patterns and Tactics which are Detected by the Selected Tool

We validate the results of applying Archie on the candidate frameworks by looking for the occurrences of those patterns/tactics, which are detected by Archie, manually in the

source code/documentation/websites of the candidate frameworks. The goal of this step is to ensure the validity of our results. For more details about this step, see [11].

B. Modeling Frameworks in terms of their Implemented Patterns and Tactics

To model frameworks in terms of their implemented patterns and tactics, we perform the following sub-steps:

1) Determining the Modeling Language to be Used to Model Frameworks

Different modeling languages have been used to model frameworks. Examples include The Goal-oriented Requirement Language (GRL) [9], the NFR-framework [16], i* (i-star) framework [17], and the softgoal modeling language [18].

In this work, we chose the GRL. The elements of the GRL notation used are shown in Figure 1. The choice of GRL was motivated by the facts that: it enables us to evaluate and compare the impact of different design choices on quality attributes, it is a part of an international standard (User Requirements Notation – URN) [9], enables the modeling of stakeholders and their goals, supports Key Performance Indicators (KPIs) for quantitative reasoning, and supports evaluation strategies and propagation algorithms to evaluate the satisfaction of goals and actors under selected conditions [19]. Giving quantitative contributions of patterns and tactics helped us calculate the satisfaction of NFRs.

2) Modeling the Patterns, Tactics, and their Contributions on the NFRs

In this sub-step, we first extract from the description of the patterns/tactics, the NFRs, the contributions of the patterns and tactics on the NFRs. Then, we extract the design decisions, which show the reason for the negative or the positive impact of a pattern/tactic on an NFR. In this work, we follow Ong et al.'s [20] approach to extract NFRs, design decisions, and the contributions of the patterns and tactics on the NFRs. We added to the description by underlining the benefits, liabilities, the affected NFRs, and reasons for the positive or negative impact of the patterns or tactics on the NFRs.

The benefits and liabilities of a pattern/tactic indicate the positive and negative contributions on the NFRs respectively. The reasons for the positive or negative impact of the patterns/tactics on the NFRs correspond to design decisions behind the application of a pattern/tactic. These design decisions are expressed as sentences starting with an active verb such as 'define,' 'register,' 'change,' 'reuse,' etc. We also have followed the same method for the tactics.

We then derive GRL models, with the NFRs and the contributions of the patterns and tactics on the NFRs, from the description of each pattern/tactic. First, we start with the patterns/tactics at the bottom of the model. Then, we put the design decisions and NFRs at the topmost level of the model. The complexity of the system dictates the number of levels of design decisions.

Figure 1. Summary of the GRL notations [9]

Based on Figure 1, we select softgoals (clouds) elements to represent NFRs and the design decisions, indicating that these cannot be achieved in an absolute manner. Tasks (hexagons) are selected to represent patterns, tactics, the parts of a framework where a pattern/tactic is implemented, and frameworks, representing ways of achieving a softgoal. An actor with Boundary (dotted circle) is used to represent an architect of a framework. Solid lines (Contribution links) indicate the desired impacts of one element on another element. Contribution types determined by labels. These labels indicate various degrees of positive (+) or negative (-) contributions (see Figure 2 for the complete set of labels). Decomposition links allow an element to be decomposed into sub-elements [9]. AND, IOR and XOR are supported decompositions. We use only AND decomposition links to represent the connection between a framework and its patterns and tactics because all the patterns are required in a framework before the NFRs are satisfied. We used it also to represent the connection between the parts of a framework and the patterns and tactics because all the patterns and tactics are needed to be implemented in a part of a framework.

We use quantitative contribution values. There are different methods to get the contribution values of a pattern/tactic to an NFR such as AHP [10], Delphi [21], or by using indicators (one of the GRL notations as we can see in Figure 1). We use a matching method between the contribution between a pattern/tactic and a given NFR from the literature [22][23][24][25][26][27][28] and the contribution values used in the GRL. More details about the calculations of the contribution values are shown in [11].

3) Modeling Framework in terms of their Implemented Patterns and Tactics

The GRL models of the patterns and tactics from the previous sub-step, are used to build a bottom up GRL model for frameworks, starting with the framework and its parts at the bottom level of the model, connected with all its implemented patterns and tactics. The parts of a framework show where its patterns and tactics are implemented. A link between a pattern and a tactic indicates that the tactic is used as part of the pattern implementation. The resulting GRL model specifies that the

design decisions explain why a pattern/tactic impacts an NFR the way it does. Consequently, the design decisions push or pull the framework towards or away from NFRs, as shown in Figure 2.

Each NFR is assigned an importance value given by architects to help compare and choose the best suited framework. We calculate these importance values using the AHP method, as shown in the [11].

C. Choosing a Framework

1) Evaluate the models of the candidate frameworks

To initially assign a satisfaction level to a pattern/tactic, we assign a tactic or a pattern to be Satisfied (100) if a framework implements a tactic or a pattern; else, if a framework does not implement it, it is then assigned to be Denied (0). The initial values are marked with a star (*) on the evaluation model. All the patterns and tactics, which are implemented in a framework, are initially assigned using a star (*). After the initial assignment of satisfaction levels to the tactics and patterns of a framework, we evaluate the satisfaction levels of the NFRs by applying different evaluation strategies on the GRL models, as we will see in Section IV(C).

2) Compare the Candidate Frameworks

In this last step, we compare the candidate frameworks based on their implemented patterns and tactics considering the importance values of the NFRs, which would be given by an architect, as we will see in Section IV(C).

IV. CASE STUDY

To validate the approach, we applied our approach to an industrial case study, which is a part of a project to develop a cyber fusion center. The case study consists in choosing a stream processing framework for big data. Architects had to choose among different candidate frameworks. The selected framework was to provide the backbone for the collection and correlation of security events. Processing the events requires routing information from sensors to various processing stages that perform analytics on the events at different levels of abstraction (such as detecting attacks and attack patterns).

Our industrial collaborators considered three candidate frameworks: Apache Storm [29] (a component in Apache Metron [30]), Apache Flink [31], and Apache Spark [32]. In the following, we apply the main steps of our approach.

A. Determining Patterns and Tactics Implemented in a Framework

We apply the following sub-steps to determine the implemented patterns and tactics of the frameworks Apache Storm, Apache Flink, and Apache Spark.

1) Determine the Context/Domain

We determined the context of this project to be as big data systems in general and data streaming frameworks in specific. All the candidate frameworks are real data streaming frameworks.

2) Choose the Patterns and Tactics that Need to be Checked for A Framework in the Determined Context/Domain

Figure 2. The general GRL model of a framework

To perform this sub-step, we conducted a literature review to find the most relevant and common patterns and tactics of a framework in the determined context.

Given the context of the problem, we conducted a literature review to find the most relevant patterns and tactics of a big data system in general and a data streaming system in specific. We also determine the most common NFRs of a data streaming framework. The results of this step and more details are shown in [11].

3) Determine the Tool to be Used to Extract the Patterns and Tactics of a Framework

We use the Archie tool [6][7][8] to extract the patterns and tactics from the frameworks source code as discussed in [7].

4) Apply the Tool on A Framework to Extract the Patterns and Tactics it Implement

Mirakhorli and Huang [6][7][8] trained a classifier in Archie to recognize specific terms that occur commonly across implemented tactics and calculate the weights of the tactics (the probability that a particular term identifies a class associated with a tactic). Archie tool considers thirteen tactics [6][7][8] from three quality attributes to be detected in any Java-based system. These tactics are *Policy-Based Access Control (PBAC), Role-Based Access Control (RBAC), Kerberos, Audit trail, Session Management, and Authenticate* from Security, *Checkpoint, Heartbeat, Ping/Echo, Active Redundancy, and Load Balancing* form Reliability, *and Resource Scheduling, and Resource Pooling* from Performance.

In addition to these thirteen tactics, we added seven other tactics and five patterns to be detected by the Archie tool. To see the added patterns and tactics, we refer to [11]. The analysis of the results of applying Archie to Storm, Flink, and Spark is shown in [11].

5) Validate the Candidate Set of Patterns and Tactics which are Detected by the Selected Tool

After applying the tool on the candidate frameworks, we validated the results by looking for the occurrences of the detected patterns/tactics, manually in the source code/documentation/websites of the candidate frameworks Storm, Flink, and Spark. The sample results of the validation are shown in [11].

B. Modeling Frameworks in terms of their Implemented Patterns and Tactics

We modeled the candidate frameworks Storm, Flink, and Spark in terms of their implemented patterns and tactics following the general model shown in Figure 2. The case study considers NFRs relevant to data streaming systems such as *Scalability, Maintainability, Performance, Portability, Availability, Reliability, Security, and Interoperability*. For the sake of readability, the presented model in Figure 3 is restricted to *Testability, Security, Reliability, Availability, and Scalability*. The high-level goal of the project is shown at the top of the model connected to alternative candidate frameworks at the bottom of the model. On top of each framework, there are several parts for each framework connected to their implemented patterns and tactics. The design decisions explain why a pattern/tactic impacts an NFR the way it does at the top of the model. Consequently, the design decisions push or pull the framework toward or away from NFRs.

C. Choosing a framework

1) Evaluate the models of the candidate frameworks

We evaluate the model to calculate the satisfaction levels of the NFRS (Figure 3). The evaluation is done by applying different evaluation strategies on the GRL model. For example, Figure 4 shows a first strategy where only the patterns and tactics implemented in the Spark framework are initially satisfied. Similarly, Figures 5 and 6 show strategies where only the patterns and tactics implemented in Flink and Storm, are initially satisfied. Color-coding is used to highlight what is satisfied and what is denied. For example, the 'Green' colour indicates that the element is satisfied, while the 'Yellow' colour indicates that the element is neutral. The 'Red' colour indicates that the element is denied.

2) Compare the candidate frameworks

Based on the evaluation results of the GRL models from the previous sub-step, we can see that the three frameworks have similar satisfaction levels of the *Testability, Security, and Scalability* requirements as shown in Figures 4, 5, and 6. The *Testability* requirement is satisfied with (42) satisfaction level for all the frameworks. While the *Security* is satisfied with (50) satisfaction level and *Scalability* with (56) satisfaction level for all the frameworks.

The Storm framework has a higher satisfaction level for *Reliability* and *Availability*, which is (63) compared to Spark and Flink. This is because of the implementation of the three

Figure 3. The GRL model of the Storm, Flink, and Spark frameworks in terms of Testability, Security, Reliability, Availability, and Scalability

reliability tactics: *Exception Handling, Heartbeat,* and *Checkpoint*. They all improve fault tolerance, which improves reliability. Spark and Flink have the same satisfaction level for Reliability, which is (42). Spark has the least satisfaction level for *Availability, which* is (5). While Storm has (32) and Flink has (30) satisfaction levels for *Availability*. This is because of applying the Observer/Publish-Subscribe pattern in Storm and Flink, which provides Asynchronous communication between components without blocking to wait for a response. This helps decouple publishers and subscribers so they can be active and available at different points in time, resulting in improving the

availability of the frameworks. Both Storm and Flink use the "Checkpoint" tactic to Record consistent states and have a path to roll back to them if necessary. While Spark uses the "Active Redundancy" tactic for recovery, preparation, and repair of the errors. The architect is more satisfied with Storm than Flink and Spark. As we see, the satisfaction value of the architect for Storm is (48), while it is (43) for Flink and (37) for Spark. If an architect favours Reliability and Availability over the other requirements, we recommend Storm. However, if Testability, Security, and Scalability are preferred, then any one of the three frameworks could be equally recommended.

Figure 4. Strategy 1: Applying only the implemented patterns and tactics of the Spark

Figure 5. Strategy 2: Applying only the implemented patterns and tactics of the Flink

Figure 6. Strategy 3: Applying only the implemented patterns and tactics of the Storm

V. PRELIMINARY VALIDATION

The previous sections discuss the application of the approach to a case study. We applied the approach in the context of an industrial project where architects had to choose among different frameworks Spark, Storm, and Flink. The results were found satisfactory (and in agreement) with the project architects. The architects confirmed that the approach was helpful in choosing the best-fit framework to provide the backbone for the collection and correlation of security events in a cyber security center. We also compared the inferred quality attributes (i.e. reliability, availability, and performance) with benchmark comparison results such as [33]. Inoubli [33] showed that both Storm and Flink use the "Checkpoint" tactic for fault tolerance. While Spark uses recovery techniques. This was compatible with our results in Section IV(C). Our results showed that both Storm and Flink implement the "Checkpoint" tactic to Record consistent states and have a path to roll back to them if necessary. While Spark uses the "Active Redundancy" tactic for recovery, preparation, and repair of errors. Inoubli also showed that Spark is the fastest framework in terms of the processing time compared to Storm and Flink. This was compatible with our results, which shown in [11], that the satisfaction level of the Performance for Spark is (86) while it is (46) for both Storm and Flink. This confirms that Spark is the fastest one while Strom and Flink are quite similar in terms of the data processing speed, as shown in Figures 12 and 13 in Section IV(C).

Inoubli also reported that Flink and Storm share similarities and characteristics with Spark. Flink, Storm, and Spark implement similar patterns, such as the Layers and Broker patterns and similar tactics, such as "Resource Pooling" and "Resource Scheduling". The compatibility with Inoubli's results offers some validation of the main tenet of our works; the link between the implemented patterns and tactics, and quality attributes.

In another case study on Gradle and Maven tools [34], we also compared the inferred quality attributes (i.e. performance) with benchmark comparison results such as [34]. The results of the experiment conducted in [34], showed that Gradle is faster than Maven. This is because of the performance features, which Gradle includes, such as the parallelism and the incremental build and subtasks. In our results, which are shown in [11], we got quite similar results to the ones in [34].

In a case study on a Healthcare-Supportive System-System of Systems (HSH-SoS) architecture [35], we use our approach to support an analysis of the HSH-SoS architecture in terms of its implemented patterns and tactics. Our objective is to confirm that the approach can be used not only to compare implementations but also to provide a rationale or documentation about a framework/system architecture.

I. THREATS TO VALIDITY

Threats to validity can be classified as construct, internal, and external validity. We discuss the threats, which potentially impact our work, and the ways in which we attempted to mitigate them.

External Validity evaluates the generalizability of the approach. The primary threat is related to the assumption that a framework inherits the aspects of quality associated to its implemented tactics/patterns. It is possible that patterns/tactics could be implemented the wrong way and not provide their expected benefits. Although our initial validation with case studies such as Gradle and Maven has showed the validity of our assumption, more case studies will however be required. As mitigation to this threat we confirmed the proper implementation by performing a manual inspection of the code. Another threat is that NFRs derived from patterns/tactics such as performance might not be sufficient to be able to compare the frameworks. We consider the result provided by our approach as one component of the criteria for a final decision on choosing a framework. Other criteria including the cost, stability, maturity, community support might also be considered.

Construct Validity evaluates the degree to which Archie was accurate in detecting the patterns and tactics of the frameworks. In our case study, we have calculated the false positives and false negatives numbers by checking if those patterns/tactics detected by Archie are implemented in the source code of a framework. We found that there were only 12% false positives in Storm, 16% in Flink, 4% in Spark, and 16% in both Gradle and Maven. The whole results showed that most of the patterns and tactics, which were detected by Archie for the frameworks, are implemented in the frameworks. This confirms the high accuracy and performance of the Archie tool. Archie also has been tested on several systems ranging from 1,000 to 20,000 java files [6][7].

Internal Validity reflects the extent to which a work minimizes systematic error or bias so that a causal conclusion can be drawn. A threat to validity is that the search for specific patterns or tactics was solely performed by the authors. In the case of the cyber fusion center project, we mitigated this threat by elicited feedback from developers and architects with extensive experience with the involved frameworks.

VI. CONCLUSION AND FUTURE WORK

The approach described in this paper extracts the implemented architectural patterns and tactics from frameworks source codes to connect frameworks to quality requirements upon which a selection can be made. We use an information retrieval approach, with a tool called Archie, to determine the implemented architectural patterns and tactics in order to enable a more informed assessment by architects. We then model the frameworks in terms of their implemented patterns and tactics using the Goal-oriented Requirements Language (GRL). This model provides architects with a rationale about the satisfaction levels and the analysis of the tradeoff of given NFRs for a framework. Providing such rationale with considering the importance values of the NFRs integrated with other criteria such as the cost, delivery time,

stability, and maturity of a framework would help an architect to choose among several candidate frameworks.

In the future, we plan to improve our modeling of frameworks with GRL indicators instead of simply matching the impact of patterns on NFRs and the contribution values in the GRL. The indicators in the GRL measure observable values and convert them to GRL satisfaction values (from zero for denied, to 100 for satisfied) that can be propagated to other model elements through links. This would allow getting the contribution values of the patterns and tactics automatically.

Another future work is to integrate the consideration of criteria such as cost, delivery time, stability, and maturity of a framework in addition to the patterns, tactics, and the importance values of the NFRs to be able to choose a framework in a more informed way.

REFERENCES

[1] H. Cervantes, P. V. Elizondo, and R. Kazman. 2013. A principled way to use frameworks in architecture design. *IEEE Software*, March/April, 46-53.

[2] G. Grau, and X. Franch. 2007. A Goal-Oriented Approach for the Generation and Evaluation of Alternative Architectures. *European Conference on Software Architecture (ECSA)*, pp 139-155.

[3] A. Zalewski. 2013. Modeling and Evaluation of Software Architecture. *Warsaw University of Technology Publishing Office*.

[4] L. Bass, P. Clements, and R. Kazman. 2012. Software Architecture in Practice. *Addison-Wesley*.

[5] H. Milhem, M. Weiss, and S. Some. 2019. Extraction of Architectural Patterns from Frameworks and Modeling their Contributions to Qualities. *Pattern Languages of Programs (PLoP)*. 17 pages, Ottawa, Canada.

[6] M. Mirakhorli. 2014. Preserving the Quality of Architectural Tactics in Source Code.

[7] M. Mirakhorli and J. Cleland-Huang. 2016. Detecting, Tracing, and Monitoring Architectural Tactics in Code. *IEEE Transactions on Software Engineering*, Volume: 42, Issue 3, pp 205-220.

[8] M. Mirakhorli, A. Fakhry, A. Grecho, M. Wieloch, and J. Cleland-Huang. 2014. Archie: A Tool for Detecting, Monitoring, and Preserving Architecturally Significant Code. *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pp 739-742, Hong Kong, China.

[9] G. Mussbacher, M. Weiss, and D. Amyot. 2007. Formalizing Architectural Patterns with the Goal-oriented Requirement Language. *Proceedings of the Fifth Nordic Conference on Pattern Languages of Programs*.

[10] https://en.wikipedia.org/wiki/Analytic_hierarchy_process

[11] http://www.site.uottawa.ca/~ssome/publis/Methodology_Frameworks _Selection.pdf

[12] B. Sena, L. Garces, A. P. Allian and E. Yumi Nakagawa. 2018. Investigating the Applicability of Architectural Patterns in Big data Systems. *Pattern Languages of Programs (PLoP)*, Portland, Oregon, USA.

[13] R. E. Johnson. 1997. How frameworks compare to other object-oriented reuse techniques. Communications of the ACM, 40(10), 39-42.

[14] A. Aguiar, and G. David. 2011. Patterns for effectively documenting frameworks. *Transactions on Pattern Languages of Programming II*, 79-124, Springer.

[15] K. Beck and R. Johnson. 1994. Patterns Generate Architecture. ECOOP '94 Proceedings of the 8th European Conference on Object-Oriented Programming, pp 139-149, London, UK.

[16] Mehta, R., Ruiz-López, T., Chung, L., & Noguera, M., "Selecting among Alternatives using Dependencies: An NFR approach",

[17] Bastos, L.R.D., & Castro, J.F.B., "Systematic Integration Between Requirements and Architecture", *Software Engineering for Multi-Agent Systems III*, pp 85-103, Volume 3390 of the series Lecture Notes in Computer Science, (2005).

[18] Zhu, M.X., Luo, X.X., Chen, X.H., & Wu, D.D., "A non-functional requirements tradeoff model in Trustworthy Software", *Information Sciences 191*, pp 61 – 75, (2012).

[19] Amyot, D., Ghanavati, S., Horkoff, J., Mussbacher, G., Peyton, L., & Yu, E., "Evaluating Goal Models within the Goal-oriented Requirement Language", *International Journal of Intelligent Systems*, pp 841-877, (August 2010).

[20] Ong, H., Weiss, M., & Araujo, I., "Rewriting a Pattern Language to Make it More Expressive", 2003.

[21] https://en.wikipedia.org/wiki/Group_decision-making

[22] N. Harrison. 2011. Improving quality attributes of software systems through software architecture patterns.

[23] S. Bode and M. Riebisch. Impact Evaluation for Quality-Oriented Architectural Decisions Regarding Evolvability. *European Conference on Software Architecture ECSA*, 2010.

[24] A. Alebrahim, S. Fassbender, M. Filipczyk, M. Goedicke, and M. Heisel. 2015. Towards a Reliable Mapping between Performance and Security Tactics, and Architectural Patterns. *EuroPLoP '15 Proceedings of the 20th European Conference on Pattern Languages of Programsv*, Article No. 39, 43 pages.

[25] G. Me, C. Calero, and P. Lago. Architectural patterns and quality attributes interaction. *2016 Qualitative Reasoning about Software Architectures (QRASA)*.

[26] N. Harrison and P. Avgeriou. 2010. Implementing Reliability: The Interaction of Requirements, Tactics and Architecture Patterns. *Architecting Dependable Systems VII* pp 97-122.

[27] M. Kassab, G. El-Boussaidi, and H. Mili. 2011. A Quantitative Evaluation of the Impact of Architectural Patterns on Quality Requirements. *Software Engineering Research,Management and Applications 2011* pp 173-184, Pp 173-184.

[28] M. Kassab and G. El-Boussaidi. 2013. Towards Quantifying Quality, Tactics and Architectural Patterns Relations. Proceedings of the International Conference on Software Engineering and Knowledge Engineering, SEKE.

[29] https://storm.apache.org

[30] Apache Metron, metron.apache.org

[31] Apache Flink, flink.apache.org

[32] https://spark.apache.org

[33] W. Inoubli, S. Aridhi, H. Mezni, M. Maddouri, E. M. Nguifo, "An experimental survey on big data frameworks", *Future Generation Computer Systems*, 546-564, 2018.

[34] https://gradle.org/maven-vs-gradle/

[35] L. Garces, B. Sena, and E. Y. Nakagawa, "Towards an architectural patterns language for System-of-Systems", HILLSIDE Proc. Of Conf. on Pattern Lang. of Prog. V (October 2019), 24 pages.

An Evaluation of Recommendation Algorithms for Tourist Attractions

Anderson Feitosa Júnior
Informatics Coordination
Federal Institute of Alagoas
Maceió, Alagoas, Brasil
amfj1@ifal.edu.br

Flávio Medeiros
Informatics Coordination
Federal Institute of Alagoas
Maceió, Alagoas, Brasil
flavio.medeiros@ifal.edu.br

Ivo Calado
Informatics Coordination
Federal Institute of Alagoas
Maceió, Alagoas, Brasil
ivo.calado@ifal.edu.br

Abstract—Tourism is an important activity for the economy of several places in the world. In this context, the use of information systems might bring some benefits for tourists, for example, by helping people to select tourist attractions, such as restaurants, beaches, and museums. In this paper, we perform two evaluations of 22 recommendation algorithms provided by existing recommendation system libraries, aiming to identify efficient algorithms in terms of prediction accuracy. In the first evaluation, we compare the algorithms by measuring different metrics, such as *RMSE, MAE, precision, recall*, and *F1 Score*. In the second evaluation, we compare the recommendation algorithms based on the answers of 172 people that participated in our survey by evaluating different kinds of tourist attractions. The results of our study show that some recommendation algorithms remain on the top list with regards to efficiency on both studies, such as the *SVD++, Baseline Only*, and *KNN Z Score with Pearson Baseline Similarity*. Others are efficient in the first evaluation, or for some metrics, but not in the second study, for example, or the other way around. The results of our study are useful for people that are creating solutions in the tourism domain.

Index Terms—Recommendation systems, Tourist attractions

I. INTRODUCTION

Tourism is an important activity for the economy of several places in the world [1]. In this context, the use of information systems might bring some benefits for tourists, which nowadays can find information about hotels, tourist attractions, and restaurants in many different places (e.g. *Four Square, Yelp*, and *Google*). Thus, the task of selecting what is relevant is difficult and time-consuming. Recommendation systems can help tourists in this task by recommending tourist attractions, restaurants, and hotels based on their profile and interests.

Considering the relevance of the topic, several approaches for system recommendation have been proposed, and many of those recommendation algorithms are available in well-known libraries, including *Surprise, Apache Mahout*, and *LibRec*. These libraries provide different algorithms to make recommendations, including neighborhood methods [2–4], matrix factorization-based techniques [5–7], and other types, such as SlopeOne [8], and CoClustering [9]. However, a little effort has been put into comparing the efficiency of these algorithms in practice, for example, by considering the perspective and opinions of human beings.

In this paper, we perform two evaluations of 22 collaborative filtering recommendation algorithms. The implementation of such algorithms were obtained from mature libraries, e.g., *Surprise, Apache Mahout*, and *LibRec*. After selecting the algorithms, we performed two complementary evaluations. In the first one, we compared the algorithms based on a set of metrics, such as *RMSE, MAE, precision, recall*, and *F1 Score*, to measure the efficiency of the algorithms in terms of prediction accuracy. Afterwards, in the second evaluation, we compared the same 22 algorithms based on the answers of 172 people that participated in our experiment by evaluating tourist attractions of different kinds. At the end, we triangulate the results of both studies with the goal of identifying the most efficient algorithms, that is, the ones that make predictions with fewer mistakes.

The results of our study show that some recommendation algorithms remain on the top list with regards to efficiency on both studies, such as the *SVD++, Baseline Only*, and *KNN Z Score with Pearson Baseline Similarity* algorithms. Other algorithms are efficient in the first evaluation, or for some metrics, but not at the second evaluation, for example, or the other way around. For instance, the *SVD* algorithm is the second more efficient for metrics *RMSE* and *MAE*, while in the second evaluation, it takes the 16 position in the ranking of more efficient algorithms. On the other hand, the *KNN Z Score with Cosine Similarity* is the most efficient in the evaluation with people, but it takes position 14 when considering metric *RMSE*. By triangulating the results of both studies, we improve evidence regarding the most efficient algorithms for recommendation systems in the tourism domain.

The remainder of this article is organized as follows. In Section II, we present some background information that is necessary to understand our study. In Section III, we discuss the setup of our study, and Sections IV and V present the results of the evaluations based on metrics and users perspective respectively. In Section VI, we compare the results of both evaluations. In Section VII, we discuss the related work, and Section VIII presents the concluding remarks and conclusions.

II. BACKGROUND

In this section, we present a brief overview of recommendation systems and machine learning. Recommendation systems have emerged in response to the huge amount of information available nowadays [10], which makes it difficult

and time-consuming for choosing between a wide variety of products and services available, for example, in websites around the world. By definition, recommendation systems are software tools and techniques that automatically provide recommendations for users based on their past experience, that is, recommending the items that are most likely to interest a particular user [11]. Recommendation systems use the available user data, information about other users, and information about the environment with the goal of making predictions.

Currently, recommendation systems are a common application area in machine learning, which is a collection of algorithms and techniques that software developers can use to create computational systems that automatically learn by analyzing the available data to make predictions and inferences [12]. The algorithms for recommendation systems in machine learning are typically classified into three categories [13]:

1) **Content-Based Filtering**: algorithms in which the user receives recommendations based on similar items that the user has shown interest in the past [14];

2) **Collaborative Filtering**: the algorithm makes recommendations for users based on items that people with a similar profile has shown interest in the past. This category is subdivided into two subcategories [15]:

    a) **User-based Collaborative Filtering**: a memory-based algorithm that uses information about users and items to generate recommendations [16]. It calculates a certain similarity score among users, and based on this score, it selects the most similar users and recommends items that these similar users have previously shown interest;

    b) **Item-based Collaborative Filtering**: it is a model-based algorithm that provides item recommendations based on a model of user ratings [16]. In this algorithm, the similarities between pairs of items are calculated, and the similarity is used to recommend items.

3) **Hybrid Filtering**: algorithms that combine collaborative and content-based methods [17].

Recommendation systems are currently applied in many different domains, such as e-commerce [18], and video streaming [19], due to information overload. In the tourism field, there is also a huge amount of information available. For example, tourist information on the internet spreads across a wide range of different websites. In addition to the official websites of destinations, attractions or services, there are a wide range of blogs, and wikis, offering additional information about tourist attractions [20].

Developers can use a number of programming libraries in different languages to implement recommendation systems, such as *Surprise*, *Apache Mahout*, and *LibRec*. These libraries provide several algorithms to make predictions, including neighborhood methods [2–4], matrix factorization-based [5–7], and other types, such as SlopeOne [8], and CoClustering [9].

However, little effort has been put into comparing the efficiency of these algorithms in practice, considering the perspective of humans. In this study, we few this gap by performing two evaluations with the goal of ranking collaborative filtering algorithms: (1) by comparing the efficiency of the collaborative filtering algorithms by using a set of five metrics; and (2) by comparing the algorithms based on the answers of 172 people that participated in our survey.

III. Study Setup

To rank the collaborative filtering recommendation algorithms based on their efficiency, we followed the process described in Figure 1. In *Step 1*, we selected a set of platforms with tourist attractions information and selected four options to use in our study: *Trip Advisor*, *Google Places*, *Yelp*, and *Foursquare*. We extracted the attractions information manually and by using Application Programming Interface (API). Overall, we collected information about 92 tourist attractions, and more than 76 thousand ratings.

Fig. 1. The Steps performed to Rank the Recommendation Algorithms.

After collecting information about tourist attractions and ratings, in *Step 2*, we implemented a tool by using the *Surprise* library. This library provides a large set of out-of-box implementations for all recommendation algorithms arbitrarily selected to our study. Moreover, the *Python* programming language was selected due to its growing community working with machine learning and recommendation systems. In Table I, we list the 22 algorithms considered in our study.

In *Step 3*, we performed the first evaluation of the recommendation algorithms based on a set of five metrics, and ranking the algorithms based on the *RMSE*, *MAE*, *F1 Score*, *Precision*, and *Recall*. We made a system on the top of Jupyter [21], which allowed us to create an interactive visualization to compare the results of the recommendation algorithms. We present the details of this step in Section IV.

Last, in *Step 4*, we integrated the recommendation system that we developed in *Python* to a *RESTful API* [22] developed in *Flask* [23]. For the user interface, we implemented a responsive website by using *ReactJS*, which connects to the *RESTful API* through *HTTP* calls. After developing this infrastructure, we sent emails asking users to participate in our survey. We present the details of this step in Section V.

Algorithm	RMSE	MAE	Precision	Recall	F1
Baseline Only	0.89	0.70	0.86	0.92	0.89
SVD	0.90	0.71	0.86	0.87	0.87
SVD++	0.91	0.71	0.86	0.89	0.88
kNN Baseline with pearson baseline similarity	0.98	0.75	0.86	0.88	0.87
kNN Means with pearson baseline similarity	0.99	0.76	0.85	0.91	0.88
kNN Z Score with pearson baseline similarity	0.99	0.76	0.85	0.91	0.88
kNN Baseline with MSD	1.02	0.77	0.88	0.81	0.84
kNN Baseline with cosine similarity	1.02	0.77	0.88	0.81	0.84
kNN Means with with cosine similarity	1.03	0.78	0.87	0.82	0.85
kNN Basic with pearson baseline similarity	1.03	0.79	0.84	0.95	0.89
kNN Means with MSD	1.03	0.78	0.87	0.82	0.85
kNN Z Score with MSD	1.03	0.78	0.87	0.82	0.85
SlopeOne	1.03	0.79	0.87	0.82	0.84
kNN Z Score with cosine similarity	1.03	0.78	0.87	0.83	0.85
kNN Baseline with pearson correlation similarity	1.04	0.79	0.87	0.82	0.85
kNN Z Score with pearson correlation similarity	1.05	0.80	0.86	0.84	0.85
kNN Means with pearson correlation similarity	1.05	0.80	0.87	0.84	0.85
CoClustering	1.05	0.80	0.87	0.81	0.84
kNN Basic with cosine similarity	1.07	0.80	0.86	0.86	0.86
kNN Basic with MSD	1.07	0.80	0.86	0.86	0.86
kNN Basic with pearson correlation similarity	1.10	0.82	0.85	0.89	0.87
NMF	1.13	0.90	0.90	0.65	0.75

TABLE I
RANKING OF RECOMMENDATION ALGORITHMS BASED ON METRICS.

IV. METRIC-BASED EVALUATION

To evaluate the 22 recommendation algorithms, we consider a set of five metrics, as we explain next.

A. Mean Absolute Error (MAE)

The Mean Absolute Error calculates the error of estimated evaluations of users. For example, if a user rated an item with a five note and the system predicted a three note, the MAE is two. The formula used to calculate MAE is presented in Figure 2.

$$\text{MAE} = \frac{1}{|\hat{R}|} \sum_{\hat{r}_{ui} \in \hat{R}} |r_{ui} - \hat{r}_{ui}|$$

Fig. 2. The MAE formula used in our study.

B. Root Mean Square Error (RMSE)

The Root Mean Square Error is a quadratic scoring rule that also calculates the error of estimated evaluations of users. It is the square root of the average of squared differences between the prediction given by the system and the user's evaluation. The RMSE assigns higher weights to larger errors, since errors are squared before the average. The formula used to calculate RMSE is presented in Figure 3.

$$\text{RMSE} = \sqrt{\frac{1}{|\hat{R}|} \sum_{\hat{r}_{ui} \in \hat{R}} (r_{ui} - \hat{r}_{ui})^2}$$

Fig. 3. The RMSE formula used in our study.

C. Precision

The precision is used to know how many items the user really liked considering all recommended items. $Precision = tp/tp + fp$, where tp is the number of recommended items the user liked (true positives) and $tp + fp$ is the total number of recommended items (true positives, and false positives).

D. Recall

The recall is used to calculate the proportion of recommended items that the user likes. $Recall = tp/tp + fn$, where tp is the number of recommended items the user likes (true positives) and $tp + fn$ is the total number of items that user likes (true positives, and false negatives).

E. F1 Score

The F1 Score uses Precision and Recall in its calculation, and can be interpreted as a weighted average of both. $F1Score = 2 * [(precision * recall)/(precision + recall)]$.

F. k-Fold Cross Validation

Together with RMSE, MAE, Precision, and Recall, we used the k-fold cross validation [24] method. In the k-fold cross-validation, the data is divided into k subsets. Thus, the metrics equations is repeated k times, in which the time one of the k subsets is used as a validation set, and the other $k-1$ subsets are used as the training set. The error estimate is calculated based on all k attempts to get the full effectiveness of the model. By exchanging the training and test sets, we increase the effectiveness of this method. In our study, we used $k = 5$. In Figure 4, we illustrate this process in detail.

Fig. 4. Cross validation process used in our study with $k = 5$.

In Table I, we list not only the 22 algorithms considered in our study, but also the metric values according to the use of statistics. Notice that the algorithms are ranked based on the value of RMSE. In this context, the most efficient algorithms are *Baseline Only*, *SVD*, and *SVD++*. In the next section, we present the second evaluation based on answers of real users, which we performed to triangulate the results and increase evidence in the results.

V. Evaluation Based on Real-User Evaluations

To evaluate the 22 recommendation algorithms by taking into account the perception of real users, we integrated our recommendation system into a tourism application. The main goal of this evaluation is to compare the evaluations of real users with the evaluations provided by the recommendation system (that is, the evaluation based on metrics) for each algorithm considered in our study.

To perform the evaluation, we integrated the recommendation system into the *Hupi* application, which provides two main functionalities: (1) a centralized database with tourist attractions information; and (2) recommendations of tourist attractions for tourists based on their profile. In Figure 5, we present the main screen of the evaluation of the recommendation algorithms based on the *Hupi* application. In this screen, the users could see the tourist attractions and evaluate the attractions based on their perceptions by clicking on the number of stars. The user could choose to not evaluate any tourist attraction, and the system automatically generates a new attraction for evaluation, as the user might not know some attractions listed in the page.

Fig. 5. The screen of the *Hupi* application used during our evaluation with real users.

The evaluation consists of two steps. In the first step, the system selects a number of tourist attractions and presents to the users, which evaluate each attraction based on their perceptions. This first step is used in our system to model the profile of the user. Based on the profile of the user, the systems runs the recommendation system to measure the possible rates for a set of tourist attractions. In the second step, the system selects a number of attractions, ask the users to evaluate again, and compare the rates of users with the rates generated by the recommendation system.

To select participants for our evaluation, we sent emails asking people to participate in the experiment. Overall, 172 users

Algorithm	Error
KNN Z Score with cosine similarity	18.08
KNN Baseline with Pearson correlation similarity	19.04
SVD++	20.04
KNN Means with with cosine similarity	20.08
KNN Means with MSD	20.62
Baseline Only	21.89
KNN Z Score with Pearson baseline similarity	22.38
KNN Means with Pearson correlation similarity	22.64
CoClustering	23.13
KNN Z Score with MSD	26.94
KNN Basic with Pearson baseline similarity	27.22
KNN Basic with Pearson correlation similarity	28.02
KNN Baseline with Pearson baseline similarity	28.85
KNN Baseline with cosine similarity	29.56
KNN Means with Pearson baseline similarity	30.27
SVD	30.76
SlopeOne	33.52
KNN Basic with cosine similarity	35.60
KNN Baseline with MSD	35.76
KNN Basic with MSD	41.49
NMF	42.30
KNN Z Score with Pearson correlation similarity	45.36

TABLE II
RANKING OF RECOMMENDATION ALGORITHMS BASED ON THE ANSWERS OF REAL USERS.

participated in the evaluation. In Table II, we show the results by ranking the algorithms considered in our study. It is important to notice here that the results of algorithms has changed when compared with the results of the first evaluation (see Table I). Error $= \int_1^n |system\ rating(i) - user\ rating(i)|$, where n is the number of evaluations, and i the current evaluation number.

VI. Discussion

We triangulate the results of both evaluations to rank the algorithms. As the evaluations and metrics ranked the algorithms differently, we compared the rankings to suggest the most efficient recommendation algorithms taking into account the results of both evaluations. Thus, we summed the ranking positions of each algorithm in both evaluations, for each metric, to identify the top algorithms based on their efficient. The top algorithms considering the results of both evaluations are *SVD++*, *Baseline Only*, and *KNN Z Score with Pearson baseline similarity*.

We summarise the results in Table III, in which we show the ranking of the algorithms in both evaluations. To rank the algorithms taking the results of the two studies, we calculated the Total Error (TE), by summing the rankings of both studies, that is $TE = RMSE + MAE + Recall + Precision + F1Score + People$, as we were interested in selecting the most efficient algorithms. So, the final ranking represents the algorithms with better ranking considering the two evaluations that we performed in this study, that is, the most efficient algorithms are the ones with the lowest total error values.

VII. Related Work

Many studies that compares recommendation algorithms focus on evaluating the quality of the recommendation system forecasts by using metrics. In [25–27], a recommendation system for tourism is evaluated by using cross-validation and several metrics, such them the Root Mean Squared Error (RMSE), which is also used in our study.

Algorithm	RMSE	MAE	People	Precision	Recall	F1	Total Error
SVD++	3	3	3	12	5	5	9
Baseline Only	1	1	6	17	2	2	8
kNN Z Score with pearson baseline similarity	6	6	7	19	3	3	19
kNN Means with with cosine similarity	9	9	4	4	17	16	22
kNN Z Score with cosine similarity	14	11	1	9	13	13	26
kNN Means with MSD	11	10	5	10	16	17	26
kNN Baseline with pearson baseline similarity	4	4	13	15	7	6	21
kNN Baseline with pearson correlation similarity	15	14	2	8	15	15	31
SVD	2	2	16	14	8	7	20
kNN Means with pearson baseline similarity	5	5	15	20	4	4	25
kNN Basic with pearson baseline similarity	10	15	11	22	1	1	36
kNN Z Score with MSD	12	12	10	6	14	14	34
kNN Baseline with cosine similarity	8	8	14	3	20	21	30
kNN Means with pearson correlation similarity	17	16	8	11	12	12	41
kNN Baseline with MSD	7	7	19	2	21	20	33
CoClustering	18	17	9	5	19	19	44
kNN Z Score with pearson correlation similarity	16	18	13	13	11	11	47
SlopeOne	13	13	17	7	18	18	43
kNN Basic with pearson correlation similarity	21	21	12	21	6	8	54
kNN Basic with cosine similarity	19	20	18	16	10	10	57
kNN Basic with MSD	20	19	20	18	9	9	59
NMF	22	22	21	1	22	22	65

TABLE III

RANKING OF RECOMMENDATION ALGORITHMS BASED ON METRICS AND ANSWERS OF REAL USERS.

Arsan et al. [28] proposes a study by using the *Mahout* framework with the goal of searching the best collaborative filtering algorithms. The study uses the metrics RMSE, and Mean Absolute Error (MAE) metric. In their study, they used movie rating dataset to compare the algorithms. In another study, Seminar and Wilson [29] used the open source *MovieLens* database to evaluate *Mahout* algorithms with the MAE metric.

Said and Bellogín [30] compared the results of some algorithms of the *Mahout*, *LensKit* and *MyMediaLite* frameworks in relation to the RMSE metric and time. In the tourism domain yet, Nilashi et al. [31] proposed a recommendation system for tourist places and evaluated it by using several metrics, such as RMSE, MAE, F1-measure, and Precision.

Noguera et al. [32] developed a mobile hybrid recommendation system by using user-based collaborative filtering combined with item-based filtering. The main contributions of this study is related to identifying the limitations of precision metrics. Liu et al. [33] proposed a new model of user similarity with the goal of improving the accuracy of collaborative filtering. It uses the metrics precision and recall to evaluate its results by claiming that the metrics RMSE and MAE, although widely used, do not guarantee user satisfaction. In our study, we complement these studies by comparing the results of RMSE with the results of a study with real users, which rated a number of tourist places.

Our study complements the existing studies by considering a big data set, by comparing 22 algorithms, using a set of five metrics, as well as by considering an evaluation with real users to get more evidence with the results.

VIII. CONCLUSION REMARKS

In this study, we performed a comparison of 22 recommendation algorithms in the tourism domain. We performed two evaluations: (1) based on metrics; and (2) based on the answers of real users, to rank the algorithms according to their efficiency. By triangulating the results of both studies, we improve evidence regarding the most efficient algorithms for recommendation systems.

