Multi-Objective Crowd Worker Selection in Crowdsourced Testing

Qiang Cui*§, Song Wang†, Junjie Wang*, Yuanzhe Hu*§, Qing Wang*‡§ and Mingshu Li*‡§
*Laboratory for Internet Software Technologies, Institute of Software Chinese Academy of Sciences
†Electrical and Computer Engineering, University of Waterloo, Canada
‡State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences
§University of Chinese Academy of Sciences
{cuiqiang, junjie, yuanzhe, wq}@nfs.iscas.ac.cn, song.wang@uwaterloo.ca, mingshu@iscas.ac.cn

***Abstract*—Crowdsourced testing is an emerging trend in software testing, which relies on crowd workers to accomplish test tasks. Typically, a crowdsourced testing task aims to detect as many bugs as possible within a limited budget. For a specific test task, not all crowd workers are qualified to perform it, and different test tasks require crowd workers to have different experiences, domain knowledge, etc. Inappropriate workers may miss true bugs, introduce false bugs, or report duplicated bugs, which could not only decrease the quality of test outcomes, but also increase the cost of hiring workers. Thus, how to select the appropriate crowd workers for specific test tasks is a challenge in crowdsourced testing.**

This paper proposes a Multi-Objective crowd wOrker SElection approach (MOOSE), which includes three objectives: maximizing the coverage of test requirement, minimizing the cost, and maximizing bug-detection experience of the selected crowd workers. Specifically, MOOSE leverages NSGA-II, a widely used multi-objective evolutionary algorithm, to optimize the three objectives when selecting workers. We evaluate MOOSE on 42 test tasks (involve 844 crowd workers and 3,984 test reports) from one of the largest crowdsourced testing platforms in China, and the experimental results show MOOSE could improve the best baseline by 17% on average in bug detection rate.

I. Introduction

Crowdsourced testing is an emerging trend in software testing. In recent years, it has received much attention from the research community [15, 16], and there are many successful crowdsourced testing platforms in industry, e.g., uTest[1], TestBirds[2], and Pay4Bugs[3]. Different from traditional testing, crowdsourced testing entrusts test tasks to crowd workers, who are available on Internet and located in different places. A crowdsourced testing task aims at detecting as many bugs as possible, meanwhile the cost of hiring workers should be controlled within the limited budget.

In current practice, crowd workers search available test tasks themselves and perform any tasks they interested. Thus, test tasks are often performed by a random set of workers. However, for a specific test task, not all crowd workers are qualified to perform it, and different test tasks require crowd workers to have different experiences, domain knowledge, etc. Inappropriate workers may miss true bugs, introduce false bugs, or report duplicated bugs, while hiring them requires nontrivial budget. Selecting inappropriate workers for a test task, not only decreases the quality of test outcome but also increases the cost of hiring workers. Thus, how to select the appropriate crowd workers for specific test tasks is a challenge in crowdsourced testing [9, 19].

The ultimate purpose of crowd workers selection is to select as few workers as possible to detect as many bugs as possible. However, there is no ideal test criterion for this purpose, because the real bug detection information of a test task can only be available until the test task is finished. In this work, we leveraged three alternative criteria for selecting workers, which are critical in crowdsourced testing, i.e., the coverage of test requirement, bug-detection experience of the selected crowd workers, and the cost of the selected crowd workers.

Specifically, *the coverage of test requirement* assures all the test requirements should be tested by the workers with similar expertise. If some test requirements were missed, corresponding bugs might not be detected. This objective is measured using the percentage of terms in test requirement mentioned in workers' historical reports. *The bug-detection experience of the selected crowd workers* assures more experienced workers should be selected to perform the test task, which is important in crowdsourced testing. This is because the experiences of workers vary significantly and experienced workers are more likely to detect bugs [19]. We use the total number of bugs detected by the selected workers to measure this criterion. *The cost of the selected crowd workers* considers the limited budget of a specific test task. Specifically, the cost is the reward for workers, which is set as a constant for each worker performing the test task in most of crowdsourced testing platforms.

