Advanced Crowdsourced Test Report Prioritization Based on Adaptive Strategy

PENGHUA ZHU 1, YING LI 1, TONGYU LI 2, HUIMIN REN 3, AND XIAOLEI SUN 4.

1North China Institute of Aerospace Engineering, China
2State Key Laboratory for Novel Software Technology, Nanjing University, China
3Beijing Aerospace Automatic Control Institute, China
4Wuhu Institute of Technology, China

Corresponding author: Huimin Ren (e-mail: appletree_rhm@163.com).

ABSTRACT Crowdsourced testing is an emerging trend in software testing, which takes advantage of the efficiency of crowdsourced and cloud platforms. Crowdsourced testing has gradually been applied in many fields. In crowdsourced software testing, after the crowdsourced workers complete the test tasks, they submit the test results in test reports. Therefore, in crowdsourced software testing, checking a large number of test reports is an arduous but unavoidable software maintenance task. Crowdsourced test reports are numerous, complex, and need to be sorted to improve inspection efficiency. There are no systematic methods for prioritizing reports in crowdsourcing test report prioritization. However, in regression testing, test case prioritization technology has matured. Therefore, we migrate the test case prioritization method to crowdsourced test report prioritization and evaluate the effectiveness of these methods. We use natural language processing technology and word segmentation to process the text in the test reports. Then we use four methods to prioritize the reports: total greedy algorithm, additional greedy algorithm, genetic algorithm, and ART. The results show that these methods all perform well in prioritizing crowdsourced test reports, with an average APFD of more than 0.8.

INDEX TERMS Crowdsourced Software Testing, Test Report Prioritization, Text Classification

I. INTRODUCTION

Crowdsourcing technology has been widely favored in the field of software engineering research [1] in recent years. In crowdsourcing software testing, the main participants include task requesters, crowdsourcing workers, and crowdsourcing platforms [2]. Crowdsourced works are required to perform test tasks and submit a test report of the system’s behavior. The test reports are usually composed of natural language descriptions, sometimes with screenshots. Since crowdsourced testing usually involves many users, the number of test reports can be great, and the task of reviewing these test reports is time-consuming and expensive. Therefore, developers are trying to find methods to prioritize and filter valuable reports.

Previous research has produced test case prioritization techniques. The test case prioritization is to arrange the execution order of the test cases in the regression test suite to execute them in the order of earlier faults. The existing prioritization algorithms can be divided into the following groups, i.e., greedy algorithm, search-based algorithm, information-retrieval-based algorithm, integrate-linear-programming-based algorithm, and machine-learning-based algorithm [3]. In the field of crowdsourced test report prioritization, there is no mature technology proposed, and the test case prioritization technology has matured. This motivates us to migrate and apply test case prioritization technology to the prioritization of crowdsourced test reports.

In this paper, we apply four test case prioritization techniques to crowdsourced test report prioritization and evaluate the effectiveness of these methods. The four methods are total greedy algorithm, additional greedy algorithm, genetic algorithm, and ART [4]. We collect a total of 723 crowdsourced test reports for five different types of applications. After obtaining the crowdsourced test reports, we use natural language processing (NLP) technology and word segmentation to process the texts in the test report. First, we put together all the texts of each report. Then, Jieba library

1https://github.com/fxsjy/jieba
is used to perform word segmentation processing on the summarized texts. After word segmentation, we filter out the stop words based on the stop word list. Finally, we use a one-hot encoded vector to represent each report. After getting the vector, we use the above four algorithms to sort the test report. For the ART method, in addition to the word segmentation method, we also use the Word2Vec model [5], and TF-IDF [6] to encode the texts. The average percentage of faults detected (APFD) [7] is introduced to compare four test report prioritization methods. The experimental results show that four methods perform well in the crowdsourced test report prioritization, and the average APFD is above 0.8.

In this paper, our main contributions are as follows:

1) We first apply the test case prioritization method to the prioritization of crowdsourced test reports, adopt corresponding text encoding strategies for different methods, and explore the feasibility of these methods.
2) We employ a diverse strategy to rank test reports, and these strategies are evaluated using five crowdsourced test report datasets.

The rest of this paper is structured as follows. In the section II, we introduce the relevant background and motivation. Section III describes four prioritization methods. In section IV and V, we conduct empirical experiments to evaluate the effectiveness of the prioritization of crowdsourced test reports. The related work is reviewed in Section VI. Finally, in section VII, we conclude this paper.

II. BACKGROUND AND MOTIVATION

In this section, we describe the background of crowdsourced software testing to motivate the prioritization of test reports. We also show some sample test reports. Prioritization of test cases is also introduced in this section.

Crowdsourced Software Testing. In 2006, Howe first proposed the concept of crowdsourcing [8]. Subsequently, many scholars defined crowdsourcing from different angles [9]. In crowdsourcing software engineering, researchers have proposed a large number of related application technologies and application scenarios. Zhang et al. derived the definition of crowdsourcing software testing from the definition of crowdsourcing software engineering [10]. Figure 1 shows the procedure of crowdsourcing testing. Testers prepare the software under test and test tasks. Test tasks are divided into sub-tasks. These tasks are released on a crowdsourced platform, and workers bid for test tasks. When a bug is detected, the worker submits a test report online.