The results of the evaluations ranked the algorithms differently. Some recommendation algorithms remain on the top list with regards to efficiency, such as the *SVD++*, *Baseline Only*, and *KNN Z Score with Pearson Baseline Similarity* algorithms, while others are efficient in the first evaluation, but not at the second one, or the other way around.

As future work, we intend to evaluate the recommendation algorithms in different domains, such as movie reviews, and music.

REFERENCES

1. Bunghez, C. L. The Importance of Tourism to a Destination's Economy. *Journal of Eastern Europe Research in Business and Economics,* 1–9 (2016).
2. Kramer, O. in *Dimensionality Reduction with Unsupervised Nearest Neighbors* 13–23 (Springer Berlin Heidelberg, 2013).
3. Koren, Y. *Factorization Meets the Neighborhood: A Multifaceted Collaborative Filtering Model* in *Proceedings of the International Conference on Knowledge Discovery and Data Mining* (ACM, 2008), 426–434.
4. Koren, Y. Factor in the Neighbors: Scalable and Accurate Collaborative Filtering. *ACM Trans. Knowl. Discov. Data* **4,** 1:1–1:24 (Jan. 2010).
5. Lee, D. D. & Seung, H. S. *Algorithms for Non-negative Matrix Factorization* in *Proceedings of the International Conference on Neural Information Processing Systems* (MIT Press, 2000), 535–541.

6. Luo, X., Zhou, M., Xia, Y. & Zhu, Q. An Efficient Non-Negative Matrix-Factorization-Based Approach to Collaborative Filtering for Recommender Systems. *IEEE Transactions on Industrial Informatics* **10,** 1273–1284 (2014).

7. Koren, Y., Bell, R. & Volinsky, C. Matrix Factorization Techniques for Recommender Systems. *Computer* **42,** 30–37 (Aug. 2009).

8. Lemire, D. & Maclachlan, A. Slope One Predictors for Online Rating-Based Collaborative Filtering. *CoRR* (2007).

9. George, T. & Merugu, S. *A Scalable Collaborative Filtering Framework Based on Co-Clustering* in *Proceedings of the IEEE International Conference on Data Mining* (IEEE Computer Society, 2005), 625–628.

10. Melinat, P., Kreuzkam, T. & Stamer, D. *Information Overload: A Systematic Literature Review* in *Perspectives in Business Informatics Research* (eds Johansson, B., Andersson, B. & Holmberg, N.) (Springer int. Pub., Cham, 2014), 72–86.

11. Ricci, F., Rokach, L. & Shapira, B. in *Recommender Systems Handbook* (eds Ricci, F., Rokach, L. & Shapira, B.) 1–34 (Springer US, 2015).

12. Swamynathan, M. *Mastering Machine Learning with Python in Six Steps. A Practical Implementation Guide to Predictive Data Analytics Using Python* 1st ed. (Apress, 2017).

13. Akhil, P. V. & Joseph, S. A Survey of Recommender System Types and Its Classification. *International Journal of Advanced Research in Computer Science* **8,** 486–491 (2017).

14. Deshpande, M. & Karypis, G. Item-based top-N Recommendation Algorithms. *ACM Trans. Inf. Syst.* **22,** 143–177. ISSN: 1046-8188 (2004).

15. Ekstrand, M. D., Riedl, J. T. & Konstan, J. A. Collaborative Filtering Recommender Systems. *Found. Trends Hum.-Comput. Interact.* **4,** 81–173 (2011).

16. Sarwar, B., Karypis, G., Konstan, J. & Riedl, J. *Item-based Collaborative Filtering Recommendation Algorithms* in *Proceedings of the International Conference on World Wide Web* (ACM, 2001), 285–295.

17. Mourão, F., Rocha, L., Konstan, J. A. & Meira Jr., W. *Exploiting Non-content Preference Attributes Through Hybrid Recommendation Method* in *Proceedings of the ACM Conference on Recommender Systems* (ACM, 2013), 177–184.

18. Linden, G., Smith, B. & York, J. Amazon.com recommendations: item-to-item collaborative filtering. *IEEE Internet Computing* **7,** 76–80 (2003).

19. Gomez-Uribe, C. A. & Hunt, N. The Netflix Recommender System: Algorithms, Business Value, and Innovation. *ACM Trans. Manage. Inf. Syst.* **6,** 13:1–13:19 (Dec. 2015).

20. Inversini, A. & Buhalis, D. *Information Convergence in the Long Tail: The Case of Tourism Destination Information* in *Information and Communication Technologies in Tourism 2009* (eds Höpken, W., Gretzel, U. & Law, R.) (Springer Vienna, Vienna, 2009), 381–392.

21. Kluyver, T. *et al. Jupyter Notebooks-a publishing format for reproducible computational workflows.* in *ELPUB* (2016), 87–90.

22. Schreier, S. *Modeling RESTful Applications* in *Proceedings of the Second International Workshop on RESTful Design* (ACM, 2011), 15–21.

23. Grinberg, M. *Flask Web Development: Developing Web Apps with Python* 1st (O'Reilly Media, Inc., 2014).

24. Rodriguez, J. D., Perez, A. & Lozano, J. A. Sensitivity Analysis of k-Fold Cross Validation in Prediction Error Estimation. *IEEE Transactions on Pattern Analysis and Machine Intelligence* **32,** 569–575 (2010).

25. Lim, K. H., Chan, J., Leckie, C. & Karunasekera, S. *Personalized Tour Recommendation Based on User Interests and Points of Interest Visit Durations* in *Proceedings of the International Conference on Artificial Intelligence* (AAAI Press, 2015), 1778–1784.

26. Lim, K. H. *Recommending Tours and Places-of-Interest Based on User Interests from Geo-tagged Photos* in *Proceedings of the ACM SIGMOD on PhD Symposium* (ACM, 2015), 33–38.

27. Lim, K. H., Chan, J., Leckie, C. & Karunasekera, S. Personalized trip recommendation for tourists based on user interests, points of interest visit durations and visit recency. *Knowl. Inf. Syst.* **54,** 375–406 (2018).

28. Arsan, T., Koksal, E. & Bozkus, Z. Comparison of Collaborative Filtering Algorithms with Various Similarity Measures for Movie Recommendation. *International Journal of Computer Science, Engineering and Applications* **6,** 1–20 (2016).

29. Seminario, C. E. & Wilson, D. C. *Case Study Evaluation of Mahout as a Recommender Platform* in *Workshop on Recommendation Utility Evaluation: Beyond RMSE* (2012).

30. Said, A. & Bellogın, A. *Comparative Recommender System Evaluation: Benchmarking Recommendation Frameworks* in *Proceedings of the 8th ACM Conference on Recommender Systems* (ACM, Foster City, Silicon Valley, California, USA, 2014), 129–136. ISBN: 978-1-4503-2668-1.

31. Nilashi, M., bin Ibrahim, O., Ithnin, N. & Sarmin, N. H. A multi-criteria collaborative filtering recommender system for the tourism domain using Expectation Maximization (EM) and PCA-ANFIS. *Electronic Commerce Research and Applications* **14,** 542–562 (2015).

32. Noguera, J. M., Barranco, M. J., Segura, R. J. & MartıNez, L. A Mobile 3D-GIS Hybrid Recommender System for Tourism. *Inf. Sci.* **215,** 37–52 (Dec. 2012).

33. Liu, H., Hu, Z., Mian, A., Tian, H. & Zhu, X. A new user similarity model to improve the accuracy of collaborative filtering. *Knowledge-Based Systems* **56,** 156–166 (Jan. 2014).

Who Should Close the Questions: Recommending Voters for Closing Questions Based on Tags

Zhang Zhang[1,2], Xinjun Mao* [1,2], Yao Lu [1,2], and Jinyu Lu[1]

[1]National University of Defense Technology, Changsha 410073, China
[2]Key Laboratory of Software Engineering for Complex Systems, Changsha 410073, China

Abstract—**Stack Overflow is the most popular community with a great number of questions daily post by users. The questions that are unfit for the community should be closed to maintain the quality of questions. Current practices of closing questions in Stack Overflow mainly rely on the votes of experienced users and community moderators, and face several challenges: (1)an increase in both the number of questions that should be closed and the proportion of these questions to all questions. (2) a decrease in participation willingness of non-moderator users to close questions. One way to tackle the problem is to extensively utilize the forces of experienced users in the community by recommending them appropriate questions against their development experiences and skills in order to increase their willingness and decrease their voting efforts. In this paper, we propose a voter recommendation method based on the tags of both users and questions, design user recommending algorithm based on the user willingness model that incorporates the quantitative and time information of user activity history in Stack Overflow. We select 1,397 questions randomly in Stack Overflow to validate the effectiveness of our method. The results show the successful recommendation probability in the top5, top10, top100, top300 users are 35.0%, 45.8%, 80.8%, and 89.8% respectively, which helps recommending users to 'closed' questions.**

Index Terms—**Stack Overflow, question quality control, closing question, voter recommendation.**

I. INTRODUCTION

Stack Overflow(SO) is the most popular community driven by questioning and answering [1] [2] and with more than 10M registered users, 19M questions and 28M answers. It maintains a strong emphasis on question-answer based format, and encourages to post the questions related to programming problems, software algorithms, coding techniques and software development tools and discussions or *chit-chat* are discouraged [3]. However, a large number of questions that are not of the Stack Overflow concern are posted every day. Therefore maintaining question quality on such a large scale social collaborative platform is a great challenge [4]. The questions that are unfit for the community should be closed. Stack Overflow guidelines clearly outline categories of questions that are deemed unfit for its Q&A format, and questions that fall into the pre-defined sets of guidelines are marked 'closed' [5]. A question can be marked as 'closed' for five reasons:

duplicate, off-topic, unclear what you're asking, too broad and primarily opinion-based.

According to the current rules of SO, the decision to 'close' a question lies completely on the shoulders of experienced users with a reputation over 3,000 and community moderators via a systematic voting process. Due to the rapid growth of Stack Overflow's questions and answers, there has been a steady increase in the workload on the experienced users and moderators [3] [6]. In contrast to thousands of questions created every day, the whole community has only 20 moderators and currently they undertake most of the voting tasks. According to the analysis on data of Stack Overflow between 2008 and 2018, the current practices of 'closed' questions faces several critical challenges: (1)the number of 'closed' questions and the ratio of 'closed' questions to total questions show an increasing trend; (2)the participation willingness of non-moderator users to close questions is declining, which has probably led to increase the workload of moderators.

One way to tackle the problem is to extensively utilize the forces of experienced users in the community, encourage them to participate in closing questions. Currently, the experienced users browse the questions in SO and vote for the questions based on their willingness, interest and correlation with their expertise. Obviously, such a method to vote by browsing thousands of questions and judging whether they need to be closed is inefficient. It is necessary to study the automatic method to recommendate users appropriate questions against their experiences and skills to increase their willingness and decrease their voting efforts. The current researches on 'closed' questions are mainly focused on predicting whether the question should be closed [3] [5] [6], and little attention is paid to the treatment of the 'closed' question, such as the study of voters.

In this paper, we propose a voter recommendation method for experienced users in Stack Overflow to improve their participation in the closing question activity. Different from the existing method, we build user willingness model against users' development experiences and skills by analyzing their tag-based activity history in SO. We design and implement the recommendation algorithm that outputs and ranks potential voting users against the tags of the question. The paper also conducts experiments to validate the effectiveness of our

method.

The rest of the paper is organized as follows. Related works are introduced in section 2. section 3 analyzes the current practices and discusses the challenging issues of closing questions in Stack Overflow. Section 4 presents and details our user recommendation method based on tags and section 5 introduces the experiments and analyzes the results. We discuss the conclusion and future work in section 6.

II. RELATED WORK

This section introduces the related works on user recommendation in Community Question Answering(CQA) and question quality control in Stack Overflow.

A. User recommendation in CQA

There have been increasing studies on user recommendation in CQA in recent years [7] [8]. Many studies model users profiles by learning their history of behaviors, and the tag-based information from the previous questions, answers and comments play an important role in this field. Yang *et al.* [9] studied the user expertise under tags and recommended a set of possible expert users for questions to help askers to get their preferable answers. And their method performs better than the up-to-date method. Wang *et al.* [10] propose a novel personalized recommendation method that considers both the topic modeling and the link structure for routing new questions to a group of experts, and the proposed method improves the recommendation performance over other methods in expert recommendation. Wang *et al.* [11] propose the Topic Professional Level Model (TPLM) to find the right experts for questions that combines both the topic model and the professional level model to calculate the user's authority under a specific topic. Their results showed that their method is superior to the traditional expert finding method in the Chinese CQA platform-Zhihu dataset. Liu *et al.* [12] propose a gating mechanism to dynamically combine structural and textual representations based on past question-answering behaviors, and their experiments on Stackexchange and Quora show that our approach can improve the performance on expert finding tasks.

B. Question quality control in Stack Overflow

The question quality control of Stack Overflow is still a great research challenge [4]. Current researches focus on prediction of question quality and suggestions for improving question quality. Correa *et al.* [3] used a machine learning framework and build a predictive model to identify a 'closed' question at the time of question creation and achieve an overall accuracy of 73%. Goyal *et al.* [5] studied the closed questions and then built a classifier that predicted whether or not a question would be closed given the question as submitted, along with the reason that the question was closed. Tóth *et al.* [13] present a novel approach for classifying questions based exclusively on their linguistic and semantic features using deep learning method and they conclude that by combining deep learning and natural language processing

methods, the maintenance of quality at Q&A forums could be supported using only the raw text of posts. Calefato *et al.* [14] investigate how information seekers can increase the chance of eliciting a successful and develop a conceptual framework of factors potentially influencing the success of questions in Stack Overflow.

III. CURRENT PRACTICES AND ISSUES ANALYSIS OF CLOSING QUESTIONS

In this section, we discuss the current practices of 'closed' questions and analyze the potential issues and challenges based on data of StackOverflow[1].

A. Current practices of closing questions

Currently, the decision to 'close' a question in Stack Overflow lies completely on the systematic voting process(seeing Fig.1). The experienced users with a reputation over 3,000 and moderators undertake the voting task. The former can cast a vote to close a question once and 5 votes can close any question, and the latter can close any question with a single vote. In addition, users with a reputation of over 250 can vote to close their questions and users who hold a gold badge for one of the question's tags can close the question as *duplicate* with a single vote. As Fig.1 shows, a question can be marked as 'closed' for five reasons: *duplicate, off-topic, unclear what you're asking, too broad and primarily opinion-based.* According to our analysis on data of SO, up to December 2019, Stack Overflow has closed more than 0.8M questions, and more than 30,000 users participated in the voting tasks.

Fig. 1. **The current practices of closing questions**

B. Issues analysis of closing questions

We collect and analyze the data related to the presence of 'closed' questions in Stack Overflow between 2008 and 2018, and find the following potential issues and challenges of closing questions.

The number of 'closed' questions. Fig.2 shows the number of questions posted each year and the 'closed' questions each

[1]https://archive.org/details/stackexchange

year. First, we find a rapidly increasing trend in the number of 'closed' questions: from 205 in 2008 to 113,292 in 2018. Then, we observe the ratio of 'closed' questions to total questions over this period in Fig.2. We can find a sharp increase in the ratio of 'closed' questions after 2011 and a sharp decrease from 2014 to 2015, finally it is at around 0.056. In other words, at least one out of every 20 questions needs to be marked as 'closed' questions in 2018, which puts tremendous pressure on questions quality control of the community.

Fig. 2. The number of questions and 'closed' questions posted each year

Community Participation We analyze the voting history of experienced users and moderators to understand community participation. A question is marked as 'closed' if it reaches 5 votes but a vote from a moderator can immediately close a question. Therefore, a question can be closed with any number of 'close' votes between 1 to 5. Table I shows the distribution of the number of 'close votes' on closed questions. More than 71% of questions require moderator intervention to close. We also observe a rise in the percentage of questions being closed only by moderators over time, and a decrease in the percentage of questions being closed by experienced users.

TABLE I
THE DISTRIBUTION OF NUMBER OF 'CLOSE VOTES'

Year	1-vote	2-votes	3-votes	4-votes	5-votes
2008	100.00%	0	0	0	0
2009	8.50%	2.05%	1.49%	7.08%	80.88%
2010	4.87%	2.72%	1.94%	1.69%	88.78%
2011	22.15%	8.18%	5.71%	4.44%	59.53%
2012	23.06%	5.56%	2.56%	1.84%	66.98%
2013	8.90%	2.97%	1.79%	1.14%	85.19%
2014	18.09%	5.19%	2.53%	1.53%	72.66%
2015	36.02%	11.66%	4.38%	1.97%	45.97%
2016	41.49%	14.15%	4.93%	2.12%	37.31%
2017	45.92%	14.62%	4.95%	2.04%	32.47%
2018	48.94%	15.42%	4.96%	2.06%	28.62%

Then we further study the difference of the number of votes between the moderators and non-moderators. Table II shows descriptive statistics on the voter distribution from 2008 to 2018. We can find that the number of moderators is much less than that of non-moderators, but they have undertaken most of the voting tasks. The maximum vote number is 15,024, showing that one moderator in SO has made his own contribution

to close at least 15,024 questions. The average vote number of moderators is 2,601, which is about 30 times the average number of non-moderators votes. However, the number of non-moderators is about 550 times the number of moderators. In conclusion, the status of community participation in voting to close the questions has led to a huge workload on moderators according to our analysis.

TABLE II
THE STATISTICS OF MODERATORS AND NON-MODERATORS

Types	Number	Mean of voting number	Median of voting number	Min of voting number	Max of voting number
Moderators	56	2601	2076	4	15024
Non-moderators	30873	86	6	1	20695

Summary. We now summarize the potential issues and challenges of closing question in SO:

- From the perspective of 'closed' questions, there is an increasing number of 'closed' questions and the percentage of 'closed' questions, which requires more votes to participate in closing the questions.
- From the perspective of users who vote to close questions, there is a decrease in community participation of non-moderator users to close the questions, which has probably led to an increase in the workload for moderators, and thereby requires an effective method to encourage non-moderator users to participate in voting.

According to the above analysis, one way to tackle the issues is to seek an effective way to help and encourage experienced users to participate in the voting. The feasible solution is to actively recommendate users appropriate questions for voting against their development experiences and skills. Such a method can increase their willingness and decrease their voting efforts, and therefore improve the efficiency of closing questions and enhance community question quality control.

IV. USER RECOMMENDATION METHOD BASED ON TAGS

This section details our method, including user willingness model incorporating the quantitative information and time information of tag-based information, and the user recommendation algorithm.

A. User willingness model based on tags

One potential voting candidate that is willing to participate in voting questions depends on several factors that are related to the questions and the user itself. These factors include his/her expertise, activeness, etc. We can build the user willingness model based on these factors to help user recommendation. User models can be established by learning their history of behaviors [15], we can analyze the tag information in the user history data to establish an accurate user willingness model.

Quantitative information of tag-based activity history. In a user's quantitative information of tag-based activity history,

509

we extract the frequency of activity about a tag(the number of questioning, answering and commenting related to a specific tag). And it represents the user's willingness on this tag [16]. The more frequently a user participates in a post related to a particular tag, the more interested he is in that tag. We use the $Fre_{tag_i(u)}$ to measure the frequency of tag i for user u, which is defined as:

$$Fre_{tag_i(u)} = number\ of\ activities\ in\ tag\ i\ for\ user\ u. \tag{1}$$

For example, if user u has commented 2 posts related to *java*, posted 1 question related to *java*, and answered 5 questions related to *java*, then $Fre_{tag_{java}(u)}$ = 5+2+1=8.

Time information of tag-based activity history. The user's willingness is dynamically changing [17], thus, the time information of tags is valuable. We extract the recency of user activity about a tag from the time information, that is, the chronological order of activities related to the tag. The activity data close to the current temporal period is usually more important than that temporally far from the current period [17]. This study defines the recency to which user u participate in tags i(abbreviated as $RecT_{tag_i(u)}$) as the following:

$$RecT_{tag_i(u)} = 1 - \frac{Current - Last_{tag_i(u)}}{Current - First_{tag_i(u)}} \tag{2}$$

where *Current* is the time point at which the user tag recency is currently measured. $Last_{tag_i(u)}$ is the last time user u participated in the posts or comments related to tag i. And $First_{tag_i(u)}$ is the first time user u participated in the posts or comments related to tag i.

We also extract the duration of user activity from the time information, that is, the length of time a user has participated in activities related to a specific tag. It is another important factor based on the time information to represent a user's willingness [18]. The long-duration activity history about a tag usually reflects a user's willingness more than the short-duration ones. We use the $Duration_{tag_i(u)}$ to measure the duration of user u's participation in posts or comments related to tag i, which is defined as follow:

$$Duration_{tag_i(u)} = Last_{tag_i(u)} - First_{tag_i(u)} \tag{3}$$

Where $Last_{tag_i(u)}$ and $First_{tag_i(u)}$ have been mentioned above.

Then we use the $ActDuration_u$ to measure the duration of user u's participation in posts or comments, which is defined as follow:

$$ActDuration_u = LastTime_u - FirstTime_u \tag{4}$$

Where $LastTime_u$ is the last time user u participated in the posts or comments and $FirstTime_u$ is the first time user u participated in the posts or comments.

After getting the above two indicators, we use their ratio to measure the duration of the user u's preference for tag i(abbreviated as $DurT_{tag_i(u)}$) as the following:

$$DurT_{tag_i}(u) = \frac{Duration_{tag_i(u)}}{ActDuration_u} \tag{5}$$

User willingness model. The user willingness model is composed of several model elements for each tag. To construct user u's model element for tag i, this paper uses $Pre_{tag_i(u)}$ to combine the frequency, recency and duration of user u's activity history about tag i, which is defined as follow:

$$Pre_{tag_i(u)} = Fre_{tag_i(u)} * (\alpha * RecT_{tag_i(u)} + \beta * DurT_{tag_i(u)}) \tag{6}$$

Where α and β are used to control the relative impact of $RecT_{tag_i(u)}$ and $ActDuration_u$, and $\alpha + \beta$ =1($0 \leq \alpha, \beta \leq 1$).

Fig.3 shows user u's activity history, and user u's willingness model elements for tag *java* are based on it. Firstly, we extract quantitative information and time information of activities related to *java* as follows:

- User u posted a question related to *java* on *2019-02-14*, answered a post related to *java* on *2019-03-15*, commented a post related to *java* on *2019-04-01*, so $Fre_{tag_{java}(u)}$ = 1 +1 +1 = 3 according to Eq.1.
- $RecT_{tag_{java}(u)} = 1 - \frac{2019-05-01 - 2019-04-01}{2019-05-01 - 2019-02-14} \approx 0.605$ with $First_{tag_{java}(u)}$ = *2019-02-14*, $Last_{tag_{java}(u)}$ = *2019-04-01*, $Current$ = *2019-05-01* according to Eq.2.
- $DurT_{tag_{java}(u)} = \frac{2019-04-01 - 2019-02-14}{2019-04-01 - 2019-02-01} \approx 0.779$ with $FirstTime_u$ = *2019-02-01*, $LastTime_u$ = *2019-04-01*, $First_{tag_{java}(u)}$ = *2019-02-14*, and $Last_{tag_{java}(u)}$ = *2019-04-01* according to Eq.3, Eq.4 and Eq.5.

Then, we get $Pre_{tag_{java}(u)}$ = $3*(0.5*0.605+0.5*0.779) \approx 2.08$ with $\alpha = 0.5$ and $\beta = 0.5$ according to Eq.6.

Fig. 3. An illustration of the user's activity history

B. User recommend algorithm

This section proposes a user recommendation algorithm based on the above user willingness model. The algorithm inputs the question with tags and the candidates with their activity history and outputs ranked candidates based on the user willingness model on the tags of the question to be voted, and the top k users will be recommended to vote for the question.

Firstly, we input the question that needs votes and get the tags of this question. We also need to get the candidates and their activity history. Then, we get the $PerScore_u$(the sum of the candidate u's user willingness model for tags of this question) for each user, and rank them based on the $PerScore_u$. Finally, we select top k users to vote for this question. The details of the algorithm are as follows:

Algorithm: User Recommend

Input:
      tag_list: the list of tags of the question that needs voters.
      user_list: users who belong to the candidate list.
      history_list: activity history of users who belong to user_list.
Output:
      RecommendedUser-list: ranked users
Procedure:
1:     let *PerScore* be a list
2:     for u in *user_list*:
3.   let $PerScore_u = 0$
4:        for i in *tag_list*:
5:           calculate the $Pre_{tag_i(u)}$[1]
6:           $PerScore_u$ += $Pre_{tag_i(u)}$
7:           $PerScore$.append($PerScore_u$)
8:     sort *PerScore* based on the $PerScore_u$
9:     return *PerScore*

[1]The user u's willingness model for tag i

V. EXPERIMENTS AND RESULTS

In this section, we will describe the design of our experiment, and analyze the results of the experiment.

A. Experiment design

In order to validate the effectiveness of our recommendation method, firstly, we construct the candidates' user willingness model based on their activity history in SO, then select the 'closed 'questions in SO and use our recommendation method to recommend experienced users to vote these questions. We then analyze the overlap between the users who actually vote to close these questions and our recommended users.

Data selection. We collect user activity history from January 2018 to September 2019 in SO, and select randomly 1,397 questions that were closed between January 2019 and September 2019 by the experienced users and use them to perform the experiment.

Evaluation metrics. We evaluate the overlap between the actual voters and the recommendation result by *r_top@k*. We define *r_top@k* as follows: if any actual voter of a 'closed' question ranks among the top k in our recommendation results, this question is called `hit question`. And *r_top@k* = (the number of `hit question`)/(the number of all tested questions). Our candidate list is consist of 30,000+ experienced users who have participated in closing the questions, so we choose the 1% of the candidates as the maximum value of k: 300. On the other hand, any question can be closed if it reaches 5 votes, so the minimum value of k is set to 5. Thus, k of *r_top@k* is varied as 5, 10, 100, and 300.

Parameter settings. We use 5 different sets of α and β in our experiment: (1)$\alpha = 0$, $\beta = 1$; (2)$\alpha = 0.3$, $\beta = 0.7$; (3)$\alpha = 0.5$, $\beta = 0.5$; (4)$\alpha = 0.7$, $\beta = 0.3$; (5)$\alpha = 1$, $\beta = 0$. For the construction of the user willingness model, we have taken the activity history of the questions, answers and comments of the users in the 30 days and 365 days before the tested question is posted respectively.

B. Results and analysis

The experiment results are shown in Table III. From the time period of the data used to build the user willingness model, the effectiveness of the user recommendation method based on the 30-day data is better than that using the 365-day data when the values of α and β are the same. We speculate that the latter contains more user activity history, but the old data may be misleading to reflect the user's willingness with the user's willingness changing over time. From the values of α and β, with the increase of α and the decrease of β, the effectiveness of the user recommendation method is gradually improved, and reaches the best when $\alpha = 1$ and $\beta = 0$. This may be because the recency of the activity is more expressive of the user's willingness than the duration of the activity in the time information or the indicator $DurT_{tag_{java}(u)}$ we use to measure the duration of the user's willingness is not accurate.

TABLE III
THE EFFECTIVENESS OF USER RECOMMENDATION METHOD WITH
DIFFERENT PARAMETER SETTINGS

Time period	α	β	r_top@5	r_top@10	r_top@100	r_top@300
30 days	0	1	0.346	0.450	0.797	0.88
30 days	0.3	0.7	0.348	0.452	0.803	0.89
30 days	0.5	0.5	0.349	0.455	0.803	0.89
30 days	0.7	0.3	0.349	0.455	0.805	0.894
30 days	1	0	0.350	0.458	0.808	0.898
365 days	0	1	0.334	0.422	0.772	0.867
365 days	0.3	0.7	0.335	0.424	0.777	0.879
365 days	0.5	0.5	0.335	0.424	0.779	0.88
365 days	0.7	0.3	0.336	0.424	0.779	0.882
365 days	1	0	0.337	0.424	0.778	0.881

Overall, the user recommendation method using the 30-day data to build the user willingness model is the most effective with $\alpha = 1$ and $\beta = 0$ in our experiment: r_top@5 = 0.350, r_top@10 = 0.458, r_top@100 = 0.808, r_top@300 = 0.898. Then we use this set of parameters to analyze the effectiveness of our method for different kinds of 'closed' questions(seeing Fig.4). For r_top@5 and r_top@10, our method is most effective for the questions closed as *Unclear what you're asking*: r_top@5 = 0.395, r_top@10 = 0.499. However, our method is not ideal for the questions closed as *Too broad* and *Primarily opinion-based*. We speculate that the user's willingness to participate in the closing questions is also affected by the ease of identifying the reasons of the closing questions.

Fig. 4. The effectiveness of our method for different kinds of 'closed' questions

We pick 10 most frequently occurring tags in the test set and verify the effectiveness of our method for questions related to these tags(seeing Table IV) with above parameter settings. Because our test set is randomly selected, the situation of the tag frequency distribution in the test set is similar to that in the community. There are significant differences in the effectiveness of our method for questions related to different tags: `c#` with r_top@5 = 0.102, r_top@10 = 0.284, `python-3.x` with r_top@5 = 0.519, r_top@10 = 0.597, and so on. This indicates that the effectiveness of our method is influenced by the tags of 'closed' questions, and this may be because the popularity of a tag affects users' willingness to participate in voting activities related to the tag.

TABLE IV
THE EFFECTIVENESS OF OUR METHOD FOR 'CLOSED' QUESTIONS
RELATED TO DIFFERENT TAGS

Tag	r_top@5	r_top@10	r_top@100	r_top@300	Number
python	0.431	0.546	0.838	0.927	260
javascript	0.289	0.371	0.811	0.962	159
java	0.194	0.3125	0.792	0.917	144
php	0.402	0.413	0.772	0.913	92
c#	0.102	0.284	0.83	0.92	88
python-3.x	0.519	0.597	0.909	0.961	77
html	0.365	0.486	0.824	0.905	74
c++	0.233	0.466	0.795	0.932	73
android	0.125	0.266	0.609	0.781	64
r	0.365	0.476	0.984	0.984	63

VI. CONCLUSION AND FUTURE WORK

To close the unfit questions in CQA is extremely significant in order to manage and guarantee the quality of the question and the whole community. Current practices of closing questions face several challenges, which requires encouraging experienced users to participate in closing questions and increases community efficiency. One way to solve these challenges is to actively recommend experienced users appropriate questions against their development experiences and skills, instead of relying on them to randomly browse the questions to determine whether they need to vote in the past. In this paper, we present an effective method to actively recommend users for questions in CQA. Our contribution of this paper is threefold:(1) obtaining some important findings about the potential issues and challenges of voting. (2)building a user willingness model based on the relationship of tags of both users and questions by extracting the quantitative and time information of user activity history. (3)proposing a user recommendation algorithm that outputs and ranks the potential voters for questions. We conduct experiment to validate the effectiveness of our proposed method. The experiment results are positive and impressive in successful recommendation.

In the future, we plan to use more dimensional indicators of tag-based information to model users, such as active time. In addition, the users' collaboration in the past vote history will also be included in the recommendation basis.

REFERENCES

[1] Zhu J, Shen B, Cai X, et al. Building a Large-scale Software Programming Taxonomy from Stackoverflow[C]//SEKE. 2015: 391-396.
[2] Wang T, Yin G, Wang H, et al. Automatic knowledge sharing across communities: A case study on android issue tracker and stack overflow[C]//2015 IEEE Symposium on Service-Oriented System Engineering. IEEE, 2015: 107-116.
[3] Correa D, Sureka A. Fit or unfit: analysis and prediction of'closed questions' on stack overflow[C]//Proceedings of the first ACM conference on Online social networks. 2013: 201-212.
[4] Ponzanelli L, Mocci A, Bacchelli A, et al. Improving low quality stack overflow post detection[C]//2014 IEEE international conference on software maintenance and evolution. IEEE, 2014: 541-544.
[5] Mukerjee A. Predicting Closed Questions on Stack Overflow[J]. 2013.
[6] Lezina C G E, Kuznetsov A M. Predict closed questions on stackoverflow[J]. 2013.
[7] Huang W, Mo W, Shen B, et al. CPDScorer: Modeling and Evaluating Developer Programming Ability across Software Communities[C]//SEKE. 2016: 87-92.
[8] Zhao Z, Yang Q, Cai D, et al. Expert Finding for Community-Based Question Answering via Ranking Metric Network Learning[C]//Ijcai. 2016, 16: 3000-3006.
[9] Yang B, Manandhar S. Tag-based expert recommendation in community question answering[C]//2014 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM 2014). IEEE, 2014: 960-963.
[10] Wang L, Wu B, Yang J, et al. Personalized recommendation for new questions in community question answering[C]//2016 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM). IEEE, 2016: 901-908.
[11] Wang S, Jiang D, Su L, et al. Expert Finding in CQA Based on Topic Professional Level Model[C]//International Conference on Data Mining and Big Data. Springer, Cham, 2018: 459-465.
[12] Liu Z, Zhang Y. Structures or texts? a dynamic gating method for expert finding in cqa services[C]//International Conference on Database Systems for Advanced Applications. Springer, Cham, 2018: 201-208.
[13] Tóth L, Nagy B, Janthó D, et al. Towards an Accurate Prediction of the Question Quality on Stack Overflow Using a Deep-Learning-Based NLP Approach[C]//14th International Conference on Software Technologies, ICSOFT 2019. SciTePress, 2019: 631-639.
[14] Calefato F, Lanubile F, Novielli N. How to ask for technical help? Evidence-based guidelines for writing questions on Stack Overflow[J]. Information and Software Technology, 2018, 94: 186-207.
[15] Yu Y, Wang H, Yin G, et al. Reviewer recommender of pull-requests in github[C]//2014 IEEE International Conference on Software Maintenance and Evolution. IEEE, 2014: 609-612.
[16] Golder S A, Huberman B A. Usage patterns of collaborative tagging systems[J]. Journal of information science, 2006, 32(2): 198-208.
[17] Zheng N, Li Q. A recommender system based on tag and time information for social tagging systems[J]. Expert Systems with Applications, 2011, 38(4): 4575-4587.
[18] Huang C L, Yeh P H, Lin C W, et al. Utilizing user tag-based interests in recommender systems for social resource sharing websites[J]. Knowledge-Based Systems, 2014, 56: 86-96.

A Deep Spatio-temporal Residual Network Model for Commercial Activeness Prediction

Ping Liang, DongjinYu*, Xiaoxiao Sun

School of Computer Science and Technology
Hangzhou Dianzi University
Hangzhou, China
liangpingprivate@gmail.com, yudj@hdu.edu.cn, sunxiaoxiao@hdu.edu.cn

Abstract—**Activeness of regional stores represents the evolvement of their corresponding commercial districts, whose prediction helps practitioners grasp the trend of commercial development and provides support for urban layout. Traditional prediction methods, however, mostly rely on time series analysis and cannot model the complex nonlinear space-time relationship of commercial development as a geographic phenomenon. Inspired by the outstanding performance of deep learning in the field of image and video processing, this paper proposes a deep spatio-temporal residual network model (DSTRN) for commercial activeness prediction using online reviews and check-in records of stores. Specifically, our model includes a spatial dimension that employs local CNN to capture the spatial relation of surrounding commercial districts, and a temporal dimension that applies 3D convolutions and LSTM to deal with the temporal characteristics of commercial activeness. Meanwhile, a residual network is introduced to eliminate gradient vanishing and exploding caused by depth increasing of neural networks. In particular, the recent variations (e.g., sequential changes) and periodic variations (e.g., seasonal changes or holiday effects) of commercial development are both taken into consideration in the model for better prediction. Experiments on public Yelp datasets of Toronto from 2013 to 2018 demonstrate that DSTRN vastly outperforms other approaches and reduces the mean square error by 51.2%, 57.5% and 8.5% compared to Historical average (HA), Autoregressive integrated moving average (ARIMA) and XGBoost, respectively.**

Keywords-Commercial activeness prediction, Commercial district, Spatio-temporal analysis, Deep learning, Yelp

I. INTRODUCTION

Urban commercial districts are key functional areas in cities with high density of shopping malls, restaurants, entertainment facilities and other commercial entities [1]. Activeness of these entities in turn represents the evolution of their corresponding commercial districts, whose prediction could help practitioners grasp the trend of business development to achieve commercial success and help the government to make accurate urban layout policies.

In recent years, the concepts of big data and smart city have been put forward, which introduce a new idea for predicting commercial activeness based on the massive amount of data collected from different aspects of cities. On the other hand, with the rise of Yelp and other online review apps, a great number of reviews and check-in records of various commercial entities given by visitors and consumers become available. Since business performance, such as revenues and profits, of individual entities are commercial secrets with only limited access, how to employ these publicly available online data to predict the variations of commercial activeness is a topic with practical significance [2]. Traditionally, time series prediction methods such as autoregressive model (ARIMA) are widely applied in commercial prediction field [3,4], which however usually fail to capture the complex non-linear spatio-temporal variations of the commercial activeness as a geographic phenomenon.

To address the above problems, we propose a novel model to predict commercial activeness as a spatio-temporal problem. Our approach is motivated by the outstanding performance of deep learning techniques on handling non-linear relations, more specifically on the prediction of spatio-temporal scenarios such as air pollution prediction [5] and taxi demand scheduling [6]. According to the first law of geography "Near things are more related than distant things", commercial entities in the same district affect the commercial activeness of each other. In order to capture these spatial relations, we apply local Convolutional Neural Networks (local CNN) on commercial activeness prediction, which ensure that close spatial relations are captured and remote spatial relations are excluded. During analysis of the Yelp datasets, we notice that the review and check-in data show periodic changes in some commercial entities. For example, the number of reviews and check-ins of ice cream stores becomes significantly larger in summer than in winter; more customers visit shopping malls in holidays than in workdays. In order to reflect this reality, we adopt the 3D convolutions to extract the recent variations and periodic rules, respectively, and employ Long Short-Term Memory (LSTM) model to synthesize them as the temporal characteristics for commercial activeness prediction.

In summary, the main contributions of this paper are summarized as follows:

- We propose a Deep Spatio-Temporal Residual Network model (DSTRN) to predict the activeness of commercial districts based on online reviews and check-in records of commercial entities.

---

*Corresponding author

- We design a spatial dimension in DSTRN that employs local CNN to capture the spatial relations of surrounding commercial districts, and a temporal dimension that applies 3D convolutions and LSTM to deal with the temporal characteristics of commercial activeness (including periodic and recent variations).

- Experiments on public Yelp datasets demonstrate that DSTRN vastly outperforms HA, ARIMA and XGBoost by 51.2%, 57.5% and 8.5% in terms of the mean square error, respectively.

The rest of the paper is structured as follows. After discussing related work in Section 2, we introduce some basic definitions and raise our problem in Section 3. Section 4 presents our prediction model in detail. Experiment results are given in Section 5. Finally, Section 6 concludes the paper and introduces the future work.

II. RELATED WORKS

Space and time are two fundamental dimensions related to all geographic researches. For a long time, spatio-temporal analysis and modeling of geographic parameters have been the main focus of GIScience, such as urban changes [7], land resources utilization [8], and environmental issues [9]. Commercial activeness, which changes with the aggregation and evolution of commercial districts, is also a geographic issue affected by complex space and time factors. Traditional studies on this issue are generally done through investigating local conditions, which is unsustainable and relies too much on field survey [10]. Recent studies attempt to explore the utilities of big data, such as social and review data generated from mobile apps (e.g., Twitter, Yelp, etc.), in commercial activeness prediction. For example, Yang et al. [11] applied a clustering algorithm to aggregate commercial districts based on multiple online data and used a linear model to predict commercial activeness. Hu et al. [2] presented a raster transformation model of check-in data from Internet to analyze commercial district. Wang et al. [12] proposed an approach to the business failure prediction with mobile location-based check-ins. These studies, however, did not recognize the variations of commercial activeness as a non-linear spatio-temporal issue, which led to relatively low accuracy on the prediction results.

Deep learning, which has been vastly applied in the field of image and video processing, is found to be able to handle complex non-linear relations and further to complete spatio-temporal prediction. He et al. [13] put forward a multi-view ensemble neural network to predict commercial hotness, and in some sub-neural networks they introduced CNN. Zhang et al. [14] presented a deep learning model with CNN to predict urban congestion. In these studies, however, the entire research area was input into CNN as one image, which failed to capture the local relations among the surrounding areas and falsely included irrelevant relations of remote entities. In addition, Ji et al. [15] demonstrated that 3D convolution can perceive not only spatial features, but also temporal features compared with 2D convolution in the field of video. Recently, LSTM has been applied to solve spatio-temporal issue due to its outstanding ability in capturing temporal relations. Chen et al. [16] applied LSTM to forecast urban housing price and Kong et al. [17] utilized LSTM to forecast urban power load. Since LSTM cannot reflect the spatial relations, researchers thought that combining CNN and LSTM can capture both spatial and temporal characteristics. For example, Huang et al. [5] applied the CNN-LSTM model to predict air particulate matter (PM2.5). However, due to the complex model architecture, the depths of deep learning network increased sharply, which led to gradient vanishing and exploding and reduced the effectiveness of capturing spatio-temporal relations [18]. On the other hand, LSTM alone cannot capture both the periodic temporal relations (e.g., seasonal changes or holiday effects) in spatio-temporal modeling, which is of vital importance in long-term prediction [14].

In conclusion, commercial activeness prediction is a complex and non-linear geographic issue, which contains both spatial and temporal variations. But until now, these two characteristics have not been adequately extracted at the same time. Furthermore, the temporal features of commercial activeness should be divided into periodic and recent dimensions. Unfortunately, existing methods, even those based on deep learning, however, cannot capture these two dimensions simultaneously and effectively. Therefore, in this paper, we are dedicated to proposing a deep learning model based on local CNN, 3D convolution, LSTM and residual network for commercial activeness prediction.

III. PRELIMINARIES

In this section, we presents the preliminaries which helps understand the model we presented.

A. Definitions

Definition 1 (Commercial District). The aggregation of commercial entities in space forms a *Commercial District*, which has a diffusion and radiation effect on its surrounding space.

Definition 2 (Commercial Activeness). The total number of reviews and check-ins from one commercial entity is defined as its *Commercial Activeness*. Similarly, the *Commercial Activeness* of one commercial district is the sum of reviews and check-ins of all commercial entities in this commercial district, defined as follows:

$$y = Sum_{\#Review} + Sum_{\#Checkin} \tag{1}$$

B. Problem Definition

Commercial Activeness Prediction: Given the commercial activeness of all grids in the commercial district before a given time t (including t), the problem is to predict the commercial activeness of any grid (i, j) at the time of $t + 1$, defined as:

$$y_{t+1}^{ij} = \mathcal{F}\left(y_{t_{start}}^{ij}, \ldots, y_t^{ij}\right) \tag{2}$$

where i and j represent a grid in the commercial district on 2D coordinate, t_{start} represents the time when the earliest data are used as input and $\mathcal{F}$ is the prediction model.