In this paper, we propose a Multi-Objective crowd wOrker SElection approach (MOOSE), which aims at selecting the appropriate crowd workers for specific test tasks by considering the above three criteria. MOOSE maximizes the coverage of test requirement, minimizes the cost, and maximizes bug-detection experience of the selected workers. It leverages a widely used multi-objective evolutionary algorithm, namely NSGA-II, to optimize the three objectives when selecting workers.

We evaluate MOOSE on an industrial dataset from Baidu CrowdTest - one of the largest crowdsourced testing platforms

[1]https://www.utest.com/
[2]https://www.testbirds.com/
[3]https://www.pay4bugs.com/

DOI reference number: 10.18293/SEKE2017-102

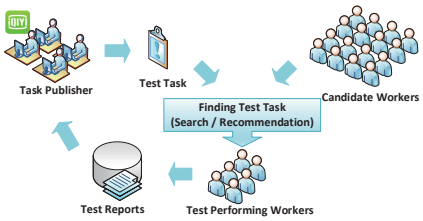

Fig. 1: The typical procedure of crowdsourced testing.

in China. The experimental results show the effectiveness and practical value of MOOSE.

The primary contributions of this paper are as follows:

- We propose a multi-objective crowd worker selection approach, which can maximize the coverage of test requirement, minimize the cost, and maximize the bug-detection experience of the selected workers. **To the best of our knowledge, this is the first approach for the multi-objective crowd worker selection problem.**
- We evaluate our approach on 42 test tasks (involve 844 crowd workers and 3,984 test reports) from one of the largest crowdsourced testing platforms in China. The results show that MOOSE could improve the best baseline on average by 17% in bug detection rate.

II. Background

A. Crowdsourced Testing

The overall procedure of crowdsourced testing is shown in Figure 1. A task publisher provides a test task for crowdsourced testing, including the software under test and test requirements. Then crowd workers could sign in to perform the task, and are required to submit crowdsourced test reports after finishing the task. Finally, the task publisher needs to further inspect the submitted reports and verify whether they are bugs or not.

In current practice, crowd workers search and take proper test tasks all by themselves. However, such process might be ineffective for bug detection, because crowd workers would perform test tasks they are not good at for financial incentive. Consequently, inappropriate workers for a test task may miss true bugs, introduce false bugs, or report duplicated bugs, which could not only decrease the quality of test outcomes, but also increase the cost of hiring workers.

Intuitively, to mitigate the above issue in crowdsourced testing, an appropriate subset of workers should be selected for a specific test task. The selected workers can be recommended to the task publishers, who could then invite them to participate in the task. In addition, the selection results can also assist crowd workers to find proper test tasks so as to save their effort in searching tasks.

In crowdsourced testing, a *test task* is provided by a task publisher, which are described with the natural language test requirements. A *test report* is the test outcome submitted by a crowd worker. It contains the natural language description about operation steps and test outcomes. Specifically, it is labeled by test engineers in the platform to indicate whether the report reveals a "bug", or whether the report is "duplicated" with other reports. A *crowd worker* is a registered worker in the crowdsourced testing platform.

B. Multi-objective Optimization

The multi-objective optimization problem seeks to simultaneously optimize multiple objective functions. It can be defined as to find a vector of variable x, which optimizes a vector of M objective functions $f_i(x)$, where $i = 1, 2, ..., M$.

The multiple objective functions are optimized using Pareto optimality [3], which is a strategy that supposed one player's situation cannot be improved without making the other player's situation worse. Specifically, a decision vector x_1 will dominate a decision vector x_2 if and only if their objective vectors $f_i(x_1)$ and $f_i(x_2)$ satisfy:

$$f_i(x_1) \geq f_i(x_2) \forall i \in 1, 2, ..., M; and$$
$$\exists i \in 1, 2, ..., M | f_i(x_1) > f_i(x_2)$$

All decision vectors that are not dominated by any other decision vectors are called to form the Pareto optimal set, and the corresponding objective vectors are called the the Pareto frontiers.