In crowdsourced software testing, workers will submit a test report after completing the task. Because crowdsourced testing involves many tasks, workers usually submit thousands of test reports. Testers manually check all test reports to judge the performance of workers, which is a time-consuming and tedious process. Therefore, this motivates us to prioritize crowdsourced test reports.

In Table 1, we show three examples of crowdsourced test reports. Each test report is mainly composed of Environment, Input, Expect Output, Actual Output, and Screenshot. Environment refers to the test environment, including hardware and software configuration. Input is test data and operation steps. Workers follows the corresponding test steps to carry out the test. Expect Output is the result that worker expects the software to run. Actual Output is the result of the software actually running. Screenshot is the state of the software when a bug occurs.

Test Case Prioritization. Prioritization of test cases is to prioritize test cases that need to be re-executed in regression testing. Test cases are executed in the prioritized order to detect more faults earlier. The test case prioritization can be defined as the following process: given any test suite \( T \), the test case prioritization is to find a permutation \( T' \) of \( T \) that satisfies \( f(T') \geq f(PT) \), where \( PT \) represents a set of possible permutations of \( T \) and \( f \) is a function from \( PT \) to the real number, when applied to an ordering, an evaluation of the ordering occurs [11].

Test case prioritization algorithms can be divided into several groups, i.e., greedy algorithm, search-based algorithm, information-retrieval-based algorithm, integrate-linear-programming-based algorithm, and machine-learning-based algorithm [12]. The greedy algorithm is widely used to solve the problem of test case prioritization, and it searches for the optimal local solution of prioritization. Since the greedy algorithm does not always get the optimal solution in the solution space, search-based algorithms are used to solve the prioritization problem. Integrate-linear programming-based algorithm transforms the prioritization of test cases into a formula construction and solution process. The information-retrieval-based algorithm uses each test case’s execution information or source code to construct a corresponding document collection for each test case. The machine-learning-based algorithm uses machine learning technology, which builds a model from sample input to make predictions on new data.

---

2 https://github.com/goto456/stopwords
TABLE 1. Examples of test reports

| No.  | Environment                                      | Input                                                                                           | Description                                                                 | Screenshot |
|------|--------------------------------------------------|-------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------|------------|
| Report1 | Operating System: Android 4.3  
|        | Phone model: Vivo X3L  
|        | Screen Resolution: 1920 × 1080  
|        | System Language: Chinese                     | 1. Click the "My" button  
|        | 2. Click the settings icon in the upper right corner  
|        | 3. Click “Personal Information”  
|        | 4. Click "Date of Birth" to select 2017.10.21 | Personal information can be stored normally.                                                 | ![Screenshot](image1.png) |
| Report2 | Operating System: Android 4.3  
|        | Phone model: Vivo X3L  
|        | Screen Resolution: 1920 × 1080  
|        | System Language: Chinese                     | 1. Disconnect from the network  
|        | 2. Use email to log in  
|        | 3. Click to log in                           | Prompt “the network is not connected”.                                                     | ![Screenshot](image2.png) |
| Report3 | Operating System: Android 4.3  
|        | Phone model: Vivo X3L  
|        | Screen Resolution: 1920 × 1080  
|        | System Language: Chinese                     | 1. Click the Lucky Draw button for three seconds  
|        | 2. Click the Stop Draw button                 | The turntable stops rotating successfully, and the lottery is successful.                | ![Screenshot](image3.png) |
| Report4 | Operating System: Android 4.3  
|        | Phone model: Vivo X3L  
|        | Screen Resolution: 1920 × 1080  
|        | System Language: Chinese                     | 1. Modify personal profile  
|        | 2. Modify background image or personal profile  
|        | 3. Click save                                | Prompt “There is an unknown error”, and the modification fails to save.                  | ![Screenshot](image4.png) |
| Report5 | Operating System: Android 4.3  
|        | Phone model: Vivo X3L  
|        | Screen Resolution: 1920 × 1080  
|        | System Language: Chinese                     | 1. View Comments  
|        | 2. Enter Comments  
|        | 3. Click Submit                              | Can’t quickly find out comments that have been posted.                                   | ![Screenshot](image5.png) |

III. METHODOLOGY

In this section, we introduce our crowdsourced test report prioritization method in detail. Figure 2 shows the overall framework of our test report prioritization.

A. PREPROCESSING

After collecting the crowdsourced test reports, we first preprocess the report. These test reports strictly include four parts: Environment, Input, Description, and Screenshot. Our preprocessing operations for the report include three processes: word segmentation, stop word removal and synonym replacement.

Word Segmentation. Word segmentation is a natural language processing (NLP) task. There are many effective word segmentation tools for different languages [13]. Since our report is composed in Chinese, we use the Jieba library to segment sentences into words. Jieba is a word segmentation algorithm specially used for Chinese word segmentation. It uses a prefix dictionary to achieve efficient word graph scanning, generates a directed acyclic graph composed of all pos-
sible generations of Chinese characters in the sentence, then uses dynamic programming to find the maximum probability path, and finds the maximum cut based on word frequency. For unregistered words, the HMM (Hidden Markov Model) model [14] based on the ability to form Chinese characters is used, and the Viterbi algorithm is used.