C. 3D convolution

3D convolution is proved to be effective to capture spatio-temporal features in the field of video convolution [15]. When 2D convolution is applied on video identification, several contiguous frames in the video are treated as multiple channels in the image, which reduces the tensor dimension and makes the network less sensitive to temporal features. However, 2D convolution applies a 2D filter, which only captures spatial features. 3D convolution, by contrast, employs a 3D filter and the increased dimension helps capture the temporal information from contiguous time spans.

D. Residual Network

Since the ability of only one convolution layer to capture information is limited, we try to capture depth-dimension information by increasing the numbers of convolution layers. However, training deeper neural networks usually leads to gradient vanishing and exploding. To overcome this problem, we introduce a residual neural network into our model. Residual neural network contains several residual units, each one of which includes two convolution layers and one batch normalization layer. One residual unit is defined as:

$$X^{l+1} = \mathcal{F}_{res}\left(X^l\right) + X^l \qquad (3)$$

where X^l and X^{l+1} are the input and output of the lth residual unit, and $\mathcal{F}_{res}$ is a residual function.

IV. PREDICTION OF COMMERCIAL ACTIVENESS

In this section, we introduce the details of DSTRN, i.e., our prediction model $\mathcal{F}$. As is shown in Fig.1, the framework is mainly composed of two parts, which are used to capture spatial correlation and temporal correlation, respectively. As for temporal correlation, both periodic variations and recent variations are considered.

Figure 1. The framework of deep spatio-temporal residual network.

A. Preprocessing

Commercial entities are spatially discrete, which are not conducive to the capture of their spatial correlations. To eliminate this obstacle, we rasterize the commercial district into a series of grids and sum the counts of reviews and check-ins about the business entities located inside a certain grid to indicate the commercial activeness of the corresponding grid as Fig.2 shows. We set the time span as one month and obtain a time sequence expressed as $\Gamma = \{T_0, T_1, T_2, T_3, \dots, T_t\}$. For each time slot T_t, we rasterize the commercial activeness and obtain a set of grid activeness represented as $\xi^t = \{P_{00}^t, P_{01}^t, P_{02}^t, \dots, P_{ij}^t\}$.

Figure 2. Superposition of reviews and check-ins to indicate commercial activeness.

B. Spatial Information Extraction

As is mentioned above, in previous spatio-temporal studies, the whole research area (i.e., the commercial district) is input into CNN as image with each grid having its commercial activeness as its value. To focus on correlations among the surrounding areas and eliminate irrelevant relations of remote areas, we introduce local CNN to extract spatial information of commercial activeness. The following steps demonstrate the process of image segmentation as is shown in Fig.3:

1) For each time slot T_t, we place the commercial district in the first quadrant of a 2D coordinate system.

2) We extract grid P_{ij} and its surrounding grids as an $S * S$ image starting from the origin of coordinates, where S is an odd number to maintain P_{ij} in the center of image.

3) We then move P_{ij} one grid to the right and repeat the segmentation operation until the right edge of the last segmented image reaches the right border of the research area.

4) Afterwards, we move P_{ij} one grid up each time until the upper edge of the segmented image reaches the upper border of the research area.

So far, we obtain an image tensor set expressed as $Y_t^{ij} \in R^{S*S}$, in which every image has commercial activeness as pixel value of each grid. The local CNN takes Y_t^{ij} as an input image to convolutional layers, which is defined as follows:

$$Y_t^{ij,k+1} = f\left(Y_t^{ij,k} * W_t^k + b_t^k\right) \qquad (4)$$

where * denotes the operation of CNN and f is an activation function. $Y_t^{ij,k}$ is the input of the kth CNN layer. W_t^k and b_t^k are learnable parameter sets. Since the task is to predict the commercial activeness of the central grid within an $S * S$ image, our model does not involve any subsampling and pooling operations

Figure 3.   Image segmentation for local CNN (when S is 3).

C. Temporal Correlation Extraction

The temporal feature extraction of commercial activeness applies a 3D convolution, in which the image tensor Y_t^{ij} in each time span is regarded as a video frame. 3D convolution is in fact an increased-dimension 2D convolution. So local CNN is applied in each 2D dimension of 3D convolution to capture spatial information. Considering the difference between periodic and recent characteristics of commercial activeness, a periodic neural network is designed together with a recent neural networks as Fig.1 shows.

When designing the periodic neural network, several 3D convolutions are applied to extract periodic spatio-temporal information. For each 3D convolution, the activation function is represented as:

$$\mathcal{Y}_t^{ij,k+1} = f\left(\mathcal{Y}_t^{ij,k} * W_t^k + b_t^k\right) \qquad (5)$$

where * denotes the operation of 3D convolution, $\mathcal{Y}_t^{ij,k}$ is the input of the kth 3D convolution layer, and W_t^k and b_t^k are learnable parameter sets. After several 3D convolution layers, we flatten the output tensor $\mathcal{Y}_t^{ij} \in R^{S*S}$ to a feature vector $v_t^{ij} \in R_t^{S^2}$ for grid (i,j) at time slot t. Finally, a fully connected layer is applied to reduce the length of spatio-temporal feature vector v_t^{ij}, which is defined as:

$$\zeta_t^{ij} = f\left(W_t^{FC3d} v_t^{ij} + b_t^{FC3d}\right) \qquad (6)$$

where f is an activation function, and W_t^{FC3d} and b_t^{FC3d} are learnable parameters sets. So far, we get a periodic spatio-temporal information vector $\zeta_{t\_periodic}^{ij} \in R^l$, where l means length of the vector. Thus, the periodic data in p periods generate p $\zeta_{t\_periodic}^{ij}$, which is then integrated into one tensor $\eta_{t\_periodic}^{ij} \in R^{l*p}$.

When designing the recent neural network, we assume that recent correlation is more relevant to the prediction result and hence we input more recent data to our model. Based on this idea, we increase the depth of recent correlation extraction neural networks, i.e., increasing the number of 3D convolution layers. Since increasing layers leads to gradients vanishing, we introduce residual network to extract recent information. As mentioned above, in each residual unit, two 3D convolution layers and one batch normalization layer are employed, together with several residual units, as Fig.1 shows. After a flatten layer and a fully connected layer, we obtain a recent spatio-temporal information tensor $\eta_{t\_recent}^{ij} \in R^{l*q}$, in which q represents the number of recent time slots.

In order to automatically assign different weights to recent correlation and periodic correlation, LSTM network is applied in our model. The concatenation of the two tensors $\eta_{t\_periodic}^{ij}$ and $\eta_{t\_recent}^{ij}$ is as follows:

$$\theta_t^{ij} = \eta_{t-period}^{ij} \oplus \eta_{t-recent}^{ij} \qquad (7)$$

Here, the tensor θ_t^{ij} contains $p + q$ periods of spatio-temporal information. θ_t^{ij} is then fed into LSTM and an outputting tensor is obtained as δ_t^{ij}.

D. Prediction

For the final prediction, our goal is to obtain all values of grid activeness for time slot $t + 1$. Since tensor δ_t^{ij} already contains spatial and temporal correlations, a fully connected layer is applied to calculate the final prediction value $\hat{y}_{t+1}^{ij}$ as follows:

$$\hat{y}_{t+1}^{ij} = f\left(W_t^{FC} \delta_t^{ij} + b_t^{FC}\right) \qquad (8)$$

where W_t^{FC} and b_t^{FC} are learnable parameters, and f is the activation function. The final result is normalized to [0,1] and denormalized back to real commercial activeness values. According to the order that local CNN segments images, the predicted values are calculated back to obtain the predicted grid values of grid activeness.

E. Loss Function

Our model is trained through minimizing the value of loss function iteratively, which is defined as the average absolute error (MAE) between the real commercial activeness value and the predicted one as Eq.(9) shows:

$$L_\theta = \frac{1}{m} \sum_{i=1}^{m} \left| y_t^{ij} - \hat{y}_t^{ij} \right| \qquad (9)$$

where θ is all learnable parameters in DSTRN, and m is the number of samples.

V. EXPERIMENTS

A. Experimental Settings

1) Datasets: We selected the open dataset[1] from Yelp for the experiments, which includes more than 668,000 reviews and more than 160,000 check-in records of stores in various cities from October 2004 to November 2018. We chose Toronto as the target city to evaluate the performance of the proposed method DSTRN.

After investigation, we selected the data from January 2013 to December 2017 as the training dataset, and the data from 2018 as the testing dataset, and adopted one month as the time span. When testing the prediction results, we used the same month in the previous 3 years and the previous 11 months to predict the commercial activeness in the next time slot. In our experiment, the activeness values smaller than 10 were filtered, which is a common practice in practical applications [6].

2) Parameters Settings: We implemented DSTRN[2] by Keras, which is a fast experimentation neural networks API running on top of TensorFlow. The experiments ran on a cluster with four NVIDIA 1080Ti GPUs. During experiments, we applied min-max normalization on training datasets to normalize input values to [0,1]. After the DSTRN prediction, we reversed the min-max normalization to recover commercial activeness values.

In our experiment, the size of surrounding grids was set to $7 * 7$, which corresponds to a 7km∗7km actual rectangle area. For the periodic part of DSTRN, we used 3 3D convolution layers with 32 filters, of which the size is $3 * 3 * 3$. For the recent part of DSTRN, we set the number of residual units to 12, and 32 filters with size of $3 * 3 * 3$ were applied in the 3D convolution layers of each residual unit. In all convolution layers, we adopted ReLU as the activation functions. For LSTM, we set the dimensions of hidden state vector as 512. In the final fully connection layer, we adopted Sigmoid as activation function. As for other parameters, the batch size was set to 64, and the learning rate was 0.001. Finally, we applied Adam as optimizer.

3) Evaluation Metrics: We use mean square error (MSE) and mean percentage error (MAPE) as accuracy indicators of the model, which are defined as follows:

$$MSE = \frac{1}{m} \sum_{i=1}^{m} \left(y_t^{ij} - \hat{y}_t^{ij}\right)^2 \qquad (10)$$

$$MAPE = \frac{1}{m} \sum_{i=1}^{m} \left|\frac{y_t^{ij} - \hat{y}_t^{ij}}{y_t^{ij}}\right| \times 100 \qquad (11)$$

where y_t^{ij} is the observed activeness value, $\hat{y}_t^{ij}$ is the predicted value for grid (i, j) at time slot t and m is the number of samples.

4) Comparison Methods: We compared our model with three other methods, i.e., Historical average (HA), Autoregressive integrated moving average (ARIMA), and XGBoost.

- **Historical average (HA):** A traditional prediction method, in which the predicted value is the average value of previous historical commercial activeness values.

- **Autoregressive integrated moving average (ARIMA):** A traditional time series analysis method, which does not consider the variations of other relevant random variables.

- **XGBoost:** A gradient boosting tree model, which has gained widely popularity recently after many winning teams of competitions used it.

B. Experiment Results

In this section, we demonstrate the experimental results of DSTRN compared with other methods in the first experiment and we further try to adjust parameters in our model to see their influence on the model performance (i.e., size of surrounding grids, the numbers of recent months and the numbers of residual units) in the second experiment.

1) Model Performance: Figure 4 shows that DSTRN achieves the lowest MSE (i.e., 26.25) and the lowest MAPE (i.e., 16.20) among all methods. Compared to the second-best model (i.e., XGBoost), DSTRN improves 8.5% in MSE and 26.9% in MAPE, respectively. Compared to traditional methods such as HA and ARIMA, machine learning and deep learning models obviously have better performances in terms of MSE and MAPE.

Figure 4.   Comparision of different methods.

2) Parameter Influence: In the second experiment, we explore how the number of residual units influences the accuracy of our model as shown in Fig.5a. When the number of residual units ranges between 11 and 13, MSE and MAPE are relatively stable. Otherwise, MSE rises significantly due to the lack of deep mining of commercial activeness. However, when the number of residual units increases to a certain amount, the gradients vanishes gradually. Since MAPE better reflects the overall prediction accuracy of the model, we finally apply 12 residual units in our model.

In addition, because the size of surrounding grids determines the input image size of local CNN, we also tried to figure out which size is best for the model. As shown in Fig.5b, when the size is selected as $7 * 7$, MSE and MAPE both obtain their optimal values. When we increase the size to $13 * 13$, MSE and MAPE rise significantly. The reason is that several unrelated locations are included in CNN and hence reduces the precision

---

[1]https://yelp.com/dataset

[2]http://dbsi.hdu.edu.cn/CommercialActiveness/

of the model. MSE and MAPE also increase slightly as the size decreases to $5 * 5$ for not considering enough surrounding relations.

Figure 5. Comparison of DSTRN with different parameters

VI. CONCLUSION

In this paper, we propose a model called DSTRN which employs online reviews and check-in records to predict commercial activeness by month. Specifically, our model includes a spatial dimension that employs local CNN to capture the spatial relation of surrounding commercial districts and a temporal dimension that applies 3D convolutions to deal with the temporal characteristics of commercial activeness. Considering the obvious periodic and recent patterns of commercial activeness, DSTRN handles periodic and recent temporal dimensions simultaneously and applies LSTM to combine the both. In addition, to avoid the gradients vanishing and exploding caused by increasing of convolution layers, the residual network is applied in DSTRN. Experiments on public Yelp datasets of Toronto from 2013 to 2018 demonstrate that DSTRN vastly outperforms other methods.

For future optimization, we plan to add more related information about commercial activeness, such as the layout of subways, as the input of the prediction other than just reviews and check-in records. Besides, the semantics of reviews should also be considered in the future to improve the performance of the model.

REFERENCES

[1] Wang, Fang, Li, Yan, and Gao, Xiaolu. "A SP survey-based method for evaluating environmental performance of urban commercial districts: A case study in Beijing." Habitat International 53 (2016): 284-291.

[2] Hu, Qingwu,Wang, Ming, and Li, Qingquan. "Urban hotspot and commercial area exploration with check-in data." Acta Geodaetica et Cartographica Sinica 43.3 (2014): 314-321.

[3] Nath, Bhola, D. S. Dhakre, and Debasis Bhattacharya. "Forecasting wheat production in India: an ARIMA modelling approach." Journal of Pharmacognosy and Phytochemistry 8.1 (2019): 2158-2165.

[4] Zhu, Bangzhu, and Julien Chevallier. "Carbon price forecasting with a hybrid arima and least squares support vector machines methodology." Pricing and Forecasting Carbon Markets. Springer, Cham, 2017. 87-107.

[5] Huang, Chiou-Jye, and Ping-Huan Kuo. "A deep cnn-lstm model for particulate matter (PM2. 5) forecasting in smart cities." Sensors 18.7 (2018): 2220.

[6] Yao, H., Wu, F., Ke, J., Tang, X., Jia, Y., Lu, S., Gong, P., Ye, J., Chuxing, D., and Li, Z. "Deep multi-view spatial-temporal network for taxi demand prediction." Thirty-Second AAAI Conference on Artificial Intelligence. 2018.

[7] Clarke, Keith C., and Leonard J. Gaydos. "Loose-coupling a cellular automaton model and GIS: long-term urban growth prediction for San Francisco and Washington/Baltimore." International journal of geographical information science 12.7 (1998): 699-714.

[8] Wrenn, Douglas H., and Abdoul G. Sam. "Geographically and temporally weighted likelihood regression: Exploring the spatiotemporal determinants of land use change." Regional Science and Urban Economics 44 (2014): 60-74.

[9] Liu, Y., Zheng, Y., Liang, Y., Liu, S., Rosenblum, D.S.: Urban water quality prediction based on multi-task multi-view learning. Proceedings of the 25th International Joint Conference on Artificial Intelligence. pp. 2576-2582 (2016).

[10] Mandhachitara, Rujirutana, and Randall Shannon. "The Formation and Sustainability of same Product Retail Store Clusters in A Modern Mega City." Tijdschrift voor economische en sociale geografie 107.5 (2016): 567-581.

[11] Yang, S., Wang, M., Wang, W., Sun, Y., Gao, J., Zhang, W., and Zhang, J. "Predicting commercial activeness over urban big data." Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies 1.3 (2017): 1-20.

[12] Wang, lei, Gopal, Ram, Shankar, Ramesh, and Pancras Joseph. "On the brink: Predicting business failure with mobile location-based checkins." Decision Support Systems 76 (2015): 3-13.

[13] He, Zhiyuan, and Su Yang. "Multi-view Commercial Hotness Prediction Using Context-aware Neural Network Ensemble." Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies 2.4 (2018): 1-19.

[14] Zhang, Junbo, Yu Zheng, and Dekang Qi. "Deep spatio-temporal residual networks for citywide crowd flows prediction." Thirty-First AAAI Conference on Artificial Intelligence. 2017.

[15] Ji, Shuiwang, Xu, Wei, Yang, Ming, and Yu, Kai. "3D convolutional neural networks for human action recognition." IEEE transactions on pattern analysis and machine intelligence 35.1 (2012): 221-231.

[16] Chen, Xiaochen, Wei, Lai, and Xu, Jiaxin. "House Price Prediction Using LSTM." arXiv preprint arXiv:1709.08432 (2017).

[17] Kong, Weicong, Dong, Zhaoyang, Jia, Youwei, Hill, D., Yan, Xu and Zhang, Yuan. "Short-term residential load forecasting based on LSTM recurrent neural network." IEEE Transactions on Smart Grid 10.1 (2017): 841-851.

[18] He, Kaiming, Zhang, Xiangyu, Ren, Shaoqing, and Sun, Jian. "Deep residual learning for image recognition." Proceedings of the IEEE conference on computer vision and pattern recognition. 2016.

Collaborative Denoising Graph Attention Autoencoders for Social Recommendation

Nan Mu*†, Daren Zha*†, Lin Zhao*, Rui Gong*

*Institute of Information Engineering, Chinese Academy of Sciences, Beijing, China
†School of Cyber Security, University of Chinese Academy of Sciences, Beijing, China
Email: {munan,zhadaren,zhaolin1,gongrui}@iie.ac.cn

Abstract—In recent years, social recommendation has attracted more and more attention from researchers, because it can effectively solve the problems of data sparsity and cold start. But social recommendation faces two problems: The first is how to deeply integrate social information with user-item preference information to obtain accurate user and item latent vectors. The second problem is how to generate a robust top-N recommendation model from the implicit feedback information. To solve these problems, we propose a novel model: collaborative denoising graph attention autoencoders for social recommendation(CDGAAE). This model uses the currently popular graph neural networks to fuse the interaction preference information graph and social information graph through a multi-head attention message passing mechanism. At the same time, this model delicately merges graph neural networks with denoising autoencoders, which uses the corrupt versions of the original data to make the model more robust and generalized. Finally, we conduct comparative experiments of our model with other baseline algorithms on two real world datasets, and the experimental results prove the superiority of our model.

Index Terms—Denoising Graph Attention Autoencoders , Social Recommendation , Implicit Feedback , Top-N Recommendation

I. Introduction

Recently, with the boom of the Internet and social media, social recommendation is a hot topic in academia and industry. Previous works [1], [2] incorporate social information into the recommendation system from different perspectives and these works have achieved good performance. However, the above methods are shallow models and they cannot deeply merge social information with user-item interaction information. In addition, the robustness and generalization of the recommendation system is also crucial, so how to resist the interference of noise is also a problem we need to consider. In real world, the user's explicit rating information for items is difficult to obtain and what we can usually get is the implicit feedback data (click, browse, etc). At the same time, for ordinary users, they prefer to get a list of items that are most interesting to them. So for the social-based top-K recommendation system, we face the following two problems: The first is how to deeply integrate social information with user-item preference information to obtain accurate user and item latent vectors. The second problem is how to generate a robust top-k recommendation model from the implicit feedback information.

The rapid development of deep learning technology has brought us new ideas for solving the above problems. Graph neural networks [9] generate the accurate representation for nodes through deeply exploring the graph structure, which has achieved significant results. The denoising autoencoders [6] provide a robust feature representation algorithm by using the corrupted version of the original data. Collaborative neural filtering [14] deeply learns the interaction behaviors between users and items.

So in this paper, we propose a novel model: Collaborative Denoising Graph Attention Autoencoders for Social Recommendation (CDGAAE). We deeply integrate the graph neural networks with the denoising autoencoders. For the user-item implicit preference interaction matrix, we use its corrupted version as input to train our model. CDGAAE includes graph attention encoder and collaborative neural filtering decoder. The encoder performs the message passing process based multi-head attention mechanism on the user-item preference graph and the social network graph. Then we can obtain the user latent vector by deeply merging the results of these two graphs, and the item latent vector is generated from the user-item preference graph. The decoder designs a multi-layer neural collaborative recommendation module which takes the latent vectors of users and items as input, and then the decoder reconstructs the original user-item preferences, which can enhance the robustness of the model. For model learning process, since the original data is implicit feedback data, we use a binary cross-entropy loss function and a stochastic gradient descent optimization method. So the main contributions of this article are as follows:

1) We propose a novel model Collaborative Denoising Graph Attention Autoencoders for Social Recommendation (CDGAAE), which can deeply integrate social information and user-item preference information to accurately represent users and items.
2) The CDGAAE model delicately integrates graph neural networks and denoising autoencoders, which making the model more robust and generalized. At the same time, it can make top-K recommendations from implicit feedback data.
3) We compare CDGAAE and many baseline algorithms on two real world datasets, and the experimental results prove the superiority of our model.

519

II. Related Work

A. Social Recommendation

With the booming development of online social media, social recommendation technology has attracted more and more researchers' attention. SocialMF [1] incorporates the mechanism of trust propagation into the social recommendation model because the trust propagation has been shown to be a crucial phenomenon in the social sciences. In order to further improve the performance of recommendation, TrustSVD [2] integrates explicit and implicit influence into the model at the same time. And then the top-K recommendation is also an important research hotspot in social recommendation. SBPR [3] extends the classical BPR [4] method with observation that users tend to assign higher rankings to their friends' favorite items. SPF [5] develops a social poisson factorization method to closely combine ratings with social information.

B. Denoising Autoencoders

The Denoising Autoencoder (DAE) [6] is an extension of the classical autoencoder and the purpose of DAE is to reconstruct the raw input data χ from its (partially) corrupted version $\tilde{\chi}$. The common corrupted methods consist of the additive Gaussian noise and the mask-out/drop-out noise. With this setup, DAE can generate more robust features than the classical autoencoder. The DAE model is widely used in recommendation systems to improve the performance of the framework. Collaborative denoising auto-encoders(CDAE) [7] proposes a top-N recommendation algorithm which utilizes the idea of DAE and the CDAE model is a generalization of several well-known collaborative filtering models but with more flexible components. And then to tackle the data sparsity and cold start problems, the Trust-aware Collaborative Denoising AutoEncoder (TDAE) [8] learn compact and effective representations from both rating and trust data for top-N recommendation.

C. Graph Neural Networks

In recent years, graph neural networks [9] have developed rapidly in the field of representation learning and make remarkable achievements. The representation of each node on the graph structure data has always been a research hotspot and graph neural networks achieve better performance in terms of accuracy and speed than the classical network repersentation methods [10] in many application scenarios. Due to the advantages of graph neural networks more and more recommendation systems adopt these algorithms. GCMC [12] provides a framework which considers matrix completion as a link prediction task and leverage graph autoencoders combining interaction data with side information. GraphRec [13] provides a principled approach to jointly capture interactions with opinions in the user-item graph and introduces the attention mechanism into the model.

III. The Proposed Model

In the classical recommendation system, we usually have a userset U with N users $\{u_1, u_2, \ldots, u_N\}$ and itemset V with M items $\{v_1, v_2, \ldots, v_M\}$. We also have a matrix $\mathbf{R} \in \mathbb{R}^{N \times M}$ that represents the users' preferences for the items. And in this paper we focus on the implicit feedback information because the explicit feedback is often hard to get. In the preference matrix $\mathbf{R}$, if there is an interaction between the user u_i and the item v_j, $r_{ij} = 1$, otherwise $r_{ij} = 0$. And then we also have a matrix $\mathbf{S} \in \mathbb{R}^{N \times N}$, which refers to the social network relationship between users. In the social graph, if u_k has a relation to u_i, $s_{ki} = 1$, otherwise $s_{ki} = 0$.

Now we have the implicit preference matrix $\mathbf{R}$ and the social matrix $\mathbf{S}$, so the goal of our social recommendation system is to pick a top-k list of the most interesting items from the unobserved item set for each user.

A. Overall Structure of the Proposed Framework

In this part, we will introduce the overall structure of our proposed model Collaborative Denoising Graph Attention Autoencoders(CDGAAE). The classical denoising autoencoder has been described in the related work II-B. The common corrupted methods consist of the additive Gaussian noise and the multiplicative mask-out/drop-out noise. And in this paper, we use the mask-out/drop-out corruption, which is widely used in the previous works [7], [8]. The drop-out corruption can be explained that the non-zero values ($r_{ij} = 1$) in the preference matrix $\mathbf{R}$ are randomly dropped out independently with probability q:

$$
\begin{aligned}
P(r_{ij} = 0) &= q \\
P(r_{ij} = r_{ij}) &= 1 - q
\end{aligned}
\tag{1}
$$

And then the autoencoder framework of CDGAAE consists of graph attention encoder and neural collaborative filtering decoder. The encoder performs the message passing process based multi-head attention mechanism on the user-item implicit preference graph and the social network information graph. Then we can obtain the user latent vector by deeply merging the results of these two graphs, and the item latent vector is generated from the user-item preference graph. The decoder designs a multi-layer neural collaborative filtering module which takes the latent vectors of users and items as input, and then the decoder reconstructs the original preferences between users and items, which can enhance the robustness and generalization of the model. For model learning process, since the original data is implicit feedback data, we use a binary cross-entropy loss function and a stochastic gradient descent optimization method.

B. Graph Attention Encoder

The purpose of the encoder module is to learn user latent vector $\mathbf{h}_i$ and and item latent vector $\mathbf{h}_j$. The graph attention encoder part shown in Fig.1 contains three message passing processes. So $\mathbf{h}_i$ is from user-item implicit preference graph and social network graph, and $\mathbf{h}_j$ from user-item preference graph. Following the initialize method of NeuMF [14], for the one hot embedding user i and item j, we pass them through two multi-Layer perceptrons, then we can get the initial embedding $\mathbf{u}_i \in \mathbb{R}^d$ and $\mathbf{v}_j \in \mathbb{R}^d$. Next we will introduce the generation method of these two latent vectors $\mathbf{h}_i$ and $\mathbf{h}_j$.

Fig. 1. Overall Structure of the Proposed Framework.

1) User Latent Vector: For each user, firstly, the features of all directly connected items in the user-item preference graph are collected to obtain the embedding vector $\mathbf{h}_i^U$. The second aggregation is the message passing of the associated users in the social graph and we can get another embedding vector $\mathbf{h}_i^S$. Finally, the deep fusion of these two vectors can form a new embedding vector, that is user latent vector $\mathbf{h}_i$.

In the user-item preference graph, for user i, $\mathcal{O}_i$ denotes the set of items which this user interacted with. Firstly we need to calculate how important the item $\mathbf{v}_j$ is to the user $\mathbf{u}_i$, which we call the attention coefficients e_{ij}:

$$e_{ij} = f(\mathbf{W}_u \mathbf{u}_i, \mathbf{W}_u \mathbf{v}_j) \qquad (2)$$

In this equation, f is a function mapping and we can implement it with a variety of neural network structures. $\mathbf{W}_u$ denotes a weight matrix, of which the purpose is to make reasonable linear transformations for the $\mathbf{u}_i$ and $\mathbf{v}_j$. For each node in the graph, weight matrix $\mathbf{W}_u$ and function f are shared. This sharing strategy is inspired by the weight sharing of convolutional neural networks. During the message passing process, we consider all the items from the set $\mathcal{O}_i$, so we pass the attention coefficients through a softmax function to get the final weight of each node:

$$\alpha_{ij} = softmax(e_{ij}) = \frac{exp(e_{ij})}{\sum_{k \in C(i)} exp(e_{ik})} \qquad (3)$$

In our model, we use a standard multi-Layer perceptron g_u to implement the mapping function f, so the calculation of specific α_{ij} is as follows:

$$\alpha_{ij} = \frac{exp(g_u(\mathbf{W}_u \mathbf{u}_i \oplus \mathbf{W}_u \mathbf{v}_j))}{\sum_{k \in \mathcal{O}_i} exp(g_u(\mathbf{W}_u \mathbf{u}_i \oplus \mathbf{W}_u \mathbf{v}_k))} \qquad (4)$$

$\oplus$ denotes the concatenation of two vectors, so Next we can get the single user embedding vector $\mathbf{h}_i^U$ from the aggregation

of items' characteristics:

$$\mathbf{h}_i^U = \sigma(\sum_{j \in \mathcal{O}_i} \alpha_{ij} \mathbf{W}_u \mathbf{v}_j) \qquad (5)$$

To make the model aggregate more information from different perspectives, we adopt a very popular multi-head attention mechanism [11], which can be described as:

$$\mathbf{h}_i^U = \mathop{\|}_{k=1}^{K} \sigma(\sum_{j \in \mathcal{O}_i} \alpha_{ij}^k \mathbf{W}_u^k \mathbf{v}_j) \qquad (6)$$

In this equation, where $\|$ represents concatenation of the vectors, and α_{ij}^k, $\mathbf{W}_u^k$ denote the k-th attention weights and linear transformation's weight matrix.

Now we hava generated the vector $\mathbf{h}_i^U$ for user u_i from the user-item preference graph. Next, we will introduce the generation method of social embedding vector $\mathbf{h}_i^S$ from the social network graph. we also use the multi-head attention for message passing and the calculation of attention coefficients is similar to Eq.3:

$$\alpha_{ij} = \frac{exp(g_s(\mathbf{W}_s \mathbf{u}_i \oplus \mathbf{W}_s \mathbf{u}_j))}{\sum_{o \in \mathcal{S}_i} exp(g_s(\mathbf{W}_s \mathbf{u}_i \oplus \mathbf{W}_s \mathbf{u}_o))} \qquad (7)$$

g_s is a multi-Layer perceptron and $\mathbf{W}_s$ denotes a weight matrix. $\mathcal{S}_i$ is the collection of users which have relationship with user $\mathbf{u}_i$, then we can generate the vector $\mathbf{h}_i^S$ through the multi-head attention mechanism:

$$\mathbf{h}_i^S = \mathop{\|}_{k=1}^{K} \sigma(\sum_{o \in \mathcal{S}_i} \alpha_{io}^k \mathbf{W}_s^k \mathbf{u}_o) \qquad (8)$$

Through the above illustration we've generated the two parts of the user latent vector: user embedding vector $\mathbf{h}_i^U$ from the user-item implicit preference graph and social embedding vector $\mathbf{h}_i^S$ from the social network graph. Both of these vectors are important components of the representation of user

characteristics,so in order to deeply merge the information of two vectors we also use a multi-layer perceptron:

$$\mathbf{h}_i = g_{us}(\mathbf{h}_i^U \oplus \mathbf{h}_i^S) \quad (9)$$

So $\mathbf{h}_i$ is the final user latent vector which deeply integrates the information from user-item implicit preference graph and social network graph.

2) Item Latent Vector: In this part, we introduce the generation of item latent vector $\mathbf{h}_j$ from the user-item interaction graph. For each item $\mathbf{v}_j$, we aggregate the characteristics of users who have interated with this item. So this process is very similar to the generation of user embedding vector $\mathbf{h}_i^U$ and due to the limit length of this article, we only list the important formulas below:

$$\alpha_{ij} = \frac{exp(g_v(\mathbf{W}_v\mathbf{v}_j \oplus \mathbf{W}_v\mathbf{u}_i))}{\sum_{t \in \mathcal{B}_j} exp(g_v(\mathbf{W}_v\mathbf{v}_j \oplus \mathbf{W}_v\mathbf{u}_t))} \quad (10)$$

$$\mathbf{h}_j = \mathop{\|}_{k=1}^{K} \sigma(\sum_{i \in \mathcal{B}_j} \alpha_{ij}^k \mathbf{W}_v^k \mathbf{u}_i) \quad (11)$$

Through the above statement, we finally get the user latent vector $\mathbf{h}_i$ and item latent vector $\mathbf{h}_j$.

C. Neural Collaborative Filtering Decoder

For reconstructing the user-item relations in the preference graph, we proposes a neural collaborative filtering decoder inspired by NeuMF framework [14]. The neural collaborative filtering decoder part is shown in Fig.1, which consists of collaboration layer and neural collaborative filtering layers.

The collaboration layer combines user latent vector $\mathbf{h}_i$, item latent vector $\mathbf{h}_j$ and the element-wise product $\mathbf{h}_i \odot \mathbf{h}_j$:

$$\mathbf{P}_{ij} = [\mathbf{h}_i \oplus \mathbf{h}_j \oplus (\mathbf{h}_i \odot \mathbf{h}_j)] \quad (12)$$

$\mathbf{h}_i, \mathbf{h}_j$ is the results from the graph attention encoder part. But in our framework we also introduce the element-wise product of these two vectors $\mathbf{h}_i \odot \mathbf{h}_j$, which depicts the shallow linear user–item interaction. Next, we take $\mathbf{P}_{ij}$ as the input for the neural collaborative filtering layers to get the deep and intrinsic interaction of user-item pairs.

We now define the neural collaborative filtering layers as a multi-layer neural network formulated as:

$$\hat{r}_{ij} = \mathcal{L}_{out}(\mathcal{L}_X(\dots \mathcal{L}_2(\mathcal{L}_1(\mathbf{P}_{ij}))\dots)) \quad (13)$$

$\hat{r}_{ij}$ is the reconstructed value of user i with item j and the $\mathcal{L}_{out}$ is the Logistic function for the output predicted value. $\mathcal{L}_1, \mathcal{L}_2, \dots, \mathcal{L}_X$ are the mapping function and in our experiment, we all use the standard multi-layer perceptron (MLP) to implement the function.

D. Model Learning

When we talk about the model learning process, we first need to generate the negative instances. The set $\mathcal{O}$ represents the user-item pairs $(u_i, v_j, r_{ij} = 1)$ with observed interactions and the set $\tilde{\mathcal{O}}$ denotes the other user-item pairs $(u_i, v_j, r_{ij} = 0)$. In general, the size of $\tilde{\mathcal{O}}$ is much larger than the size of $\mathcal{O}$, so we can't just take the set $\tilde{\mathcal{O}}$ as negative instances.

To balance the positive and negative instances, we randomly sample set $\tilde{\mathcal{O}}$ and get a new set $\mathcal{O}^-$, which size matches the size of set $\mathcal{O}$.

Then we will introduce the model learning process. For recommendation system based on explicit feedback, the loss function is mainly a regression with squared loss. But for the implicit feedback data used in our paper, the square loss function doesn't perform very well because the target value r_{ij} is a binarized 1 or 0, which refers to whether there is any interaction between user i and item j. So in order to learn the parameters of the model better, we constrain the output value $\hat{r}_{ij}$ in the range of [0, 1] with the Logistic function in the output layer. Just like the NCF model [14] described by He et al, we define the finall objective function as follows:

$$L = - \sum_{(i,j) \in (\mathcal{O} \cup \mathcal{O}^-)} r_{ij} \log \hat{r}_{ij} + (1 - r_{ij}) \log(1 - \hat{r}_{ij}) \quad (14)$$

This function is the classical binary cross-entropy loss and we use the stochastic gradient descent algorithm to optimize the model.

IV. EXPERIMENT

A. Experimental Settings

1) Datasets: In our experiments, we choose two real-world public datasets: Ciao[1] and Epinions[2]. These datasets are crawled from two famous commerce website, Ciao.com and Epinions.com, which contain user-item interaction information and social relationships. The ratings in Ciao and Epinions are integers from 1 to 5: $\{1, 2, 3, 4, 5\}$ and the statistics of datasets are illustrated in TABLE I.

To generate the implicit feedback data for our model, we take records greater than or equal to 4 as observed preference interactions and other records as the unobserved preference. Then we iteratively drop users and items with less than 5 interactions. This data processing method is widely used in the previous works [7], [8]. For negative sampling, we randomly sample the unobserved set $\tilde{\mathcal{O}}$ to get the negative instances set $\mathcal{O}^-$. In our experiment, the sampling strategy is that for each user the number of negative instances is 5 times the number of observed instances of this user.

TABLE I
GENERAL STATISTICS OF THE CIAO AND EPINIONS

statistics	Ciao	Epinions
Users	7,375	40,163
Items	106,797	139,738
Ratings	284,086	664,824
Density(Ratings)	0.036%	0.051%
Social Relations	111,781	487,183
Density(Social Relations)	0.205%	0.029%

[1]https://www.cse.msu.edu/ tangjili/datasetcode/ciao.zip
[2]www.trustlet.org/downloaded_epinions.html

2) Evaluation Metrics: We use two classical metrics to evaluate the performance of our top-k recommendation system: NDCG@K and MAP@K.

DCG@K is computed by:

$$DCG@K = \sum_{i=1}^{K} \frac{2^{rel_i} - 1}{\log_2(i+1)} \tag{15}$$

and then NDCG@K is the normalized DCG@K over the ideal iDCG@K.

AP@K is computed by:

$$AP@K = \frac{\sum_{m=1}^{K} P@m \times rel_m}{\min\{K, |\mathcal{Y}_i|\}} \tag{16}$$

$P@m$ represents the precision with m recommended items and by calculating the average of AP@K from all the users we can get the MAP@K.

3) Baselines: We choose the following baselines compared with our model:

- **BPR** [4]. BPR is a typical pairwise ranking method for item recommendation and it achieved competitive results in many datasets.
- **SBPR** [3]. This model extends the BPR algorithm by adding the social network information. SBPR is based on the same idea that a user's behavior can be influenced by the users associated with him on social networks.
- **NCF** [14]. NCF is a neural network based method which learn the internal interactions between users and items through the multi-layer perceptrons.
- **CDAE** [7]. CDAE formulates the top-N recommendation problem using the Denoising Auto-Encoder framework and learns distributed representations of the users and items from the implicit feedback data.
- **TDAE** [8]. TDAE extends the CDAE model and learns compact and effective representations from both rating and trust data for top-N recommendation.
- **GraphRec** [13]. GraphRec provides a state-of-the-art model for the social recommender system, which captures interactions in the user-item graph and social graph.
- **CDGAAE$_{den}$**. It is a variant of CDGAAE, which uses the uncorrupted input for the whole model.

4) Parameter Settings: We implement our model CDGAAE with the famous framework Pytorch and in view of the effectiveness and efficiency we set the final parameters of our model with the following values: For each user, we select 80% of the data as train set to learn the model parameters, 10% for validation and 10% for test. Moreover, we set the batch size and embedding size to 128 and 64, and also the learning rate is 0.001. Through multiple experiments, we set the number of attention heads to 4. Then we use the Gaussian distribution to randomly initialize the model parameters and the activation function is ReLU. For all the baseline algorithms, we read the articles and implement methods carefully to get the best performance.

B. Performance

Table. II shows the perfomance comparison of our model and baseline methods. $*$ represents the best performace except for our method and the boldface represents the best result among all the algorithms. By careful comparison, we can find the following conclusions:

- Deep neural networks have better performance than the shallow models. In the absence of social network information, the results of NCF and CDAE are better than that of BPR, and if we add the social network data, the models TDAE and CDGAAE perform better than SBPR.
- The social network can improve the performace of the recommendation system. SBPR add social information to BPR and in the table we can find that SBPR performs better than BPR. The same conclusion can be drawn from the comparison of CDAE and TDAE.
- GraphRec shows best experimental results apart from CDGAAE$_{den}$ and CDGAAE. This model uses graph neural networks to generate more accurate embedding for users and items. At the same time, GraphRec uses attention mechanism for user-item graph and social graph.
- CDGAAE$_{den}$ uses the uncorrupted input for the model, so it's less robust than CDGAAE. From the results, we can clearly find that the performance of CDGAAE$_{den}$ is worse than that of CDGAAE, and even worse than that of GraphRec method under certain metrics.
- It is clear from the table II that our method CDGAAE performs best among all the algorithms. We deeply fuse the graph neural networks with the Denoising Auto-Encoder, and we also adopt multi-head attention mechanism for message passing.

C. Model Analysis

We now study the performance of our approach under different parameter settings. We mainly analyze the multi-head attention and the embedding size of the latent vector.

1) The impact of multi-head attention: In our graph attention encoder part, user aggregation, item aggregation in user-item interaction graph and social aggregation in social graph all use multi-head attention mechanism. In this method, the number of attention heads is an important parameter, which has a crucial impact on the performance of the model. So we will compare the effects of different quantities of heads on the results. In the parameter analysis setting, we set the number of heads k=1, 2, 4, 8, 16 and the experimental performance is shown in Fig. 2.

From the figure, we can clearly see that experimental performance of multi-head attetion is better than single-head attention. In our experimental environment, when the number of heads is 4, the performance is best for the two datasets. however, when k=16, the results are worse than the baseline GraphRec bacause the dimension of single attention layer is small, which is difficult to learn all the useful information in the two graphs. So for the multi-head attetion mechanism, an appropriate number of heads is the key to improve performance.

TABLE II
PERFOMANCE COMPARISON OF OUR MODEL AND BASELINE METHODS

Datasets	Metrics	Algorithms								
		BPR	SBPR	NCF	CDAE	TDAE	GraphRec	CDGAAE$_{den}$	*CDGAAE*	**Improve**
Ciao	NDCG@10	0.0369	0.0421	0.0437	0.0461	0.0496	0.0503	0.0506*	**0.0532**	**4.89%**
	MAP@10	0.0210	0.0237	0.0241	0.0261	0.0299	0.0307*	0.0305	**0.0315**	**2.61%**
Epinions	NDCG@10	0.0153	0.0188	0.0191	0.0218	0.0244	0.0240	0.0248*	**0.0258**	**4.03%**
	MAP@10	0.0080	0.0105	0.0107	0.0104	0.0127	0.0132*	0.0120	**0.0135**	**2.27%**

Fig. 2. Experimental results under different number of attention heads on two datasets.

Fig. 3. Experimental results under different embedding size d on two datasets.

2) The impact of the embedding size: In this part, we will discuss the effect of embedding size on model performance. We adopt six different embedding sizes 8, 16, 32, 64, 126, 256 for parameter analysis on the two datasets and the experimental performance comparison is shown in Fig.3. In general, with the embedding size increases, the recommendation performance of our model first increases and then decreases. The embedding size of less than 8 and greater then 128 significantly degrades the model performance. This phenomenon demonstrates that if embedding size is small, the model can not fully and accurately represent user and item characteristics, but if the size is large, the complexity of the model is high, leading to performance degradation. So we need to find a suitable embedding size to balance the representation performance and complexity of the model.