III. MOOSE

We first present a definition of multi-objective crowd worker selection problem. Given a set of candidate workers W, a vector of M objective functions f_i, where $i = 1, 2, ..., M$, the problem is to find a subset of workers S that is a Pareto efficient set with respect to the objective functions.

To solve the problem, we propose a multi-objective worker selection approach (MOOSE), which maximizes the coverage of test requirement, minimizes the cost, and maximizes the bug-detection experience of the selected workers. Specifically, the cost is the inevitable constraint for worker selection, the other two objectives, i.e., the coverage of test requirement and the bug-detection experience of the selected workers, are critical criteria in crowdsourced testing. We use search-based method to solve the multi-objective optimization. Like other search-based software engineering tasks [4, 7, 10, 13, 20], MOOSE contains representation, fitness function, and computational search algorithm. It also contains a fourth part to illustrate how to handle multiple objectives.

A. Representation

Like other selection problems [4, 6, 14], we encode each worker as a binary variable. If the worker is selected, the value is one; otherwise, the value is zero. The solution is a vector of binary variables, which includes all the candidate workers in crowdsourced testing. The initial population is generated randomly, and the feasible solution would be selected when their values are positive for all objective functions.

B. Fitness

To evaluate the fitness of each solution, we employed a multi-objective function to simultaneously maximize *the coverage of test requirement*, *the bug-detection experience of the selected workers*, and minimize *the cost*.

1) Coverage of test requirement: Test requirement coverage is an important test criterion in software testing. In crowdsourced testing, to cover the test requirements, on the one hand, the selected workers should have similar expertise with the requirements; on the other hand, the workers should be different with each other, so as to cover different parts of the requirements. In this work, we use the conducted crowdsourced tasks and accomplished reports of a worker to represent his/her expertise. Therefore, we use the technical terms in one's historical test reports to represent his expertise. The coverage of test requirement (TR) is measured by the percentage of all terms in the test requirement covered by the expertise of the selected workers. Note that, not all the natural language terms are meaningful for testing, we only use the technical terms obtained based on the method in Section IV-D. Formally, the coverage of test requirement is defined as follows.

$$Coverage = \frac{\textit{\# unique terms of TR mentioned by workers}}{\textit{\# unique terms in TR}} \quad (1)$$

2) Bug-detection experience of the selected workers: In crowdsourced testing, it often has a very distinguished situation that the experiences of crowd workers vary a lot. Some workers have detected many bugs. With the rich experience, they hardly miss true bugs during testing. While some other workers are almost new and do not have much experience, which makes them easily miss true bugs, introduce false bugs, and report duplicate bugs. Thus, it is important to select the experienced workers to perform a specific test task. The bug-detection experience is measured as the total number of bugs the worker detected before. Since duplicated bugs detected by different workers are useless to the overall test outcomes, we use the number of unique bugs detected by a set of workers to represent their bug-detection experience. As mentioned in Section II, duplicated bugs are labeled by test engineers in the platform, thus the unique bugs can be easily picked out.

3) Cost: The cost is an unavoidable objective in crowd worker selection, because the constraint of the cost must be considered when selecting workers in crowdsourced tasks. The straightforward cost in crowdsourced testing is the reward for workers. We suppose all the workers who participate in a test task are equally paid, which is a common practice in real-world crowdsourced testing platforms. The cost is measured as the total reward of the selected workers.

C. Computational Search

We employed a widely used evolutionary algorithm, namely NSGA-II, to simultaneously optimize the three objectives. NSGA-II is a genetic algorithm based optimization technique developed by Deb et al. [3], which is the state-of-the-art optimization technique and has already been widely used in other multi-objective optimization tasks [10].

D. Handling Multiple Objectives

In MOOSE, the three objectives are handled on orthogonal scales. Then we employed Pareto optimality: a solution x_1 is said to dominate another solution x_2, if x_1 is no worse than x_2 in all objectives and x_1 is strictly better than x_2 in at least one objective. According to Pareto optimality, we can search for the set of solutions which are non-dominating.