**Stop Word Removal.** In crowdsourced testing, workers usually come from different workplaces, and they have different language preferences and expression habits. Therefore, the test report will contain some meaningless words, and these words do not need to be part of the features of the test report. We filter out these useless words according to the stop word list, usually called “stop words” in the NLP literature.

**Synonym Replacement.** In crowdsourced testing, workers may use different words to express the same concept in the test report. For example, “turn on” and “open” refer to the same semantics. To alleviate this problem, we perform the synonym replacement operation. We adopt the synonym library of Language Technology Platform (LTP) [15], which is considered to be one of the best Chinese NLP platforms.

### B. TEXT FEATURE EXTRACTION

We extract three types of report text features, i.e., TF-IDF (Term Frequency and Inverse Document Frequency) feature, word embedding feature, and one-hot encoding feature.

**TF-IDF Feature.** TF-IDF is a statistical method used to evaluate the importance of a word to a document set or a document in a corpus. The main idea of TF-IDF is that if a word or phrase appears in an article with a high frequency of TF and rarely appears in other articles, it is considered that the word or phrase has good classification ability and is suitable for classification. Specifically, given a term $T$ and a report $R$, $TF(T, R)$ is the frequency of the term $T$ in the report $R$. $IDF(T)$ is obtained by dividing the total number of reports by the number of reports containing the term $T$, and then taking the logarithm of the obtained quotient. The calculation formula of TF-IDF is:

$$TF-IDF = TF(T, R) \times IDF(T)$$  \hspace{1cm} (1)

Through the above formula, the text description of report $R$ can be expressed as a TF-IDF vector, i.e., $R = (W_1, W_2, ..., W_i)$, where $W_i$ represents the TF-IDF value of the $i$-th item in the report $R$.

**Word Embedding Feature.** Word embedding is a feature learning technology in natural language processing. A single word is represented as an absolute number vector in a predefined vector space, and each word is mapped to a vector [16]. We use the Word2Vec model to obtain the word embedding of a report. Word2Vec is one of the methods of word embedding. The trained word embedding model will convert each word into a 100-dimensional vector. A crowdsourced test report contains multiple words, which can be converted into a matrix. Each row of the matrix represents a term in the report. Then, we convert the report matrix into a vector by averaging all word vectors contained in the report.

Specifically, given a reporting matrix with a total of $M$ rows, the $i$-th row of the matrix is denoted as $r_i$, and the converted report vector $R_w$ is generated as follows:

$$R_w = \frac{\sum_i r_i}{M}$$  \hspace{1cm} (2)

According to the above formula, each crowdsourced test report can be expressed as a word embedding vector.

**One-hot Encoding Feature.** One-hot encoding is also known as one-bit effective encoding. One-hot encoding is a word represented by a vector of length $V$, where only one position is 1, and the rest are 0. $V$ is the size of the lexicon in the corpus. We construct a corpus based on the collected crowdsourced test reports and convert each report into a report vector encoded by 0 and 1.

### C. PRIORITIZATION STRATEGY

In this section, we use four prioritization strategies to prioritize test reports. These four strategies are total greedy algorithm, additional greedy algorithm, genetic algorithm, and ART.

**Total Greedy Algorithm.** The total greedy algorithm is widely used to solve the test case prioritization problem, which focuses on always selecting the current best test case during the test case prioritization period. The goal of the total greedy algorithm is to select tests that contain more sentences and prioritize the test cases according to the descending order of the sentences covered by each test case. The total greedy algorithm directly calculates the program entity coverage of each test case and sorts it from high to low. Assuming that there are $m$ test cases and $n$ program entities that need to be covered, the time complexity of this strategy is $O(mn)$. For this strategy, we use the one-hot encoding feature to prioritize reports.

**Additional Greedy Algorithm.** The additional greedy algorithm introduces a feedback mechanism. This algorithm believes that when the executed test cases cover some program entities, the remaining test cases do not need to be considered for coverage of the above program entities. Specifically, each time a test case is executed, the coverage information of the remaining test cases needs to be updated in real-time. When all program entities are covered, reset these program entities to uncovered and iteratively apply the above process to the remaining test cases. The time complexity of the strategy is $O(m^2n)$. This algorithm uses the one-hot encoding feature to prioritize reports.

The greedy method is a standard method to solve the test case prioritization problem. After collecting the historical coverage information of test cases, according to the coverage ability of each test case’s program entities (such as statements, branches, or functions), set the weight for it, and apply the greedy method to guide the ordering of test cases. The assumption is that improving the early coverage of program entities helps improve the early detection rate of defects. The total greedy and additional greedy algorithms focus on
searching for the optimal local solution for prioritization, so the prioritization result may not be optimal.

**Genetic Algorithm.** The genetic algorithm is an evolutionary algorithm. Its basic principle is to imitate the evolutionary law of “natural selection in competition and survival of the fittest” in Darwin’s biological evolution theory. The genetic algorithm encodes the problem parameters into chromosomes. It then uses iterative methods to perform selection, crossover, and mutation to exchange information about chromosomes in the population and finally evolve into chromosomes that meet the optimization goals. In this algorithm, each test sequence is encoded in an $n$-size array representing an instance of a chromosome. A set of test sequences is randomly generated as the initial individual in the initial step. Combine the selected individuals guided by the fitness function to generate a new generation. This process is iterative. The entire search process stops until the requirements are met. The feature of this algorithm for prioritization is a one-hot encoding feature.