V. CONCLUSION AND FUTURE WORK

In this paper, we propose a novel model: Collaborative Denoising Graph Attention Autoencoders for Social Recommendation (CDGAAE). CDGAAE uses a corrupted version of the original data as input, making the representation of the model more robust. At the same time, this autoencoder framework consists of graph attention encoder and collaborative neural decoder, which is used to deeply merge the user-item preference information and social network information. The final experiments are conducted on two real-world datasets, and the results show the superiority of our proposed model.

In the future, in order to improve the accuracy of the recommendation list, we can use the side information of users and items, which is a significant supplement to depict the rich characteristics of users and items. Moreover, in addition to denoising autoencoders, variational autoencoders are also widely used in recommendation systems. Therefore, we will integrate the idea of variational autoencoders into the social recommendation systems in the future, expecting to get better performance.

REFERENCES

[1] Jamali M, Ester M. A matrix factorization technique with trust propagation for recommendation in social networks[C]//Proceedings of the fourth ACM conference on Recommender systems. ACM, 2010: 135-142.

[2] Guo G, Zhang J, Yorke-Smith N. TrustSVD: collaborative filtering with both the explicit and implicit influence of user trust and of item ratings[C]//Twenty-Ninth AAAI Conference on Artificial Intelligence. 2015.

[3] Zhao T, McAuley J, King I. Leveraging social connections to improve personalized ranking for collaborative filtering[C]//Proceedings of the 23rd ACM international conference on conference on information and knowledge management. ACM, 2014: 261-270.

[4] Rendle S, Freudenthaler C, Gantner Z, et al. BPR: Bayesian personalized ranking from implicit feedback[C]//Proceedings of the twenty-fifth conference on uncertainty in artificial intelligence. AUAI Press, 2009: 452-461.

[5] Chaney A J B, Blei D M, Eliassi-Rad T. A probabilistic model for using social networks in personalized item recommendation[C]//Proceedings of the 9th ACM Conference on Recommender Systems. ACM, 2015: 43-50.

[6] Vincent P, Larochelle H, Bengio Y, et al. Extracting and composing robust features with denoising autoencoders[C]//Proceedings of the 25th international conference on Machine learning. ACM, 2008: 1096-1103.

[7] Wu Y, DuBois C, Zheng A X, et al. Collaborative denoising autoencoders for top-n recommender systems[C]//Proceedings of the Ninth ACM International Conference on Web Search and Data Mining. ACM, 2016: 153-162.

[8] Pan Y, He F, Yu H. Trust-aware collaborative denoising auto-encoder for top-n recommendation[J]. arXiv preprint arXiv:1703.01760, 2017.

[9] Zhou J, Cui G, Zhang Z, et al. Graph neural networks: A review of methods and applications[J]. arXiv preprint arXiv:1812.08434, 2018.

[10] Tang J, Qu M, Wang M, et al. Line: Large-scale information network embedding[C]//Proceedings of the 24th international conference on world wide web. International World Wide Web Conferences Steering Committee, 2015: 1067-1077.

[11] Veličković P, Cucurull G, Casanova A, et al. Graph attention networks[J]. arXiv preprint arXiv:1710.10903, 2017.

[12] Berg R, Kipf T N, Welling M. Graph convolutional matrix completion[J]. arXiv preprint arXiv:1706.02263, 2017.

[13] Fan W, Ma Y, Li Q, et al. Graph Neural Networks for Social Recommendation[C]//The World Wide Web Conference. ACM, 2019: 417-426.

[14] He X, Liao L, Zhang H, et al. Neural collaborative filtering[C]//Proceedings of the 26th International Conference on World Wide Web. International World Wide Web Conferences Steering Committee, 2017: 173-182.

Identifying Similar Users Based on Metagraph of Check-in Trajectory Data

Rui Song, Tong Li, Xin Dong, Zhiming Ding
Faculty of Information Technology
Beijing University of Technology
songrui@emails.bjut.edu.cn, litong@bjut.edu.cn, dongxin19@foxmail.com, zmding@bjut.edu.cn

Abstract— **Identifying similar users lay the foundation in many fields, such as friend recommendation, user-based collaborative filtering, and community discovery. It is useful to analyze users' similarity based on check-in data, especially the analysis of spatiotemporal and semantic information. The existing works pursue semantic similarity of user trajectories and cannot distinguish the effects of geographical factors in a fine-grained way. This paper proposes a graph embedding approach to identify similar users based on their check-in data. We firstly identify meaningful concepts of user check-in data, based on which we design a metagraph for representing features of similar user behaviors. Then we characterize each user with a sequence of nodes that are derived through a metagraph-guided random walk strategy. Finally, the sequences are embedded to generate meaningful user vectors that are used to the similarity among users and thus identify similar users. We evaluate our proposal on two datasets, the results of which show that our proposal can outperform the baselines.**

Keywords-user similarity; metagraph; random walk; embedding; check-in data

I. INTRODUCTION

With the fast development of Location-Based Social Network (LBSN) platforms, an increasing number of users can check-in at various Point of Interests (POIs) conveniently and share their experiences, resulting in massive user check-in trajectory data. Such check-in trajectory data contains information about when and where a user visited a POI, which indicates users' behaviors and preferences. Analyzing check-in data has contributed to identifying similar users, laying the foundation in many fields, such as location prediction [1], community discovery [3], user-based collaborative filtering and location recommendation [2].

Some early user similarity studies focus only on the geographic features of the trajectory [4][5][6], which is limited by the geographic distance and ignores the semantics [7]. Recent works have measured the similarity of users by mining the semantics of GPS trajectories [7][8][9], but the geographical impacts are not considered deeply enough. For example, in Fig.1, the three different user trajectories are typical to be treated as the same, and not further distinguished in terms of their detailed geographical information [7][8][9]. However, users who live closer to each other are supposed to be more similar than others. As shown Fig.1, affected by geographical location, *trajectory1* is more similar to *trajectory2* than *trajectory3*. As such, we

argue that the semantic and geographic features should all be considered when calculating user similarity.

Figure 1. Examples of three users' semantic trajectories

To consider both semantic and geographic features, we leverage the graph embedding technique to measure user similarity. Graph embedding techniques have been proven as a practical approach for representing profound meanings of entities and relations. Specifically, the trajectory check-in data contains different types of nodes (user, time, POI, etc.), which establishes a graph with heterogeneous nodes. The detailed semantics of check-in data can be obtained by embedding such a graph based on metagraph, which contributes to calculating the similarity among user trajectories. Some pioneer studies have been done regarding this topic. Dong et al. [14] and Zhang et al. [15] transformed the structure into the input of the embedded model by recording the node sequences of the random walk in the graph. They use metagraph-guided random walk sequences to capture semantic information between different types of nodes and thus to improve the quality of transformation. However, they do not consider spatiotemporal features.

In this paper, we propose a metagraph embedding based approach to identify similar users using their check-in trajectory. The main contributions of this paper are summarized as follows:

1) We identify meaningful concepts of user check-in data, based on which we design a metagraph for representing features of similar user behaviors.

2) We propose an improved metagraph-guided random walk algorithm to adapt to time and location similarity, which is used to characterize each user with a sequence

of nodes. The sequences are embedded to generate meaningful user vectors that can be used to effectively calculate user similarity.

3) We compare our approach with the state-of-the-arts based on two datasets, the result of which show that our approach can outperform others.

The rest of this paper is organized as follows: Section II describes some related works. Section III describes the details of our method, including metagraph of check-in data, the improved metagraph-guided random walk, embedding and user similarity calculation. Section IV evaluates our approach. Section V summarizes the paper.

II. RELATED WORK

This paper identifies similar users by combining the semantic and spatio-temporal features of check-in trajectories. We leverage the metagraph-guided heterogeneous graph embedding to calculate the user's representation and then calculate the similarity. This section will introduce related works of calculating similarity based on semantic trajectory and graph embedding algorithms.

A. User Similarity Based on Semantic Trajectory

Horozov et al. [10] first construct the user activity's vectors by using the user's votes on POIs, and then calculate the Pearson similarity to represent the user similarity. Mazumdar et al. [11] compare user similarity by combining the length, support and check-in distribution of common location sequences using GPS data. However, these methods can't measure users who have similar preferences but live far away. To solve this problem, Ying et al. [7] propose a method to measure the user semantic similarity by Maximal Semantic Trajectory Pattern (MSTP). They first identify the stopping point from the GPS trajectory data, and use the landmarks collected from Google Map to form the semantic trajectory. Then the user similarity is calculated by the weighted average of the maximum semantic trajectories. Chen et al. [8] found that the similarity between two identical users is not equal to 1 in [7]'s work. To this end, Chen et al. propose a method called Maximal Trajectory Pattern (MTP) to fix the shortcomings in [7] by using the longest common semantic patterns. Later, Chen et al. [9] propose a method to calculate the user similarity according to the Common Pattern Set (CPS), they introduce the support value distribution of common patterns to solve the problem of indistinguishable pattern frequencies in literature [8]. However, these methods ignore the influence of geographical factor while studying semantics. We argue that semantic and geographic features should be considered when calculating user similarity.

B. Embedding Learning

Embedding is a way to transform discrete variables into continuous vector representations [23]. In neural networks or graphs, embedding can not only reduce the spatial dimension of nodes, but also represent the nodes in a meaningful way. Embedding learning is to learn the vector representation of nodes in the metric space through specific methods, such as LINE [12], DeepWalk [13], etc.. Deepwalk uses a neural language model (skip-gram) to embed graph. The authors first use random walks to uniformly sample the neighbors of the nodes from the graph as a path. The paths are treated as sentences, and nodes are treated as words. Then the skip-gram model is used to train the representation of the nodes. To better preserve the relationship between nodes during the embedding process, LINE proposes the concepts of the first-order similarity and second-order similarity. However, these works focus on homogeneous networks which have a single type of node or relationship. Some pioneer studies have been done regarding this topic. Dong et al. [14] propose a method (named as Metapath2vec) for embedding in heterogeneous networks. They use a random walk based on metapath to get the sequence of nodes, and then improved the skip-gram model to learn the embeddings. Zhang et al. [15] argue that metagraph has richer semantics than meta-path. They use metagraph to analyze the behaviors of authors' published papers and classify similar authors. However, there is no time and location limit for the author to publish papers, thus their work can not apply to check-in data. Based on these works, this paper leverages the metagraph to study the behavior of users visiting POIs, introduces time and location constraints in random walk algorithms, and learns user's embedded representation.

III. METHODOLOGY

The framework of our method, as shown in Fig.2, named as Metagraph-Guided Embedding (MGE). We first design the metagraph of check-in trajectory data to represent similar user behaviors. Then we characterize each user with a sequence of nodes that are derived through a metagraph-guided random walk strategy. Finally, the sequences are embedded to generate meaningful vectors that are used to calculate user similarity. Here are the basics concepts in this paper, and the details of our framework.

Figure 2.   The framework of our method (MGE).

A. Concepts

We briefly introduce the basic concepts of check-in data and metagraph.

POI: A POI is defined as a place that has a special function or meaning to users (e.g., a school or a bank). In our method, a POI has four attributes: identifier, name, geographical location, and category.

Trajectory: A trajectory is a path that a user takes in space over a period of time. It can be regarded as a spatial point with chronological order, which records the user's geographical locations at different times. The check-in data studied in this paper has POI profiles, which is a trajectory data with location semantic.

Check-in: A check-in is a quadruple *(u, t, g, p)*, where *u* is a user, *t* is a time slot, *g* is a geographical location coordinate (including latitude and longitude), and *p* is a POI. A check-in means a user *u* visiting a POI *p* at a time slot *t* and a geographical location *g*. It is worth noting that we discretize timestamps associated with check-in records into 24-time slots based on hours, as other works have done [16].

Metagraph: A metagraph is a relational hypergraph representing multi-relational and multi-dimensional data [17]. It is a graph with its nodes denoting the entities and its edges representing the interaction between nodes. Different from the traditional graph concepts, each node in metagraph represent a set of an entity. For example, the user, POI, time slot, and geographical location can be regarded as entities. Fig.3 shows the metagraph of our paper.

B. Designing Metagraph

Mining check-in trajectory data can discover user behavior features. We argue that users with similar behaviors have similar check-in characteristics. For instance, users will check-in at similar time slots, geographical locations, or POIs. In order to measure the similarity of user behaviors, we design a metagraph of check-in, which can reflect the meanings that two users visit the same POI when they are at similar time slots and geographical locations. We use *U, T, G, P* to represent the set of users, time slots, geographical locations and POIs, respectively. In particular, user, time slot, geographical location, and POI are defined as entities, and serve as different types of nodes in the metagraph.

Figure 3.   Metagraph of check-in data

Fig.3 shows the metagraph of check-in data, which describes that two users are relevant in check-in activity if they have similar time slots and geographical locations in the same POI. At this point, the readers only need to understand the meaning of the metagraph. The method of defining the similar time slots and similar locations will be described in detail in the next subsection.

C. Metagraph Guided Random Walk

Using a random walk based on metagraph, we can capture meaningful semantics from the data, which find similar user behaviors. Our metagraph of check-in data has the constraints of similar time slots and similar locations. Therefore, we improved

the random walk strategy, defining similar time slots and similar locations to limit the random walk process. Here we show the definition.

Similar time slots. Given two time slots *t1* and *t2* , a time threshold τ, if $|t1\text{-}t2| \leq \tau$, then *t1* and *t2* are similar time slots.

Similarity locations. Given two geographical coordinates *g1* and *g2*, a distance threshold δ, the function *distance(a, b)* represents the geographic distance between the location *a* and *b*, if $distance(g1, g2) \leq \delta$, then *g1* and *g2* are similar locations.

To meet the constraints of time and location, we propose an improved random walk strategy and list the following four principles.

1) *Every node that random walks exists in the instantiated graph network.* Otherwise, it is meaningless.

2) *Random walk starts with user type nodes.* Because this paper studies the similarity of users, we focus on user-type nodes.

3) *Random walks are limited by the structure of the metagraph.* The metagraph is used to describe similar user behaviors. Walking in the structure of metagraph can capture the semantics of metagraph.

4) *Random walks are constrained by time and location nodes.* To ensure similar time slots, the time threshold must be met when randomly walking time-type nodes. In the same way, the distance threshold is met when randomly walking the location nodes to ensure similar locations.

To help understand the above principles, given a graph network of check-in data in Fig.4, Table I shows the correct and incorrect walking sequences, we assume the time threshold is 3 (hours), and locations are similar between nodes. For ease of observation, the example sequences omit the time and location nodes, only leaving the user and POI nodes.

Figure 4.   Example of a graph network.

TABLE I.   EXAMPLES OF RANDOM WALK SEQUENCES

Example	Remark
u1, p1, u2	Correct
u1, p3, u2	Incorrect. Violate the principle 1: p3 is not in graph
p1, u2, p2	Incorrect. Violate the principle 2: the start node is not a user
u3, u2, p2	Incorrect. Violate the principle 3: does not satisfy the metagraph structure
u1, p1, u2, p2	Incorrect. Violate the principle 4: the time difference does not meet the threshold

Next, we formally describe the node transition probability of random walks. Different types of nodes (i.e. user, time, location, POI) constitute a heterogeneous information network (HIN) [18]. Given a HIN $H = (V, E)$ and the metagraph m, where V is a vertex set, E is an edge set. Equation (1) defines the node transition probability.

$$P(v_i | v_{i-1}; m, H) = \frac{1}{N^{\varphi_{v_{i-1}}(v_i)}} \tag{1}$$

Where $\varphi(\cdot)$ is a function of the node type, $\varphi_{v_{i-1}}(v_i)$ represents the type of v_i and the previous node is v_{i-1}, $N^{\varphi_{v_{i-1}}(v_i)}$ represents the number of nodes which of type $\varphi_{v_{i-1}}(v_i)$. If $v_i \notin V$, the probability is 0. It is worth noting that the higher frequency of a user visiting a POI or visiting at a time slot, the higher the probability of such a time slot or POI node being selected. A walk will follow the structure of the metagraph repetitively until it reaches the pre-defined length.

The pseudocode of random walk with time and location constraints algorithm is shown in Algorithm 1. In particular, given a graph network, the random walk starts at the user-type node, walk randomly according to the node transition probability and ends at a given length. Each starting node generates a given number of paths. The output is a file containing meaningful sequences of nodes.

Algorithm 1 Random Walk with Time and Location Constraints	
Input	The HIN $H = (V, E)$; the metagraph m; the walk length wl; the number of walks per node n.
Output	A path.txt that records sequences of random walks
1.	**For** each u in U **do**
2.	random walk to time node $t0$, and location node $g0$
3.	**For** $i \leftarrow 1$ to n **do**
4.	path = [u]
5.	**For** $j \leftarrow 1$ to wl **do**
6.	**While** (1) **do**
7.	walk to the node according (1)
8.	**If** time **or** location similarity are satisfied
9.	append the node into *path*; **break**
10.	**End while**
11.	write path into path.txt
12.	**End for**
13.	**End for**
14.	**End for**

D. Embedded Learning

Through random walks under time and location constraints, we obtain sequences of each node. We aim to convert the sequences into vectors to calculate similarity. With embedded models, given a HIN $H = (V, E)$, the task is to calculate latent representations in d- dimension $X \in R^{|V|*d}$ (a.k.a. embeddings), $d \ll |V|$. Then, we choose the skip-gram model to learn the latent embeddings of nodes. This model has been validated in [14][15]. Specifically, the skip-gram learn node representation by maximizing the probability of the occurrence node v's context nodes *Context(v)* within w window size, as shown in (2).

$$\min \sum_{v \in V} \sum_{v' \in Context(v)} -\log P(v'|v;\theta) \tag{2}$$

Where *Context(v)* denotes v's neighborhood based on the random walks guided by metagraph, $P(v'|v;\theta)$ is modeled via softmax. To speed up training, like other works [19], negative sampling is used to approximate the objective function (3):

$$\log \sigma(X_{v'} \cdot X_v) + \sum_{k=1}^{K} \log \sigma(-X_{v'_k} \cdot X_v) \tag{3}$$

Where X_v is the v_{th} row of X, representing the embedding vector of node v. $\sigma(\cdot)$ is the sigmoid function, v'_k is the k_{th} negative node sampled for node v', and K is the number of negative samples.

E. Calculating User Similarity

The higher the frequency with which two users visit the same POI at similar times and locations, the higher the behavior similarity. Based on this meaning, we get the embedded representation of user nodes. In particular, the closer the user embeddings are in the vector space, the more similar the users are. In vector space, the cosine distance pays more attention to the difference from the vector direction, which helps to distinguish and measure the similarity of users. Therefore, equation (4) use the cosine distance to calculate the user's similarity and normalization to make the result in the range [0,1].

$$\text{sim}(u_a, u_b) = 0.5 + 0.5 \frac{u_a \cdot u_b}{|u_a||u_b|} \tag{4}$$

Where $u_a, u_b \in X$, sim is the user similarity.

IV. EVALUATION

In this section, we evaluate our method (MGE) on two datasets. Specifically, we focus on two research questions (RQ) and designed experiments for each one.

- **RQ1.** Can our method (MGE) calculate user similarity effectively?

To answer this question, we compare our method with the MTP [8] and CPS [9] that focus on the semantic trajectory similarity calculation.

- **RQ2.** Can our method (MGE) identify similar users effectively?

To answer this question, we compared MGE with popular network representation learning methods LINE [12] and Deepwalk [13].

A. Experiment 1

1) Dataset: to observe the results of pairwise user similarity more specifically, and to accurately compare with other literature which using the same dataset, we use the synthetic dataset, which is derived from [9]. The dataset is constructed based on six users' behaviors that first five being from the literature [9], we constructed the same behavior of *u6* as *u3*, but they live far away. The dataset consists of 76 check-ins and 4

POIs from 6 users. Fig.5 shows the six users' behaviors. We use circles to indicate POIs and arrows between POIs to represent the trajectory transition direction, and thicker arrows indicate higher transition frequencies.

Figure 5.   The six users' behaviors (The u6 living far from others).

2)   Similarity metrics. the effective similarity calculation should meet the following principles, with the first four being from [9]. The last is used to verify the geographical impact.

a) sim(u, u') is in range [0,1]

b) sim(u, u') = sim(u', u)

c) sim(u, u) = 1

d) The frequency of visiting the same place affects the similarity of user behavior, i.e. sim(u2, u3) > sim(u1, u3).

e) According to the common sense, the user's geographic location should have an impact on similarity. The similarity between geographic and semantic combination should be higher than considering only semantic similarity, i.e. sim(u3, u6) ≠ 1.

3)   Comparison with Baselines: in terms of calculating user similarity, we choose MTP [8] and CPS [9] as the baselines for comparison. The reason is that after our research, most of the papers that identify similar users are based on GPS data [20][21], and the algorithms of literature [22] based on check-in data are unknown or incomplete, making it difficult to reproduce. Although [8][9] use GPS trajectory data, and their later works are to calculate user similarity by mining trajectory semantics. Since neither of them disclosed semantic trajectory datasets and codes, in order to avoid the negative impact of different datasets' types and the reproduction process on the results, we use the data from [9] and prove their weaknesses and illustrate the rationality of our method. The results of the different methods are given below.

TABLE II.   PAIRWISE USER SIMILARITY

	u1			u2			u3			u4			u5			u6		
	MTP	*CPS*	*MGE*	*MTP*	*CPS*	*MGE*	*MTP*	*CPS*	*MGE*	*MTP*	*CPS*	*MGE*	*MTP*	*CPS*	*MGE*	*MTP*	*CPS*	*MGE*
u1	1.0	1.0	1.0	1.0	0.96	0.88	1.0	0.93	0.83	0.83	0.76	0.81	0.83	0.50	0.62	1.0	0.93	0.66
u2	1.0	0.96	0.88	1.0	1.0	1.0	1.0	0.97	0.89	0.58	0.71	0.80	0.58	0.47	0.63	1.0	0.97	0.67
u3	1.0	0.93	0.83	1.0	0.97	0.89	1.0	1.0	1.0	0.79	0.67	0.77	0.79	0.44	0.57	1.0	1.0	0.79
u4	0.83	0.76	0.81	0.58	0.71	0.80	0.79	0.67	0.77	1.0	1.0	1.0	0.79	0.44	0.58	0.79	0.67	0.59
u5	0.83	0.50	0.62	0.58	0.47	0.63	0.79	0.44	0.57	0.79	0.44	0.58	1.0	1.0	1.0	0.79	0.44	0.57
u6	1.0	0.93	0.66	1.0	0.97	0.67	1.0	1.0	0.79	0.79	0.67	0.59	0.79	0.44	0.57	1.0	1.0	1.0

TABLE III.   COMPARING THE METRICS OF METHODS ACCORDING TO THE VALUES IN TABLE II

	Similarity metrics					Result		
Methods	*a)*	*b)*	*c)*	*d)*	*e)*	*Description*	*User behavior examples*	*Example values in Table II*
MTP	√	√	√	×	×	Violate d): The similarity of *u1*, *u2* and *u3* should not be the same.		sim(u1,u2) = sim(u1,u3) = sim(u2,u3)
						Violate e): The similarity of *u3* and *u6* should not be the same.		sim(u3,u6) = 1
CPS	√	√	√	√	×	Violate e): The similarity of *u3* and *u6* should not be the same.		sim(u3,u6) = 1
MGE	√	√	√	√	√	Meet all metrics		

Results and analysis. In Table II, and we describe the results of different methods separately. For the sake of observation, we list the measurement results of each method in Table III, which satisfy with "√" and do not satisfy with "×", and give detail description and examples.

For MTP, the first three users *u1*, *u2* and *u3*, are not distinguished, which violates the metric *d)*. MTP's weakness has been proven in [9]. For CPS, the similarity between *u3* and *u6* is 1, which violates metric *e)* and cannot find the difference in geographical location.

MGE can distinguish *u3* and *u6* and satisfy all metrics. The reason is that we randomly walked nodes with similar geographical locations while considering semantics. Randomness can reduce but not eliminate the similarities of faraway users.

B. Experiment 2

1) Dataset: we use the Foursquare datasets that is one of the most popular online location-based social networks. This dataset consists of 372,387 check-ins and 90,089 POIs from 4,144 users over four years (December 2009 to July 2013). We construct the graph with users, check-in time and location, and POI as different types of nodes. We use 80% data as the training dataset and the rest as the test dataset.

2) The experiment design: the node classification is used to evaluate MGE. We leverage third-party labels to determine the class of each node. Foursquare offers ten categories of POI (including Arts & Entertainment, College & University, Event, Food, Nightlife Spot, Outdoors & Recreation, Professional & Other Places, Residence, Shop & Service and Travel & Transport), which can be used as labels for POI node. Like literature [15], the user's label is assigned to the category of the user's most visited POIs. In this paper, the node embeddings are as the input to the logistic regression classifier. **F1-measure**, **Precision** and **Recall** are applied to evaluate the performance [24].

3) Parameter selection: in random walks and embeddings, there are several parameters, such as the number of walks per node, walk length, the vector dimension and location threshold δ. We perform an analysis of these parameters in MGE and select appropriate parameters by observing the F1-measure. Fig. 6 shows the results using the control variable method. It can be seen from Fig. 6(a) and Fig. 6(b) that a larger value of walk length and numbers does not mean that the effect is better. F1 peaks at 20 and 50 respectively, but overall the parameters have little effect on the result, and the extreme value of the result is around 0.03. For the location threshold in Fig. 6(c), 10km works best. This shows that users within 10km of their current location are more likely to visit the same place, so their similarity is high. In Fig. 6(d), the dimension of the vector peaked at 96. As the dimension increased, the result did not change much.

4) Comparison with Baselines: in terms of embedding, we compare two popular network representation learning methods, LINE [12], DeepWalk [13]. After parameter selection, we use the same parameters in Table IV for all embedding methods.

Results and analysis. From Fig. 7, the results of LINE and DeepWalk are similar, with the F1-measure of about 0.54. MGE is higher than the baselines, with the F1-measure of about 0.63, indicating that considering similar time slots and similar locations can identify users in the same category better. Overall, our method has better performance than all baselines in F1-measure, Precision and Recall.

(a)

(b)

(c)

Figure 6.   Comparison of different parameters

TABLE IV.   PARAMETER SETTINGS

Parameter	Number of walks per node	Walk length	Vector dimension	τ	δ
value	50	20	96	3h	10km

Figure 7. Comparison of different methods.

V. CONCLUSION

This paper proposes a systematic method for identifying similar users based on user check-in trajectory data. In particular, we base our approach on metagraph embeddings, in which we first designed a metagraph to represent the check-in behavior of similar users. Then, we apply a customized metagraph-guided random walk algorithm to integrate semantic and geographic similarity into our analysis. Finally, the heterogeneous skip-gram model is used to graph embedding so that we can calculate representation vectors of users, and calculate user similarity. We have designed and conducted a series of experiments which have shown the effectiveness of our methods over existing approaches.

REFERENCES

[1] R. Wu, G. Luo, Q. Yang, and J. Shao, "Learning individual moving preference and social interaction for location prediction," IEEE Access, vol. 6, pp. 10675-10687, 2018.

[2] J. Zhang and C. Chow, "TICRec: a probabilistic framework to utilize temporal influence correlations for time-aware location recommendations," IEEE Transactions on Services Computing, vol. 9, no. 4, pp. 633-646, 2016.

[3] C. Xu, L. Zhu, Y. Liu, J. Guan and S. Yu, "Dp-ltod: differential privacy latent trajectory community discovering services over location-based social networks," IEEE Transactions on Services Computing, pp.1-1, 2018.

[4] Y. Zheng, L. Zhang, Z. Ma, X. Xie, and W. Ma, "Recommending friends and locations based on individual location history," ACM Transactions on The Web, vol. 5, no. 1, pp. 109-152, 2011.

[5] E. H. Lu and V. S. Tseng, "Mining cluster-based mobile sequential patterns in location-based service environments," Tenth International Conference on Mobile Data Management: Systems, Services and Middleware, pp. 273-278, 2009.

[6] Q. Li, Y. Zheng, X. Xie, Y. Chen, W. Liu, and W. Y. Ma, "Mining user similarity based on location history," Proceedings of the 16th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, pp. 34.1-34.10, 2008.

[7] J. C. Ying, H. C. Lu, W. C. Lee, T. C. Weng, and V. S. Tseng, "Mining user similarity from semantic trajectories," Proceedings of the 2nd ACM SIGSPATIAL International Workshop on Location Based Social Networks, ACM, pp. 19-26, 2010.

[8] X. Chen, J. Pang, and R. Xue, "Constructing and comparing user mobility profiles for location-based services," Proceedings of the 28th Annual ACM Symposium on Applied Computing, pp. 261-266, 2013.

[9] X. Chen, R. Lu, X. Ma, and J. Pang, "Measuring user similarity with trajectory patterns: principles and new metrics," Web Technologies and Applications, pp. 437-448, 2014.

[10] T. Horozov, N. Narasimhan, and V. Vasudevan, "Using location for personalized POI recommendations in mobile environments," International Symposium on Applications and the Internet (SAINT'06), pp. 6-129, 2006.

[11] P. Mazumdar, B. K. Patra, R. Lock, and S. B. Korra, "An approach to compute user similarity for gps applications," Knowledge-Based Systems, vol. 113, pp. 125-142, 2016.

[12] J. Tang, M. Qu, M. Wang, M. Zhang, J. Yan, and Q. Mei, "LINE: large-scale information network embedding," Proceedings of the 24th International Conference on World Wide Web, pp. 1067-1077, 2015.

[13] B. Perozzi, R. Al-Rfou, and S. Skiena, "Deepwalk: online learning of social representations," Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp.701-710, 2014.

[14] Y. Dong, N. V. Chawla, and A. Swami, "Metapath2vec: scalable representation learning for heterogeneous networks," Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, pp. 135-144, 2017.

[15] D. Zhang, J. Yin, X. Zhu, and C. Zhang, "MetaGraph2Vec: complex semantic path augmented heterogeneous network embedding," Advances in Knowledge Discovery and Data Mining, Springer, Cham, pp.196-208, 2018.

[16] T. Qian, B. Liu, Q. V. H. Nguyen, and H. Yin, "Spatiotemporal representation learning for translation-based poi recommendation," ACM Transactions on Information Systems, vol. 37, no. 2, pp. 18.1-18.24, 2019.

[17] Z. Huang, Y. Zheng, R. Cheng, Y. Sun, N. Mamoulis, and X. Li, "Meta structure: computing relevance in large heterogeneous information networks," Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 1595-1604, 2016.

[18] Y. R. Lin, J. Sun, P. Castro, R. B. Konuru, H. Sundaram, and A. Kelliher, "MetaFac: community discovery via relational hypergraph factorization," Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp.527-536, 2009.

[19] T. Mikolov, I. Sutskever, K. Chen, G. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," Proceedings of the 26th International Conference on Neural Information Processing Systems, vol. 26, pp. 3111-3119, 2013.

[20] L. Guo, X. Gao, B. Wu, H. Guo, H. Y. Xu, and Y. Wei, "Discovering common behavior using staying duration on semantic trajectory," Journal of Computer Research and Development, vol. 54, no. 1 pp. 111-122, 2017.

[21] D. Yao, C. Zhang, Z. Zhu, J. Huang, and J. Bi, "Trajectory clustering via deep representation learning," 2017 International Joint Conference on Neural Networks (IJCNN), IEEE, pp. 3880-3887, 2017.

[22] W. You, Z. Chenghu, and P. Tao, "Semantic-geographic trajectory pattern mining based on a new similarity measurement," ISPRS International Journal of Geo-Information, vol. 6, no. 7, pp. 212-, 2017.

[23] H. Y. Cai, V. W. Zheng, and K. C. Chang, "A comprehensive survey of graph embedding: problems, techniques, and applications," IEEE Transactions on Knowledge & Data Engineering, vol. 30, no. 9, pp.1616-1637, 2018.

[24] X. W. Meng, R. C. Li, Y. J. Zhang, and W. Y. Ji, "Survey on mobile recommender systems based on user trajectory data," Journal of Software, vol. 29, pp. 3111-3133, 2018.

Formal Security Analysis for Blockchain-based Software Architecture

Nacha Chondamrongkul*, Jing Sun†, Ian Warren‡

Department of Computer Science
The University of Auckland
Auckland, New Zealand
*ncho604@aucklanduni.ac.nz † jing.sun@auckland.ac.nz ‡ i.warren@auckland.ac.nz

Abstract—During the design phase, security as a non-functional requirement needs to be analysed to address vulnerabilities in the architecture design. Without such analysis, security vulnerabilities can be propagated to the implementation. However, security analysis is an error-prone task, especially in complex systems that apply blockchain technology. Without proper security controls applied, the interaction among software components and the blockchain may pose security risks. This paper presents a security analysis approach based on a formal model of blockchain-based architecture design. Our approach can automatically identify specific security vulnerabilities and generate informative scenarios that show how attacks may impact the blockchain. We have evaluated our approach with an example system and found it performs well in identifying an extensible class of security vulnerabilities.

Index Terms—Software Architecture, Security Analysis, Blockchain, Formal Method

I. Introduction

Blockchain is an emerging technology that many software engineers apply to secure data and resources in software systems. To protect data against tampering, blockchain is usually applied as software components that provide data storage, computation services and control functions [1]. Designing the software architecture for blockchain-based systems involves off-chain components that are outside the blockchain network and on-chain components that have access to a node in the blockchain network. With the combination of on-chain and off-chain components, protecting a blockchain-based system is not limited to the blockchain, but the structure and behaviour of other components that interface with the blockchain should also be considered to eliminate risks of attacks [2]. At the design stage, beside verifying the design against the functional requirements, the architecture security analysis task is conducted using software architecture design models [3]. This task involves tracing through the components and security configuration to pinpoint any security flaws in the architecture design. It helps to prevent the architectural security flaws propagating to the implementation.

There are approaches proposed to verify security in the software architecture design in general, such as [4] and [5]. These approaches apply logical rules and metrics that are hard-coded in source code. Almorsy et al. [6] presented an extensible tool to identify different security attacks based on

the signatures representing security metrics and vulnerabilities. However, the blockchain-based system has specific interaction behaviour between on-chain and off-chain components that requires tracing through different attack scenarios to find both direct and indirect impact. The scenario-based traceability is not yet addressed by the existing approaches. There are approaches particularly proposed to analyse security in the blockchain. Luu et al. [7] enhanced the operational semantics of Ethereum to prevent security bugs in the smart contract. Chaieb et al. [8] proposed a verifiable protocol that applies encryption to ensure privacy and security properties. Some approaches [9] [10] [11] have been proposed to verify security in the smart contract by analysing source code. However, these approaches focus either on the structure within the blockchain or the implementation of the blockchain that does not yet exist at the design stage. There is still lack of approaches that can analyse security in blockchain-based application at the software architecture level.

This paper presents an approach that supports architecture security analysis based on a formal model of blockchain-based architecture design. At the design stage, our analysis approach aims at verifying security as a non-functional requirement after the design has complied with functional requirements. Our approach can automatically identify security characteristics and generate scenarios that show how attacks may have direct or indirect impact on the blockchain. The contribution of our approach can be summarised as follows. First, the formal modelling of blockchain-based software architecture design is proposed to describe the structural and behavioural aspect of blockchain-based systems. Second, a set of formally described security characteristics representing security metrics and vulnerabilities is presented. This set is extensible to support other characteristics not addressed in this paper. Last, our approach has been implemented as a tool. This tool allows users to seamlessly perform modelling and security analysis of the blockchain-based system at the architectural level.

The rest of this paper is organized as follows. Section II explains a motivating example of blockchain-based system and its security challenges. Section III presents the modelling and analysis approach. Section IV presents the evaluation of our approach. This paper is concluded in Section V.

II. Security in Blockchain-based Architecture

This section discusses an architecture design for an example application that applies blockchain. In addition, this section identifies the security threats in the architecture design.

A. Motivating Example

AgriDigital is a motivating example that we use in this work. In this paper, we briefly introduce the system, but more details can be found in [12]. The system applies blockchain technology to build digital trust between parties in the agriculture supply chain such as farmers, suppliers and transporters. Digital trust allows different parties to track and transfer commodities while financial transactions can be made transparently. The architecture design of AgriDigital can be found in Figure 1. The *AD Web App* and *Provenance Web App* are web applications that allow users to perform administration operations and enter the record of commodities. After a user enters or updates information, *Provenance Integration* publishes it to involved components such as *AD Message Bus* and *Blockchain Message Bus*. *Blockchain Message Bus* notifies *Blockchain Integrator* about updated information. This design applies the Oracle pattern [13] that allows the off-chain component to push information from the external world to the blockchain. Therefore, when *Blockchain Integrator* is notified about the new information, it creates a new block that keeps financial transactions on the *Public Blockchain*, while another block is created and appended on *Private blockchain* for an updated status of commodities. Some status of commodities can be detected and updated through *IOT Sensor* that helps to measure the condition of commodities such as temperature and humidity. These records of updated status are also created as a block on *Private Blockchain*. Reverse Oracle pattern has been applied between *Digital Wallet* and *Public Blockchain*. This pattern allows *Digital Wallet* to fetch financial information from the blockchain for processing.

Some off-chain components, such as *AD AppServer*, *AD WebServer* and *IOT WebApp* are deployed on local dedicated servers, while others, such as *Provenance Integration*, *Provenance WebApp* and *Blockchain Integrator*, are deployed on the public cloud infrastructure. The deployment configuration of these components poses security threats to the on-chain components such as *Private Blockchain*, implemented with Quorum, and *Public Blockchain* implemented with Ethereum.

B. Attack Scenarios

The on-chain components are vulnerable to attack as they are in the request flow triggered by the off-chain components. In this work, we focus on prominence attack scenarios that usually have security impacts on the blockchain, namely data disclosure and data tampering.

1) Data Disclosure: A key decision in designing blockchain-based software is determining whether the data should be placed on-chain or kept off-chain [1]. The same copy of data in the blockchain is shared among all nodes that run in the blockchain network. If adversaries have access to any node, they can also access the data stored in the blockchain.

Fig. 1. AgriDigital System

Even though data on the blockchain can be encrypted, the data could be disclosed if the private key is stolen [2]. Securing the connections and components that access the blockchain is therefore important. In our example system, if *Digital Wallet* or its connection to *Public blockchain* is compromised, the financial transaction can be disclosed.

2) Data Tampering: Data in the blockchain is known to be nearly immutable and tamper-proof since hash functions are one-way and collision resistant [14]. However, when the oracle component feeds data to the blockchain, the data are assumed to be trusted by all participants. If the oracle is compromised, the adversaries could modify data before it is stored on the blockchain. The oracle should be safeguarded by appropriate security controls to prevent this scenario. In our example system, the *Blockchain Integrator* serves as an oracle component. If the *Blockchain Integrator* or the components that feed data to it such as *IOTWebAPI* and *Provenance Integrator* are compromised, the data on *Private Blockchain* and *Public Blockchain* can be tampered with.

C. Security Metrics

Different security metrics are used to measure the security in the software architecture design. In this work, we focus on the security metrics that suit assessing blockchain-based software architecture.

1) Attack Surface: This metric measures the number of weaknesses in the system that the adversaries can use to attack the system. The attack surface is usually where the system is open to the external environment such as where data are entered or the components that are publicly accessible. The lower the number of attack surface, the more secure the system is. In our example, *Public Blockchain* is an attack surface as it allows any entity to join and run a node in the blockchain network. *BCIntegrator* and *IOTWebAPI* are also attack surfaces as they are accessible from the public network.

2) Least Priviledge: This metric ensures that minimal access to critical data or operations is granted to users or other components in the system. From an architectural perspective,

the number of components that can access critical data should be limited. In blockchain-based software, the on-chain components are critical components that should be accessed by no more than necessary off-chain components. In our example, *BCIntegrator* and *DigitalWallet* are the only components that have direct access to the on-chain components.

3) Defense In Depth: This metric measures how security controls are applied at different points in the system. To protect data in the blockchain, the components that access the blockchain should employ security controls at the component, host and network layers. For architectural analysis, we can calculate the ratio of off-chain components that access on-chain components and which apply security controls compared to the total number of off-chain components that access on-chain components. The higher the ratio value is, the more secure the system is. In our example, *BCIntegrator* and *DigitalWallet* should apply authentication and authorization controls, as well as a firewall that prevents incoming malicious traffic.

III. FORMAL SECURITY ANALYSIS

Our formal security analysis for blockchain-based software architecture combines ontology reasoning and model checking techniques, as shown in Figure 2. First, the architecture design is formally modelled as the component and connector (C&C) view and the deployment view. Second, the ontology reasoner is used to identify security vulnerabilities in the model, based on the ontology description of architecture patterns and security characteristics. These ontology descriptions are defined as classes kept in the ontology library. Third, assertions are inserted into the behavioural model based on the identified vulnerabilities. Finally, the model checker processes the assertions against the model and generates the security scenarios.

Fig. 2. Overall Analysis Process

A. Blockchain Architecture Modelling

Both structure and behaviour of the model are essential to analyse the security in the design phase. Structure in architecture design can be formally defined as ontology representation using the Ontology Web Language (OWL). Behaviour respect

Fig. 3. Architecture Patterns for Blockchain

to interaction among components, can be defined using the Architecture Description Language (ADL).

Architecture security analysis is usually performed on the component and connector view (C&C) and the deployment view of software architecture design. These views are therefore semantically described in the design model. Ontology classes are predefined to support the modelling of the architecture design [15]. As shown in Figure 3, we have defined four connector types to support three architecture patterns, namely Oracle, Reverse Oracle and Off-Chain Storage. These patterns, proposed by Xu et al. [13], have been applied to blockchain-based software. In AgriDigital, we applied Oracle and Reverse Oracle, as well as other basic patterns such as Client-Server and Publish-Subscribe (more details can be found in [15]).

The design model of a blockchain-based system can be defined by creating ontology individuals based on defined classes to represent different entities in the C&C view, such as Component, Connector, Port and Role. Another set of individuals is created to represent different entities in the deployment view such as device, execution environment and communication link. These entities are linked to describe how components are deployed within the infrastructure. Due to the page limit, we present a subset of the AgriDigital model as shown below[1].

