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.

DOI reference number: 10.18293/SEKE2020-117.

[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 predicte the probability of link existence through RNN, in order to solve the "the term mismatch" problem [9].

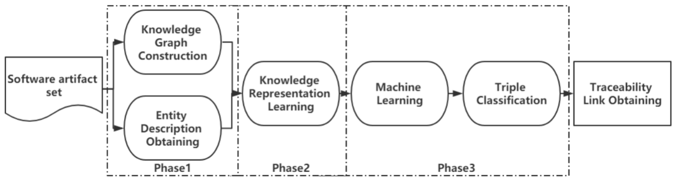
2) Knowledge Graph and Knowledge Representation Learning: The main goal of the knowledge graph is to describe the various entities and concepts that exist in the real world and the explicit relationships between them. Relations are used to describe the relationship between two entities. The knowledge 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.



Software artifact set
Knowledge Graph Construction
Entity Description Obtaining
Knowledge Representation Learning
Machine Learning
Triple Classification
Traceability Link Obtaining
Phase1
Phase2
Phase3

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.

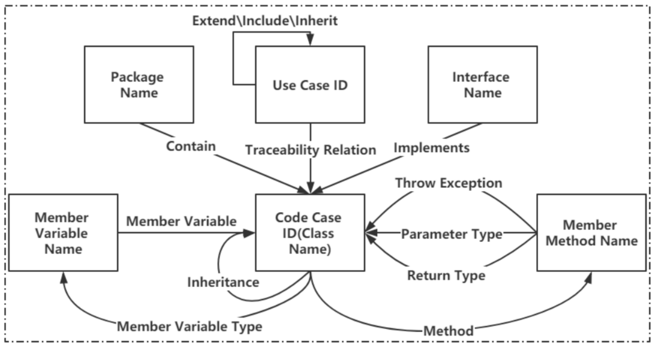


Fig. 2. Entity Type and Relation Definition in KG

- *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",

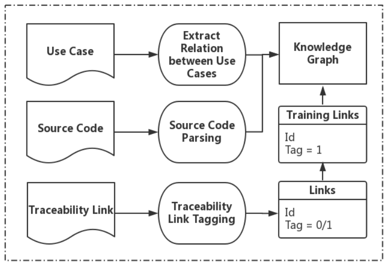


Fig. 3. Software Artifact Knowledge Graph Construction

"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.

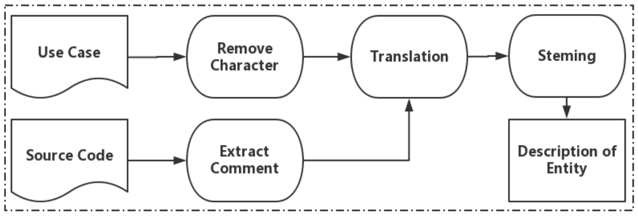


Fig. 4. Entity Description Obtaining

- *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" 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.

[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

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

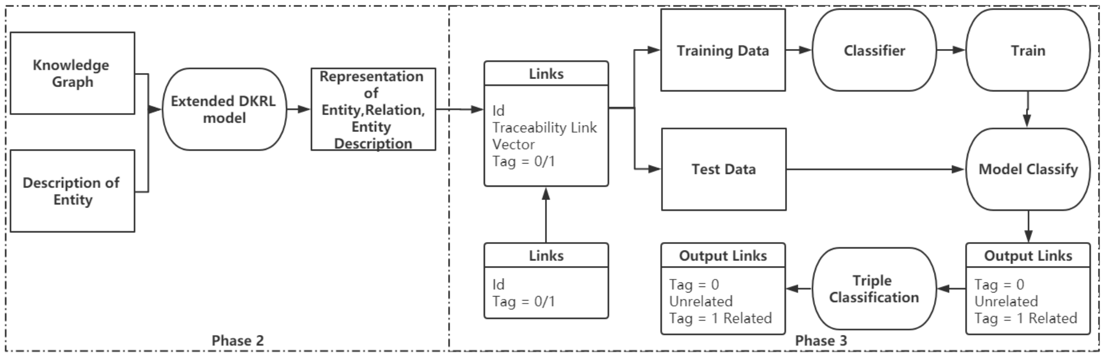


Fig. 5. Phase 2. Knowledge representation learning and Phase 3. Recover Traceability Link

shown in Table 1). Our improved negative sample construction process is as follows:

$$S'_{(h,r,t)} = \begin{cases} \{(h',r,t) \mid (h',r,t) \in T'\} \cup \\ \{(h,r,t') \mid (h,r,t') \in T'\} \\ \textit{if } r \neq r_{Trace} \\ \{(h',r,t) \,|h' \in E\} \cup \{(h,r,t') \,|t' \in E\} \\ \textit{if } r = r_{Trace}" \end{cases} \quad (1)$$

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 $F1$-$score$ can be calculated, which are shown in equation (3)-(5)