The genetic algorithm belongs to search-based prioritization, characterized by finding the optimal solution under the guidance of a predefined fitness function in the search space.

**ART.** ART was proposed by Jiang et al. in 2009. Its primary purpose is to prioritize a given set of test cases. It iteratively builds a set of candidate test cases and then selects a test case from the candidate set until all the given cases are selected. It calls a generated process to build the candidate set. The generation process iteratively constructs a set of unselected test cases by randomly adding the remaining test cases to the candidate set, as long as they can increase the program’s coverage and the candidate set is not complete yet. Another procedure selection function needs to be called to decide which candidate test case to choose. The latter requires a function $f_1$ to calculate the distance between a pair of test cases and another function $f_2$ to return the index of the selected test case furthest from the priority set. The function $f_1$ uses the Jaccard distance between two test cases based on the coverage structure. For a test suite with $m$ test cases and $n$ sentences, the time complexity of the algorithm is $O(n^2)$ in the best case and $O(m^3n)$ in the worst case. We use the TF-IDF feature, word embedding feature, and one-hot encoding feature for the ART algorithm.

**IV. EXPERIMENT**

In this section, we evaluate the effectiveness of four test report prioritization algorithms. We introduce the experimental settings and the baseline method compared with the four prioritization algorithms. We also set up two research questions (RQ). Finally, we introduce the evaluation metric.

**A. EXPERIMENT SETUP**

To evaluate the four prioritization algorithms, we design an empirical experiment. We collect 723 crowdsourced test reports from 5 different mobile applications to complete the experiment. Table 2 shows the detailed information. The labels of these applications range from A1 to A5, and the number of test reports for different applications ranges from 39 to 338.

**B. BASELINE**

We select two baselines for comparison to verify the effectiveness of four prioritization methods. The first baseline is the Random strategy, which is widely used in software testing. Random strategy randomly selects a test report for...
TABLE 2. Experiment Application

| App | App Category | Reports |
|-----|--------------|---------|
| A1  | Education    | 338     |
| A2  | Finance      | 145     |
| A3  | Music        | 41      |
| A4  | Reading      | 39      |
| A5  | Life         | 160     |

sorting, forming a priority ranking list. The second baseline is the Best strategy, which is an ideal prioritization technique. To compare these prioritization methods fairly, we repeat the experiment 10 times to collect experimental data.

C. RESEARCH QUESTION
In the experiment, we design the following two research questions.

- **RQ1**: How effective are these four prioritization algorithms?
- **RQ2**: How far are these four methods from the Best strategy?

When the priority method is not applied, testers will apply the Random strategy to prioritize the test reports. Only when the prioritization method can exceed the Random strategy will testers use it. RQ1 evaluates the effectiveness of four prioritization methods.

In practice, it is difficult to design effective methods in all situations. Therefore, it is valuable to understand the gap between the current four methods and the theoretically Best method. RQ2 evaluates the room for improvement in the four prioritization methods.

D. EVALUATION METRIC
In order to measure the effectiveness of four prioritization methods, we adopt APFD (Average Percentage of Failure Detection) [17]. Test report prioritization simulates the ideal test case prioritization, and APFD is an evaluation metric that is widely used in test case prioritization [18]. Therefore, we use APFD to evaluate the effectiveness of four test report prioritization methods. We will record the index of the first test report that showed it for each bug. According to the order of the test reports and which test reports show which fault information, we can calculate the APFD value to measure the effectiveness of the prioritization method. The value of APFD ranges from 0 to 1. The higher the APFD value, the better the prioritization result. The calculation formula of APFD is as follows,

$$APFD = 1 - \frac{\sum_{i=1}^{M} T_{fi}}{n \times M} + \frac{1}{2 \times n}$$  \hspace{1cm} (3)

where \(n\) is the number of test reports, \(M\) is the total number of bugs shown in the test report, and \(T_{fi}\) is the index of the first test report showing the bug \(i\).

V. RESULT ANALYSIS
In this section, we analyze the experimental results to answer RQ1 and RQ2. The results of all prioritization methods are shown in Figure 3. Figure 3 shows a box plot of APFD results for ten experimental runs of five applications. The application label is displayed on the horizontal axis, and the APFD value is displayed on the vertical axis.

A. ANSWER TO RQ1
RQ1: How effective are these four prioritization algorithms?

According to the results shown in Figure 3, we can find that these four priority ranking methods are better than Random. For the greedy strategy, the average APFD calculated by the additional algorithm is slightly better than the total on these five applications. There is little difference in the results obtained by using three different features to prioritize the ART strategy. On the two applications, A2 and A5, the calculated average APFD value is the same. In the remaining three applications, the calculated average APFD values are within 3%. In three of these strategies, the average APFD value of the genetic algorithm in each application is above 0.9. In addition, the box plot and Table 3 show that these four methods are more stable than the random method. In summary, we found that these four prioritization methods can improve the effectiveness of the prioritization of test reports.