TABLE I: Running Example.

	Technical terms of TRs of example tasks						Experience (unique bug)				Cost
	t1	t2	t3	t4	t5	t6	b1	b2	b3	b4	value
w_1	√	√							√	√	1
w_2				√	√	√	√	√			1
w_3	√				√	√	√	√		√	1
w_4		√	√	√			√				1

E. Running example

To make MOOSE more clear, we describe it using a running example. As shown in Table I, there are four candidate workers (i.e., w_1 to w_4).

- The workers' historical test reports and test requirements are pre-processed to extract the technical terms. Then for each test task, whether a technical term of test requirements is mentioned by a worker is encoded into a binary vector. For simplicity, we only use six terms as shown in Table I.
- The bug-detection experience of a worker is extracted and encoded into a binary vector. For simplicity, we only use four historical bugs as shown in Table I.
- The cost of selecting each worker is set to "1" and also encoded into a vector.
- The above three binary vectors of a worker are merged into one vector. Then the vectors of workers for a specific test task are passed into the NSGA-II. Through the several iterative generation, selection, and evaluation of NSGA-II, the non-dominating solutions are finally obtained.

In our running example, the solution $\{w1, w2\}$ is dominant $\{w1, w3\}$, because it has a better coverage, a better bug-detection experience, and equal cost. Similarly, $\{w_3\}$ dominates all the solutions containing one worker, $\{w_1, w_2\}$, $\{w_1, w_2, w_4\}$ are the best solutions for selecting two or three workers respectively.

IV. Empirical Study Design

A. Research Questions

- **RQ1**: (Effectiveness of MOOSE) How effective is MOOSE for crowd worker selection?
- **RQ2**: (Necessity of the objectives) Look inside of MOOSE, is each of the three objectives necessary in MOOSE?
- **RQ3**: (Quality of results) Do the results of MOOSE achieve high quality?

B. Evaluation Metric

In this work, bug detection data are used to evaluate the performance of our approach. Given a test task, we measure the performance of a worker selection approach according to whether it can select the "right" workers, who have performed this test task and detected true bugs.

Bug Detection Rate (BDR) is the percentage of bugs detected by the selected crowd workers in a test task out of all bugs historically detected in the same test task. Formally,

TABLE II: Statistics of the dataset used in this work.

Statistic	Number
# of test tasks	42
# of categories	13
# of candidate workers	844
# of test reports	3,984
average # of test reports per task	40
average # of bugs per task	25

given a set of selected workers, i.e., W, and a test task, i.e., T, the bug detection rate is defined as follows:

$$BDR = \frac{\#bugs\ detected\ by\ workers\ in\ W}{\#all\ bugs\ of\ T} \quad (2)$$

Since a smaller subset is usually preferred in crowd worker selection due to the limited budget, we investigate the BDR when selecting from 1% to 50% of the total number of candidate workers for a test task.

C. Baselines

To evaluate the performance of MOOSE, we introduce three baselines. **Random** simulates the current practice in most crowdsourced testing platforms, where workers search test tasks to perform. It randomly selects workers from the candidate set of workers. We run Random for 10 times and record the best performance as its performance.

To further evaluate MOOSE, we also introduce two common ranking baselines, i.e., **Bug Ranking** and **IR Ranking**. **Bug Ranking** ranks all the candidate workers according to the number of historical bugs the worker detected before, and recommends the top workers. **IR Ranking** ranks all the candidate workers according to the textual similarity between the workers' historical test reports and test requirement (Similarity is calculated using Euclidean distance between technical term vectors [18]), and recommends the top workers.

D. Baidu CrowdTest Dataset

We collected crowdsourced testing data from an industrial crowdsourced testing platform, namely Baidu CrowdTest. We collected the test tasks that are closed between Nov. 1st 2015 and Nov. 30th 2015. In total, there are 42 tasks covering 13 categories, such as utilities, lifestyle, finance, etc. For each test task, we manually collected all the detected true bugs of it. The detail statistics of data set are shown in Table II.