> *Individual(ex : pubwire*
>   *value(ex : hasRole ex : pubextsupplier)*
>   *value(ex : hasRole ex : pubblockstorage))*
> *Individual(ex : walletwire*
>   *value(ex : hasRole ex : wlextquerier)*
>   *value(ex : hasRole ex : wlblocksupplier))*
> *Individual(ex : pubextsupplier type(ex : Extsupplier))*
> *Individual(ex : pubblockstorage type(ex : Blockstorage))*
> *Individual(ex : wlextquerier type(ex : Extquerier))*
> *Individual(ex : wlblocksupplier type(ex : Blocksupplier))*

The *pubwire* individual represents a connector that links *Public Blockchain* and *Blockchain Integrator*. The *walletwire* individual represents a connector that links *Public Blockchain* and *Digital Wallet*. The code below shows some individuals

---

[1]The complete OWL model of AgriDigital can be found at http://bit.ly/3bm18YB

representing components and ports. The individual representing the component *BlockchainIntegrator* has *bcint* port that attaches to *pubextsupplier* defined above. This definition applies the Oracle pattern. The *checkfinance* port attached to the *wlextquerier* role applies the reverse oracle pattern.

> Individual(ex : BlockchainIntegrator
>   value(ex : hasPort ex : bcint))
> Individual(ex : PublicBlockchain
>   value(ex : hasPort ex : logfinance)
>   value(ex : hasPort ex : checkfinance))
> Individual(ex : bcint
>   value(ex : hasAttachment ex : pubextsupplier))
> Individual(ex : logfinance
>   value(ex : hasAttachment ex : pubblockstorage))
> Individual(ex : checkfinance
>   value(ex : hasAttachment ex : wlextquerier))

For the deployment view, another set of individuals are defined partially shown below. *Docker1* represents the container situated on *PublicCloud* where *BlockChainIntegrator* is deployed. *Port12037* represents a communication port that *bcint* uses to communicate to *Public Blockchain*. *Link6* represents the network communication that links the communication ports that the ports of *BlockchainIntegrator* and *Public Blockchain* are bound to.

> Individual(ex : Docker1
>   value(ex : isNodeOf ex : PublicCloud)
>   value(ex : hasDeployment ex : BlockchainIntegrator))
> Individual(ex : Port12037
>   value(ex : hasBind ex : bcint))
> Individual(ex : Link6
>   value(ex : hasCommPort ex : Port12037)
>   value(ex : hasCommPort ex : Port8889))

The interaction behaviour of components can be defined in ADL as presented in our previous work [16]. This behaviour model allows the model checker to trace through different states occurring in the blockchain-based software. The tracing helps to generate a scenario that shows how an attack happens. Below is part of model in ADL describing the behaviour of the Oracle and Reverse Oracle pattern[2]. The ADL model also includes definition of the component and system configuration, which defines role and port attachment, as well as how they are executed at runtime.

```
connector IOConnector {
    role blockstorage() = token?j → process
        → stored → blockstorage();
    role extsupplier(j) = process
        → token!j → Skip; }
connector ROConnector {
    role extquerier(j) = request → uid!j
        → res?j → process → Skip;
    role blocksupplier() = uid?j → process
        → res!j → blocksupplier();  }
```

B. Security Characteristic Analysis

Beside the ontology classes supporting blockchain-based software architecture modelling, we also define ontology classes for classifying security characteristics in the model. Three characteristics, namely Attack Surface, Defence in Depth and Least Privilege, are used to calculate metric values that measure how secure the system is. Two characteristics,

namely Data Tampering and Data Disclosure, are used to trace attack scenarios. These ontology classes are kept in the ontology library. Other characteristics not addressed here can be defined by creating new class inherited from existing classes, or conditionally capturing different properties in the class definition.

1) Attack Surface: To formally define the attack surface, an ontology class called *AttackSurface* is created with a logic to describe the components that are publicly accessible through the internet or public network such as public blockchain. In other words, a component is an attack surface if it has an incoming communication port that binds to the internet link.

$$AttackSurface \equiv Component \sqcap \exists hasPort$$
$$(Port \sqcap \exists isBindTo\ (IncomingCommPort$$
$$\sqcap \exists isCommPortOf\ InternetLink))$$

2) Least Privilege: In a blockchain-based system, an on-chain component is considered as a critical component. We use an ontology rule to select the components that have access to the on-chain components. This rule is defined in Semantic Web Rule Language (SWRL) as shown below. It describes the connection between two components: *comp1* and *comp2*, which *comp2* is a *Blockchain*.

$$hasPort(comp1, p1) \wedge hasPort(comp2, p2)$$
$$\wedge hasAttachment(p1, r1) \wedge hasAttachment(p2, r2)$$
$$\wedge hasRole(con, r1) \wedge hasRole(con, r2)$$
$$\wedge Blockchain(comp2) \rightarrow LeastPriviledge(comp1)$$

3) Defence in Depth: To classify the communication ports that use security controls, the ontology class, namely *AuthenticatedCommPort*, *AuthorizedCommPort*, *FirewalledCommPort* and *InputSantizedCommPort* are defined. As we aim to capture the components that have access to the blockchain and apply security controls, *DefenseInDepth* is defined as a subset of *LeastPriviledge* that has its port bound to an incoming secured communication port.

$$AuthenticatedCommPort, AuthorizedCommPort \sqsubseteq SecureCommPort$$
$$FirewalledComPort, InputSantizedCommPort \sqsubseteq SecureCommPort$$
$$DefenceInDepth \equiv LeastPriviledge \sqcap \exists hasPort\ (Port \sqcap$$
$$\exists isBindTo(IncomingCommPort \sqcap SecureCommPort))$$

4) Data Disclosure: Data disclosure occurs on a connection that transfers data as plain text over unencrypted protocols such as *http* and *ftp*. *PlainLink* is defined to represent connectors that communicate using the insecure protocols. An ontology class called *DataDisclosureConnector* is defined as below to describe the connector that is vulnerable to data disclosure, as it transfers data in plain text.

$$HTTPLink, FTPLink \sqsubseteq PlainLink$$
$$DataDisclosureConnector \equiv Connector \sqcap (\exists hasLinkVia$$
$$(PlainLink \sqcap InternetLink))$$

5) Data Tampering: When data is transferred over a connector that is vulnerable to data disclosure, the data is also be vulnerable to tampering if the connector is on a communication link that has no input sanitisation or authorisation. Without input sanitisation the data may come from an unknown source, and the data can be changed during the transmission without any authorisation. *DataTamperingConnector* class is defined as below.

$$NoInputSanitizedCommPort \equiv hasInputSantization <= 0$$
$$UnauthorizedCommPort \equiv hasAuthorization <= 0$$
$$DataTamperingConnector \equiv DataDisclosureConnector$$
$$\sqcap \exists(hasLinkVia(CommunicationLink \sqcap \exists hasCommPort$$
$$(NoInputSanitizedCommPort \sqcap UnauthorizedCommPort)))$$

C. Security Attack Scenarios Analysis

With the ontology classes defined as previously described, the ontology reasoner can pinpoint which connector is vulnerable to data tampering and data disclosure. We use this information to generate attack scenarios by inserting attacker components into the design model. These attacker components represent software components that adversaries use. Then, Linear Temporal Logic (LTL) assertions are generated and inserted into the behavioural model in ADL. This allows the model checker to trace how the components interact with each other in response to the attacker's request. Algorithm 1 shows how the attacker component and LTL assertions are generated. This algorithm loops through *VulnConnSet* that contains inferred individuals that are of type *DataTamperingConnector* or *DataDisclosureConnector*. The attacker component is added to the model, and its *attack* port is attached to the outbound role of the vulnerable connector. The outbound role is where the request is initiated to make system responses. All inbound roles that handle the requests are iterated in the second loop. This iteration finds the port attached to an inbound role and its component to generate a LTL assertion.

Algorithm 1 Attack Scenarios Generation

1: **Input** *model* is a design model
2: **Input** *VulnConnSet* is a set of vulnerable connectors
3: **for** *vulconn* ∈ *VulnConnSet* **do**
4:     **create** *attacker* as an attacker component
5:     **create** *attack* port of *attacker*
6:     **attach** *attack* port to outbound role of *vulconn*
7:     **for** *inRole* ∈ *vulconn.getInboundRole()* **do**
8:         **for** *comp* ∈ *model.getComponent()* **do**
9:             **for** *port* ∈ *comp.getPort()* **do**
10:                 **if** *port* has *inRole* attached **then**
11:                     define a LTL assertion with
12:                     *vulconn* as vulnerable connector
13:                     *comp* as target component

The LTL assertion that proves the attack scenario is created according to the formula below. The *vevnt* represents the event triggered from the attached outbound role (*outrole*) of the vulnerable connector (*vulconn*). The *cevnt* represents the event triggered by the target component (*targetcomp*) that responds to the request issued by the attacker component (*attacker*). In other words, this LTL assertion checks whether the target component is always eventually invoked when the attacker makes a request.

$$\Box(attacker.vulconn.outrole.vevnt \rightarrow \Diamond \ targetcomp.inport.cevnt)$$

For example, if the connector between *Provenance Integration* and *Blockchain Message Bus* carries the plain text over the internet through the cloud-based container, this is vulnerable to data tampering. The attacker component could be added here. The LTL assertion is defined to generate the scenarios of this attack as $\Box(Attacker.prvmsgwire.publisher.process \rightarrow \Diamond \ BlockchainMessageBus.bclog.evntlogged)$. The model checker verifies this assertion to be valid. The negation of

this assertion gives a counterexample showing a state trace, as shown below.

    init → *Attacker‿ attack‿ attacked*
    → *Attacker‿ prvmsgwire‿ publisher‿ process*
    → *prvmsgwire‿ pevt!87* → *prvmsgwire‿ pevt?87*
    → *BlockchainMessageBus‿ bclog‿ evntlogged ...*
    → *BlockchainIntegrator‿ bcint‿ sendtobc*
    .. → *PublicBlockChain‿ logfinance‿ finlogged*
    → *PrivateBlockChain‿ logstate‿ statelogged ...*

This state trace illustrates the sequence of how components and connectors are involved in the scenario. It shows that both *Public Blockchain* and *Private Blockchain* are affected as they are consequently invoked by *Blockchain MessageBus* and *Blockchain Integrator* respectively. This information supports software engineers to analyse and fix the configuration in the design model. In this case, the communication link to the *Blockchain Message Bus* should employ an encrypted protocol like https. Furthermore, an authorisation control should be applied to *Blockchain MessageBus* and *Blockchain Integrator* to prevent data tampering. Hence, the state trace helps to pinpoint where the security configuration should be fixed.

IV. EVALUATION

This section presents how we evaluated our approach using the motivating example. The detail of how we conducted the evaluation is explained, followed by the results and discussion.

A. Experimental Setup

We have implemented our approach as a software framework to automate the security analysis process. *Arch Modeller*[3] is implemented as a graphical user interface tool to support modelling the architecture design and performing security analysis. This tool allows users to draw the graphical diagrams representing the architecture design model using the Eclipse Modelling Framework (EMF), and converts them into the structural model in OWL and the behavioural model in Wright#. The model in OWL can be processed by the ontology reasoner to identify the security characteristics. Then, the attacker components are automatically inserted and linked to the vulnerable connectors; at the same time the LTL assertions are inserted into the behavioural model. The behavioural model is processed by PAT ADL [16] and returns state traces as output, which is not possible using results from ontology reasoning alone.

The model of AgriDigital[4] has been created using Arch Modeller. In our evaluation, we assume the design model serves functional requirements correctly before conducting the security analysis. We aim to assess the completeness and soundness of the security analysis approach. After the model has been processed, the result is analysed to determine whether there are true-positives (TP) or false-positives (FP). We also analysed the design model to find false-negative (FN) results that are missing from the results given by the ontology reasoning. The precision and recall rate has been calculated

---

[3]Arch Modeller can be found at http://bit.ly/2m3LITT
[4]The model of AgriDigital can be found at http://bit.ly/2SfxHjE

according to [6]. This evaluation was carried out using an Intel Core i7 CPU with 8.00 GB Ram computer.

B. Evaluation Result

The evaluation result is summarised in Table I. The ontology reasoner took 9,123 milliseconds to detect all five security characteristics in the AgriDigital model. We have calculated the precision and recall rate to prove the soundness and completeness of the results. It can be seen that most of the detection could achieve a 100% recall rate for all characteristics. The precision rate can achieve 100% for most characteristics except data tampering, as we have found a false-positive result. It is important to note that no false-negative has been found, as the breach to the whole system may be caused by a missing security flaw. However, the accuracy of the detection relies on how accurately the security characteristics are defined. Also, the design model needs to be checked against functional requirements before the security analysis is performed.

TABLE I
EVALUATION RESULTS

Characters	TP	FP	FN	Precision	Recall
Attack Surface	7	0	0	1.0	1.0
Least Priviledge	2	0	0	1.0	1.0
Defence in Depth	2	0	0	1.0	1.0
Data Disclosure	5	0	0	1.0	1.0
Data Tampering	3	1	0	0.75	1.0

Table II shows the statistics when the scenarios have been generated based on identified connectors that are vulnerable to different scenarios. It can be seen that some assertions give state traces that show the on-chain components have indirect impacts (as indicated in the last column). These on-chain components are not the target directly connected to the vulnerable connectors, but they have a consequent impact from the attacks as some part of the request flow leads to them. In addition, the time taken by the model checker to process is reasonable for this size of model, however more comprehensive evaluation is required to better understand its performance.

TABLE II
SCENARIO GENERATION

Assertion	Scenario	Time(ms)	State#	Impact?
#1	Data Tampering	687	22772	None
#2	Data Tampering	930	38014	Indirect
#3	Data Tampering	19	781	Indirect
#4	Data Disclosure	565	22772	None
#5	Data Disclosure	937	38014	Indirect
#6	Data Disclosure	213	781	Indirect
#7	Data Disclosure	5	13	Direct
#8	Data Disclosure	3462	79	Indirect

V. CONCLUSION

This paper presents a security analysis approach for blockchain-based software architecture design. Based on the ontology description of the blockchain architecture pattern and security characteristics, our approach can identify vulnerabilities in the design model. Attack scenarios can be generated based on the identified vulnerabilities using the model checking technique. The result can be used to determine whether the blockchain has any impact from the attacks. We have evaluated our approach with an example system, and the results showed that it performs reasonably well. Our set of ontology descriptions can be extended by inheriting or defining ontology classes to describe other security metrics or vulnerabilities. Security engineers can extend our ontology library, which allows software engineers to analyse security in the blockchain-based system in a standardised way.

For future work, we plan to conduct a more comprehensive evaluation of our approach and explore how it can be applied in the software construction stage.

REFERENCES

[1] X. Xu, I. Weber, and M. Staples, *Blockchain in Software Architecture*. Cham: Springer International Publishing, 2019, pp. 83–92.

[2] M. Saad, J. Spaulding, L. Njilla, C. A. Kamhoua, S. Shetty, D. Nyang, and A. Mohaisen, "Exploring the attack surface of blockchain: A systematic overview," *CoRR*, vol. abs/1904.03487, 2019.

[3] W. D. Yu and K. Le, "Towards a secure software development lifecycle with square+r," in *2012 IEEE 36th Annual Computer Software and Applications Conference Workshops*, July 2012, pp. 565–570.

[4] J. Gennari and D. Garlan, "Measuring attack surface in software architecture (cmu-isr-11-121)," 2012.

[5] R. Vanciu and M. Abi-Antoun, "Finding architectural flaws using constraints," in *2013 28th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, Nov 2013, pp. 334–344.

[6] M. Almorsy, J. Grundy, and A. S. Ibrahim, "Automated software architecture security risk analysis using formalized signatures," in *2013 35th International Conference on Software Engineering (ICSE)*, 2013, pp. 662–671.

[7] L. Luu, D.-H. Chu, H. Olickel, P. Saxena, and A. Hobor, "Making smart contracts smarter," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS 16. New York, NY, USA: Association for Computing Machinery, 2016, p. 254269.

[8] M. Chaieb, S. Yousfi, P. Lafourcade, and R. Robbana, "Verify-your-vote: A verifiable blockchain-based online voting protocol," in *Information Systems*, M. Themistocleous and P. Rupino da Cunha, Eds. Cham: Springer International Publishing, 2019, pp. 16–30.

[9] S. Bragagnolo, H. Rocha, M. Denker, and S. Ducasse, "Smartinspect: solidity smart contract inspector," in *2018 International Workshop on Blockchain Oriented Software Engineering*, March 2018, pp. 9–18.

[10] E. Zhou, S. Hua, B. Pi, J. Sun, Y. Nomura, K. Yamashita, and H. Kurihara, "Security assurance for smart contract," in *2018 9th IFIP International Conference on New Technologies, Mobility and Security (NTMS)*, Feb 2018, pp. 1–5.

[11] Á. Hajdu and D. Jovanovic, "solc-verify: A modular verifier for solidity smart contracts," *CoRR*, vol. abs/1907.04262, 2019. [Online]. Available: http://arxiv.org/abs/1907.04262

[12] X. Xu, I. Weber, and M. Staples, *Case Study: AgriDigital*. Cham: Springer International Publishing, 2019, pp. 239–255.

[13] X. Xu, C. Pautasso, L. Zhu, Q. Lu, and I. Weber, "A pattern collection for blockchain-based applications," in *Proceedings of the 23rd European Conference on Pattern Languages of Programs*, ser. EuroPLoP 18. New York, NY, USA: Association for Computing Machinery, 2018.

[14] F. Chen, Z. Liu, Y. Long, Z. Liu, and N. Ding, "Secure scheme against compromised hash in proof-of-work blockchain," in *Network and System Security*, M. H. Au, S. M. Yiu, J. Li, X. Luo, C. Wang, A. Castiglione, and K. Kluczniak, Eds. Cham: Springer International Publishing, 2018, pp. 1–15.

[15] N. Chondamrongkul, J. Sun, and I. Warren, "Ontology-based software architectural pattern recognition and reasoning," in *30th International Conference on Software Engineering and Knowledge Engineering*, June 2018, pp. 25–34.

[16] N. Chondamrongkul, J. Sun, and I. Warren, "Pat approach to architecture behavioural verification," in *31th International Conference on Software Engineering and Knowledge Engineering*, July 2019, pp. 187–192.

Characterizing Vulnerabilities in a Major Linux Distribution

Stephen R. Tate   Moulika Bollinadi   Joshua Moore

Department of Computer Science, UNC Greensboro, Greensboro, NC 27402

Contact email: `srtate@uncg.edu`

Abstract

This paper reports on a careful study of vulnerabilities in open-source software, performing both a longitudinal study over 7 years of data and an in-depth exploration of a particular type of vulnerability. First, data was mined from Ubuntu security notices from 2012 to 2019, specifically pulling security notices published within the first year of each of the four stable releases during that time. This provided a dataset covering 3,232 security vulnerabilities, which were cross-referenced with other information, allowing us to identify trends in types of vulnerabilities over the past 7 years. Within these results, we see that out-of-bounds memory access (which includes the classic "buffer overflow" vulnerability) has consistently been the most pernicious security weakness, so in the second part of this research we performed an in-depth study of a random sample of 30 recent out-of-bounds access vulnerabilities. Beginning by evaluating each vulnerability in terms of seven features, we identified trends and patterns and expanded the analysis to a total of eleven features. These results help further understanding of how out-of-bounds access vulnerabilities occur in real software, which can help both researchers looking to improve tools for vulnerability analysis and developers learning how to avoid common pitfalls.

1 Introduction

The enthusiasm of programmers to create new software has led to an explosion in the size and complexity of programs that has greatly exceeded our ability to ensure that the programs are free of errors. Of particular interest in our highly-connected society are errors that can be exploited to subvert security goals, a kind of error that is called a "vulnerability." In this project, we gain insight into what kinds of vulnerabilities occur across a wide range of open-source software that is used in Linux and Unix-like operating systems, how the frequency of various vulnerability types has changed over time, and what specific characteristics are present in the most pernicious type of vulnerability, the out-of-bounds access vulnerability. Our goal is to improve knowledge about vulnerabilities in real-world software, focusing on situations in which sourcecode is available so that patterns and trends can provide useful information for software developers.

Vulnerabilities are cataloged and classified in the "National Vulnerability Database," or NVD, maintained by the National Institute of Standards and Technology (NIST) [6]. While statistics on the entire NVD database are available, including a break-down by type of vulnerability [7], these statistics aggregate vulnerabilities across all environments, including open-source software, closed-source and proprietary software, and dedicated devices and device firmware. Since our primary motivation for this work is to find information that will be useful in settings with access to sourcecode (specifically, source-based analysis tools and developer practices), we look specifically at vulnerabilities where sourcecode is published so it can be analyzed for patterns and characteristics.

To study open-source software, we need to identify a set of projects that we can study, and from which we can extract useful information about vulnerabilities. We initially considered mining information from GitHub [4] or from the Software Heritage Graph dataset [8], but the range of software maturity and the *ad hoc* nature of vulnerability identification in these sources quickly showed this to be an unproductive approach. We then looked to major Linux distributions, since distribution maintainers have already identified interesting and mature projects for inclusion in the distribution, and they track security vulnerabilities so that their users can stay up-to-date with security patches. Exploring three major distributions, Debian [13], RedHat [10], and Ubuntu [3], we found Ubuntu to be the easiest and most effective to work with for three main reasons: the Ubuntu maintainers track a huge collection of open-source software, with over 29,000 source packages in the 18.04 release; there is a well-maintained CVE tracker and source of security notifications (the Ubuntu Security notices, or USNs); and the practice of producing stable long-term-

support (LTS) releases on precise two-year intervals provides a meaningful way to perform a longitudinal study. In the end we selected four such LTS releases, 12.04, 14.04, 16.04, and 18.04, and studied both general changes between releases and specific vulnerability characteristics from the most recent release. While we use data from Ubuntu, all major Linux distributions use the same base of open-source software, so we believe these results are representative of the broader open-source software community.

Contributions: We make the following contributions in this paper.

- We provide a clear picture of how the prevalence of various types of vulnerabilities has changed over time. The data show quantitatively that out-of-bounds memory access has remained the most pernicious type of security vulnerability over time, consistently accounting for around a third of all security vulnerabilities.

- We report on a careful study of a random sample of 30 out-of-bounds memory access vulnerabilities, defining the notion of an "exploit flow" to capture and analyze features in the sourcecode that characterize each exploit. Our statistical breakdown reveals several interesting and useful observations, including the pervasiveness of exploit flows that cross compilation-unit boundaries (implying that analysis tools must similarly consider multi-module analysis) and the importance of pre and post condition statements at function boundaries to simplify understanding of vulnerabilities.

We believe that the results reported here will be beneficial both for researchers working to improve source-based vulnerability identification tools, and for developers learning about the types of security-related errors made in real-world software systems.

2 Types Vulnerabilities Over Time

In order to study how vulnerabilities have changed over time, we considered the four most recent Ubuntu Long-Term Support (LTS) releases, which were released two years apart in April, starting in 2012 with release 12.04. We processed archived Ubuntu Security Notices (USNs), extending exactly one year after each initial release, extracting CVE references and information on "affected distributions" from each security notice to get CVE/distribution pairs.

After manually examining some of these results, we discovered that there were a number of CVEs listed with distributions that were not actually affected by that CVE. The problem is that a USN may reference a large number of CVEs, and the USN flags a distribution as "affected" if any one of those is a vulnerability in the distribution. In one extreme case, a single USN (USN 3681-1) referenced 124

different CVEs! This leaves the possibility of a large number of CVEs in a USN not being relevant to that distribution. To correct this problem, we pulled information on all CVEs from the Ubuntu "CVE Tracker," which indicates which distribution(s) are affected by an individual CVE, and dropped a CVE/distribution pair if the CVE tracker has a distribution classification for that CVE as "DNE" (does not exist), "not affected," or "ignore." After processing, there was a modest drop in CVE/distribution pairs, with the number of CVEs affecting release 18.04 dropping from 911 (based on USN alone) to 843 (based on both USN and the more accurate CVE tracker classification).

For each CVE that affects an Ubuntu distribution (and was fixed within the first year of that distribution's release), we cross-referenced the CVE with the corresponding entry in NIST's National Vulnerability Database (NVD) to extract the severity of the vulnerability, as given by the CVSS v3.0 base score, and the type of vulnerability, as given by the Common Weakness Enumeration (CWE) code associated with the CVE. While there are 839 defined CWE codes, only 62 appear in our dataset, and these can further be categorized into a few broad classes of vulnerability types. Our starting-point was the CWE cluster definitions from MITRE, such as CWE-970 ("SFP Secondary Cluster: Faulty Buffer Access") that lists 11 fine-grained CWEs related to buffer access. CWEs that were not listed in such pre-defined clusters were examined for obvious matches, such as CWE-787 (Out-of-bounds Write). In the end, the 7 CWEs that occurred in our data set from the CWE-970 cluster along with CWE-787 formed our "Out-of-Bounds vulnerability" class, as summarized in Table 2. Occurrences of each vulnerability type in the four Ubuntu releases was collated, and information on the "top 5" most prevalent vulnerability types is given in Table 1. Full data, including full CWE-to-class mapping, occurrence rates for each class, as well as the scripts used to analyze the Ubuntu data, is available at the project web site [12].

Discussion: As can be seen in Table 1, the number of vulnerabilities has been generally increasing, with an anomalous spike at version 16.04. However the number of high or critical severity vulnerabilities has decreased, resulting in the percentage of vulnerabilities classified as high or critical severity decreasing by over half (from 44% to 19%). CVE evaluation has dramatically improved over the years, with the percentage of CVEs giving a CWE classification increasing 64% to 95%. The Out-of-Bounds access vulnerability has been the top vulnerability type in every version. This is particularly frustrating as it is one of the oldest forms of security vulnerability and a significant amount of research has gone into locating and fixing such vulnerabilities. Furthermore, pointer issues, also a memory safety violation, have also gotten much more common in the past few years. Of issues that are not related to memory safety, per-

Distribution	Ubuntu 12.04	Ubuntu 14.04	Ubuntu 16.04	Ubuntu 18.04
Release date	*April 26, 2012*	*April 17, 2014*	*April 21, 2016*	*April 26, 2018*
Total CVEs fixed in Year 1	646	701	1042	843
High/Critical severity	287 (44%)	268 (38%)	258 (25%)	156 (19%)
CWE classified	415 (64%)	455 (65%)	952 (91%)	802 (95%)
Out-of-bounds access	24.3% (1)	30.8% (1)	38.9% (1)	30.2% (1)
Permissions	18.1% (3)	10.3% (3)	9.5% (3)	15.5% (2)
Pointer issues	—	2.6% (10)	8.8% (4)	11.0% (3)
Input validation	16.6% (4)	14.3% (2)	13.1% (2)	10.8% (4)
Resource management	21.9% (2)	9.0% (4)	4.8% (7)	8.5% (5)
Numeric errors	7.0% (5)	8.1% (5)	1.7% (9)	0.2% (19)

Table 1. Vulnerabilities by distribution (rank of each type in parenthesis)

CWEs in the general "out-of-bounds access" class	
CWE-118	Incorrect Access of Indexable Resource ('Range Error')
CWE-119	Improper Restriction of Operations within the Bounds of a Memory Buffer
CWE-122	Heap-based Buffer Overflow
CWE-123	Write-what-where Condition
CWE-125	Out-of-bounds Read
CWE-126	Buffer Over-read
CWE-129	Improper Validation of Array Index
CWE-787	Out-of-bounds Write

Table 2. Out-of-bounds access CWEs

mission and input validation issues remained in the top four vulnerability types throughout the seven years, although the percentage of input validation issues has been slowly declining. Numeric errors have also fallen dramatically over the years. Some of the changes in pointer issues and numeric errors could be due to a change in classification criteria (e.g., an integer overflow leading to out-of-bounds access might have been classified as an integer overflow in the past, but is now classified as out-of-bounds access), but digging deeper into the CVE classification criteria is beyond the scope of this project and is left as an open question.

3 Out-of-bounds Access Vulnerabilities

Given the consistent top ranking of Out-of-Bounds access vulnerabilities (abbreviated as "OoB access"), we next undertook a study to gain deeper understanding into how these vulnerabilities manifest in real-world systems. As a first step, we pulled the full list of OoB access CVEs that affected Ubuntu 18.04 in its first year of release, and randomly sorted them so that we could select the first CVEs in our random ordering as a random sample of OoB access vulnerabilities. Early in our process we discovered that

the rapid release model of Firefox and family (Thunderbird, mozjs, etc.) differed greatly from other packages, and the huge patch updates made it practically impossible to locate specific vulnerabilities based on public information. For example, the fix for CVE-2018-18493 (a critical-severity OoB vulnerability) was only included as part of the Firefox update from version 63.0.3 to 64, where the diff between these versions contains close to 1.9 million lines. As a result, we excluded vulnerabilities in these packages from our list before taking our random sample.

We started our evaluation based on 7 characteristics suggested by our past experience. The authors studied 14 CVEs over a period of 6 weeks, with group discussions that identified 4 additional recurring patterns, giving a final set of 11 relevant characteristics. The 14 initial vulnerabilities were re-examined along with 16 others. Of these 30 vulnerabilities, six were excluded for various reasons: three did not apply to Ubuntu 18.04, two were mis-classified in the NVD as out-of-bounds access vulnerabilities, and one did not have enough public information available to analyze. While this process gave deep insight into vulnerability characteristics, it was highly labor-intensive, and an interesting future research direction could explore ways to automate or at least provide tools to assist in this analysis.

3.1 Exploit Flow Definition

To characterize out-of-bounds access vulnerabilities, we introduce the idea of an *exploit flow*: the shortest execution path through the program that fully explains to an informed reader how the vulnerability arises and what causes the out-of bounds access, where we allow irrelevant portions of the flow to be redacted. This definition is inherently subjective, with reference to "an informed reader," but in general it will include code that calculates an array index or a pointer that is subsequently used in the out-of-bounds access. When the out-of-bounds access involves a dynamically-sized block of memory, the exploit flow will also typically include the size

```
5658    first_object=(p[0] << 8) | p[1];
5659    last_object=(p[2] << 8) | p[3];
5660    p+=4;
5661
5662    for (i=(int) first_object;
            i <= (int) last_object; i++)
5663    {
5664      if (mng_info->exists[i] &&
            !mng_info->frozen[i])
5665        {
5666        ...
```

Figure 1. Vulnerability CVE-2017-13139

calculation and memory allocation.

An example of an out-of-bounds access vulnerability is shown in Figure 1, where the code comes from PNG image processing module for the ImageMagick library (code has been somewhat reformatted to fit in one column, but is otherwise directly taken from the `coders/png.c` file of ImageMagick version 6.9.7-4). Note, this was actually one of the vulnerabilities that ended up being excluded from our study, since it was patched prior to the official Ubuntu 18.04 release; however, due to its simplicity it serves as the best example to explain exploit flows in this paper.

The exploit flow is the following description, which traces an execution through the code to demonstrate clearly how the vulnerability can be exploited:

1. *Comment:* Variable `p` points to an input buffer containing unsigned characters, read directly from (possibly malicious) input.

2. *Lines 5658–5659:* 16-bit binary values are loaded in to unsigned int variables `first_object` and `second_object`, and are unchecked so they can be any possible 16-bit values (e.g., these variables can have values 1000 and 2000, respectively).

3. *Line 5662:* A for loop starts on line 5662, with index `i` ranging from `first_object` to `last_object`.

4. *Line 5664:* Array `mng_info->exists[i]` is accessed, which is a statically-sized array of size 256.

5. *Out-of-bounds access, line 5664:* With the sample values above, on the first iteration of the for loop, `i` is 1000 and when used as an index into an array of size 256 this results in an out-of-bounds read.

There are a few important observations to make from this example. First, it is not entirely self-contained, since the first comment simply mentions that buffer `p` is unconstrained data read from the user. This should be perfectly clear to the "informed reader" that is part of the exploit flow definition, and allows us to leave out sometimes-confusing I/O buffering code from the flow. Second, we include

specific example values that could be used in a proof-of-concept exploit for the vulnerability — in all but a few cases, we have constructed actual proof-of-concept inputs to test our exploit flows. Finally, we do not include code that defines sizes of memory blocks in the exploit flow when they follow from static type declarations, and instead simply state the size of the data block (as we did with the 256-entry array in the example exploit flow).

In terms of vulnerability characteristics, this exploit flow is a very simple example of a pattern we saw regularly in out-of-bounds access vulnerabilities: An offset into a binary structure is read from user data, unpacked from a straight binary value (typically 16 or 32 bits), and then used without being checked to ensure that it is a sensible value.

3.2 Exploit Flow Characteristics

As described above, we evaluated all vulnerabilities in terms of 11 characteristics or features. In this section we define and describe these characteristics.

Spans multiple compilation units or files: Does the exploit flow include code from multiple compilation units, where "compilation unit" is the unit of source code processed by one pass of a compiler? For C and C++, a compilation unit consists of both a main source file (`.c` or `.cpp` file) and any included header files (`.h` files). This characteristic is important when considering code analysis tools, as some tools (e.g., the Clang static analyzer [5]) restrict analysis to a single compilation unit, whereas others (e.g., Infer [2] and Klee [1]) perform analysis using information from multiple compilation units.

Spans multiple functions: Does the exploit flow include code from multiple functions, where these functions may be library functions, user-defined functions, or both? Unlike the previous characteristic, functions may be in the same file or compilation unit. This characteristic is important while considering code analysis tools as some tools do not trace across function call boundaries.

Involves typecasting or type confusion: Does critical data change types during the exploit flow? Code that initializes a variable using one type and converts it into another type is type casting. If that resource is accessed using an incompatible type with the original type then it is type confusion. This can result in triggering logical errors in the source code.

Simplified with a function pre/post condition: Would a stated function pre/post condition shorten or simplify the exploit flow? Given the frequency of exploit flows that span multiple functions, it is not surprising that such conditions could help analysis. In particular, a function that fills a block of memory could have a pre-condition that the buffer must have sufficient size, and then the exploit flow

does not need to extend into the function. A static analysis tool could use such pre/post conditions to check both that pre-conditions are met when a function is called, and to start path analysis in the function using the pre-condition as a starting condition.

Simplified with a data structure invariant: Would a stated data structure invariant shorten or simplify the exploit flow? The most common way that this is relevant in our study would be an invariant that relates to the size of a dynamically allocated buffer, so that the buffer size is known to a static analyzer without requiring the exploit flow to trace all the way back to the actual size calculation and allocation.

Dynamically-sized memory block: Are memory needs determined at runtime, such as when they are dependent on user input? Dynamic allocation poses a particular problem for static analysis tools, as the tool often does not have any information about the size of the memory block (this is related to, and potentially addressed by, a data structure invariant as mentioned above). In addition, with dynamic memory allocation there is a possibility that the user/attacker introduces a large value and the system cannot allocate enough memory for it, so no buffer at all is allocated. While programs can detect this situation by checking the allocation return value, it is unfortunately common that programmers omit this error-check.

Binary data format processing: Does the data being handled in the exploit flow come from a raw binary data format, such as image or audio files? Vulnerabilities can arise from code that unpacks and uses values such as sizes or offsets without first checking them for validity. While legitimate data in such formats is produced by software that will only output data conforming to certain rules, attacker-supplied data is not restricted to sensible values.

Other characteristics: We evaluated all vulnerabilities with respect to several other characteristics which ended up being less significant. In particular, we considered whether the exploit flow was asynchronous, as might be common in event-driven programming; whether bugs were involved in parsing textual input; whether there are a significant number of branching decisions (including loop iterations), which would lead to path explosion in analysis; and whether the exploit flow is determined by dynamic type resolution, such as through a method dispatch table in C++. Any of these characteristics would complicate the task of code analysis, but all turned out to be rare in our random sample of OoB vulnerabilities. This is encouraging because it implies that significant improvements in static analysis can be made without having to address these particularly challenging situations. For example, only one CVE had an asynchronous exploit flow, and most had very limited branching (some included loops, but the vulnerability could typically be triggered on the first iteration of the loop).

OoB Access Characteristics		
Multiple Functions	17/24	71%
Multiple Compilation Units	12/24	50%
Type confusion/casting	9/24	38%
Simplifying Pre/Post-condition	15/24	63%
Simplifying Invariant	10/24	42%
Dynamically-sized memory	19/24	79%
Binary data formats	19/24	79%

Table 3. Out-of-Bounds Characteristics

As we investigated characteristics of exploit flows, another interesting fact emerged: the prominence of fuzz testing in detecting vulnerabilities. The method used to detect the vulnerability was clearly stated in slightly over half of the studied CVEs, and in every single one of those cases the vulnerability was found by fuzzing (typically using AFL [14] or a variant). In several other cases, while not stated explicitly there was evidence that fuzzing was used to locate the vulnerability, and in only a single case did the evidence suggest that the vulnerability was found in some other way (probably manual code review). This does raise an interesting question, which would require further study to resolve: Is improved success in finding OoB vulnerabilities masking a decline in frequency of occurrence? In other words, how much of the OoB frequency in Table 1 is due to actual prevalence of the type of vulnerability and how much is due to our success in locating such vulnerabilities? The widespread use of fuzzing could also explain the predominance of vulnerabilities in binary file format processing, as fuzzing works particularly well on such data.

3.3 Results

The prevalence of the various vulnerability characteristics described above, across the 24 usable samples, is given in Table 3. The frequency of exploit flows that cross function and even compilation-unit boundaries clearly demonstrates that effective analysis tools must be able to reason about multi-unit execution paths. However, a promising aspect of this analysis is that a large number of such flows (roughly two-thirds of the multi-function exploit flows) could be greatly simplified through proper use of function pre/post conditions. Programmers naturally divide programs into pieces with clear logical requirements so that they can cope with complexity as a developer. Making these logical requirements explicit could significantly help analysis in most of the OoB vulnerabilities that we studied.

Furthermore, dynamic memory allocation is very common in modern software, and our results show that it also plays an important role in many vulnerabilities. Again,

buffer sizes are generally logically designed by programmers, even if such sizes are not available (in C and C++) to static analysis tools. Taking this logical design out of the mind of the programmer and turning it into an explicitly-stated data structure invariant could help identify vulnerabilities in almost half of the vulnerabilities studied.

Finally, type confusion and casting is a less common but still significant problem. Casting issues, such as from an unsigned size to a signed size, could be easily identified by even simple static analysis tools. As a next step in our work, we will study how often such casting issues occur and how often they lead to vulnerabilities. We suspect that the number of safe uses of typecasting far exceeds the unsafe uses, which would lead such a static analysis tool to produce too many false positives to be useful. Analysis of how to identify just the unsafe uses is an interesting open question.

4   Related Work

Rigorous work classifying vulnerabilities across a wide span of open-source software is uncommon, but a few recent projects are related to our work. Ponta *et al.* manually curated a large collection of 624 vulnerabilities and associated patches in open-source Java packages related to their area of interest [9]. As their work focuses specifically on Java code in a niche area, it does not provide the broad picture of open-source vulnerabilities that our study seeks.

In a similar effort to ours, reviewing reported buffer-overflow vulnerabilities, Shuckert *et al.* performed a review of such vulnerabilities that had been reported in the Firefox web browser [11]. This study provides some interesting observations, but results are reported in a more qualitative than quantitative fashion, with no statistics on the prevalence of various vulnerability characteristics reported. Furthermore, focusing on a single software package provides excellent insight into that package, but it is unclear how much the results reflect overall open-source development characteristics as opposed to particular coding practices and standards for that particular development team.

Other work on buffer-overflow vulnerabilities considers, as part of the background to their work, investigation of certain characteristics related to their work, but these are not broadly-focused studies that provide insight into bigger picture of open-source vulnerabilities.

5   Conclusion

In this paper, we examined characteristics of real-world vulnerabilities in open-source software. Mining a large set of 3,232 vulnerabilities over 7 years revealed several interesting trends, and clearly showed that out-of-bounds access vulnerabilities have remained the most commonly-occurring danger. With an eye toward identifying promising directions for program analysis tools and techniques, we

carefully studied a random sample of out-of-bounds access vulnerabilities, identifying several particularly promising directions for future work. First, there is a strong need for analysis that spans function or even compilation unit boundaries. Second, the use of pre/post conditions and data structure invariants, perhaps provided to analysis tools through sourcecode annotations, could greatly simply the reasoning required to identify vulnerabilities. And finally, exploring how tools can distinguish between safe and unsafe uses of typecasting could produce interesting and practical results.

References

[1] CADAR, C., DUNBAR, D., AND ENGLER, D. KLEE: Unassisted and Automatic Generation of High-coverage Tests for Complex Systems Programs. In *Proc. of the 8th USENIX Conference on Operating Systems Design and Implementation* (2008), OSDI'08, pp. 209–224.

[2] CALCAGNO, C., AND DISTEFANO, D. Infer: An Automatic Program Verifier for Memory Safety of C Programs. In *NASA Formal Methods*. 2011, pp. 459–465.

[3] CANONICAL, LTD. Ubuntu website. `https://ubuntu.com/`.

[4] GITHUB, INC. Github website. `https://github.com/`. Accessed: 2020-01-30.

[5] LATTNER, C., AND ADVE, V. LLVM: A Compilation Framework for Lifelong Program Analysis & Transformation. In *Proc. of the 2004 International Symposium on Code Generation and Optimization* (2004).

[6] NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY. National vulnerability database. `https://nvd.nist.gov/`.

[7] ÖZKAN, S. CVE Details website. `https://www.cvedetails.com/`. Accessed: 2020-01-30.

[8] PIETRI, A., SPINELLIS, D., AND ZACCHIROLI, S. The Software Heritage Graph Dataset: Public Software Development under One Roof. In *Proc. of the 16th International Conference on Mining Software Repositories* (2019), MSR 19, pp. 138–142.

[9] PONTA, S. E., PLATE, H., SABETTA, A., BEZZI, M., AND DANGREMONT, C. A Manually-curated Dataset of Fixes to Vulnerabilities of Open-source Software. In *Proc. of the 16th International Conference on Mining Software Repositories* (2019), MSR '19, pp. 383–387.

[10] REDHAT. Website. `http://www.redhat.com/`.

[11] SCHUCKERT, F., HILDNER, M., KATT, B., AND LANGWEG, H. *Source Code Patterns of Buffer Overflow Vulnerabilities in Firefox*. Gesellschaft fr Informatik e.V., 2018.

[12] TATE, S., BOLLINADI, M., AND MOORE, J. Ubuntu vulnerability study project. `https://span.uncg.edu/vulnerabilities`.

[13] THE DEBIAN PROJECT. Website. `https://www.debian.org/`.

[14] ZALEWSKI, M. American Fuzzy Lop. `http://lcamtuf.coredump.cx/afl/`.