D. Analysis of the Results

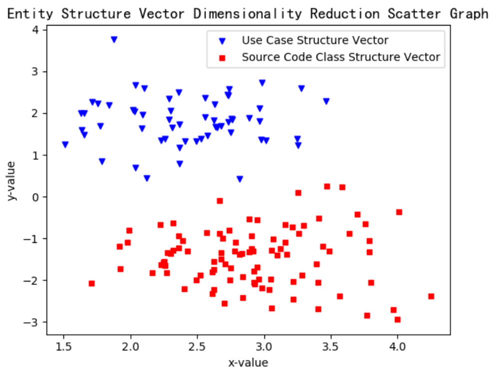


Fig. 6. Entity Structure Vector Dimensionality Reduction Scatter Graph

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.

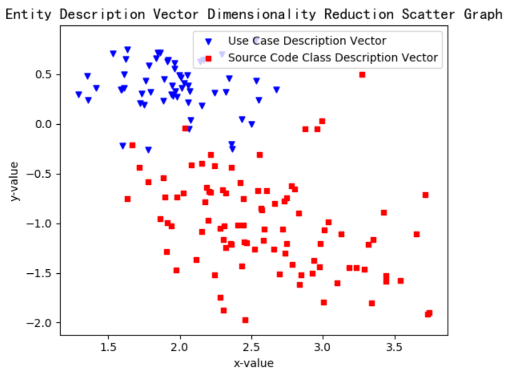


Fig. 7. Entity Description Vector Dimensionality Reduction Scatter Graph

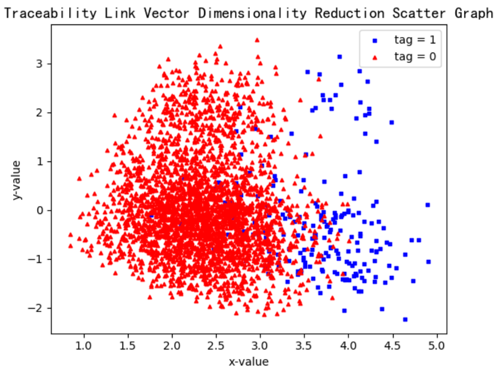


Fig. 8. Traceability Link Vector Dimensionality Reduction Scatter Graph

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.

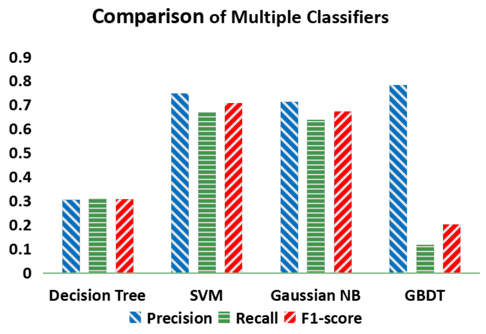


Fig. 9. Comparison of Multiple Classifiers

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.

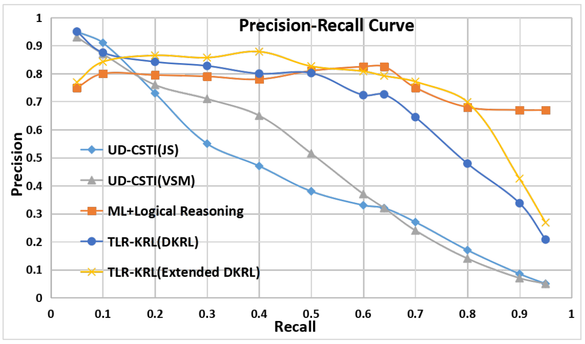


Fig. 10. The Precision-Recall Curves Graph

3) *Experiment 3:* When evaluating a traceability link recovery work, it is common to compare $precision$ at different $recall$ level. Figure 10 shows the $precision\text{-}recall$ 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 $precision\text{-}recall$ of downstream tasks has been improved.

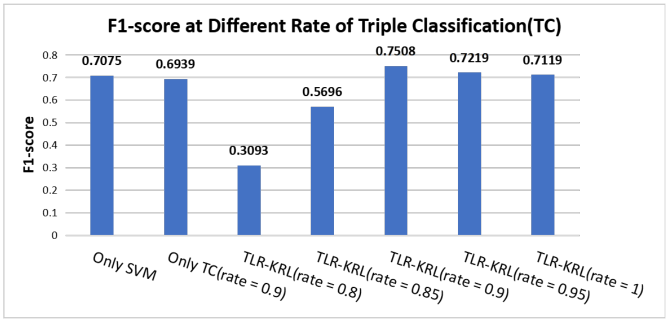


Fig. 11. F1-score at Different Rate of Triple Classification(TC)

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.

Acknowledgement

This work is supported by National Key R&D Program of China (No. 2018YFB0804703), National Natural Science of Foundation of China (No.61902010), and Beijing Excellent Talent Funding-Youth Project (No.2018000020124G039)

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.