To process the dataset, we employed Natural Language Processing to extract a technical term vocabulary for measuring the coverage of test requirements in Section III-B1. Firstly, ICTCLAS[4] is used for word segmentation, then stopwords are removed, and part-of-speech tags are conducted. Finally terms except verbs and nouns are removed. Filtering meaningless terms like existing work [15], we obtain a vocabulary of technical terms.

E. Experimental Setup

To evaluate MOOSE, following existing work [12], we use cross-validation in our experiments. We first randomly selected 70% test tasks as a training dataset and the remaining ones as the test dataset. To mitigate the randomness, the experiment is repeated 20 times, and we use the average BDR to evaluate MOOSE's performance.

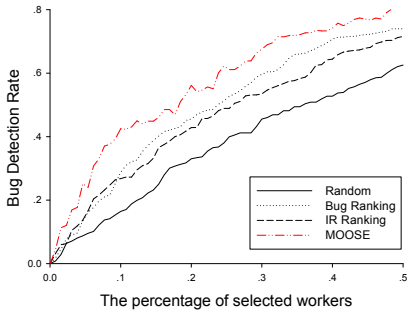

Fig. 2: Comparison of MOOSE with different baseline approaches (RQ1).

For NSGA-II used in MOOSE , we used a random initial population of size 200, and we iterated the algorithm for 200,000 max evaluation, with single point crossover and bit-flip mutation. We set the max evaluation to 200,000, since extended number of evaluations does not show any noticeable improvement in performance. To take into account the inherent randomness of the algorithm, for each test task, we executed 20 independent runs of the algorithm. Note that the running time for NSGA-II is 3,000 ms on average for each test task, which is negligible. We implemented MOOSE based on jMetal[5], a widely used Java framework aiming at solving multi-objective optimization problems.

V. Results

RQ1: How effective is MOOSE for crowd worker selection?

We first compare **MOOSE** with **Random**, which represents the current practice of worker selection in crowdsourced testing platforms. It can be easily observed from Figure 2 that **MOOSE** outperforms **Random**. Specifically, compared with **Random**, the BDR improvement ((MOOSE-Random)/Random) achieved by **MOOSE** is 158% (0.16 vs. 0.42) when selecting only 10% workers, and the BDR improvement is 48% (0.45 vs. 0.67) when selecting 30% workers. In addition, from selecting 1% to 50% workers, the average improvement of BDR is 58%.

Furthermore, **MOOSE** also outperforms both **Bug Ranking** and **IR ranking**, which are two commonly-used ranking methods. The average improvement of BDR for **Bug Ranking** is 17% and for **IR Ranking** is 25%. It can be found that **Bug Ranking** and **IR Ranking** largely outperform **Random**, and in some test tasks, they are close to **MOOSE**. They leverage the measurements that are also about the workers' bug-detection experience and expertise, which implies these two criteria are really useful for selecting workers.

In addition, Table III shows the average BDR when selecting between 1% and 50% workers for each of the 42 test tasks. In most of test tasks (37/42, 88%), **MOOSE** achieves the best performance compared with the three baseline methods. In the

[4]http://ictclas.org

[5]http://jmetal.github.io/jMetal/

TABLE III: The detailed results of MOOSE and other compared methods.