B. ANSWER TO RQ2
RQ2: How far are these four methods from the Best strategy?

In order to avoid the contingency of the experimental results, we run the random method ten times on the test report data set of each application and calculate the average APFD. Table 4 to 9 shows the comparison results of the four prioritization strategies and the baseline method. In all the tables, "Improvement" means the improvement of the method compared to Random. The calculation formula is \(\frac{X_{\text{Best}} - X_{\text{Random}}}{X_{\text{Random}}}\), where \(X\) is our method. "Gap" means the gap between the method and the Best. The calculation formula is \(\frac{X_{\text{Random}} - X_{\text{Best}}}{X_{\text{Best}}}\), where \(X\) is our method.

As can be seen from the table, we observe that the results of these four strategies are better than random and close to the best results. Among the two greedy strategies algorithms, the additional greedy algorithm performs better than the total greedy algorithm in these five applications, and the gap with the Best is smaller. For the ART algorithm, we used three features. It can be seen from the results that the APFD calculated using these three features is not very different. Overall, the APFDs calculated by the four strategies are similar, and all are better than randomly ordered test reports.

| Application | Best | Random |
|-------------|------|--------|
| A1          | 0.9512 | 0.9119 |
| A2          | 0.9526 | 0.9321 |
| A3          | 0.9098 | 0.8551 |
| A4          | 0.9513 | 0.8841 |
| A5          | 0.9665 | 0.9157 |
**Figure 3.** Experimental Results

(a) Total Greedy

(b) Additional Greedy

(c) Genetic

(d) ART(TF-IDF)

(e) ART(Word Embedding)

(f) ART(One-hot Encoding)
TABLE 4. Total Greedy

| Application | Average APFD | Improvement | Gap |
|-------------|--------------|-------------|-----|
| A1          | 0.93         | 1.97%       | 2.30% |
| A2          | 0.91         | 0.67%       | 5.02% |
| A3          | 0.84         | 1.63%       | 8.17% |
| A4          | 0.89         | 0.41%       | 7.17% |
| A5          | 0.90         | 4.34%       | 7.98% |

TABLE 5. Additional Greedy

| Application | Average APFD | Improvement | Gap |
|-------------|--------------|-------------|-----|
| A1          | 0.95         | 4.17%       | 0.13% |
| A2          | 0.95         | 1.93%       | 0.26% |
| A3          | 0.89         | 4.52%       | 1.80% |
| A4          | 0.91         | 3.83%       | 3.63% |
| A5          | 0.93         | 5.79%       | 3.79% |

TABLE 6. Genetic

| Application | Average APFD | Improvement | Gap |
|-------------|--------------|-------------|-----|
| A1          | 0.92         | 1.43%       | 2.84% |
| A2          | 0.94         | 1.05%       | 1.10% |
| A3          | 0.90         | 5.50%       | 0.85% |
| A4          | 0.91         | 3.48%       | 3.98% |
| A5          | 0.93         | 1.73%       | 3.35% |

TABLE 7. ART(One-hot Encoding)

| Application | Average APFD | Improvement | Gap |
|-------------|--------------|-------------|-----|
| A1          | 0.95         | 3.61%       | 0.67% |
| A2          | 0.95         | 2.18%       | 0.01% |
| A3          | 0.86         | 3.59%       | 2.65% |
| A4          | 0.94         | 5.86%       | 1.62% |
| A5          | 0.97         | 3.21%       | 0.08% |

TABLE 8. ART(TF-IDF)

| Application | Average APFD | Improvement | Gap |
|-------------|--------------|-------------|-----|
| A1          | 0.95         | 4.01%       | 0.29% |
| A2          | 0.95         | 1.53%       | 0.64% |
| A3          | 0.87         | 1.51%       | 4.02% |
| A4          | 0.94         | 6.79%       | 0.76% |
| A5          | 0.97         | 3.48%       | 0.18% |

TABLE 9. ART(Word Embedding)

| Application | Average APFD | Improvement | Gap |
|-------------|--------------|-------------|-----|
| A1          | 0.92         | 1.34%       | 2.93% |
| A2          | 0.95         | 2.34%       | 0.14% |
| A3          | 0.88         | 3.42%       | 2.88% |
| A4          | 0.91         | 3.19%       | 4.72% |
| A5          | 0.97         | 3.37%       | 0.07% |

C. THREATS TO VALIDITY

In our research, there are some general threats to its effectiveness.

Crowdsourced worker capacity. The capacity of the crowdsourced workers is not controlled, so there may be some low-quality reports. However, these four strategies identified the errors described in the reports even when some reports were of low quality. If the test report contains no errors, these policies will classify the report into a single category and will not affect the priority of other reports.

Experimental software selection. We selected five applications widely used on the Internet that are not specifically designed for our research. Due to time and cost constraints, we only collected crowdsourced test reports for these five applications. The amount and type of software may threaten the generalization of our conclusions. However, there are different types of software in our choice, which may reduce the threat to a certain extent.

Test report text. In our experiment, all test reports are written in Chinese, which may threaten to apply these methods to other natural language test reports. However, the current natural language processing technology is very mature, and there are different toolkits for processing different languages. Moreover, our preprocessing operations do not limit the type of language. In summary, we adopt different toolkits for different languages to deal with them, which can reduce this threat.