Mining DApp Repositories: Towards In-Depth Comprehension and Accurate Classification

Yeming Lin*†, Jianbo Gao‡, Tong Li*†, Jingguo Ge *†, Bingzhen Wu*†

*Institute of Information Engineering, Chinese Academy of Science, Beijing, 100093, China
†School of Cyber Security, University of Chinese Academy of Sciences, Beijing, 100049, China
‡School of Electronics Engineering and Computer Science, Peking University, Beijing, 100871, China
*†{linyeming,litong,gejingguo,wubingzhen}@iie.ac.cn, ‡gaojianbo@pku.edu.cn

Abstract—Blockchain has recently attracted great interest from both academia and industry. Ethereum introduces programmability into blockchain through smart contracts and provides an open-source computing platform for blockchain-based decentralized applications (DApps). There are currently thousands of DApps pertaining to different application domains, including games, gambling and finance. In order to better comprehend blockchain application scenarios and help developers understand DApps better, clear DApps classification criteria are needed. However, many DApps that can be found through collection websites (commonly known as DApp Stores) are not classified properly, making these datasets imprecise. This issue has motivated the present empirical study of DApp categories, as a part of which over 2,500 DApps in three DApp Stores are investigated, allowing us to produce and publicly release a high-quality dataset in which misclassified DApps are relabeled manually, facilitating their more precise classification. We also propose DAppClassifier, a novel technique for classifying DApps based on their actual functionalities. When developing the new classifier, we extracted features from source code, bytecode and historical transactions, and trained neural networks to classify DApps. Our approach was evaluated on the released dataset and achieved good precision.

Index Terms—Blockchain, Decentralized Application, Comprehension, Ethereum, Smart Contract

I. INTRODUCTION

Owing to the growing interest in Bitcoin, considerable efforts have been invested into the development of blockchain techniques. Ethereum is a public open-source blockchain-based distributed computing platform, providing a Turing-complete virtual machine for executing smart contracts. Smart contract refers to open-source programs that can be automatically executed without any centralized control. Consequently, it is the most important innovation of Ethereum.

Blockchain-base decentralized applications (abbreviated as DApps), are the emerging trend in the blockchain development, as they rely on smart contracts instead of traditional centralized server as the back-end. Owing to its transparency, decentralization, and security, this infrastructure offers immense potential for use in a variety of fields, including finance, governance, supply chain management, etc. Thus, it is not surprising that the number of DApps has already surpassed 2,500 within 4 years, with the value of the DApp market estimated at billions of dollars [1].

DApp Store is a repository that facilitates DApp management. Akin to AppStore for the iOS system, it provides a convenient platform for users to browse DApps and select those that meet their needs. To expedite searching, DApp Stores classify DApps into a set of categories, such as Games, Gambling, Exchange, and High-risk.

Such categorization can be helpful for both users and developers, as users can browse the appropriate category to find relevant DApps, whereas developers can determine the most optimal category for their DApps prior to submission.

At present, DApp categories are manually selected by the developers at the time of submission. The maintainers of DApp Stores will subsequently manually check if the classification is correct, aiming to detect high-risk DApps (such as those hiding a Ponzi scheme). However, as such classification relies on human judgment, it can be error-prone. On one hand, the predefined categories may be ambiguous, and the developers may find it difficult to locate a proper category for their DApp. On the other hand, as manually verifying classification for each DApp is time-consuming, there is a risk of exposing DApp users to security issues.

To provide a comprehensive understanding of the DApp classification status, as a part of this investigation, an empirical study involving over 2500 DApps in three DApp Stores is conducted. Based on the findings obtained, a dataset is constructed and its accuracy is optimized by manually relabeling misclassified DApps.

However, to the best of our knowledge, no effort has been devoted to the DApp classification problem. Although several approaches have been proposed for labelling and identifying smart contracts (a component of DApp), they can only be utilized to cluster similar smart contracts [2] or identify special smart contract types, such as Ponzi scheme or honeypot [3] [4]. Thus, none of them can be directly applied to classify multifarious DApps. Even though, some approaches to mobile App classification can be potentially modified for use in DApp classification, as they do not take advantage of the unique characteristics of DApps, the classification performance would be compromised.

To overcome these shortcomings, in this paper, we propose DAppClassifier, a novel technique for classifying DApps based

on easily obtainable rich and comprehensive features that represent the actual DApp functionalities. DAppClassifier first extracts features from source code, bytecode and historical transactions, which is the communication between the user and the DApp. Next, hybrid neural networks are devised to classify DApps.

To evaluate DAppClassifier performance, it is applied to the optimized dataset (obtained in the first phase of this study as a part of which misclassified DApps were relabeled manually), achieving >84% average precision.

The main contributions of this work are as follows:

- **We conduct a systematic empirical study on DApp status,** extensively investigate DApp classification problem and obtain the different categories characteristics.
- **We construct a high-quality dataset** by relabeling misclassified DApps. The revised dataset is open sourced and can be accessed from https://bit.ly/2JFmtiS.
- To eliminate the need for manual verification and reclassification, **we propose DAppClassifier that can automatically classify DApps based on their rich and comprehensive features.** The effectiveness of DAppClassifier is subsequently confirmed by testing its performance against other available techniques.

II. BACKGROUND

In this section, we provide the background information required for understanding our work.

A. Ethereum and Smart Contract

Ethereum is a public open-source blockchain-based distributed computing platform that provides a running environment for decentralized applications.

A smart contract is a computer program outlining the rules under which the participants agree to interact with each other. If the pre-defined rules are met, the agreement is automatically enforced.

B. Ethereum Virtual Machine (EVM)

Ethereum provides EVM to support the compilation and execution of smart contracts. Technically, it is the runtime environment for smart contracts in Ethereum. It is a stack-based, register-less virtual machine, where operators and operands are all pushed onto the stack indistinguishably, with the exception of the data that requires persistent storage space on Ethereum. Solc compiler will translate readable solidity code into bytecode, only EVM can understand.

C. Decentralized Applications

Centralized systems directly control the operation of the individual units and flow of information from a single center. Decentralized applications (DApps) are applications that run on a decentralized network rather than a single computer. Technically, a DApp is composed of front-end and back-end code, whereby the front-end is an Internet-based interface, typically a web page, and the back-end contains the key data

and operations, typically based on one or more smart contracts in a blockchain. When users interact with blockchain-based DApps, a transaction is initiated and recorded on the blockchain permanently. So we can get all the interactions between DApps and users from the blockchain.

III. EMPIRICAL STUDY

In this section, current DApp classification status is briefly outlined.

For this analysis, three DApp Stores: State of the DApps [1], DappRadar [5], and Dapp.com [6] are chosen as the representatives of the current DApp market (based on the Google Search results).

A. Overview of DApp Categories

This section commences with an overview of DApp categories, to help readers better understand the current classification criteria. DApp Stores utilize predefined categories, allowing users to find the DApps they need. However, these are not standardized, making comparisons across different platforms difficult, while introducing the risk of misclassification.

TABLE I
DAPP CATEGORIES ON THE WEBSITES. FOR EACH CATEGORY, THE COLUMNS OF THE TABLE SHOW, FROM LEFT TO RIGHT: THE NUMBER OF DAPPS FROM *State of the DApps* (**#S**), *DappRadar* (**#R**), AND *Dapp.com* (**#D**).

Category	#S	#R	#D
Games	303	402	455
Gambling	211	375	266
High-risk	148	333	256
Exchanges	106	57	65
Finance	101	-	53
Social	86	-	45
Media	55	-	-
Development	49	-	-
Marketplaces	41	15	-
Property	29	-	-
Governance	27	-	-
Wallet	20	-	-
Security	20	-	-
Storage	17	-	-
Identity	15	-	-
Health	4	-	-
Insurance	4	-	-
Energy	3	-	-
Collectibles	-	52	-
Tools	-	-	57
Art	-	-	33
Others	-	260	84
Total	1239	1494	1314

B. Misclassification of DApps

In this section, we provide a descriptive analysis on the DApps in the three most widely used platforms: State of the DApps [1], DappRadar [5], and Dapp.com [6] to determine if these are properly classified by the developers. We crawled all DApps information from three platforms and matched them by front-end URL and contract address. If they have the same front-end URL or similar front-end URL with the same contract address, we consider them to be the same DApp, for some DApps have different URL parameters in order to identify the source of the request. Our investigation has uncovered

Overlap Categories	#DApps
Games, High-risk, Gambling	8
Gambling, High-risk	77
Games, High-risk	71
Games, Gambling	37
Games, Collectibles	25
Games, Marketplaces	13
High-risk, Finance	12
Finance, Others	20
Social, Others	24
Tools, Others	12

two important problems with the current classification system, which are discussed below.

1) The classification criteria are ambiguous and differ across DApp Stores: We compare DApps and the categories utilized by the aforementioned three DApp Stores, and find that the classification criteria are non-uniform and ambiguous, for the following two reasons.

First, each DApp Store uses a different number of categories, at 18, 7, and 9, respectively (Table I). While Games, Gambling, High-risk and Exchanges are utilized in all three cases, *Health* and *Insurance*, for example, only exist in *State of the DApps*.

Second, even if DApp Stores utilize the same category, such as Games or Gambling, the classification criteria can be ambiguous. In an ideal scenario, the same DApp should be classified into the same category on all DApp Stores. However, according to our investigation, this is not always the case, as shown in Table II. For example, 37 DApps classified as Games on one DApp Store are classified as Gambling on the other.

Moreover, as no clear description of the categories is provided on the DApp Stores, developers would find it challenging to determine the most appropriate category for their DApps.

2) DApps are often misclassified by DApp Stores: As a part of our investigation, we examine the characteristics of DApps and check if these corresponded to their classification by DApp Stores.

For this purpose, we conducte a preliminary experiment on smart contracts of DApps, which is guided by three hypotheses: (1) Intuitively, DApps with the same smart contracts should be in the same category; (2) DApps with the same run-time code should be in the same category; and (3) DApps with the same opcode list should be in the same category, because opcode (rather than the operand) determines the program execution logic.

To test these hypotheses, we extract smart contracts from DApps. When comparing run-time code, we match the code textually, whereas for opcode list comparison, we split the run-time code into opcode list and operand list, and remove the latter.

Our findings revealed that, in the three DApp Stores included in our analysis, 41, 29, and 36 DApps, respectively, violate the three hypotheses given above, suggesting high degree of misclassification. We manually analyze these DApps and found that more than 40% of DApps changed from High-risk to other categories, in order to trick users into using. We manually corrected these classification errors in our dataset.

C. The Difficulties of DApp Classification

As mentioned earlier, due to the lack of clarity in the classification criteria used by DApp Stores, developers often struggle with interpreting the requirements for each category, which may result in misclassification of their DApps.

To mitigate these issues, an automatic classification approach is needed, as it would allow developers to identify the most suitable category for their DApps, while aiding the DApp Store maintainers in the verification process, which is currently conducted manually.

However, objective DApp classification is challenging for several reasons, as explained below.

1) Limited EVM features: EVM is a runtime environment for smart contracts based on a 256-bit register stack. Unlike in traditional operating systems, in EVM, program's operators and operands are all pushed onto the stack indistinguishably. As EVM relies on fewer operator types, it is not as complex as traditional operating systems. For example, Dalvik Instruction Set Architecture (ISA) defines 218 Android application instructions, and Java runtime machine (JVM) provides about 202 instructions, whereas only 137 are provided by the EVM.

Traditional program classification techniques are mostly based on static and dynamic features of the code. Due to the simplicity of smart contracts, it is much harder to generalize features to classify DApps.

2) Redundancies in libraries: To reduce human effort and avoid source code duplication, DApp developers tend to invoke public libraries developed by some third-party organizations (like openzeppelin[1] or oraclize[2]). Most developers and companies also store proprietary code in private libraries for subsequent reuse. In the context of this investigation, a library can be regarded as a special kind of smart contract. A typical smart contract is composed of multiple functions with different access scope, such as Internal, External, Public, Private, etc. However, in contrast to traditional programming languages, when compiling a smart contract that relies on libraries, all public and external functions will be compiled into application binary interfaces (abbreviated as ABIs), which serve as the inference for library invocations, regardless of whether the function is needed by the DApp.

We analyze library invocation by all DApps from the three DApp Stores, whereby Table III provides all libraries that are invoked by more than one category. As can be seen, most public or external functions are never used. Due to such redundancy, many conventional program classification techniques (which are often based on program similarity) cannot be applied effectively.

Moreover, if a library is invoked by multiple DApps from different categories, these DApps will have similar sets of

---

[1] https://openzeppelin.org/
[2] http://provable.xyz/

Library Name	#DC	#SC	#Int	#Ext	#Used
Math	11	47	12	4	0
ECRecovery	8	20	2	1	0
MathLib	3	8	4	3	0
PaymentLib	2	6	6	21	0
CommUtils	2	6	19	8	0
DLL	2	6	0	8	0
Player	2	6	9	8	1
AttributeStore	2	6	0	2	0
StringLib	2	6	1	1	0
SortitionSumTreeFactory	2	4	1	8	0
Helper	2	4	4	17	0
LinkedListLib	2	2	11	6	0

For each library, the columns of the table show, (from left to right): the number of DApp categories (#DC) and smart contracts (#SC) invoking the library, internal and private functions (#Int), external and public functions (#Ext), and public and external functions (#Used) in the library that are actually used.

ABIs. As a result, ABI-based classification techniques which are commonly used in Mobile App classification cannot be directly applied to classify DApps.

As illustrated above, current DApp classification is time-consuming and error-prone, as it is challenging to utilize the limited information from the DApp itself for this purpose. To address this issue, we develop an automated classification method, DAppClassifier, as described in the subsequent section.

IV. DAPPCLASSIFIER

A. An Overview of DAppClassifier

In this section, we use machine learning to solve the DApp classification problem. The resulting DAppClassifier predicts DApp category based on easy-to-access features representing the actual DApp functionalities. The overall DAppClassifier structure is shown in Figure 1. Given the input of all information pertaining to a particular DApp—including DApp bytecode, transactions in history, and DApp source code (optional)—the classification is performed in two steps: the feature extraction process and the classification process. In the following sections, we provide detailed description of these two processes.

Fig. 1. An Overview of DAppClassifier

B. Feature Extraction Process

In this step, various features that are related to the DApp category are identified, ensuring that they are easy-to-access and can reflect the program functionalities.

The feature extraction process is guided by three hypotheses: (1) Function names reflect their actual functionalities; (2)

Similar execution logic is more likely to imply similar program functionalities; and (3) The developer-defined names in source code are indicative of the actual functionalities.

1) Function name features: In practice, function names should reflect program functionalities. However, these can be difficult to obtain. Intuitively, function names can be deduced from function calls incorporated into the execution traces during transactions, but this is unreliable, as not all functions will be called during execution, especially for infamous DApps. As an alternative, features can be extracted from binary code and transactions. However, this approach leads to two difficulties discussed below.

First, only the 4-byte digital signatures (hash values of function names) can be extracted from binary code. These signatures cannot be directly used as features, because one of the classification criteria mandates that similar function names should imply similar functionalities. Thus, it is necessary to convert each signature into the initial word-form. Second, as explained in Section III, a significant number of unused library functions are compiled into the bytecode, which would compromise the effectiveness of the selected features.

Our feature extraction approach can mitigate these issues. We retrieve word-form function names by utilizing Ethereum Function Signature Database[3] —a large-scale open-source database that records common function names and corresponding signatures. This allows those 4-byte signatures to be mapped to their human-readable versions (in our experiment, about 60% function names can be matched). In addition, we perform an additional filtering process, whereby we identify all library functions before collecting all the function names invoked through transaction. Finally, we remove all the un-invoked public library function names to eliminate redundancies.

2) Bytecode block features: The bytecode reflects DApp behavior, and can thus be used in classification. However, the bytecode cannot be used directly due to several reasons.

First, bytecode is composed of three components: creation code, swarm code, and runtime code. As the first two elements are used for the creation and distributed storage, they have no contribution to the program runtime behavior. Second, runtime code consists of operators and operands (i.e., PUSH 0X80). If the operands are not filtered out, the extracted feature will be too sensitive to the operands. Third, runtime code comprises of multiple blocks, each of which indicates a logical unit. Thus, to maximize the feature utility, runtime code should be divided into basic blocks.

To alleviate these issues, DAppClassifier extracts refined and subtle features from bytecode in three steps: [2]

1) Redundant code elimination: In this step, all superfluous creation and swarm code parts, which easily identified, are removed.
2) Desensitization: All the operands (i.e., the immediate numbers after operators) are removed. Note that, as a special kind of operand, the signature of function name

[3]https://www.4byte.directory/

is also removed. Those have already been discussed in function name features.

3) Division into basic blocks: In runtime code, operators JUMP, JUMPI, REVERT, STOP, and RETURN are the indicators of an interruption in a logical relationship. DAppClassifier divides the runtime code according to the opcodes.

Following the above process, DAppClassifier can identify refined and subtle block-level sub-sequences of bytecode as a feature.

3) Source code features: Compared with bytecode, source code can better convey the intent of developers. Not only the function names and statements are in high-level language, which is similar to the human language, but code also contains abundant programming notes, which convey intention of developers.

For this reason, DAppClassifier applies tokenization and embeddings features from source code, which is a two-layer neural network that processes text, whereby its input is a text corpus and its output is a set of vectors.

C. Classification Process

In the classification model, (1) three units are used to handle the three kinds of features, which are subsequently combined by (2) applying another Deep Neural Network (DNN) model.

1) DNN for Feature Name features: DNN model is a learning method with multiple layers of neural networks. It is particularly suitable for classification prediction problems where inputs are assigned a class or label.

Specifically, our network is a fully connected DNN model with RELU activations, N layers, and M units per layer. We swept over N = [3, 4, 5, 6, 7] and M = [32, 64, 128, 256, 512]. Our best performing model has N = 3 layers and M = 64 units, yielding a learning rate of 3×10^{-5}.

2) CNN for Function Body features: Convolutional Neural Network model (abbreviated as CNN) is a neural network that uses convolution in place of general matrix multiplication.

Specifically, our network is a CNN model with an embedding layer (embedding dim = 128), a convolutional layer (256 filters with kernel_size = 5) and a MaxPool layer (each layer contains 64 neurons).

3) CNN for Source Code features: In line with the approach adopted for sentence classification in the *NLP* scenario, to handle features, CNN with the same specifications as given above is utilized to extract meaningful sub-structures from source code.

4) DNN to Conjunct the Units: The intermediate results of the above three models are concatenated by a model composed of three layers: an input layer combining the output of the three units, a 128-dimensional fully connected layer, and a *softmax* layer to output the final category.

V. EVALUATION

In this section, we describe the experimental design adopted for evaluating DAppClassifier, followed by the experimental results demonstrating its correctness and effectiveness.

TABLE IV
EXPERIMENTAL RESULTS

Dataset	DAppClassifier	FuncName	ByteBlocks	Source
#S	84.6%	78.2%	77.2%	79.9%
#R	83.5%	78.9%	76.3%	79.3%
#D	84.0%	80.1%	79.6%	70.3%

A. Evaluation Design

We conduct large-scale experiments on DApp classification based on our self-constructed open-sourced dataset.

Dataset: Data on the three most commonly-used DApp Stores, namely State of the DApps, DappRadar, and Dapp.com are crawled, manually checked and all misclassified DApps are relabeled or removed(As explained in Section III-B),we matched them by front-end URL and contract address. If they have the same front-end URL or similar front-end URL with the same contract address, we consider them to be the same DApp, for some DApps have different URL parameters in order to identify the source of the request. **Yielding a large-scale dataset covering about 2,573 DApps including 11,230 smart contracts deployed on the Ethereum Environment.**

For each DApp, we download the bytecode and transactions, and retrieve its source code (if available) by Etherscan. To collect data on these smart contracts, we manually run an Ethereum node, starting from the Genesis block to the latest block to identify all the transactions and extract bytecode and runtime code. We focus on the DApp categories containing at least 20 DApps, since these categories are more representative of the general classification status. The revised dataset has been released (https://bit.ly/2JFmtiS), it contains the DApp name, contract address, category, DApp url, etc..

B. Experimental Results

In this section, we present the experimental results related to the two research questions.

RQ1: How many DApps can be accurately classified into the correct category? The experimental results of our classifier reported in Table IV indicate that its accuracy surpasses 84% when applied to each of the DApp Stores, confirming that our approach is technically promising. As precision, recall and F-score are consistent for multiclassification problem, only the precision of our approach is reported.

RQ2: How does each feature contribute to the DApp classification performance?

The columns 3–5 of Table IV show the performance (measured in terms of precision) of DAppClassifier based on a single feature (with a single classification unit).

Features extracted from source code should aid in accurate classification, as the programmer's intent is typically conveyed through function and routine names and developer comments. However, the results reported in the last column of Table IV counter this intuitive expectation, requiring further investigation.

Note that the experiments focusing on source code only are conducted on DApps for which source code is available. As nearly half of the DApps lacked source code, the dataset

used for training in this experiment is substantially reduced. In particular, in *Dapp.com* only 50.4% of DApps are released with source code, compared with 63.9% and 56.7% DApps on *State of the DApps* and *DappRadar* respectively. Thus, all three features should be utilized to improve the classification accuracy.

VI. DISCUSSIONS

In this section, we will discuss the benefits and shortcomings of our approach.

Function name feature extraction benefits: In the proposed approach, prior to extracting function name features, based on the unique DApp characteristics, all the un-invoked public library function names are removed. To evaluate the effectiveness of this strategy, we conduct another experiment on the DApps with historical transactions, applying only the function name feature unit. Besides the feature extraction process described in Section IV, we conduct comparison experiments, each incorporating the following feature extraction processes:

- *Names from Invocation & Bytecode removed unused libraries*: The features we used in our approach.
- *All names from Bytecode*: All function names extracted from bytecode are utilized, i.e., the unused public library functions are not removed.
- *All names from Invocation*: All function names from invocations in the history of transactions are identified and utilized.
- *Names from Invocation & Bytecode*:The function names collected by adopting the strategy described in Section IV are combined with those from the invocations.

As can be seen in Figure. 2, the features collected by utilizing *All names from Invocation* process achieve the lowest accuracy. This finding supports our hypothesis that function names identified through invocations of historical transactions are not sufficient for meaningful classification, due to users inadequate. The features collected via the strategy denoted as *All names from Bytecode* yield a higher precision because all the functions in the application are utilized, not just those that have been called. However, lack of understanding of user behavior leads to insufficient accuracy.

We take the advantage of both and combine them as input. The features collected through *Names from Invocation & Bytecode* and *Names from Invocation & Bytecode removed unused libraries* can both reach the top accuracy given a long training time. However, by removing all the unused library methods, the learning convergence can be expedited.

VII. RELATED WORK

Several studies have been conducted to evaluate the Ethereum ecosystem. For example, some authors characterized money transfer [7], contract invocation [8], code similarity [9] of Ethereum. Other researchers focused on financial activities on Ethereum, including Ponzi schemes [3], Honeypots [4] detect and ICO behavior finding [10], which might be a complement to our work. There is an empirical study on

Fig. 2. Comparison within different features

distributed applications [11]; however, our research is more systematic and is conducted on a larger scale. Moreover, machine learning has been used to label similar smart contracts with source code [2], even though its notion of a "cluster" cannot be precisely defined.

VIII. CONCLUSION AND FUTURE WORK

Owing to the development of blockchain and mobile technology, the number of DApps has already surpassed 2,500, and the scale of DApp market is estimated at billions of dollars [1]. To provide a better comprehension of decentralized applications (DApps), we have conducted a systematic empirical study on DApp classification status, and have constructed a dataset by relabeling misclassified DApps. Based on our empirical findings, we have proposed DAppClassifier—a novel approach for classifying DApps based on their real functionalities. Extensive evaluations have demonstrated that DAppClassifier can achieve an average precision of greater than 84%.

REFERENCES

[1] S. of the DApps, "Stateofthedapps," 2020. [Online]. Available: https://www.stateofthedapps.com

[2] R. Norvill, B. B. F. Pontiveros, R. State, I. Awan, and A. Cullen, "Automated labeling of unknown contracts in ethereum," in *2017 26th International Conference on Computer Communication and Networks (ICCCN).* IEEE, 2017, pp. 1–6.

[3] W. Chen, Z. Zheng, J. Cui, E. Ngai, P. Zheng, and Y. Zhou, "Detecting ponzi schemes on ethereum: Towards healthier blockchain technology," in *Proceedings of the 2018 World Wide Web Conference.* International World Wide Web Conferences Steering Committee, 2018, pp. 1409–1418.

[4] C. F. Torres and M. Steichen, "The art of the scam: Demystifying honeypots in ethereum smart contracts," *arXiv preprint arXiv:1902.06976*, 2019.

[5] D. Radar, "Dappradar," 2020. [Online]. Available: https://dappradar.com

[6] DApp.com, "Dappcom," 2020. [Online]. Available: https://www.dapp.com

[7] T. Chen, Y. Zhu, Z. Li, J. Chen, X. Li, X. Luo, X. Lin, and X. Zhange, "Understanding ethereum via graph analysis," in *IEEE INFOCOM 2018-IEEE Conference on Computer Communications.* IEEE, 2018, pp. 1484–1492.

[8] L. Kiffer, D. Levin, and A. Mislove, "Analyzing ethereum's contract topology," in *Proceedings of the Internet Measurement Conference 2018.* ACM, 2018, pp. 494–499.

[9] N. He, L. Wu, H. Wang, Y. Guo, and X. Jiang, "Characterizing code clones in the ethereum smart contract ecosystem," *arXiv preprint arXiv:1905.00272*, 2019.

[10] G. Fenu, L. Marchesi, M. Marchesi, and R. Tonelli, "The ico phenomenon and its relationships with ethereum smart contract environment," in *2018 International Workshop on Blockchain Oriented Software Engineering (IWBOSE).* IEEE, 2018, pp. 26–32.

[11] K. Wu, "An empirical study of blockchain-based decentralized applications," *arXiv preprint arXiv:1902.04969*, 2019.

Automated Rogue Behavior Detection for Android Applications

Shuangmin Zhang, Ruixuan Li✉, Junwei Tang, Xiwu Gu
School of Computer Science and Technology
Huazhong University of Science and Technology
Wuhan, China
E-mail: {shmzhang, rxli, jwtang, guxiwu}@hust.edu.cn

Abstract—There are a large number of third-party application markets that provide application download services for Android users. In order to improve users' satisfaction, major application markets urgently need an automated solution to avoid some rogue behaviors that affect users' experience, such as rogue advertisements that induce users to click, rogue pop-up boxes that cannot be closed normally, and rogue floating windows that affect users' experience. To address such problems, we propose a rogue behavior detection framework. We use the object detection approach to identify advertisements in screen views, the random forest method to identify the pop-up views, and then combine image analysis, natural language processing and heuristic methods to detect rogue behaviors. The proposed framework can also record evidences of rogue behaviors in applications, so that application markets can ask developers to rectify. The experimental results show that the precision of rogue application detection reached 96.7% and the recall reached 90.1%.

I. INTRODUCTION

In the last decade, it has entered the era of mobile devices. According to the mobile operating system market share in March 2020, Android system has the highest share at 72.26% [1]. The huge user base of the Android system has also spawned a large number of feature-rich Android applications. Users with high security awareness generally choose to download applications from large application markets, because most of them perform virus detection before allowing the applications to be online. However, there are still a large number of problematic applications that can cause trouble to users in application markets.

Research on Android applications mainly focuses on safety issues that pose a significant risk, such as malware detection [2], privacy leak detection [3] and ad fraud detection [4]. Little attention has been paid to user experience of using Android applications. For the detection of rogue ads (a kind of rogue behavior), the detection of ads in the screen views is a key point. Ad detection methods mainly adopt traffic analysis method [4] [5]. These methods can only identify the screen views containing ads, but cannot give the specific location of the ads. Hence, an efficient method is needed to detect the location of the ads in screen views. In order to improve users' experience and satisfaction, an automated solution is urgently

needed to identify rogue behaviors that affect users' experience in Android applications, including some rogue behaviors that only affect users' experience without causing major security issues.

In this paper, we propose a framework to find rogue behaviors that affect users' experience. The contributions of our work are as follows. We propose an object detection approach based on deep learning approach to detect ads in screen views. It can be used to serve rogue ad detection. We present a classifier that can divide the screen views into those containing and not containing pop-up boxes based on random forest. It can be used to serve rogue pop-up box and rogue ad detection. We implement the rogue ad detection module based on natural language processing and heuristic rules, the rogue pop-up box detection module based on heuristic rules, and the rogue floating window detection module based on image analysis and heuristic rules.

II. DEFINITION AND CATEGORIES OF ROGUE BEHAVIORS

A. *Definition of Rogue Behavior*

Rogue Behavior: A kind of application behavior that indirectly affects the user's mobile device, making the user unable to use the mobile device conveniently, and bringing potential threat to the user's mobile device. It has no direct damage to the system after the execution, nor causing the infringement of user's personal information and fees [6].

B. *Categories of Rogue Behavior*

1) Rogue Ad: **Ads with contents that induce users to click.** Some ads often display some inductive information to induce users to click. As shown in Fig. 1(a), the bottom of the left screen view is an ad. The "Clear Memory" and "Close" in the ad view are fake buttons. Such ads can easily mislead users to think that their devices need to clear memory. When a user clicks on the green fake button, the installation package starts to be downloaded. **Ads that overlay clickable components.** To improve the click rate of ads, applications may pop up an ad pop-up box immediately after a normal pop-up box. The original intention of the user is to click the button of the normal pop-up box, but at this time, the ad pop-up box pops out suddenly, and the user is highly likely to click on the ad by mistake, as shown in Fig. 1(b). **Ads that appear**

before application exit. After a user's click on the exit key, the application will usually pop up a prompt box to confirm if the user really wants to exit. When the user clicks the "Exit" button, the user's intention is to leave the application. However, some applications show an ad after clicking the "Exit" button, as shown in Fig. 1(c).

2) Rogue Pop-up Box: Some developers may use pop-up box inappropriately, causing trouble to users. As shown in Fig. 1(d), the prompt pop-up box of the application only provides one button means "update", without the button to close the pop-up box. Through manual testing, even clicking the back key of Android device still cannot exit, which will cause great trouble to users.

3) Rogue Floating Window: Some applications misuse the floating window to place ads for profit. Because a floating window is floating on the normal application screen view, it will cover part of the contents in the application screen view, which will affect users' experience. Android application developers generally like to design the floating window used for advertising purposes as red-envelope-style to attract users to click on the ad, as shown in Fig. 1(e).

Fig. 1. Rogue Behaviors.

III. METHOD

The overall scheme of rogue behavior detection for Android applications is shown in Fig. 2. The Android application traversal module automatically runs applications and records information based on Appium[1]. The information of the screen views and the events that caused the views' transition are recorded during the traversal process, as shown in Fig. 3. The main purpose of the Android application screen view classification module is to classify the application screen views. This module will train a deep learning-based object detection model and a random forest-based classifier. The object detection model is used to identify ads in the screen views. The random forest classifier is used to divide the screen views into views containing pop-up box and those not containing pop-up box. The rogue ad detection module, the rogue pop-up box detection module and the rogue floating window detection module are used to identify rogue behaviors in Android applications.

[1]Appium. http://appium.io/

Fig. 2. Overall Scheme of Rogue Behavior Detection.

Fig. 3. Recorded Information.

A. Ad Detection

We design a lightweight network based on RetinaNet [7], as shown in Fig. 4. The left is backbone, the medium is Feature Pyramid Networks (FPN) structure [8], the right are classification and bound regression subnetworks. Each of L1 and L2 is a convolution. L3, L4 and L5 are made up of 4, 8 and 4 cells respectively, and each cell is a residual module composed of three deep separable convolutions [9]. By FPN structure, features of different scales are fused to combine the high-level information with the low-level information. Each of the classification and bound regression branches contains two residual modules consisting of two convolutions. Finally, a convolution is added as the output layer. We design 24 anchors of different scales and aspect ratios at each location. For the classification branch, each anchor is classified as an ad or non-ad, so the number of the output channels is 48. For the bound regression branch, every four numbers make up the upper-left and lower-right coordinates of the border, so the number of the output channels is 96.

Fig. 4. Ad Detection Model.

B. Pop-up View and Non-Popup View Classification

We divide the Android application screen views into views containing pop-up box and those not containing pop-up box. We analyze the information of 1,000 Android applications, and summarize a series of features. These features include some 0-1 features as well as continuous value features. These features can be roughly divided into the following categories: component size features, component quantity features, component location features, keyword features and the proportions of components with specific attributes. The extracted features are suitable for decision tree classifier, so we use a random forest classifier based on decision tree. The random forest classifier has higher accuracy by combining the results of multiple base classifiers.

C. Heuristics-based Detection of Rogue Behaviors

1) Rogue Ad Detection: **Detection of ads with contents that induce users to click.** We first get the ad images cropped from the screenshots according to the ad detection result, and then we use the Optical Character Recognition (OCR) API provided by Baidu[2] to extract the texts in ad images. Then, we analyze the semantics of the advertising texts. In order to remove the interference of the texts outside the ad, we adopt difference set of OCR texts and view texts. Due to Chinese has multiple expressions of the same meaning, we need to understand the degree of similarity between different words. We adopt neural network word vectors for word representation, so we can identify other inductive sentences of similar meaning. The detection can be considered as an advertising text classification task. We adopt FastText [10] as the classification model, because it is based on word vectors and maintains high accuracy while training and testing time are greatly reduced. **Detection of ads that overlay clickable components.** In order to overlay the clickable components, the ad view should have an ad that occupies more than 30% of the screen space. The previous screen view of the ad view is a pop-up view that contains at least one clickable component. **Detection of ads that appear before the application exits.** Generally, before exiting the application, users will be asked if they really want to quit. There are two situations at this time. One is that the user clicks the "Exit" button, then will lead to exit. In normal circumstances, this prompt view should be the

last view in the traversal process. If an ad view appears after that, this ad is a rogue ad. Another situation is that the user clicks the "Cancel" button, then will go back to the previous view. Because the previous view may contain ads, the ads that appear at this time should not be considered as rogue ads.

2) Rogue Pop-up Box Detection: The detection of pop-up boxes that force applications to upgrade requires the cooperation of the automated traversal module. In automated traversal process, we remove events of update buttons. If a pop-up view contains "update", "upgrade", "download" or other texts with similar meanings, and the event executed on this view is "Back" type event, and the ID of the next view is equal to this view, the view is considered to have a pop-up box that forces users to update the application.

3) Rogue Floating Window Detection: The most intuitive feature of rogue floating windows is that they have red-envelope-style small icons that are approximately square. We consider components with an aspect ratio between 0.8 and 1.2 to be regarded as approximate square, and extract the pixel value of each pixel for such components. The screen views that contain such components with visual red pixels accounting for more than 20% and visual yellow pixels accounting for more than 3% are considered as those likely to have red-envelope-style floating windows. It is found that rogue floating windows often exist in multiple screen views. Therefore, we further consider the context information of the current screen view. If one of the surrounding screen views also contains red-envelope-style icon, we believe it contains rogue floating window.

IV. EXPERIMENTAL EVALUATION

A. Dataset

The dataset used in our work is from the application market named "Mango Download Station"[3]. We collected a total of 4,000 applications. For Android application ad detection, 500 applications were randomly selected as the dataset. For the classification of views containing pop-up box and views not containing pop-up box, 1,000 applications were randomly selected as the dataset. For the detection of rogue behaviors, the remaining 3,000 applications were selected as the test set, which did not include the training set used for ad detection and the dataset used for the classification of screen views.

B. Result of Ad Detection

We use 300 applications as the training set, and the remaining 200 applications as the validation set. The validation results are shown in Fig. 5. Fig. 5(a) shows the precision curve. Fig. 5(b) shows the recall curve. Fig. 5(c) shows the F1 curve. Fig. 5(d) shows the precision-recall curve. The abscissa represents the recall under different score thresholds, the ordinate represents the corresponding accuracy, and the area under the curve represents the Average Precision (AP). Here, the score threshold means that during evaluation, a predicted box with a score lower than the threshold will be

---

[2]Baidu OCR. https://ai.baidu.com/tech/ocr/general

[3]Mango Download Station. http://www.90370.com

thought as a negative case, otherwise as a positive case. The evaluation was conducted with an Intersection Over Union (IOU) of 0.75, which means that the predicted box and the ground truth box are considered to be matched if their IOU is greater than 0.75. Based on the results and combined with the need to identify ads in our work, the score threshold with higher recall rate is selected under the condition that the precision and F1 value are not too low. The score threshold we use for ad detection is 0.42. The AP is 0.808 on the validation set. The effect of ad detection is good enough to be used in our work to serve rogue ad detection.

Fig. 5.   Precision, Recall, F1 and P-R Curve.

C. Result of Pop-up View and Non-Popup View Classification

In this experiment, 1,000 applications were randomly selected as the dataset. For the evaluation of the classifier, the 10-fold cross-validation method is used. We randomly divided the data into 10 parts, one of which was taken as the validation set and the other 9 parts as the training set. The experiment was conducted for 10 times, and the arithmetic mean value of the results was taken as the final result. The results show that the average accuracy of the classification is 98%. The effect of the classifier is good enough to be used in our work to serve rogue pop-up box detection and rogue ad detection.

D. Result of Rogue Behavior Detection

By manually marking the data set, 131 of the 3000 applications in the test set were marked as having rogue behaviors. Using our method to test the 3,000 applications, the results of rogue behavior detection are shown in Table I. The results show that 122 applications have rogue behaviors, of which 118 applications do have rogue behaviors, and 4 do not. 13 applications with rogue behaviors were not identified. In summary, the precision of rogue behavior detection in our work is 96.7%, and the recall is 90.1%.

TABLE I
RESULT OF ROGUE BEHAVIOR DETECTION.

		Predict Labels	
		Rogue Apps	Normal Apps
Actual	Rogue Apps	118	13
Labels	Normal Apps	4	2865

V. CONCLUSION AND FUTURE WORK

In this paper, we propose an effective rogue behavior detection framework for Android applications. We implement ad detection, screen view classifier, rogue ad detection, rogue pop-up box detection, and rogue floating window detection. Using the methods we proposed, 118 Android applications containing rogue behaviors were found in 3,000 applications. The results show that the precision of rogue application detection reached 96.7% and the recall reached 90.1%.

We only identify five types of rogue behaviors, and other rogue behaviors may exist in reality. Meanwhile, new rogue behavior is still emerging. The methods proposed in this paper is scalable, and this study can be easily expanded to new rogue behaviors.

VI. ACKNOWLEDGEMENT

REFERENCES

[1] statcounter, "Mobile operating system market share worldwide," https://gs.statcounter.com/os-market-share/mobile/worldwide, 2020.

[2] Y. S. Sun, C.-C. Chen, S.-W. Hsiao, and M. C. Chen, "Antsdroid: automatic malware family behaviour generation and analysis for android apps," in *Australasian Conference on Information Security and Privacy*. Springer, 2018, pp. 796–804.

[3] W. Enck, P. Gilbert, S. Han, V. Tendulkar, B.-G. Chun, L. P. Cox, J. Jung, P. McDaniel, and A. N. Sheth, "Taintdroid: an information-flow tracking system for realtime privacy monitoring on smartphones," *ACM Transactions on Computer Systems (TOCS)*, vol. 32, no. 2, pp. 1–29, 2014.

[4] F. Dong, H. Wang, L. Li, Y. Guo, T. F. Bissyandé, T. Liu, G. Xu, and J. Klein, "Frauddroid: Automated ad fraud detection for android apps," in *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2018, pp. 257–268.

[5] J. Crussell, R. Stevens, and H. Chen, "Madfraud: Investigating ad fraud in android applications," in *Proceedings of the 12th annual international conference on Mobile systems, applications, and services*, 2014, pp. 123–134.

[6] C. C. S. Association, "Description format for mobile internet malicious code," http://www.ptsn.net.cn/standard/std_query/show-yd-3983-1.htm, 2013.

[7] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, "Focal loss for dense object detection," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 2980–2988.

[8] Q. Zhao, T. Sheng, Y. Wang, Z. Tang, Y. Chen, L. Cai, and H. Ling, "M2det: A single-shot object detector based on multi-level feature pyramid network," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, 2019, pp. 9259–9266.

[9] F. Chollet, "Xception: Deep learning with depthwise separable convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 1251–1258.

[10] A. Joulin, E. Grave, P. Bojanowski, M. Douze, H. Jégou, and T. Mikolov, "Fasttext. zip: Compressing text classification models," *arXiv preprint arXiv:1612.03651*, 2016.

Benchmarking the efficiency of RDF-based access for blockchain environments

Juan Cano-Benito   Andrea Cimmino   Raúl García-Castro

Ontology Engineering Group
Universidad Politécnica de Madrid
E-mail: {jcano,cimmino,rgarcia}@fi.upm.es

Abstract

Blockchain and knowledge graphs are technologies that have become pervasive in several domains where web services have been developed relying on them. The inmutability of the data offered by the blockchain together with the capabilities of the knowledge graph when consuming data, enables web services to provide richer functionalities. Literature has explored the benefits of combining both qualitatively, and only a few works have exposed quantitatively the feasibility of combining these technologies. In particular, as far as we know no work reports the cost of storing knowledge graphs serialized in RDF into blockchains, or analyses alternatives such as virtualisers that transform on the fly data from different formats into RDF. In this paper we present an empirical analysis of the cost of storing into a blockchain in comparison with storing JSON, and the benefits when solving SPARQL queries by reading directly the RDF or using a virtualiser fed with RDF. For the sake of our experiments, we rely on different sensors that store their data into two blockchains, on top of which we perform our analysis.

Keywords: *Blockchain, Knowledge Graph, Semantic Web, RDF, RDF Virtualisation*

1 Introduction

Nowadays blockchain has become a pervasive technology in a wide range of sectors [1]. The reason is due to the fact that it allows to store data ensuring its inmutability [14]. The data stored into a blockchain may be expressed in any format and under any model. As a result, a large number of services have decided to publish knowledge graphs (KGs) relying on blockchain to store their data [21].

Blockchain has many implementations, such as Ethereum, Bitcoin, or Hyperledger Fabric. These implementations often associate a cost to the amount of data that peers write in the chain. As a result, the same data

written in the chain with a verbose format have a higher cost to be paid by a peer, in comparison with having the data represented with a simpler format.

The cost of writing becomes especially relevant when blockchain is storing KGs since their data format is Resource Description Framework (RDF), which is known to be verbose. Therefore, although a KG stored in a blockchain has clear benefits when consuming data due to the RDF, this format will entail a higher cost in comparison with other lighter formats, it has also an expected higher cost. There is an ever-growing number of proposals that store a KG in a blockchain, but there is a lack of knowledge about how suitable this approach is and if other alternatives could work better.