	Bug Detection Rate of all compared methods (RQ1,RQ2)							Bug Detection Rate of all compared methods (RQ1,RQ2)					
	Random	Bug Rank	IR Rank	Bug+Cost	Cov+Cost	MOOSE		Random	Bug Rank	IR Rank	Bug+Cost	Cov+Cost	MOOSE
T1	0.235	0.334	0.274	0.494	0.417	**0.529**	**T22**	0.337	0.498	0.606	0.548	**0.636**	0.631
T2	0.480	0.467	0.413	0.411	0.506	**0.520**	**T23**	0.473	0.492	0.534	0.542	0.492	**0.503**
T3	0.374	0.429	0.518	0.474	0.537	**0.554**	**T24**	0.400	0.617	0.676	0.525	**0.700**	0.659
T4	0.536	**0.677**	0.501	0.667	0.670	0.671	**T25**	0.315	0.150	0.396	0.198	0.492	**0.519**
T5	0.516	0.803	0.810	0.756	0.809	**0.823**	**T26**	0.302	0.584	0.579	0.492	0.635	**0.646**
T6	0.000	0.684	0.292	0.861	0.846	**0.881**	**T27**	0.307	0.405	0.307	0.282	0.452	**0.472**
T7	0.469	0.475	0.381	0.609	0.593	**0.611**	**T28**	0.717	0.620	0.487	0.605	0.703	**0.723**
T8	0.193	0.401	0.204	**0.404**	0.360	0.391	**T29**	0.310	0.117	0.338	0.328	0.220	**0.338**
T9	0.598	0.723	0.560	0.681	0.756	**0.763**	**T30**	0.113	0.009	0.360	0.126	**0.175**	0.107
T10	0.246	0.713	0.600	0.612	0.732	**0.744**	**T31**	0.876	0.909	0.707	0.928	0.723	**0.933**
T11	0.157	0.492	0.500	0.740	0.719	**0.742**	**T32**	0.376	0.390	0.221	0.471	0.344	**0.425**
T12	0.169	0.633	0.083	0.480	0.547	**0.636**	**T33**	0.138	0.000	0.000	**0.169**	0.000	0.153
T13	0.400	0.415	0.451	0.512	**0.569**	0.507	**T34**	0.217	0.378	0.323	0.362	0.347	**0.397**
T14	0.405	0.674	0.623	**0.756**	0.615	0.728	**T35**	0.326	0.325	0.314	0.304	**0.369**	0.345
T15	0.249	0.633	0.486	0.616	0.520	**0.651**	**T36**	0.309	0.493	0.440	0.540	0.575	**0.593**
T16	0.371	0.348	0.225	0.348	**0.389**	0.371	**T37**	0.404	0.419	0.438	0.461	**0.506**	0.436
T17	0.000	0.598	0.576	0.630	0.640	**0.659**	**T38**	0.528	0.579	0.600	0.574	0.771	**0.783**
T18	0.030	0.104	0.070	0.144	0.092	**0.150**	**T39**	0.750	0.764	0.723	0.810	0.707	**0.826**
T19	0.365	0.614	**0.786**	0.587	0.548	0.558	**T40**	0.584	0.510	0.560	**0.676**	0.630	0.609
T20	0.380	0.407	0.368	**0.462**	0.440	0.440	**T41**	0.292	0.617	0.558	0.780	0.817	**0.822**
T21	0.357	0.605	0.619	0.365	0.523	**0.620**	**T42**	0.646	0.476	0.680	0.638	0.668	**0.685**

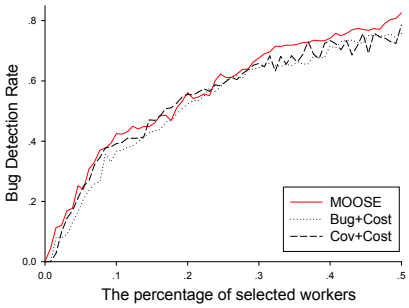

Fig. 3: Comparison between MOOSE and two bi-objective approaches: Bug+Cost and Cov+Cost (RQ2).

rest test tasks, the performance of **MOOSE** is also relatively high.

For all experimental results reported above, we conduct Mann-Whitney U test, and the p-value is much smaller than 0.05. This further implies that **MOOSE** can significantly improve the three baselines.

In summary, **MOOSE** can significantly outperform the current practice in worker selection, as well as two commonly-used ranking methods. Therefore, **MOOSE** is effective for crowd worker selection.

RQ2: Is each of the three objectives necessary in MOOSE?

In crowd worker selection problem, the objective of cost is indispensable, which cannot be removed. Hence, we compare the performance of **MOOSE** when removing either of the other two objectives. We therefore design two bi-objective approaches: **Bug+Cost** (Bug experience of the selected workers+Cost) and **Cov+Cost** (Coverage of the test requirements+Cost).