VI. RELATED WORK

In this section, we introduce the work related to our research, including crowdsourced testing, crowdsourced test report processing, and test case prioritization.

A. CROWDSOURCED TESTING

Crowdsourcing refers to a company or organization that outsources work tasks performed by employees in the past to non-specific and usually large mass networks in a free and voluntary manner. Since crowdsourcing can simulate different conditions of use and the economic cost is relatively low [19] [20], crowdsourcing has been applied in many fields.

Crowdsourced testing is a relatively new trend in software engineering [21]. Liu et al. [22] studied the results of applying crowdsourced testing to usability testing and also examined empirical comparisons between crowdsourced usability testing and more traditional face-to-face usability testing results. Through experiments, they found that crowdsourcing exhibits some limitations compared to traditional lab environments, but its applicability and value in usability testing are clear. In addition to applying crowdsourced testing to usability testing, Dolstra et al. applied crowdsourced testing to GUI testing by instantiating a virtual machine running the system under test and giving testers access to the virtual machine through their web browsers. Outsourced to a large number of workers from the Internet [23]. Nebeling et al. [24] developed a toolkit for crowdsourced testing of web interfaces that not only efficiently recruits large numbers of...
Another part of the research focuses on the selection of crowdsourced workers. Not all workers are qualified to perform a particular test task. Cui et al. [35] proposed a multi-objective population worker selection method (MOOSE), which utilizes the widely used multi-objective evolutionary algorithm NSGA-II to optimize three objectives when selecting workers. ExReDiv [36], a novel hybrid method consisting of an empirical strategy, a correlation strategy, and a diversity strategy, is used to select a set of workers for testing tasks. Furthermore, Xie et al. [37] propose a new research problem called Crowdsourced Test Quality Maximization under Context Coverage Constraints.

The above researches apply crowdsourced testing methods to traditional software testing activities to solve some problems in traditional software testing. Instead, we propose to migrate the test case prioritization method to test report prioritization to solve the problem in crowdsourced testing.

### B. CROWDSOURCED TEST REPORT PROCESSING

In crowdsourced testing, a large number of test reports will be generated, which requires developers to spend a lot of time checking, and the key issue is to improve the efficiency of developers reviewing test reports. There has been extensive research on handling crowdsourced test reports to better assist developers in reviewing reports.

Wang et al. [25] proposed a clustering-based classification method that first clusters similar reports together and then uses an ensemble method to build a classifier based on the most similar clusters, which can reduce the effort required for manual inspection and facilitate project management in crowdsourced testing. This is the first work to address the classification of test reports in practical industrial crowdsourcing testing practice. Wang et al. [26] proposed Local Based Active Classification (LOAF) to classify accurate faults in crowdsourced test reports. LOAF recommends a small set of the most informative instances in the local neighborhood, asks the user their labels, and then learns a classifier based on the local neighborhood. Furthermore, Chen et al. proposed a new framework named Test Report Augmentation Framework (TRAF) to improve the quality of inspection test reports [27]. They enhance test reports with additional useful information in repeated test reports, which can help developers better understand and fix bugs. Sun et al. [28] used an information retrieval model to more accurately detect duplicate bug reports.

The above methods include report classification and duplicate detection, all of which select part of the test report to represent all the test reports. In addition to this, some studies have attempted to assist in the generation of test reports. Liu et al. used the spatial pyramid matching technique to measure similarity and extract features from screenshot images to generate descriptions for screenshots, which could allow developers to understand mobile crowdsourced test reports better [29]. Yu et al. generate crowdsourced bug reports by analyzing crowdsourced uploaded error screenshots using image understanding techniques to provide operational guidance for crowdsourcing [30].

### C. TEST CASE PRIORITIZATION

Test case prioritization is proposed in regression testing to handle the execution order of test cases by scheduling to detect more failures earlier [31]. Rothermel et al. [32] proposed various basic test case prioritization techniques, including total techniques and additional techniques. Zhang et al. [33] [34] proposed a unified priority model that uses a probabilistic model to bridge the gap between total and additional algorithms.

Li et al. [38] applied a meta-heuristic search-based algorithm to test case prioritization, and they applied the steepest hill-climbing algorithm and genetic algorithm. In addition, Zhang et al. [39] first applied Integrated linear programming to solve the time-aware test case prioritization problem. Nguyen et al. [40] proposed an information retrieval-based method to prioritize test cases for web services, which used an identifier document extracted from the execution trace to represent each test case and a web service change description as input queries for information retrieval. Tonella et al. [41] proposed a test case prioritization method based on machine learning, using the priority indicators defined by user use cases, as well as test case information such as coverage and failure propensity indicators as features, to train a model to predict test cases priority. Ledru et al. [42] used string distance comparisons on test case texts to prioritize test cases and provided empirical results on different distances. In addition to this, [43] proposed ESBS, which iteratively selects test cases from each cluster and uses spectral information in cluster test selection.

Test case prioritization techniques have matured, which inspired us to migrate test case prioritization techniques into test report prioritization. Most test case prioritization techniques use execution profiles, whereas we use test reports in natural language. Test case prioritization enables more failures to be detected earlier by executing test cases. Prioritizing test reports can improve tester review efficiency by checking test reports.