In this paper a case study is presented in which an empirical analysis is performed in order to establish the benefits and costs of storing a KG in a blockchain. In addition, a virtualisation approach that generates virtual RDF from data expressed in JavaScript Object Notation (JSON) that is stored in a blockchain is considered. The scope of this paper is establishing how costly is storing RDF instead of JSON, and if a virtualisation approach is a better alternative that directly storing RDF in the blockchain.

The case study is contextualised in a simulated research laboratory that counts with 15 light bulb sensors, an occupancy sensor, and a temperature sensor. The sensors send data to an agent that writes such data into two different Ethereum blockchains. In one of them, data is written as plain JSON, whereas on the other one, data is expressed in RDF using the VICINITY ontology [7]. The analysis consists in measuring how costly is storing RDF and JSON in terms of gas, and how effective is querying the data is querying either data.

The analysis carried out aims at exploring the following research questions:

- H1: What has a higher cost when writing data in the blockchain, RDF or JSON?

- H2: What is faster when reading from the blockchain, RDF or JSON?

- H3: Considering a virtualiser that transforms on the fly JSON data into RDF. What is faster to query, RDF or virtual RDF?

The rest of the article is organised as follows: Section 2 reports proposals in the literature combining these technologies; Section 3 introduces concepts used across the paper; Section 4 presents the architecture followed in our experimental analysis; Section 5 explains how the experimentation was carried out and reports its results; and, finally, Section 6 recaps our conclusions and main findings.

2 Related Work

The approach of storing the RDF data of a KG in a blockchain has been addressed mainly from a theoretical point of view without reporting any quantitative analysis [4, 6, 8, 10, 13, 16, 24, 26, 27]. Although different proposals provide a preliminary qualitative analysis [15, 17, 22, 23], most of the works describe specific applications that have stored their KGs in a blockchain without analysing the efficiency of this decision over other alternatives [9, 11, 2, 25].

The majority of proposals address how semantic web and blockchain technologies could work jointly in order to enhance their benefits without providing any analysis of its feasibility [6, 8, 10, 16, 27]. Some proposals report a qualitatively analysis of how some specific domains could benefit from using these two technologies together. For instance, for chemistry [26], smart cities [24], publications [13], or government [4] domains.

Several proposals provide a quantitative analysis of the combination of these two technologies. Ruta et al. [22, 23] performed an analysis over the discovery of Internet of Things (IoT) resources whose meta-descriptions were stored in a blockchain using RDF. They reported discovery and query processing time over the RDF involved in such task. However, the results have not enough granularity to establish only the reading time of the RDF, nor they provide a comparison with other alternatives.

Le-Tuan et al. [17] presented a scenario of a small network of lightweight nodes. Each node processes 1 billion triples, but those triples are not stored in the blockchain that contains instead a hash pointing to an RDF online documents. Therefore, although the proposal reports the time for writing and querying data, these results do not involve directly the blockchain. As a result, the cost of writing is neither analysed or reported.

Ibañez et al. [15] studied the verbosity of RDF expressing data. They reported the number of bytes that different serialisations of RDF have when expressing the same data. In addition, authors considered the same information compressed with different algorithms. However, RDF was not stored in any blockchain, nor any cost was reported.

As a conclusion, the literature currently lacks to determine the benefits of storing KGs inside a blockchain from the point of view of the cost of writing RDF instead of other serialisations, e.g., JSON. Additionally, no work has explored alternatives like using RDF virtualisers in order to have the benefits of RDF when consuming data while storing in the chain less-verbose formats like JSON.

3 Background

Most of the concepts on top of which this paper is build are well-known, namely: RDF [5], the SPARQL Protocol and RDF Query Language (SPARQL) [12], JSON [3], and blockchain [20]. Nevertheless, others concepts are not terms widely known and, therefore, in this section they are defined.

Transaction: is the name of the operation that writes or stores some data inside a blockchain. Depending on the data size that is been written, it requires more or less space in one block. As a result, if transactions require more space than the one available in a block, they will be written in more than one block.

Usually, a transaction has a virtual cost since it requires a certain amount of computing power. As a result, performing a transaction has an associated cost in public blockchains and, depending on the implementation, it may have different names; for example, for Ethereum it is called Gas [28].

Software agent Autonomous actions in a tailored-domain environment can be done [29]. The means of the actions performed by an agent have as goal to meet a set of design requirements. A system with two or more agents is known as Multi-Agent System.

In the context of this paper, a proactive agent with simple reflexes based on condition-action is used.

RDF Virtualisation is a technique used in the semantic data integration context [18]. Usually, it refers to a piece of software connected to a data source and with a set of translation rules called mappings. These techniques are able to translate on the fly data from heterogeneous formats and models into RDF expressed according to a specific ontology, allowing to solve SPARQL queries over such data.

Virtual RDF is the one generated as an RDF virtualisation technique [18]. It receives such name due to the fact that the RDF is not stored anywhere and is consumed as produced; unless a software agent stores it somewhere.

4 Experimental Architecture

The scope of this paper is to provide an empirical analysis of how suitable is to store RDF inside a blockchain, due to the cost that it entails. Alternatively, storing JSON and using a virtualiser could bring the same benefits without the drawbacks of the former approach.

The scenario presented in Figure 1 has been endowed in order to perform the desired analysis. The scenario consists of a set of sensors which data is stored inside a blockchain using the JSON format. Besides, the same data is stored using RDF inside another blockchain. Then, relying on this infrastructure, a set of tests have been performed to find answers to the research questions reported in the introduction.

Next, the different components of the architecture are explained:

IoT Infrastructure: the sensors within the architecture are 15 light bulb sensors, 1 temperature sensor and 1 occupancy sensor. These devices send their data to the *IoT Collector* that forwards such information to the *Agent JSON Writer* and to the *Agent RDF Parser*. The data is reported by the sensors in JSON format.

Agent RDF Parser: this agent receives the JSON data from the *IoT Infrastructure* and using a fixed RDF template injects such data into the template using JSONPath expressions. Then, it forwards the instantiated RDF template to the *Agent RDF Writer*.

Agent RDF/JSON Writer: although the architecture counts with two different agents for this task, conceptually they perform the same function. Both receive a document and write it in the blockchain. The *Agent RDF Writer* stores the received RDF documents in the RDF blockchain serialised as Turtle, and the *Agent JSON Writer* stores the JSON documents received in the JSON blockchain.

Blockchain: in the architecture, data (either in RDF or in JSON) is stored in a different Ethereum blockchain. Their functionality is the same since the blockchain is agnostic to the data format.

Agent JSON/RDF Reader: the architecture counts with two different agents for this tasks that perform the same

function. These agents read the information within their respectively blockchain. As a parameter, they can receive the number of transactions to be read, providing as a result the collection of documents stored in those transactions.

Virtualiser: this component in the architecture is implemented with a software called Helio[1]. It reads a number of transactions from the blockchain and, relying on a set of translation rules, generates an RDF document with all the JSON documents stored in the RDF blockchain, i.e., the *Virtual RDF*. The virtual RDF is generated so it is exactly the same of the one provided by the *Agent RDF Reader*.

SPARQL Agent: this agent receives a SPARQL query and returns the query result. Depending on how it is configured, it relies on the *Virtual RDF* or on the RDF output by the *Agent RDF Reader* to answer the query.

The goal of this architecture is to provide a playground were different measurements can be taken. First, the gas consumption when storing the RDF or the JSON documents, relying on the *Agent Writers*. Secondly, the time that takes reading RDF and JSON documents from the blockchain, relying on the *Agent Readers*. Third, the time that takes answering a query with the data stored in a set of transactions when such RDF is provided by the *Agent RDF Reader* or the *Virtualisation* component.

As a result, by performing these measurements, the research questions introduced in Section 1 will be validated, analysing the feasibility of storing RDF or JSON directly on the chain, and using a virtualiser to obtain the RDF benefits.

---

[1]https://helio.linkeddata.es/

Figure 1. Experimental architecture

5 Experimental Analysis

The different experiments designed to address the three research questions formulated in this paper are reported in this section. First, the gas spent when storing RDF and JSON is measured in order to validate the first research question. Second, the time required to read from the JSON and the RDF blockchains, respectively, is measured in order to validate the second research question. Finally, the time taken to perform the same query relying on the one provided by the virtual RDF and the one provided by the RDF blockchain were measured to validate the third research question.

The best effort was done in order to prevent information loss due to the bespoke characteristics of each format. Therefore, the JSON and the RDF documents stored in both blockchains contain the same information. During the experiments we compared the results considering the same amount of transactions, namely: 2,000, 4,000, 6,000, 8,000, 10,000, 12,000, and 14,000.

All these tests have been carried out on a computer with the following characteristics: intel i7 6700k, 32 Gb RAM and 1Tb SSD.

Finally, all the times reported as box plots are measured in seconds reporting the results of executing 10 times each experiment. The test performed to establish if the results have statistically significant differences is the well-known Iman–Davenport test [19], with a confidence level of 95%. This test outputs a p-value; if this value is below 0.05 it means there are no statistically differences between the results, i.e., they can be considered the same.

5.1 Gas consumed storing RDF vs JSON

In this experiment RDF and JSON documents were stored in different blockchains. Both documents contained the same information; however, data expressed in RDF required around 6,000 characters, whereas JSON data required approximately 550 characters to encode the same information. Figure 2 depicts the gas consumed storing sets of RDF and JSON documents containing equivalent information.

As it can be observed, storing data in RDF requires for each transaction, on average, an amount of gas that is 10 times more than the one required by the information serialised in JSON. In this case there is no need of applying any statistical test since the magnitude of such difference makes results clear.

5.2 Time required to read transactions

In this experiment we measured the time that took reading a set of transactions from the JSON and the RDF

Figure 2. Gas consumed by RDF and JSON

blockchains, respectively. In addition, the time for the JSON data to be sent to the Helio virtualiser is included in the results. Figure 3 depicts the results of this experiment.

Figure 3. Time for reading the transactions

As it can be observed, the reading times for the three cases are close enough. The statistical test applied over their average values in order to ensure their statistical equivalence returns the following p-values: between JSON and Helio is 0.13, between JSON and RDF is 2.66×10^{-4}, and between Helio and RDF is 0.02×10^{-4}. With this p-values, it can be concluded that reading times are statistically equivalent between JSON and Helio. Instead, between JSON and RDF, and Helio with RDF, there is a statistical difference. As a result, reading JSON, and optionally feeding the Helio virtualiser, is faster than just reading the RDF.

5.3 Issuing SPARQL queries

In this experiment the time that took reading the blockchain plus the time that takes solving a SPARQL query was measured. The query issued asked about all the

known data in the blockchain. Figure 4 depicts the results of this experiment.

Figure 4. Time for querying all the data in the blockchain

At the light of these results, using the virtualiser Helio allows to solve the SPARQL queries faster than just aggregating RDF documents from the *Agent RDF Reader*. The difference is due to the fact that the translation is fast and produces a whole RDF document; instead, the *Agent RDF Reader* needs to aggregate all the documents into a single one before solving the query. This behaviour explains the linear growth of the results in the chart.

The statistical test outputs a p-value of 0.02; therefore, it can be concluded that there are no statistical differences and, thus, using a virtualiser in this context is the same than reading and querying the RDF directly.

6 Conclusions

This article presents an empirical study that aims at answering the research questions proposed in section 1. These questions revolve around if storing RDF in a blockchain is efficient, and if alternatives exist in order to keep the benefits of RDF but avoiding its drawbacks. The experimental results led to the following answers:

RQ1: at the light of the results reported in sub-section 5.1 we can conclude that writing RDF is more than 10 times more expensive than writing JSON.

RQ2: results from sub-section 5.2 advocate that reading JSON is faster reading than RDF; even feeding with the read data a virtualiser is faster than reading RDF.

RQ3: sub-section 5.3 proofs that querying the virtualiser is faster than reading and querying the RDF from the blockchain.

As a conclusion of our empirical analysis, storing RDF in a blockchain brings clear benefits for consuming data, e.g., been able to query semantic data, use standardized models or bring the benefits of link-data. However, RDF has some drawbacks: i) reading the data from the blockchain takes more time than reading the same data in other format like JSON, and also, ii) writing RDF in a blockchain has an elevated cost in terms of gas.

As a result, in this paper a virtualiser to translate on the fly JSON into RDF was analysed. The experimental results achieved proof that using a virtualiser under the studied circumstances is more efficient than using RDF. It has the same benefits, but none of its drawbacks.

In the future, this analysis will be extended considering SPARQL query rewriting techniques, which could be even more efficient than using virtualisers. Also other parameters will be studied, such as scalability and memory usage.

7 Acknowledgements

References

[1] J. Al-Jaroodi and N. Mohamed. Blockchain in industries: A survey. *IEEE Access*, 7:36500–36515, 2019.

[2] A. Banerjee and K. Joshi. Link before you share: Managing privacy policies through blockchain. In *2017 IEEE International Conference on Big Data*, pages 4438–4447. IEEE, 2017.

[3] L. Bassett. *Introduction to JavaScript object notation: a to-the-point guide to JSON*. O'Reilly Media, Inc., 2015.

[4] T. Beris and M. Koubarakis. Modeling and preserving Greek government decisions using semantic web technologies and permissionless blockchains. In *European Semantic Web Conference*, pages 81–96. Springer, 2018.

[5] D. Brickley, R. V Guha, and B. McBride. RDF schema 1.1. *W3C recommendation*, 25, 2014.

[6] J. Cano-Benito, A. Cimmino, and R. García-Castro. Towards blockchain and semantic web. In *International Conference on Business Information Systems*, pages 220–231. Springer, 2019.

[7] A. Cimmino, V. Oravec, F. Serena, P. Kostelnik, M. Poveda-Villalón, A. Tryferidis, R. García-Castro,

S. Vanya, D. Tzovaras, and C. Grimm. VICINITY: IoT semantic interoperability based on the Web of Things. In *2019 15th International Conference on Distributed Computing in Sensor Systems (DCOSS)*, pages 241–247. IEEE, 2019.

[8] J. de Kruijff and H. Weigand. Understanding the blockchain using enterprise ontology. In *International Conference on Advanced Information Systems Engineering*, pages 29–43. Springer, 2017.

[9] M. Demir, O. Turetken, and A. Ferwom. Blockchain and IoT for delivery assurance on supply chain (BIDAS). In *2019 IEEE International Conference on Big Data*, pages 5213–5222. IEEE, 2019.

[10] M. English, S. Auer, and J. Domingue. Blockchain technologies & the semantic web: A framework for symbiotic development. In *Computer Science Conference for University of Bonn Students*, pages 47–61, 2016.

[11] D. Graux, G. Sejdiu, H. Jabeen, J. Lehmann, D. Sui, D. Muhs, and J. Pfeffer. Profiting from kitties on ethereum: Leveraging blockchain RDF data with SANSA. SEMANTiCS Conference, 2018.

[12] S. Harris, A. Seaborne, and E. Prud'hommeaux. SPARQL 1.1 query language. *W3C recommendation*, 21(10):778, 2013.

[13] M. R Hoffman, L. Ibáñez, H. Fryer, and E. Simperl. Smart papers: Dynamic publications on the blockchain. In *European Semantic Web Conference*, pages 304–318. Springer, 2018.

[14] F. Hofmann, S. Wurster, E. Ron, and M. Böhmecke-Schwafert. The immutability concept of blockchains and benefits of early standardization. In *2017 ITU Kaleidoscope: Challenges for a Data-Driven Society (ITU K)*, pages 1–8. IEEE, 2017.

[15] Luis Daniel Ibáñez, Huw Fryer, and Elena Paslaru Bontas Simperl. Attaching semantic metadata to cryptocurrency transactions. In *DeSemWeb@ISWC*, 2017.

[16] Henry Kim, Marek Laskowski, and Ning Nan. A first step in the co-evolution of blockchain and ontologies: Towards engineering an ontology of governance at the blockchain protocol level. *SSRN Electronic Journal*, 2018.

[17] A. Le-Tuan, D. Hingu, M. Hauswirth, and D. Le-Phuoc. Incorporating blockchain into RDF store at the lightweight edge devices. In *International Conference on Semantic Systems*, pages 369–375. Springer, 2019.

[18] M. Lefrançois, A. Zimmermann, and N. Bakerally. A SPARQL extension for generating RDF from heterogeneous formats. In *European Semantic Web Conference*, pages 35–50. Springer, 2017.

[19] D. G Pereira, A. Afonso, and F. Medeiros. Overview of friedman's test and post-hoc analysis. *Communications in Statistics-Simulation and Computation*, 44(10):2636–2653, 2015.

[20] M. Pilkington. Blockchain technology: principles and applications. In *Research handbook on digital transformations*. Edward Elgar Publishing, 2016.

[21] S. Porru, A. Pinna, M. Marchesi, and R. Tonelli. Blockchain-oriented software engineering: challenges and new directions. In *2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C)*, pages 169–171. IEEE, 2017.

[22] M. Ruta, F. Scioscia, S. Ieva, G. Capurso, and E. Di Sciascio. Semantic blockchain to improve scalability in the Internet of Things. *Open Journal of Internet Of Things (OJIOT)*, 3(1):46–61, 2017.

[23] M. Ruta, F. Scioscia, S. Ieva, G. Capurso, and E. Di Sciascio. Supply chain object discovery with semantic-enhanced blockchain. In *Proceedings of the 15th ACM Conference on Embedded Network Sensor Systems*, pages 1–2, 2017.

[24] M. Ruta, F. Scioscia, S. Ieva, G. Capurso, G. Loseto, F. Gramegna, A. Pinto, and E. Di Sciascio. Semantic-enhanced blockchain technology for smart cities and communities. In *3rd Italian Conference on ICT for Smart Cities & Communities (I-CiTies 2017)*, 2017.

[25] M. Sicilia and A. Visvizi. Blockchain and OECD data repositories: opportunities and policymaking implications. *Library hi tech Journal*, 2019.

[26] J. J Sikorski, J. Haughton, and M. Kraft. Blockchain technology in the chemical industry: Machine-to-machine electricity market. *Applied Energy*, 195:234–246, 2017.

[27] H. Ugarte. A more pragmatic Web 3.0: Linked blockchain data. 2017.

[28] G. Wood et al. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum project yellow paper*, 2014.

[29] M. Wooldridge and N. Jennings. Intelligent agents: Theory and practice. *The knowledge engineering review*, 10(2):115–152, 1995.

SecureChange: An Automated Framework to Guide Programmers in Fixing Vulnerability

Sayem Mohammad Imtiaz[*1], Kazi Zakia Sultana[†2] and Tanmay Bhowmik[‡3]

[1]Department of Computer Science, Iowa State University, IA, USA
[2]Department of Computer Science, Montclair State University, NJ, USA
[3]Department of Computer Science and Engineering, Mississippi State University, MS, USA

Abstract

When developers fix a defect, they may change multiple files. The number of files changed for resolving the defect depends on how strongly the files are coupled with each other. In earlier works, researchers leveraged this coupling for better understanding and analyzing software as well as for guiding developers to quickly find all probable code areas to complete fixing a defect. In some studies, researchers generated association rules reflecting the coupling among files and built tools to automate the discovery of the related changes in the files. Such tools, however, do not consider the type of defects resolved earlier for generating the rules as a result of which many unrelated files may come up while changing a file in later releases for resolving a specific type of defect. Therefore, in our study, we consider only security defects or vulnerabilities to generate the rules and then automate the finding process of other related files while fixing a vulnerability. Our tool "SecureChange" suggests the developers a number of related files that might need to be changed while fixing a particular vulnerability based on the mined association rules from the revision history. This approach will have a significant role in guiding the developers in fixing a vulnerability. Furthermore, this will be an effective endeavor for training new developers based on the vulnerability history of a system, which will in turn help them to develop secure code. The proposed approach will also be helpful in educating new developers about software vulnerabilities. Finding all the related files which have been modified to fix a vulnerability, the new developers will be able to learn how the faults in a file can be the root cause
of a vulnerability and how it can propagate to other related files and ultimately emerge as a vulnerability to the outside world. As a demonstration of our approach, we generate association rules based on the revision history of three systems: Android, Mozilla Firefox, and Apache Tomcat. The average precision and recall of 44% and 44% respectively for three systems indicate the feasibility of our approach.

1. Introduction

Coupling among files in a software becomes crucial for program understanding and resolving issues in software maintenance [1–3]. Developers fix some software issues in every revision and record the related changes which are stored as revision histories. Mining this revision history can be a good source of discovering coupling among the files. Revision history also tells us how the program evolves over time, which can later be used for identifying the versions having or not having a particular issue. Researchers have used such historical data to support code navigation [4]. In [5,6], coupling has been used to analyze and get insights of the program. In order to guide developers in fixing defects, [7] suggested a technique to provide related changes by mining revision histories.

The state of the art does not focus on the revision histories that are targeted to fix particular types of issues (general bugs or vulnerabilities or adding new functionalities) [1–3, 7]. Here, a potential drawback is that there is a higher probability of increased false positives if the developer is not fixing the same type of issue compared to the earlier versions. Therefore, following such a generalized association rule based technique could be misleading for a developer as he/she might end up wasting time concentrat-

---
[*]sayem@iastate.edu
[†]sultanak@montclair.edu
[‡]tbhowmik@cse.msstate.edu

ing on irrelevant files considering the task at hand. In order to address this gap, in this paper, we consider the revisions that were made for resolving vulnerabilities and then extract the files that were changed together. We leverage the extracted components to generate association rules so that we can guide the developers later on fixing vulnerabilities. Software vulnerability is a mistake in software that can be directly used by a hacker to gain access to a system or network[1]. As a vulnerability can put the security of a software at risk, in our study, we considered software vulnerability and applied the framework for fixing the security issues.

Let us consider an example. According to the revision log[2] of Apache Tomcat (an open-source Java Servlet Container developed by the Apache Software Foundation (ASF)), $NamingContextListener.java$ has been modified 50 times since 2006 as a part of fixing different issues. But it was coupled with only one file $ResourceLinkFactory.java$ in Revision 1757271[3] where the developer fixed the "Unrestricted Access to Global Resources" vulnerability. This vulnerability[4] has been fixed by the developers in 2016. A developer does not need to consider all files related to $NamingContextListener.java$ since 2006 for fixing a vulnerability in this file. This story tells us that mining the revision log of a system could be of no use if we do not consider the type of issue to be resolved. Therefore, we concentrate on the revisions related to fixing vulnerabilities so that when the developers will try to fix any vulnerability later, they can be guided based on only the vulnerability-fixing related revisions of that file.

Another motivation of the paper comes from the need to educate new developers on vulnerability for a specific system. As a new developer may be unfamiliar with the system she is working on, this can be an effective guidance for her to fix the faults in all the related files and help her to be acquainted to the new system.

This paper, primarily motivated by the work of Zimmermann et al. [7], uses the concept of association rule mining to produce the list of coupled files from the revision history. In contrast to the state of the art, it focuses on quick fixing software vulnerability and ensuring secure coding. The objectives of the paper are as follows:

1. to obtain association rules reflecting the coupling among the related vulnerable files so that programmers can be guided in fixing vulnerability in later releases of the same system.

2. to assist the developers in finding and fixing vulnerabilities in an efficient and effective way, thereby ensuring secure software evolution.

3. to identify coupling among vulnerable files and thus increase the ability to understand and analyze vulnerable code for software maintenance.

4. to evaluate the proposed approach that suggests related changes that might be needed for fixing a particular type of vulnerability in a system.

Section 2 shows the related works. In Section 3, we discuss Apriori algorithm which we used to generate association rules from the vulnerability revision histories. Section 4 presents the methodology followed in *SecureChange* to perform the experiments. Finally, Section 5 presents the results and Section 6 concludes the study.

2. Related Work

In [8], Ying et al. applied data mining techniques in the change history and determined sets of files that were frequently changed together in the past. They hypothesized that the change patterns (pertinent set of files) can be recommended to the developers performing a modification task. They revealed valuable dependencies among the files in the Eclipse and Mozilla open source projects and evaluated the performance of the recommendations that were produced by their approach for actual modification tasks. Xing et al. [9] used association rule mining at the design level on versions of UML diagrams to detect class co-evolution. They presented three potential applications of class co-evolution discovery in the context of software maintenance: finding the scope of future maintenance activities, guiding refactoring activities and identifying system instabilities. Although they showed promising initial results of their approach, it still lacks in large scale evaluation. In [10], the authors investigated how a change in one source code entity propagates to other entities. They applied several heuristics to predict change propagation and validated their approach in five open source software systems. Gall et al. [2] first used release history of a system to uncover logical dependencies and common change patterns among modules in order to detect potential structural shortcomings. The CVS history has also been used to detect more fine-grained logical coupling between classes [5], files, and functions [6]. In [11–13], authors used inductive learning (a relevance relation identifying two files that are updated together) to learn different concepts between logically coupled files.

Michail [14, 15] applied data mining technique to discover library reuse patterns (for example, how library functions are used together or how library functions are overwritten by the applications classes). [15] considered the inheritance relationship and generated generalized association rules to find how the descendent classes have been invoked or instantiated. In SecureChange, we use association

rule mining for mining vulnerability revision histories and guiding developers in secure software development.

3. Introducing the Apriori Algorithm

In order to suggest relevant co-occurring changes for a particular change, *SecureChange* leverages an association rule learning technique known as Apriori algorithm [16]. Let us assume that we have a set of transactions, $T = \{T_1, T_2, T_3, ..., T_n\}$, where each transaction records a set of co-occurring change items, $C = \{C_1, C_2, C_3, ...\}$. Apriori algorithm generates a set of frequent items based on a support threshold. The frequent item set is then leveraged to learn association rules among individual items, C_i, in the set of all transactions. A rule is denoted as follows:

$$X \rightarrow Y \; where \; X, Y \subseteq C$$

Here, X is called antecedent and Y is called consequent. In other words, if X occurs, then Y follows. The rules are derived based on some parameters:

- **Support Count:** Support count indicates the popularity of an item. It measures how frequently a particular item appears in all transactions. It can be expressed as:

$$Support(X) = \frac{|\{t \in T, X \subseteq t\}|}{|T|}$$

- **Confidence:** Confidence measures the relative importance of the consequent in a rule. It indicates how frequently a consequent appears in all transactions that contain antecedent.

$$Confidence(X \rightarrow Y) = \frac{Support(X \cup Y)}{Support(X)}$$

Higher the confidence, better the likelihood of occurring Y, given X.

- **Lift:** Lift takes the popularity of both antecedent and consequent into account. It is possible that an item is generally very popular. It may occur in many transactions without maintaining a particular pattern with any antecedent. In such a case, confidence provides a poor feedback on a rule. However, lift resolves this problem by considering popularity of both the antecedent and consequent. In other words,

$$Lift(X \rightarrow Y) = \frac{Support(X \cup Y)}{Support(X) \times Support(Y)}$$

A lift value of 1 indicates that item-sets are independent of each other. Whereas, a value higher than 1 indicates a tie between item-sets. In other words, the higher the lift, the stronger the tie. On the other hand, lift value lower than 1 indicates a negative tie.

Table 1: Datasets

System	# Transactions	# Curated Transactions	Versions
Tomcat	76	48	5 - 8.5.38
Firefox	595	249	3.6 - 62.0.2
Android	1485	381	4.4 - 9.0

4. Methodology

This section describes the methodology of *SecureChange* in details.

4.1 Research Question

The primary challenge with mining association rules for secure development is that the occurrence of a vulnerability is very infrequent compared to regular bugs. As a result, vulnerability-fixing transactions are less in amount compared to other general transactions. Abundance of data is very crucial for the success of a data mining technique such as association rule mining. Therefore, in this study, we attempt to answer the following research question (RQ):

Given the limited availability of the vulnerability-fixing transactions, is association rule mining for secure software development as effective as it has been shown in guiding general software change tasks [7]?

4.2. Training Data Preparation

To evaluate *SecureChange*, we have performed case studies on three widely known real world open-source software systems, namely, Apache Tomcat[5], Mozilla Firefox[6], and Android Open Source Project (AOSP)[7]. The vulnerabilities detected and fixed in these systems are publicly accessible. We begin by mining respective security advisories of all the systems. A security advisory typically records detailed information of past vulnerabilities, including the source code changes for fixing the vulnerability and time of the fix. We treat set of all files changed for fixing a vulnerability as a single transaction.

Table 1 shows the demography of the collected data. We found that most of the transactions have only one change. These transactions can be easily identified by *Apriori* algorithm, hence leading to a bloated estimation of the performance. For instance, in our experiment, *SecureChange* provided two or three times better performance estimation than the one trained without such transactions. Therefore, we further curated the dataset to eliminate transactions with

---

[5]http://tomcat.apache.org/
[6]https://www.mozilla.org/
[7]https://source.android.com/

a single change and then assessed the effectiveness of *SecureChange* in predicting co-occurring changes.

4.3. Validation Technique

We have validated *SecureChange* in a repeated validation setup. The experiment has been repeated 100 times on a randomly shuffled dataset for each project. In each experiment, *Apriori* algorithm was trained with 90% of the data and remaining 10% was retained for testing. The data were randomly shuffled before each experiment. The random shuffling and repeating many times minimized the bias in the reported result. During every experiment, we performed the following steps:

1. Shuffle dataset randomly and split into two 90%-10% folds.

2. Train *SecureChange* with *Apriori* algorithm on the fold with 90% data.

3. After generating the rules or training, iterate through every transaction in the test dataset. For every transaction in the test dataset, we evaluate every pair of rules. For instance, if a certain transaction contains three files which have been changed to fix a vulnerability, $T_i = \{File_1, File_2, File_3\}$, then following rules are evaluated: $File_1 \rightarrow File_2$, $File_2 \rightarrow File_3$, $File_1 \rightarrow File_3$, $File_2 \rightarrow File_1$, $File_3 \rightarrow File_2$, and $File_3 \rightarrow File_1$. Then we obtain the average of the performance metrics for all queries in the test dataset.

4. Final performance metrics (i.e., Recall, Precision and Feedback) are reported by taking average over all repeated experiments.

4.4. Performance Metrics

The performance of *SecureChange* is evaluated based on the following three metrics:

- **Precision:** For a given transaction, precision refers to the percentage of rules predicted correctly out of all predictions. For instance, consider, a transaction contains two files, $T_i = \{File_1, File_2\}$. This transaction suggests two rules:

$$File_1 \rightarrow File_2 \qquad (1)$$

$$File_2 \rightarrow File_1 \qquad (2)$$

Rule 1 tells us that if $File_1$ is changed, then $File_2$ changes and the impact is bidirectional; therefore, we have the rule 2.

Assume, following rules have been predicted by *SecureChange* for $File_1$ and $File_2$:

$$File_1 \rightarrow File_2 \qquad (3)$$

$$File_1 \rightarrow File_3 \qquad (4)$$

$$File_2 \rightarrow File_4 \qquad (5)$$

The rules 3 and 4 basically tell us that with a certain support, confidence and lift, if $File_1$ is changed, then $File_2$ and $File_3$ are also changed. The rule 5 can be interpreted similarly.

A transaction containing N items or files will have N number of queries. For example, we have two queries to validate: $File_1$ and $File_2$ as in the rules 1 and 2 generated from the transaction T_i. For query 1, only the rule 3 is correct out of two predictions made for $File_1$. Therefore, the precision, in this case, is 50%. Similarly, the precision for query 2 is 0 since the prediction is wrong for $File_2$. Finally, an average precision for all queries (25%) is reported by *SecureChange*. If *SecureChange* generates no rules for a query, the precision is considered to be 100%. However, considering such queries distorts the average precision. Therefore, we did not measure precision and recall for such queries and also did not consider them in performance evaluation.

- **Recall:** For a given transaction, recall refers to the percentage of rules predicted correctly out of all ground truth rules. In the example, for query 1, the recall is 100% as one out of one possible original changes for $File_1$ has been correctly predicted. Similarly, for query 2, it is 0. Finally, an average recall (50%) is reported by *SecureChange*.

- **Feedback:** A query can be left unreported by *SecureChange*. For instance, consider the transaction consisting of two files in the above example. Assume, following rules have been predicted by *SecureChange*:

$$File_1 \rightarrow File_2 \qquad (6)$$

$$File_1 \rightarrow File_3 \qquad (7)$$

As we can see, rule 2 has not been predicted by *SecureChange*. The precision, in this case, is 100% since no false positive reported and recall is 0 since no actual change has been picked. Similarly, there can be many queries for which *SecureChange* may not respond. It would take a toll on the final average precision and recall and undermine the actual performance of the *SecureChange*. Therefore, we omit such queries in the final performance evaluation and instead incorporate another complementary performance metric,

Table 2: Performance Overview

System	Feedback	Precision	Recall
Firefox	42%	20%	24%
Android	37%	49%	56%
Tomcat	44%	62%	51%
Average	41%	44%	44%

Figure 1: Precision, Recall, and Feedback for varying support and confidence (First row showing result for Firefox, second row for Android, and third row for Tomcat)

feedback. Feedback informs us about the fraction of queries that have been responded to by *SecureChange*. It allows us to evaluate the actual performance of the *SecureChange* even in the case of inadequate training data. It also highlights the overall responsiveness of the framework. The feedback for this example is 50% since one query out of two queries has been responded.

5. Results and Implications

To answer the research question mentioned in Section 4.1, we have conducted case studies on three open-source software systems, Firefox, Tomcat, and Android Open Source Project, which publicly report vulnerabilities detected and fixed in their system. For three systems, *SecureChange* achieved approximately 41% feedback and 44% precision and recall on average as in Table 2. Different values for support, confidence, and lift parameters provide different results in each system as shown in Figure 1.

Zimmermann et al. [7] achieved 66% feedback in transactions where any kind of changes by the developers were considered. Although their feedback is better than the feedback we found for vulnerability-fixing transactions, they achieved 33% precision and 29% recall on average. The feedback found in our study can be explained from the fact that, considering all kind of changes allowed them to include more transactions in the experiments. On the other hand, security-related transactions are less frequent, and therefore, our curated security transactions are significantly less than the transactions considered in [7]. Intuitively, more the transactions, better the response rate would be.

However, the better recall and precision we obtained on a comparatively smaller set of transactions suggest that the vulnerability fixes often involve same set of files, hence they are frequently co-occurring. It implies that correlation studies on finer-granularities (e.g. statements, functions, classes etc.) is worth-exploring which could provide more precise vulnerability localization in the source code.

Figure 1 presents a comparative overview of the precision, recall, and feedback at different minimum supports and confidences for *Firefox*, *Android*, and *Tomcat* respectively. A general observation is that precision and recall tend to increase as the support and confidence increase. Feedback, on the other hand, tends to decrease as the sup-

port and confidence increase. A higher support and confidence ensures that only highly relevant rules are considered, therefore, as these parameters go up, the number of queries responded decreases for stricter filtering conditions. However, on the positive side, it only returns highly likely changes with lower false positives. Therefore, there is an obvious trade-off between the provided results and the accurate results.

Our results imply that we can effectively apply association rule mining to guide developers in secure software development.

6. Threats to Validity and Conclusion

In this study, we first analyzed vulnerability revision histories of three large systems: Android, Firefox, and Tomcat. Then we generated association rules to figure out which files are frequently changed together in a vulnerability fix. When a developer starts to fix a vulnerability by making changes in a vulnerable file, our approach will suggest relevant set of files to be modified based on the generated association rules. We found precision and recall of 44% and 44% respectively on an average for three systems.

In our experiment, a threat to internal validity could be, to what extent, the reported result can be trusted or whether it is biased. To mitigate this threat, we have introduced randomization in the experimental procedure and repeated

the experiments 100 times. On the other hand, an external threat to *SecureChange* is whether it generalizes across application domains. In order to mitigate this threat, we have performed experiments on three different kinds of open-source systems, namely a web server (Tomcat), a desktop application (Firefox), and an operating system (Android). However, it is possible that *SecureChange* might not replicate the similar performance in closed-source systems for different architecture or other open-source systems for the quality of the captured vulnerability data. Also, a reasonable amount of preexisting vulnerability data has to be available to have useful feedback from *SecureChange*.

In future, we plan to extend this research for other systems so that our proposed framework, *SecureChange*, can be used by the developers of all systems. In addition, we will work on building a plug-in of the proposed framework for different source code editors (e.g., Eclipse, Netbeans) so that developers can find it more useful for secure coding.

References

[1] T. Ball, J. min Kim, A. A. Porter, and H. P. Siy, "If your version control system could talk..." 1997.

[2] H. Gall, K. Hajek, and M. Jazayeri, "Detection of logical coupling based on product release history," in *Proceedings. International Conference on Software Maintenance*, Nov 1998, pp. 190–198.

[3] J. M. Bieman, A. A. Andrews, and H. J. Yang, "Understanding change-proneness in oo software through visualization," in *11th IEEE International Workshop on Program Comprehension*, May 2003, pp. 44–53.

[4] D. Čubranić and G. C. Murphy, "Hipikat: Recommending pertinent software development artifacts," in *Proceedings of the 25th International Conference on Software Engineering*, ser. ICSE, 2003, pp. 408–418.

[5] H. Gall, M. Jazayeri, and J. Krajewski, "Cvs release history data for detecting logical couplings," in *Sixth International Workshop on Principles of Software Evolution, 2003. Proceedings.*, Sep. 2003, pp. 13–23.

[6] T. Zimmermann, S. Diehl, and A. Zeller, "How history justifies system architecture (or not)," in *Sixth International Workshop on Principles of Software Evolution, 2003. Proceedings.*, Sep. 2003, pp. 73–83.

[7] T. Zimmermann, P. Weissgerber, A. Zeller, and S. Diehl, "Mining version histories to guide software changes," *IEEE Transactions on Software Engineering*, vol. 31, no. 6, pp. 429–445, June 2005.

[8] A. T. T. Ying, G. C. Murphy, R. Ng, and M. C. Chu-Carroll, "Predicting source code changes by mining change history," *IEEE Trans. Softw. Eng.*, vol. 30, no. 9, pp. 574–586, Sep. 2004.

[9] Z. Xing and E. Stroulia, "Data-mining in support of detecting class co-evolution," in *The 16th International Conference on Software Engineering and Knowledge Engineering*, Alberta, Canada, June 20-24, 2004, 2004, pp. 123–128.

[10] A. E. Hassan and R. C. Holt, "Predicting change propagation in software systems," in *20th IEEE International Conference on Software Maintenance, 2004. Proceedings.*, Sep. 2004, pp. 284–293.

[11] J. S. Shirabad, T. C. Lethbridge, and S. Matwin, "Supporting maintenance of legacy software with data mining techniques," in *Proceedings of the 2000 Conference of the Centre for Advanced Studies on Collaborative Research*, ser. CASCON '00, 2000, pp. 11–.

[12] ——, "Mining the maintenance history of a legacy software system," in *International Conference on Software Maintenance, 2003. ICSM 2003. Proceedings.*, Sep. 2003, pp. 95–104.

[13] ——, "Mining the software change repository of a legacy telephony," in *Proceedings of International Workshop on Mining Software Repositories (MSR '04*, 2004, pp. 53–57.

[14] A. Michail, "Data mining library reuse patterns in user-selected applications," in *14th IEEE International Conference on Automated Software Engineering*, Oct 1999, pp. 24–33.

[15] ——, "Data mining library reuse patterns using generalized association rules," in *Proceedings of the 2000 International Conference on Software Engineering.*, June 2000, pp. 167–176.

[16] R. Agrawal, R. Srikant *et al.*, "Fast algorithms for mining association rules," in *Proc. 20th int. conf. very large data bases, VLDB*, vol. 1215, 1994, pp. 487–499.

Significant API Calls in Android Malware Detection

Using Feature Selection Techniques and Correlation Based Feature Elimination

Asadullah Hill Galib

Institute of Information Technology
University of Dhaka
Dhaka, Bangladesh
bsse0712@iit.du.ac.bd

B M Mainul Hossain

Institute of Information Technology
University of Dhaka
Dhaka, Bangladesh
mainul@iit.du.ac.bd

Abstract— **Android API Calls are an important factor in differentiating malware from benign applications. Due to the increasing number of API Calls and considering computational complexity, the number of API calls in Android malware detection should be assessed and reduced without affecting detection performance. This study tries to figure out a feature reduction approach for identifying significant API Calls in Android malware detection. It incrementally analyzes various feature selection techniques to find out the minimal feature set and the most suitable technique. Also, it incorporated a correlation-based feature elimination strategy for further reduction of API Calls. Experiments on two benchmark datasets show that the Recursive Feature Elimination with Random Forest Classifier causes the minimal number of API Calls. Evaluation results indicate that the reduced set of significant API Calls (SigAPI) will perform relatively close to the full set of features in terms of accuracy, accuracy, recall, f-1 performance, AUC, and execution time. It also compares the performance with the existing malware detection works and the SigAPI outperforms most of the work regarding malware detection rate. Furthermore, it reports the top significant API Calls in malware detection. Finally, this work suggests that reduced features set of significant API Calls would be useful in classifying Android malware effectively.**

Keywords- Significant API Calls, Android Malware Detection, Feature Selection

I. Introduction

Android API (Application Programming Interface) is a series of specifications and guidelines that programs can follow to communicate with each other. Using API Calls this communication happens. APIs are growing exponentially every year [1]. Due to the wide-ranging applicability of API Calls, they are commonly used for characterizing and separating malware from benign applications.

However, the Android operating system uses a large number of API Calls and the number continues to expand. So, handling this large number of API calls in malware detection for Android is challenging. This would overfit the classifier model or complicated the classification method by providing a large number of features set. It would be useful to boost this problem by reducing features of the API Calls using feature selection techniques.

This study dealt with examining API Calls for reducing the irrelevant ones without tampering significant API Calls. It presents an approach for significant API Calls identification. Primarily, it incrementally employed several feature selection techniques. It trialed with Mutual Information Gain, Univariate ROC-AUC scores, Recursive Feature Elimination (RFE) with Gradient Boosting Classifier, and Random Forest Classifier, SelectKBest using chi scoring function, SelectFromModel using Random Forest and Extra Trees classifiers for exploring the effect of incremental feature selection. Subsequently, according to the performance evaluation, it infers a minimal range of features for different techniques and determines the best selection technique. Further, by incorporating a correlation-based feature elimination strategy, it reduces the minimal range of feature sets. Finally, the selected features are evaluated on two benchmark datasets based on five performance metrics (accuracy, precision, recall, f-1 score, AUC), execution time, and comparison with existing works.