We can easily see from Figure 3 that our **MOOSE** outperforms the two bi-objective approaches in terms of BDR. The average improvement of BDR for **Bug+Cost** is 9.7%, for **Cov+Cost** is 5.7%.

Table III also presents the average BDRs when selecting between 1% and 50% workers for each of the 42 test tasks. In most of test tasks (29/42, 69%), **MOOSE** is better than both of the bi-objective approaches. In the rest test tasks, the performance of **MOOSE** is also relatively high. For all these experimental results reported above, we conduct Mann-Whitney U test, and the p-value is much smaller than 0.05.

In summary, all the three objectives, which are the coverage of test requirement, bug-detection experience of the selected workers, and cost, are necessary in **MOOSE**.

RQ3: Do the results of MOOSE achieve high quality?

Since MOOSE is a search-based approach, which produces Pareto fronts. To evaluate the quality of Pareto fronts, existing studies in Search-based Software Engineering have applied quality indicators, such as Contribution (I_C), Hypervolume (I_{HV}), and Generational Distance (I_{GD}) [17]. To illustrate the quality of an algorithm, these quality indicators compare the results of the algorithm with the reference Pareto front, which consists of best solutions. To assess the quality of results of MOOSE, we employed these three quality indicators. The set of non-dominated solutions found by **MOOSE**, **Bug+Cost**, and **Cov+Cost** are used as Reference Set (RS) [17].

I_C is the proportion of solutions given by an algorithm that lie on the reference front. The higher this proportion, the more contribution the algorithm to the best solutions and the better the corresponding algorithm. I_{HV} calculates the volume covered by members of a non-dominated set of solutions from an algorithm. The larger this volume, the better the corresponding algorithm. I_{GD} computes the average distance between set of solutions from the algorithm and the reference set. The smaller this distance, the better the corresponding algorithm. Due to the limited space, for details about the three quality indicators, please refer to [17].

For **MOOSE**, we measure these three quality indicators for each test task, then the average values are obtained. The average I_C of **MOOSE** is 0.89, the average I_{HV} is 0.90, and the average I_{GD} is 0.00. The results suggest that the results of **MOOSE** achieve high quality.

VI. Threats to Validity

The threats to external validity concern the generality of this study. First, our experimental data consists of 42 test tasks collected from one of the largest crowdsourced testing platforms in China. The results of our study may not generalize beyond this environment where our experiments were conducted. However, we used different sizes and a variety of data to control this threat. Second, all crowdsourced reports investigated in this study are written in Chinese, and it is not guaranteed that similar results can be observed on crowdsourced projects in other languages. This threat, however, is alleviated as we did

not conduct semantic comprehension, but instead we simply tokenized sentences and used words as tokens for modeling.

The main threat to construct validity in this study involves three objectives in multi-objective formulation. These three objectives are designed from different test criteria: such as the coverage of test requirements, bug detection experience of the selected workers, and cost, but other objectives may also contribute to bug detection in crowdsourced testing. Further exploration of other objectives would address this threat.

VII. Related Work

Crowdsourced testing has been applied to generate test cases [2], solve the oracle problem [11], help usability testing [8], etc. All these studies use crowdsourced testing to solve the problems in traditional software testing activities. There are studies focusing on solving the new encountered problem in crowdsourced testing, e.g., crowdsourced reports prioritization [5] and crowdsourced reports classification [15, 16]. Our approach is also to solve the new encountered and important problem in crowdsourced testing.

The Search Based Software Engineering (SBSE) is an increasingly trend in software engineering. In SBSE, many software engineering problems are reformulated as search problems, such as test case selection [4, 10, 20], and mutation testing [7, 13].

There are several related researches focusing on selecting workers for various software engineering tasks, such as bug triage [6], mentor recommendation [1], expert recommendation [14], etc. All the aforementioned studies either select one worker, or assume the selected set of workers are independent with each other. However, our work selects a set of workers who are dependent on each other, because their performance can together influence the final test outcomes.