### VII. CONCLUSION

We employ a diversity strategy to prioritize test reports to help developers prioritize and filter useful report information. We process the text in the test reports using NLP techniques and use three different features to represent each report. These three features are the one-hot encoding, TF-IDF, and word embedding. After obtaining the feature vector, we use the total greedy algorithm, additional greedy algorithm, genetic algorithm, and ART to prioritize the test report. Five mobile crowdsourced test report datasets are used to evaluate the effectiveness of the test report prioritization method. The results of the empirical study show that these four prioritization methods all have good performance and can effectively help developers reduce inspection costs.
In the future, we will continue to dig into the adaptive test report prioritization methods, not only by borrowing techniques for prioritizing test cases. In addition, we will also try to design a new test report prioritization technology, starting from the test report itself, analyzing the images and text information contained in the test report, and extracting the corresponding features for report prioritization. And we will apply these techniques to actual production to solve test report-related tasks.

ACKNOWLEDGMENT
This work is supported partially by the Special Fund for Military-civilian Integration Development of Hebei Province (JMYB-2020-01) and the Key Project of Natural Science Research in Anhui Higher Education Institutions (KJ2019ZD67).

REFERENCES
[1] Mao K, Capra L, Harman M, et al. “A survey of the use of crowdsourcing in software engineering,” Journal of Systems and Software, 2017, 126: 57-84.
[2] Wu W, Tsai W T, Li W. “An evaluation framework for software crowdsourcing,” Frontiers of Computer Science, 2013, 7(5): 694-709.
[3] Lou Y, Chen J, Zhang L, et al. “A survey on regression test-case prioritization,” Advances in Computers. Elsevier, 2019, 113: 1-46.
[4] B. Jiang, Z. Zhang, W. K. Chan and T. H. Tse, “Adaptive Random Test Case Prioritization,” 2009 IEEE/ACM International Conference on Automated Software Engineering, 2009, pp. 233-244, doi: 10.1109/ASE.2009.77.
[5] Mikolov T, Sutskever I, Chen K, et al., “Distributed representations of words and phrases and their compositionality,” Advances in neural information processing systems. 2013: 3111-3119.
[6] Salton G, Yu C T. “On the construction of effective vocabularies for information retrieval,” ACM Sigplan Notices, 1973, 10(1): 48-60.
[7] G. Rothermel, R. H. Untch, Chengyun Chu and M. J. Harrold, “Prioritizing test cases for regression testing,” in IEEE Transactions on Software Engineering, vol. 27, no. 10, pp. 929-948, Oct. 2001, doi: 10.1109/32.962562.
[8] Howe J. “The rise of crowdsourcing,” Wired magazine, 2006, 14(6): 1–4.
[9] Estellés-Arolas E, González-Ladrón-de-Guevara F. “Towards an integrated crowdsourcing definition,” Journal of Information science, 2012, 38(2): 189-200.
[10] Zhang X, Feng Y, Liu D, et al., “Research progress of crowdsourced software testing,” Journal of Software, 2018, 29(1): 69-88.
[11] J.A. Jones, M.J. Harrold, “Test-suite reduction and prioritization for modified condition/decision coverage,” IEEE Transactions on software engineering, 2003, 39(3): 195-209.
[12] Lou Y, Chen J, Zhang L, et al., “A survey on regression test-case prioritization,” Advances in Computers. Elsevier, 2019, 113: 1-46.
[13] S. Foo and H. Li, “Chinese word segmentation and its effect on information retrieval,” Information processing & management, 40(1):161–190, 2004.
[14] L. R. Rabiner, “A tutorial on hidden markov models and selected applications in speech recognition,” Proceedings of the IEEE, vol. 77, no. 2, pp. 257–286, 1989.
[15] Che W, Li Z, Liu T, “Ltp: A chinese language technology platform,” Coling 2010: Demonstrations. 2010: 13-16.
[16] T. Mikolov, I. Sutskever, K. Chen, G. Corrado, J. Dean, “Distributed representations of words and phrases and their compositionality,” Advances in neural information processing systems, 2013, pp. 3111–3119.
[17] G. Rothermel, R. H. Untch, C. Chu, and M. J. Harrold. “Prioritizing test cases for regression testing,” IEEE Transactions on Software Engineering, 27(10):929–948, 2001.
[18] Ledru Y, Petrenko A, Boroday S, “Using string distances for test case prioritization,” 2009 IEEE/ACM International Conference on Automated Software Engineering, IEEE, 2009: 510-514.
[19] Gao R, Wang Y, Feng Y, et al., “Successes, challenges, and rethinking—an industrial investigation on crowdsourced mobile application testing,” Empirical Software Engineering, 2019, 24(2): 537-561.
[20] Mao K, Capra L, Harman M, et al., “A survey of the use of crowdsourcing in software engineering,” Journal of Systems and Software, 2017, 126: 57-84.
[21] Alyahya S. Crowdsourced software testing: A systematic literature review, “Information and Software Technology,” 2020, 127: 106363.
[22] Liu D, Bias R G, Lease M, et al., “Crowdsourcing for usability testing,” Proceedings of the American Society for Information Science and Technology, 2012, 49(1): 1-10.
[23] E. Dolstra, R. Vliegendhart and J. Pourselle, “Crowdsourcing GUI Tests,” 2013 IEEE Sixth International Conference on Software Testing, Verification and Validation, pp. 332-341, doi: 10.1109/ICST.2013.68.