Results show that from all the API Calls, 15-25 API Calls are significant in malware detection according to RFE with Random Forest Classifier. Evaluation illustrates that using those significant API calls, the performance is close enough to the full API Calls set. For instance, using the top 25 significant API Calls derived from the feature selection technique, the performance metrics are as follows for a particular dataset: accuracy - 97.07%, precision - 97.41%, recall - 94.60%, f-1 score - 0.960, and AUC – 0.993. Likewise, as far as the execution time is concerned, the significant API Calls take fairly less time. In comparison with existing works, this work outperforms other studies while using only a few numbers of API Calls. Also, the top significant API Calls are reported.

There were no prior works on reducing or defining significant API calls. To the best of our knowledge, this is the first study on significant API Calls in Android malware detection. The main contributions of the study are as follow:

- It proposes and assesses a feature reduction approach for identifying significant API Calls in Android Malware Detection effectively.

- It's reduced significant API Calls performs notably with respect to the full features set in terms of accuracy, precision, recall, f-1 score, AUC, and execution time. Also, it outperforms most of the existing works.

- It provides the top significant API Calls list in Android malware detection.

The rest of the paper is organized as follows. Section II presents the significant API Calls identification approach in detail. Section III gives the evaluation of the approach. Section IV gives the limitation. Section V gives related works. Section VI concludes the paper and guides future work.

II. SIGNIFICANT API CALLS IDENTIFICATION APPROACH

The overall approach of significant API Calls identification consists of five steps. The overview of the approach is depicted in Fig. 1. The details of each step are as follow:

Figure 1. *Significant API Calls Identification Approach*

A. Data Preprocessing

The dataset is preprocessed using traditional data preprocessing techniques. Other features are excluded from the dataset except for the API Calls. Subsequently, missing value treatment and label encoding are incorporated. For, incremental feature selection, the dataset is split into training and validation sets.

B. Incremental Feature Selection (IFS)

In identifying the significant API Calls, two aspects are considered. First, how many numbers of API Calls should we choose? In this regard, it is avoided to set any predefined parameters like a certain number of API Calls to be selected. Rather, it is intended to determine the optimal/minimal number of API Calls by analyzing performance metrics for different numbers of API Calls. In doing so, a feature selection technique is employed in an incremental way. For each feature selection technique, from one to the highest number of API Calls are assessed separately based on performance metrics.

Second, which feature selection technique is more suitable in reducing API Calls while maintaining the performance in Android malware detection? Again, various feature selection techniques are analyzed to figure out the most suitable technique, rather than imposing a predetermined feature selection technique. In this study, the following feature selection techniques are examined:

1. *Feature Selection using Mutual Information Gain (Entropy-Based):* Mutual Information is a non-negative value between two random variables, which measures dependency between variables. It measures the quantity of

information gained by analyzing the other random variable involving one random variable. It is equal to zero if there are two independent random variables, and higher values mean higher dependence. The function is based on nonparametric methods based on entropy estimation of the distances from k-nearest neighbors as defined in [2] and [3].

2. *Feature Selection Based on Univariate ROC-AUC Score*: A ROC curve (receiver operating characteristic curve) is a graph representing a classification model output at all classification thresholds. This curve maps two parameters: True Positive Rate and False Positive Rate. AUC stands for "Area under the ROC Curve," meaning that AUC measures the whole two-dimensional space under the ROC Curve. The region under the curve (AUC) is proportional to the probability that a classifier ranks a randomly selected positive instance higher than a randomly selected negative one by using normalized units [4]. Univariate ROC-AUC involves the analysis of a single variable. An AUC equal to 0.5 corresponds to a type of random classification. For a model to be acceptable AUC will be greater than 0.5.

3. *Feature Selection using Recursive Feature Elimination (RFE):* The goal of recursive feature elimination (RFE) is to pick features by recursively considering smaller and smaller sets of features, given an external estimator that assigns weights to features. First, the estimator is trained on the initial collection of features and the importance of each function is obtained. The least significant characteristics are then pruned from the present range of characteristics. The process is repeated recursively on the pruned collection before finally achieving the required number of features to be chosen [5]. In this work, two classifiers are used as the estimators of the RFE.

   3.1. *RFE with Gradient Boosting Classifier*: As the base estimator of the RFE, Gradient Boosting Classifier is employed. Gradient Boosting Classifier builds an additive model in forward-stage-wise fashion; enables arbitrary differentiable loss functions to be optimized. Regression trees are fit on the negative gradient of the function of binomial or multinomial loss of deviance in each point [6].

   3.2. *RFE with Random Forest Classifier*: Random Forest Classifier is also used as the base estimator of the RFE. It is a meta-estimator that fits multiple decision tree classifiers on various dataset sub-samples and uses an average to improve predictive [7].

4. Feature Selection using SelectKBest with chi2: SelectKBest scores the features according to the k highest scores. It takes a score function as a parameter, which would be specific to a pair. The score function retains the features of the first k with the highest scores [8]. In this study, the chi2 scoring function is employed. This scoring function computes the chi-squared stats between each non-negative feature and class scores accordingly. It tests for which the distribution of the test statistic approaches the $\chi2$ (Chi-Squared) distribution asymptotically [9].

5. Feature Selection using SelectFromModel (Tree-Based): SelectFromModel is a meta-transformer that can be used

along with any tree-based estimator. It calculates the feature importance of each feature according to fitting the estimator into the data. Based on the feature importance it selects the top N features, where N is predefined [10]. Tree-based estimators are used here as it can classify the significant features by selecting the classification features on the basis of how well they boost the node's purity [11]. In this case, every possible value of N is evaluated. Also, two tree-based estimators are incorporated here: Random Forest Classifier and Extra Trees Classifier.

C. Determining Feature Selection Technique using the Minimal Range of Features

After implementing the incremental feature selection using different feature selection techniques, analysis of performance metrics is carried out to identify the minimal range of features. The minimal range of features implies a range of features from which segment the performances of Android malware detection are not increased significantly with respect to the increase of features. In other words, before the minimal range, the performances are increased. But, after the minimal range, the performances are quite unchanged with the increase in the number of features. To draw a conclusion from the analysis, a self-explanatory plot is generated using the performance metrics (accuracy, precision, recall, f-1 score) with respect to the increasing number of features. According to the plots for different techniques, the minimal range of features are deduced.

These minimal ranges are conducive to determine the best feature selection technique. The feature selection technique with the lowest minimal range is carefully chosen for identifying significant API Calls in Android Malware Detection.

D. Correlation-based Feature Elimination

After selecting the important features using the suitable feature selection technique, a final feature elimination strategy is performed for further reduction of API Calls without affecting the performances notably. Here, a correlation-based feature elimination strategy is applied.

A pair-wise Pearson correlation coefficient is calculated for all pairs of important API Calls. It is a measure of the linear correlation between two variables X and Y. It is calculated using the following equation [12]:

$$\rho_{xy} = \frac{Cov(x,y)}{\sigma_x \, \sigma_y} \tag{1}$$

Where,

ρ_{xy} = Pearson correlation coefficient

Cov (x,y) = covariance of variable x and y

σ_x = standard deviation of x

σ_y = standard deviation of y

Then all the pairs with a Pearson correlation coefficient greater than 0.85 are filtered out for the feature elimination process. As the two features in each pair are highly correlated, so removing one of them would not affect the classification performances.

The elimination strategy here is to remove the less important feature from each pair. To measure the relative importance of the features, a tree-based estimator – Random Forest Classifier is used. According to the relative feature importance, the more important feature in each pair is intact, and the less important feature is eliminated.

E. Evaluating Significant API Calls

Finally, the reduced set of API Calls are evaluated according to five performance metrics – accuracy, precision, recall, f-1 score, AUC. Here, the minimal range of features derived from the best feature selection technique is assessed. In the final assessment, the Random Forest classifier is trained and evaluated with 10-fold cross-validation. Along with the five-performance metrics, execution time, and comparison with the existing approach are also evaluated. Also, significant API Calls are determined and reported.

III. EVALUATION

In the evaluation of this study, two benchmark datasets are used. The experimental results are analyzed based on five performance metrics - accuracy, precision, recall, f-1 score, and AUC. These metrics are widely used in performance measure of Android malware detection. Besides, the execution time of detection is considered for evaluation. Three research questions are being answered here regarding significant API Calls in malware detection.

A. Dataset

In this study, the Drebin [13] and the Android Malware Genome Project [14] datasets are used. The datasets are used separately to ensure the applicability and generalizability of the approach.

The Drebin dataset contains 5,560 malware applications from 179 different malware families. Also, 9470 benign applications derived from the Google Play Store are incorporated here for classifying the malware properly.

The Android Malware Genome Project dataset contains 1,200 malware samples that cover most existing Android malware families. Here, 2539 benign applications derived from the Google Play Store are incorporated. In the rest of the paper, this dataset is referred to as Malgenome.

Only the API Calls are considered in this work. In total, 73 API Calls are found in the Drebin dataset and 69 API Calls are found in the Android Malware Genome Project dataset.

B. RQ1: How can we sort out the significant API Calls (features) in Android malware detection?

According to the different incremental feature selection techniques, the minimal ranges are analyzed to determine the best feature selection technique in identifying significant API Calls. The near minimal ranges are as depicted in Table I.

From the experiments, the best feature selection technique for API Calls reduction is Recursive Feature Elimination (RFE) with Random Forest Classifier as it has the lowest minimal range among other techniques for both datasets (see Table I).

TABLE I.   MINIMAL RANGE OF FEATURES FOR DIFFERENT FEATURE SELECTION TECHNIQUES

Feature Selection Technique	Minimal Range for Drebin	Minimal Range for Malgenome
Mutual Information Gain	37-42	23-28
Univariate ROC-AUC Score	35-38	25-30
RFE with Gradient Boosting Classifier	23-28	20-25
RFE with Random Forest Classifier	18-25	18-21
SelectKBest with chi2	47-50	30-33
SelectFromModel with Random Forest Classifier	25-30	17-22
SelectFromModel with Extra Trees Classifier	28-33	20-23

The minimal range of features using RFE with Random Forest Classifier for the Drebin dataset can be deduced from Fig.2 and Fig. 3. According to Fig. 2., with the number of features, are increased, the performance metrics are also increased initially. However, in the range between 18-25 features (approximately), the performance metrics are going to be stable and remain constant (the lines are almost horizontal) for the following selected features. So, it can be inferred that by taking those 18-25 features, the performances are close enough to the actual performances of all the 73 features. In the next research question, this minimal range of features are evaluated.

Figure 2.   Recursive Feature Elimination (RFE) using Random Forest Classifier (Performance Metrics vs Number of Features)

Before proceeding to the performance evaluation, the correlation-based feature elimination (CFE) is performed on the minimal range of features. And, experiments show that

CFE can reduce the features as depicted in Table II. For instance, CFE can reduce 18 features to 15 and 16 features respectively for the Drebin and Malgenome datasets.

TABLE II.   CORRELATION-BASED FEATURE ELIMINATION (CFE) ON THE MINIMAL FEATURE SETS

Number of API Calls		
Minimal Feature Sets	With CFE for Drebin	With CFE for Malgenome
18	15	16
19	15	17
20	17	18
21	19	19
22	20	20
23	21	21
24	21	21
25	22	22

C. RQ2: How do the significant API Calls perform in detecting Android Malware?

In this research question, the reduced set of Significant API Calls (SigAPI) are evaluated based on the performance metrics, execution time, and comparison with existing works.

1) Performance Evaluation of the SigAPI

The performance evaluation for the significant API Calls is based on five metrics. The evaluation is described in Table III and Table IV for the number of API Calls– 15, 17, 19, 21, 23, 25, and all API Calls to sidestep redundancy.

Table III shows that for the Drebin dataset, the performance metrics using significant API Calls are close to the performance metrics of using all the API Calls (73).

Table IV also shows that for the Malgenome dataset, the significant API Calls performs almost identically to the full feature set of API Calls (69).

Specifically, how many significant API calls should be selected? - It depends on the requirement of the stakeholders. However, it is suggested to use the range of 15-25 significant API Calls based on the prerequisite of performance metrics.

TABLE III.   PERFORMANCE EVALUATION OF THE SIGNIFICANT API CALLS (DREBIN)

Feature Selection Technique	# of API Calls	Acc (%)	Pre (%)	Rec (%)	F-1	AUC
All Features	73	98.32	98.62	96.17	0.974	0.996
RFE with Random Forest Classifier	25	97.07	97.41	94.60	0.960	0.993
	23	96.69	96.92	94.06	0.955	0.993
	21	96.26	96.64	93.15	0.949	0.992
	19	96.17	96.26	93.27	0.947	0.991
	17	95.62	95.58	92.43	0.940	0.988
	15	95.38	96.03	91.29	0.936	0.986

TABLE IV.   PERFORMANCE EVALUATION OF THE SIGNIFICANT API CALLS (MALGENOME)

Feature Selection Technique	# of API Calls	Acc (%)	Pre (%)	Rec (%)	F-1	AUC
All Features	69	98.71	98.87	97.22	0.980	0.998
RFE with Random Forest Clas-sifier	25	98.12	98.16	96.19	0.972	0.998
	23	98.03	98.15	95.87	0.970	0.998
	21	98.10	98.07	96.10	0.971	0.996
	19	96.16	97.92	96.51	0.972	0.996
	17	97.97	97.82	96.03	0.969	0.995
	15	97.26	97.62	94.05	0.958	0.993

2)   Execution Time of the SigAPI

Table V shows the comparative execution time of malware detection. The result shows that using the significant API Calls (ranges between 15-25), the execution time of the malware detection is considerably lower than using all the features. For large data sets, this time would be substantially higher.

TABLE V.   EXECUTION TIME OF THE SIGNIFICANT API CALLS

# of API Calls	Execution Time (s) for Drebin	Execution Time (s) for Malgenome
All	6.48	5.76
25	4.15	4.02
23	4.11	3.98
21	4.11	3.88
19	3.95	3.85
17	3.91	3.78
15	3.90	3.74

3)   Comparison with Existing Works

Table VI shows the comparative analysis of the detection rate using significant 20 API Calls – SigAPI (20) for the Drebin dataset with respect to some existing works on Android malware detection. The result shows that SigAPI (20) outperformed all the existing works except two.

TABLE VI.   COMPARISON WITH THE EXISTING WORKS

Works	Detection Rate (%)
SigAPI (20)	96.30
Drebin [10]	93.90
SigPID [15]	93.62
Yerima et al. [11]	92.1%
Yerima et al. [12]	97.5%
Peiravian et al. [13]	95.75%
DroidAPIMiner [14]	~99%
Altaher et al. [16]	91%

A. RQ3: Which API Calls are Significant in Android Malware Detection?

Table VII shows the top 25 significant API Calls in Malware Detection for the Drebin dataset. These API Calls are derived from the feature selection technique – RFE with Random Forest Classifier. Also, these 25 API Calls are almost identical to the Malgenome dataset except 3 API Calls. More data instances would be conducive to generating identical API Calls. Yet, as this study primarily suggests a feature reduction approach for significant API Calls, the dataset to dataset it may slightly vary due to the inconsistency and time period of datasets.

TABLE VII.   TOP 25 SIGNIFICANT API CALLS (DREBIN)

TOP 25 SIGNIFICANT API CALLS (DREBIN)	
transact	ClassLoader
onServiceConnected	Landroid.content.Context.registerReceiver
bindService	Ljava.lang.Class.getField
attachInterface	android.content.pm.PackageInfo
ServiceConnection	TelephonyManager.getLine1Number
android.os.Binder	Ljava.lang.Class.getMethod
Ljava.lang.Class.getCanonicalName	android.telephony.gsm.SmsManager
Ljava.lang.Class.getMethods	TelephonyManager.getSubscriberId
Ljava.lang.Class.cast	Ljava.lang.Object.getClass
Ljava.net.URLDecoder	TelephonyManager.getDeviceId
android.content.pm.Signature	HttpUriRequest
android.telephony.SmsManager	Runtime.exec

IV.   LIMITATION

In this study, only the Drebin and Malgenome datasets have been analyzed, which is subject to bias and lack of generalizability, threatening external validity. In terms of threats to internal validity, the parameters of different techniques and algorithms, execution time measurement are susceptible to bias and can be examined differently by different analysts and machines.

V.   RELATED WORK

Several works dealt with API Calls in Android malware detection. For instance, Drebin incorporated API Calls with other features and obtained an accuracy of 93.90% in malware detection [13]. Yerima et al. combined API Calls and Permissions with Bayesian Classifier and attained 92.1% accuracy [15]. In another work, they achieved an accuracy of 97.5% and an AUC of 0.953 by using a composite parallel classifier approach [16]. Using API Calls modeled with SVM (Support Vector Machine), Peiravian et al. gained an accuracy of 95.75% and an AUC of 0.957 [17]. Likewise, DroidAPIMiner integrated API level features and reached an accuracy as high as 99% using the KNN classifier [18].

Though a handful number of works employed API Calls, none dealt with reducing API Calls or identifying important API Calls. However, feature reduction technique is applied in Permission features previously. Li et al. successfully reduced 135 Permission features to 22 features. They used Permission ranking with negative rate, support based Permission ranking, and Permission mining with association rules for feature selection. Their reduced Permission features have higher recall value, close enough accuracy value with the full features set. But their precision was lower and false positive rate (FPR) was higher significantly with respect to all Permission features [19].

Altaher et al. proposed an approach based on ANFIS with fuzzy c-means clustering using significant application permissions. Their classification accuracy was 91%, with the lowest false positive and false negative rates of 0.5% and 0.4%, respectively [20].

Wang et al. evaluated individual permissions and collective permissions and implemented three measures of scoring on the permission features. They discovered dangerous permission subsets using Sequential Forward Selection (SFS) and Principal Component Analysis (PCA). They got a 94.62% detection rate [21].

To the best of our knowledge, there is no such work on feature reduction of API Calls in Android malware detection.

VI. Conclusion

In this study, a feature reduction approach is proposed for identifying significant API Calls in Android Malware detection. Evaluation of the significant API Calls shows that API Calls can be reduced without affecting performance so much. Therefore, reduced features set of significant API Calls would be convenient in classifying Android malware considering performance and computational complexity.

In the future, the approach will be evaluated using different and large datasets. Also, other feature selection techniques using deep learning, ensemble learning, etc. will be employed.

References

[1] J. Fernando, "What is an API ? How to call an API from Android ?," DroidMentor, 12-Oct-2016. [Online]. Available: https://droidmentor.com/api-call-api-android/. [Accessed: 11-Jan-2020].

[2] A. Kraskov, H. Stögbauer, and P. Grassberger, "Erratum: Estimating mutual information [Phys. Rev. E69, 066138 (2004)]," Physical Review E, vol. 83, no. 1, 2011.

[3] B. C. Ross, "Mutual Information between Discrete and Continuous Data Sets," PLoS ONE, vol. 9, no. 2, 2014.

[4] T. Fawcett, "An introduction to ROC analysis," Pattern Recognition Letters, vol. 27, no. 8, pp. 861–874, 2006.

[5] "sklearn.feature_selection.RFE¶," scikit. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.RFE.html. [Accessed: 11-Feb-2020].

[6] M. B. Fraj, "In Depth: Parameter tuning for Gradient Boosting," Medium, 24-Dec-2017. [Online]. Available: https://medium.com/all-things-ai/in-depth-parameter-tuning-for-gradient-boosting-3363992e9bae. [Accessed: 03-Mar-2020].

[7] Ho, Tin Kam. "Random decision forests." In Proceedings of 3rd international conference on document analysis and recognition, vol. 1, pp. 278-282. IEEE, 1995.

[8] "sklearn.feature_selection.SelectKBest¶," scikit. [Online]. Available: http://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.SelectKBest.html. [Accessed: 11-Jan-2020].

[9] Pearson K. X. On the criterion that a given system of deviations from the probable in the case of a correlated system of variables is such that it can be reasonably supposed to have arisen from random sampling. The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science. 1900 Jul 1;50(302):157-75.

[10] "1.13. Feature selection¶," scikit. [Online]. Available: https://scikit-learn.org/stable/modules/feature_selection.html. [Accessed: 01-Mar-2020].

[11] "Using Scikit-Learn in python for feature selection," Data Science Beginners, 26-Nov-2018. [Online]. Available: https://datasciencebeginners.com/2018/11/26/using-scikit-learn-in-python-for-feature-selection/. [Accessed: 01-Mar-2020].

[12] "Basic Concepts of Correlation," Real Statistics Using Excel. [Online]. Available: http://www.real-statistics.com/correlation/basic-concepts-correlation/. [Accessed: 04-Mar-2020].

[13] D. Arp, M. Spreitzenbarth, M. Hübner, H. Gascon, and K. Rieck, "Drebin: Effective and Explainable Detection of Android Malware in Your Pocket," Proceedings 2014 Network and Distributed System Security Symposium, 2014.

[14] Yajin Zhou, Xuxian Jiang, "Dissecting Android Malware: Characterization and Evolution," Proceedings of the 33rd IEEE Symposium on Security and Privacy (Oakland 2012), San Francisco, CA, May 2012

[15] Yerima, S. Y., Sezer, S., McWilliams, G., & Muttik, I. (2013, March). A new android malware detection approach using bayesian classification. In 2013 IEEE 27th international conference on advanced information networking and applications (AINA) (pp. 121-128). IEEE.

[16] Peiravian, N., & Zhu, X. (2013, November). Machine learning for android malware detection using permission and api calls. In 2013 IEEE 25th international conference on tools with artificial intelligence (pp. 300-305). IEEE.

[17] Yerima, S. Y., Sezer, S., & Muttik, I. (2014, September). Android malware detection using parallel machine learning classifiers. In 2014 Eighth International Conference on Next Generation Mobile Apps, Services and Technologies (pp. 37-42). IEEE.

[18] Aafer, Y., Du, W., & Yin, H. (2013, September). Droidapiminer: Mining api-level features for robust malware detection in android. In International conference on security and privacy in communication systems (pp. 86-103). Springer, Cham.

[19] Li, J., Sun, L., Yan, Q., Li, Z., Srisa-an, W., & Ye, H. (2018). Significant permission identification for machine-learning-based android malware detection. IEEE Transactions on Industrial Informatics, 14(7), 3216-3225.

[20] Altaher, A., & BaRukab, O. (2017). Android malware classification based on ANFIS with fuzzy c-means clustering using significant application permissions. Turkish Journal of Electrical Engineering & Computer Sciences, 25(3), 2232-2242.

[21] Wang, W., Wang, X., Feng, D., Liu, J., Han, Z., & Zhang, X. (2014). Exploring permission-induced risk in android applications for malicious application detection. IEEE Transactions on Information Forensics and Security, 9(11), 1869-1882

Threat Intelligence Relationship Extraction Based on Distant Supervision and Reinforcement Learning

Xuren Wang, Jie Yang
Information Engineering College
Capital Normal University
Beijing, China
wangxuren@cnu.edu.cn

Qiuyun Wang, Changxin Su
Key Laboratory of Network Assessment Technology
Institute of Information Engineering
Chinese Academy of Sciences
Beijing, China

*Abstract—*In recent years, threat intelligence has become a new hotspot in cybersecurity. It analyzes and predicts attacks that have occurred and have not occurred, and plays an important role in building an efficient defense system. Traditional threat intelligence relies on a manual collection and its efficiency is relatively low. Therefore, the efficient sharing of threat intelligence has important research value. For the information extraction technology of threat intelligence, we focus on the construction of threat intelligence labeling data sets and the extraction technology of threat intelligence relationship. The specific content and research results include two aspects: (1) Research on the construction of threat intelligence information extraction data set. The threat intelligence extraction data set is constructed by a distantly supervised labeling method. In this paper, more than 900 threat intelligence reports are used as a corpus. We finally obtain a relation extraction data set containing 10,000 sentence instances of 30 relationships. (2) Research on the extraction of threat intelligence relationships. To mitigate noise labeling data in the relation extraction data, we propose a distant supervision relationship extraction method based on DRL-ET-PCNN-ATT (Deep Reinforcement Learning Entity Type Piecewise Convolution Neural Network-Attention) based on the PCNN-ATT (Piecewise Convolution Neural Network-Attention) model. The experimental results show that compared with the CNN (Convolution Neural Network), PCNN (Piecewise Convolution Neural Network), RL-CNN (Reinforcement Learning Convolution Neural Network) models, the accuracy of the extraction model used in this paper has increased by 16.77%, 5.88%, and 4.97%, and the recall rate has increased by 16.39%, 2.83%, and 4.49%.

Keywords-threat intelligence; relationship extraction; distant supervision; reinforcement learning

I. INTRODUCTION

Threat intelligence sharing research faces two major challenges: First, when there is a large amount of threat intelligence report, it is very inefficient to rely solely on manual analysis and sharing of critical information. It is impossible to synchronize and share real-time threat intelligence on time, resulting in Threat information lags. Second, unlike the natural language processing corpus in the general domain, the tagging corpus in the field of threat intelligence is scarce, which makes research on threat intelligence extraction very difficult. Therefore, information extraction on threat intelligence has important practical significance and application value.

In summary, we make contributions in this work include: (1) Constructing a threat intelligence extraction data set through a distantly supervised labeling method; (2) We evaluate our model and achieve the best result compared with several state-of-the-art relation extraction models.

II. RELATED WORK

In recent years, there has been an increasing amount of literature on threat intelligence data, and the dataset for threat intelligence is increasing. Varish Mulwad et al. proposed a framework to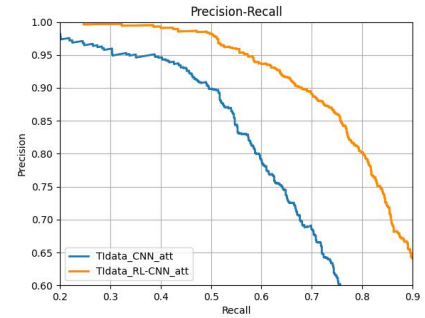 extract vulnerability and attack information from web text, and generate machine-understandable languages. The data set was from 107 vulnerability description documents and was not publicly available [1]; Nikki McNeil et al. proposed a new entity extraction guidance algorithm PACE is used to extract valuable network security concepts. The data set is manually annotated 10 documents from online open source websites with a total of seven entity types [2]; Corinne L. Jones and others proposed a bootstrapping algorithm to extract security entities and their relationships from the text. The dataset is a corpus of 62 documents made from various security-related websites. The dataset is not open source [3]; Arnav Joshi et al. The research of linked data completed an experimental data set through professional annotations. The training set consists of 3800 entities and 38,000 instances. The test set consists of 1200 entities and 9,000 instances. The dataset is not public [4]; Ravendar Lal et al. Researched extracting secure entities and concepts from unstructured text, and they built datasets from more than 100 select reports After screening and the fact that sampling CVE eventually got 60, 12 and 12 Dobe Microsoft bulletin announcement of the composition of the data set, it is not open to the public [5]. In summary, threat intelligence related datasets are very rare and most of them are not public. Hence, we propose an annotation method based on distant supervision to help security analysts to label OSINT data more quickly and efficiently. Then we propose the relationship extraction method combined with reinforcement learning to research threat intelligence information extraction on this data set.

III. DATASET

After distant supervision labeling and manual verification, the label definition and quantity distribution for each relationship are shown in Table 1.

The final threat intelligence relationship extraction data set contains 10,000 sentence examples of 30 types of relationships.

TABLE I RELATIONSHIP LABEL DEFINITION AND QUANTITY DISTRIBUTION

Head entity	Relation	Tail entity	Relation number
Hacker group	Background	Region	386
Hacker group	Target	Region	1155
Hacker group	Target	Industry	1218
Hacker group	Target	Organization	257
Hacker group	Target	User	179
Hacker group	Attack	Way	759
Hacker group	Use	Tool	1227
Hacker group	Use	Loophole	97
Hacker group	Oldest active	Time	167
Hacker group	First found	Time	103
Hacker group	Attack	Time	458
Hacker group	Attack	Purpose	325
Hacker group	Have	Alias	153
Hacker group	Launch	Attack action	238
Hacker group	Use	Tool	146
Hacker group	Attack	Purpose	91
Sample file	Generate	Time	109
Sample file	Use	Loophole	96
Sample file	Have	File type	85
Sample file	Propagation	Way	444
Sample file	Have	Features	238
Sample file	Target	Region	91
Sample file	Target	Industry	295
Sample file	Related	Sample file	111
Sample file	Have	Alias	222
Security Team	Found	Sample file	112
Security Team	Found	Attack activity	248
Security Team	Release	Time	123
Security Team	Found	Hacker group	115
Offensive action	Attack	Time	752

IV. THREAT INTELLIGENCE RELATIONSHIP EXTRACTION FRAMEWORK

Aiming at the complicated threat data in a large number of threat intelligence reports, as shown in section III, the entity-relationship is marked based on the method of distant supervision, which solves the problem of marking threat intelligence data. However, this method generally classifies sentences at the sentence set level, and cannot map relationships to sentences one by one. The main reason for this problem is the noisy data in the distantly supervised labeled data set, which has a great effect on relationship extraction great influence. To solve this problem, based on the distant supervised model PCNN, we propose a distant supervised extraction model based on DRL-ET-PCNN-ATT. The model is shown in Fig. 1, which is mainly composed of the input vector layer and piecewise convolution neural network and sentence instance selector. The model first inputs three layers of feature vectors, including pre-trained word vectors, the vector of the relative position between each word and the entity, and the entity type vector; the next step is the piecewise convolutional neural network to extract the context information related to the entity, and add the attention mechanism to the sentence vector, and get the classification result of the relationship label. To alleviate the problem of noisy sentences, we introduce a sentence selector based on reinforcement learning.

Figure 1. DRL-ET-PCNN-ATT relationship extraction model diagram

A. Input Vector Layer

- In this paper, before inputting into the neural network layer, it is necessary to characterize the word vector and obtain the context relationship between words. Here, the word2vec word vector language model is used to convert each word in the corpus into a d-dimensional vector. Thus, we get a single vector representation of each word.

- In the input vector feature, to highlight the relative position of the entity in the sentence and make full use of the position information in the sentence, this article adds the vector feature of the entity position and uses the relative position of each word and the entity position in the sentence as an important feature input. Here, position embedding proposed by Zeng [6] are used. As shown in Fig. 2, the relative distance between each word and the entities E_1 and E_2 in the sentence is stitched together as the position vector feature.

Figure 2. Example of the position feature vector

- Due to the difference in the order of magnitude of each entity type in the threat intelligence data set, consider adding entity type features based on a common model. First label the text with a BIO labeling scheme, that is, label each element as "BX", "IX" or "O". Among them, "BX" indicates that the fragment in which this element is located belongs to type X and this element is at the beginning of this fragment, "IX" indicates that the fragment in which this element exists belongs to type X and that the element is in the middle position of this fragment, and "O" indicates that it does not belong to any type, and then quantify the labeled entities and words to give them corresponding features, and then stitch them together with word features and location features as input features of the convolutional neural network. That is, if the dimension of the word vector is, the dimension of the position vector is, the dimension of the entity type feature vector is, and the dimension of the input vector layer is d:

$$d = d_w + 2 * d_p + d_e \qquad (1)$$

B. Convolutional neural network layer And Attention layer

The construction process of the convolutional neural network layer and attention layer uses the baseline model proposed by Lin et al. [7].

C. Sentence selector based on reinforcement learning

Reinforcement learning is an area in machine learning that emphasizes how to act based on the environment to achieve the maximum expected benefits. The problem that reinforcement learning solves is to get an optimal action for a specific problem so that the reward obtained under this strategy is the largest.

Definition of the problem to be solved by the instance selector: given a set [sentences, relationship labels], expressed as $X = \{(x_1, r_1), (x_2, r_2), \dots, (x_n, r_n)\}$, X include the noise annotation generated by the distant supervision method, and the task of the selector is to determine which sentence correctly describes the relationship, then select the sentence and hand it to the convolutional neural network classifier.

According to the task requirements of this problem, we construct a reinforcement learning selector for relation extraction tasks. As shown in Fig. 1, the state, action, and reward are defined as follows:

- The state contains the current sentence, selected sentences, and entities. The author uses a continuous function $\emptyset(s_i)$ to represent the state, which will output a vector. Among them, the vector representation of the current sentence is obtained from the non-linear layer of the PCNN used for relation classification; the vector representation of the selected sentence set is the average of each sentence vector; the vector representation of a pair of entities is pre-trained word vector.

- The value of action is {0, 1}, indicating whether to select the current sentence. The a_i obtained according to the policy function $\pi_\theta(s_i, a_i)$ where θ is the parameter to be learned. The following logical function is used here as the policy function definition, where $\emptyset(s_i)$ is the state feature mentioned earlier.

$$\pi_\theta(s_i, a_i) = P_\theta(a_i, s_i) = a_i \sigma(W * \emptyset(s_i) + b) + (1 - a_i)(1 - \sigma(W * \emptyset(s_i) + b))$$

$$(2)$$

- The reward is a quality representation of the selected sentence. When a round of sentences is selected, there will be final feedback, that is, final feedback is set at the final state. The definition of the feedback function is as follows, where Q is the selected sentence set, which is a subset of state, r represents the relationship label of the current sentence, and $p(r|s_j)$ is the label probability output by the relationship classifier.

$$r(s_i|Q) = \begin{cases} 0, & i < |Q+1| \\ \frac{1}{Q}\Sigma_{S_j \in Q} logp(r|s_j), & i = |Q+1| \end{cases}$$

$$(3)$$

- The optimization function of sentence selector for maximizing feedback is defined as:

$$J(\theta) = V_\theta(s_0|Q) = E_{s_0, a_0, s_1, a_1 \dots s_i, a_i, \dots}[\Sigma_{i=0}^{|Q|+1} r(si|Q)]$$

$$(4)$$

- According to Actor-critic algorithms [8], we add value function Q_ω after state and reward calculations to reduce the error of the policy function. The value function is defined as follows:

$$Q_\omega(s,a) = \emptyset(s_i)^T \omega \qquad (5)$$

- Where $\emptyset(s_i)$ is the initial state vector, $\emptyset(s_i')$ is the state vector after the sentence is selected, input these two vectors to the value function to get the Q value output $Q_\omega(s_i)$ and $Q_\omega(s_i')$, the TD error δ is used as the parameter update error of the policy function and value function, γ is the decay.

$$\delta = r + \gamma Q_\omega(s_i') - Q_\omega(s_i) \qquad (6)$$

- The parameter ω of the value function is updated as follows, β is the training step.

$$\omega = \omega + \beta \delta \emptyset(s_i) \qquad (7)$$

- The parameter θ of the policy function is updated as follows, α is the training step.

$$\theta = \theta + \alpha \Sigma_{i=1}^{|Q|} \nabla_\theta log\pi_\theta(s_i, a_i)\delta \qquad (8)$$

V. EXPERIMENTS

In this section, we evaluated the model on the threat intelligence data set constructed in part III. We first introduce the experiment dataset and parameter settings. To verify the advantages of this model, we conducted experiments on CNN, PCNN, and RL-CNN separately. The experimental results show that the model used in this paper has a higher accuracy of extracting threat intelligence relationships than other models, and gives this comparison of experimental results and PR (precision-recall) curves of these four models.

A. Dataset

As shown in the third part, 10000 sentence instances containing 30 relationships are established, and the data set is randomly divided into a training set of 90% and a test set of 10%, that is, the training data contains 9,000 sentence examples, and the test data contains 1000 sentence examples. Relationship extraction usually has three types of evaluation indicators: precision, recall, and F1 measure. We will use

these three indicators to compare the performance of our model with the baseline extraction model.

B. Parameter Settings

Some key parameters need to be set during model training. For the parameter settings of the relational classifier part, the word vector dimension is set to 50, the position vector dimension is set to 5, and the entity type feature vector dimension is set to 3. In the convolutional neural network layer, the size of the convolution window is set to 3. The number of neurons in the hidden layer of the convolutional layer is set to 230. In the instance selector section, set batch size 40 and learning rate 0.1. To alleviate the problem of overfitting the model, we add a dropout unit.

C. Experiment Results

To verify the effectiveness of the DRL-ET-PCNN-ATT based relationship extraction model used in the extraction of threat intelligence relationships, this paper compares this with general models CNN [9], PCNN [6], and RL-CNN [10]. Second, we also compare the processing of sentence information in the package. There are 4 ways, namely ATT, AVE, ONE, CROSS_MAX, and AVE. All the sentence weights in a package are regarded as the same, that is, the vector is taken. The average value; ONE takes the sentence instance vector with the highest confidence in the bag as the input calculation; CROSS_MAX [11] performs an instance-max-pooling operation on all sentence vectors inside the bag. The triples and sentences are converted into a dictionary format and input to the above model for training and testing. The accuracy, recall, and F1 values are shown in Table 2. By analyzing the experimental results in Table 2, we can see the advantages of the model used in this article. The DRL-ET-PCNN-ATT model has the highest accuracy rate, reaching 92.31%, and the recall rate is 83.24%. The ATT method is also used in the package example. Compared with the CNN / PCNN / RL-CNN model in the field of relation extraction, the accuracy rate has increased by 16.77%, 5.88%, and 4.97%, and the recall rate has increased by 16.39%, 2.83%, and 4.49%.

The precision/recall curves for each method are shown in Fig. 3. By analyzing the PR curve of Fig. 3 (a)(b), the extraction effect of DRL-ET-PCNN-ATT is significantly better than PCNN-ATT, and the extraction effect of RL-CNN-ATT is significantly better than CNN-ATT. The processor eliminates some noise data and improves the accuracy of relation extraction. From Fig. 3 (c), the extraction effect of PCNN is better than the CNN model, and we can see the advantage of the segmented pooling method in the extraction effect. From Fig. 3 (d), it can be seen that the advantages of DRL-ET-PCNN-ATT for the other three models reflect the advantages of adding entity type features and combining the PCNN model with reinforcement learning, making full use of the distribution characteristics of entity type threat intelligence data and the advantages of joint training. The combination of the two greatly improves the accuracy of relation extraction, as shown in Fig. 3 (e), the extraction performance comparison of the four bag instance processing methods on the DRL-ET-PCNN model shows that the ATT method is the most suitable for the model used in this paper, maximizing the extraction accuracy.

TABLE II. EXPERIMENTAL RESULTS OF EACH MODEL ON FOUR BAG PROCESSING METHODS

Model	Bag Way	ACCURACY	AUC	F1
CNN	ATT	0.7554	0.6685	0.7092
	AVE	0.7585	0.6745	0.7140
	ONE	0.7773	0.6723	0.7210
	CROSS_MAX	0.7626	0.6854	0.7219
PCNN	ATT	0.8643	0.8041	0.8331
	AVE	0.8639	0.7843	0.8222
	ONE	0.8723	0.7587	0.8115
	CROSS_MAX	0.8745	0.7743	0.8213
RL-CNN	ATT	0.8734	0.7875	0.8282
	AVE	0.8830	0.8047	0.8420
	ONE	0.8942	0.8102	0.8501
	CROSS_MAX	0.8864	0.7957	0.8386
DRL-ET-PCNN	ATT	**0.9231**	**0.8324**	**0.8754**
	AVE	0.8943	0.8075	0.8487
	ONE	0.9018	0.8186	0.8582
	CROSS_MAX	0.9113	0.8265	0.8679

(a)

(b)

(c)

(d)

(e)

Figure 3. Comparison of Experimental P-R Curves of Threat Intelligence Relation Extraction Data Sets on Different Models

VI. CONCLUSION AND FUTURE WORK

In this paper, we use the knowledge base to distantly supervise structured threat intelligence data to construct a relationship extraction dataset and compare the number of sentences and relationship types with the classic data set in the field of relationship extraction. In the research of the construction of the threat intelligence relationship extraction model, based on the PCNN-ATT model, we propose a distant supervision relationship extraction method based on DRL-ET-PCNN-ATT, The extraction accuracy is significantly improved.

In the future, we will explore the following directions:

The dataset used in this paper is extracted from unstructured text. It is limited to text corpora such as hacker organizations, security teams, and sample files. It ignores charts in threat intelligence reports, threat information in pictures. In future research, we can consider building a set of report pre-processing process and image recognition model, which can extract this non-text information and enrich the shared information of threat intelligence.

In relation extraction, the assumption in distant supervision is too positive, and it is inevitable to introduce a lot of noise data. To alleviate the problem of mislabeling, at present, the typical model of entity-relationship extraction is PCNN-ATT, but it mainly uses the semantic information of the sentence and does not involve grammatical information. Therefore, how to effectively fuse the semantic and sentence grammatical information to extract entity relationships is also one of the main directions to optimize the extraction model in future work.

REFERENCES

[1] Mulwad V, Li W, Joshi A, et al, "Extracting information about security vulnerabilities from web text". 2011 IEEE/WIC/ACM International Conferences on Web Intelligence and Intelligent Agent Technology. USA: IEEE, 2011, pp. 257-260.

[2] McNeil N, Bridges R A, Iannacone M D, et al, "Pace: Pattern accurate computationally efficient bootstrapping for timely discovery of cyber-security concepts". 2013 12th International Conference on Machine Learning and Applications. USA: IEEE, 2013, pp. 60-65.

[3] Jones C L, Bridges R A, Huffer K M T, et al, "Towards a relation extraction framework for cyber-security concepts". Proceedings of the 10th Annual Cyber and Information Security Research Conference. USA: ACM, 2015, pp. 1-4.

[4] Joshi A, Lal R, Finin T, et al, "Extracting cybersecurity related linked data from text". 2013 IEEE Seventh International Conference on Semantic Computing. USA: IEEE, 2013, pp. 252-259.

[5] Lal R, "Information Extraction of Security related entities and concepts from unstructured text". 44(3), pp. 127-131, 2013.

[6] Zeng D, Liu K, Chen Y, et al, "Distant supervision for relation extraction via piecewise convolutional neural networks". Proceedings of the 2015 conference on empirical methods in natural language processing. Portugal: Association for Computational Linguistics, 2015, pp. 1753-1762.

[7] Lin Y, Shen S, Liu Z, et al, "Neural relation extraction with selective attention over instances". Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers). Germany: Association for Computational Linguistics, 2016, pp. 2124-2133.

[8] Konda V R, Tsitsiklis J N, "Actor-critic algorithms". Advances in neural information processing systems. USA: NIPS, 2000, pp. 1008-1014.

[9] Zeng D, Liu K, Lai S, et al, "Relation classification via convolutional deep neural network". Ireland: ACL 2014, 2014, pp. 2335-2344.

[10] Feng J, Huang M, Zhao L, et al, "Reinforcement learning for relation classification from noisy data". Thirty-Second AAAI Conference on Artificial Intelligence. USA: AAAI, 2018.

[11] Jiang X, Wang Q, Li P, et al, "Relation extraction with multi-instance multi-label convolutional neural networks". Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers. Japan: The COLING 2016 Organizing Committee 2016, pp. 1471-1480.

A

B

C