VIII. Conclusion

In this paper, we propose MOOSE for crowd worker selection, which maximizes the coverage of test requirement, minimizes the cost and maximizes the bug-detection experience of the selected workers. Experimental results show that MOOSE is effective in bug detection.

In the future, we plan to explore more test criteria that may be helpful for worker selection in crowdsourced testing. Collaborating with Baidu CrowdTest, we are on the way to deploying MOOSE online in order to better evaluate its performance in practice.

IX. Acknowledgments

This work is supported by the National Natural Science Foundation of China under grant No.61432001, No.61602450. We would like to thank the testers in Baidu for their great efforts in supporting this work.

References

[1] G. Canfora, M. Di Penta, R. Oliveto, and S. Panichella. Who is going to mentor newcomers in open source projects? In *FSE'12*, pages 44:1–44:11.

[2] N. Chen and S. Kim. Puzzle-based automatic testing: Bringing humans into the loop by solving puzzles. In *ASE '12*, pages 140–149.

[3] K. Deb, S. Agrawal, A. Pratap, and T. Meyarivan. A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: NSGA-II. In *ICPPSFN'00*, pages 849–858.

[4] M. G. Epitropakis, S. Yoo, M. Harman, and E. K. Burke. Empirical evaluation of pareto efficient multi-objective regression test case prioritisation. In *ISSTA'15*, pages 234–245.

[5] Y. Feng, J. A. Jones, Z. Chen, and C. Fang. Multi-objective test report prioritization using image understanding. In *ASE'16*, pages 202–213.

[6] G. Jeong, S. Kim, and T. Zimmermann. Improving bug triage with bug tossing graphs. In *FSE'09*, pages 111–120, 2009.

[7] W. B. Langdon, M. Harman, and Y. Jia. Efficient multi-objective higher order mutation testing with genetic programming. *JSS'10*, 83(12):2416 – 2430.

[8] D. Liu, R. G. Bias, M. Lease, and R. Kuipers. Crowdsourcing for usability testing. *ASIS&T'12*, 49(1):1–10.

[9] K. Mao, L. Capra, M. Harman, and Y. Jia. A survey of the use of crowdsourcing in software engineering. *JSS'16*, 126:57–84.

[10] D. Mondal, H. Hemmati, and S. Durocher. Exploring test suite diversification and code coverage in multi-objective test case selection. In *ICST'15*, pages 1–10.

[11] F. Pastore, L. Mariani, and G. Fraser. CrowdOracles: Can the crowd solve the oracle problem? In *ICST'2013*, pages 342–351, March 2013.

[12] F. Sarro, A. Petrozziello, and M. Harman. Multi-objective software effort estimation. In *ICSE '16*, pages 619–630.

[13] R. A. Silva, S. d. R. S. de Souza, and P. S. L. de Souza. A systematic review on search based mutation testing. *IST'17*, 81:19 – 35.

[14] A. Tamrawi, T. T. Nguyen, J. M. Al-Kofahi, and T. N. Nguyen. Fuzzy set and cache-based approach for bug triaging. In *FSE'11*, pages 365–375.

[15] J. Wang, Q. Cui, Q. Wang, and S. Wang. Towards effectively test report classification to assist crowdsourced testing. In *ESEM'16*, pages 6:1–6:10.

[16] J. Wang, S. Wang, Q. Cui, and Q. Wang. Local-based active classification of test report to assist crowdsourced testing. In *ASE'16*, pages 190–201.

[17] S. Wang, S. Ali, T. Yue, Y. Li, and M. Liaaen. A practical guide to select quality indicators for assessing pareto-based search algorithms in search-based software engineering. In *ICSE '16*, pages 631–642.

[18] I. H. Witten and E. Frank. *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, 2005.

[19] Y. Yang, M. R. Karim, R. Saremi, and G. Ruhe. Who should take this task?: Dynamic decision support for crowd workers. In *ESEM'16*, page 8.

[20] S. Yoo and M. Harman. Pareto efficient multi-objective test case selection. In *ISSTA'07*, pages 140–150.