[24] Nebeling M, Speicher M, Grossniklaus M, et al., “Crowdsourced web site evaluation with crowdsudy,” International Conference on Web Engineering. Springer, Berlin, Heidelberg, 2012: 494-497.
[25] Wang J, Cui Q, Wang Q, et al., “Towards effectively test report classification to assist crowdsourced testing,” Proceedings of the 10th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement. 2016: 1-10.
[26] Wang J, Wang S, Cui Q, et al., “Local-based active classification of test report to assist crowdsourced testing,” Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering. 2016: 190-201.
[27] Chen X, Jiang H, Chen Z, et al., “Automatic test report augmentation to assist crowdsourced testing,” Frontiers of Computer Science, 2019, 13(5): 943-959.
[28] Sun C, Lo D, Wang X, et al., “A discriminative model approach for accurate duplicate bug report retrieval,” Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering-Volume 1, 2010: 45-54.
[29] Liu D, Zhang X, Feng Y, et al., “Generating descriptions for screenshots to assist crowdsourced testing,” 2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER). IEEE, 2018: 149-156.
[30] Yu S. “Crowdsourced report generation via bug screenshot understanding,” 2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE). IEEE: 2019: 1277-1279.
[31] W.E. Wong, J.R. Horgan, S. London, H. Agrawal, “A study of effective regression testing in practice,” International Symposium on Software Reliability Engineering, 1997, pp. 264–274.
[32] G. Rothermel, R.H. Untch, C. Chu, M.J. Harrold, “Test case prioritization: an empirical study,” IEEE International Conference on Software Maintenance, IEEE, 1999, pp. 123–133.
[33] D. Hao, L. Zhang, L. Zhang, G. Rothermel, H. Mei, “A unified test case prioritization approach,” ACM Trans. Softw. Eng. Methodol. 24 (2) (2014) 10.
[34] L. Zhang, D. Hao, L. Zhang, G. Rothermel, H. Mei, “Bridging the gap between the total and additional test-case prioritization strategies,” Proceedings of the 2013 International Conference on Software Engineering, IEEE Press, 2013, pp. 192-201.
[35] Cui Q, Wang S, Wang J, et al., “Multi-Objective Crowd Worker Selection in Crowdsourced Testing,” SEKE, 2017: 17: 218-223.
[36] Cui Q, Wang J, Yang G, et al., “Who should be selected to perform a task in crowdsourced testing?,” 2017 IEEE 41st Annual Computer Software and Applications Conference (COMPSAC). IEEE, 2017: 75-84.
[37] Xie M, Wang Q, Yang G, et al., “Cocoon: Crowdsourced testing quality maximization under context coverage constraint,” 2017 IEEE 28th International Symposium on Software Reliability Engineering (ISSRE). IEEE, 2017: 316-327.
[38] Z. Li, M. Harman, R.M. Hierons, “Search algorithms for regression test case prioritization,” IEEE Trans. Softw. Eng. 33 (4) (2007) 225–237.
[39] L. Zhang, S.-S. Hou, C. Guo, T. Xie, H. Mei, “Time-aware test-case prioritization using integer linear programming,” Proceedings of the Eighteenth International Symposium on Software Testing and Analysis, ACM, 2009, pp. 213–224.
[40] C.D. Nguyen, A. Marchetto, P. Tonella, “Test case prioritization for audit testing of evolving web services using information retrieval techniques,” 2011 IEEE International Conference on Web Services, IEEE, 2011, pp. 636–643.
[41] P. Tonella, P. Avesani, A. Susi, “Using the case-based ranking methodology for test case prioritization,” IEEE International Conference on Software Maintenance, 2006, pp. 125–133.
[42] Ledru Y, Petrenko A, Boroday S, “Using string distances for test case prioritisation,” 2009 IEEE/ACM International Conference on Automated Software Engineering. IEEE, 2009: 510-514.
PENGHUA ZHU was born in Handan City, Hebei Province, China in 1980. He works at North China Institute of Aerospace Engineering and is the principal of the School of Computer. His research interests are software engineering and computer application technology. His title is senior experi-
menter.

YING LI was born in Langfang City, Hebei Province, China in 1982. Her title is associate pro-
fessor. Her research direction is software testing, software quality assurance and computer application technology.

TONGYU LI received the B.E. degree in computer science and technology from Northwestern Poly-
technical University in 2016. She is currently pur-
suing the master degree in software engineering at Nanjing University.

HUIMIN REN was born in Hohhot, Inner Mon-
golia, China, in 1981. She is currently working as a senior engineer at the Beijing Aerospace Automatic Control Institute. She is also the deputy director of the System Software Engineering Re-
search Department of Beijing Aerospace Auto-
matic Control Institute. Her research interests in-
clude software engineering and computer application technology.

XIAOLEI SUN, professor, graduated from Hefei University of Technology with a bachelor’s degree in automation and a master’s degree in control theory and control engineering. He is currently a member of the party committee and vice president of Wuhu Vocational and Technical College. He has been engaged in electronic information and vocational education research for a long time, and has published more than 20 high-level academic papers and authorized 6 invention patents.