The Journal of Systems & Software 204 (2023) 111799

Contents lists available at ScienceDirect

The Journal of Systems & Software

journal homepage: www.elsevier.com/locate/jss

An effective fault localization approach based on PageRank and mutation analysis☆

Yue Yan, Shujuan Jiang *, Yanmei Zhang, Cheng Zhang

Engineering Research Center of Mine Digitalization (China University of Mining and Technology), Ministry of Education, Xuzhou 221116, China
School of Computer Science and Technology, China University of Mining and Technology, Xuzhou 221116, China

ARTICLE INFO

Article history:
Received 30 September 2022
Received in revised form 18 March 2023
Accepted 1 July 2023
Available online 4 July 2023

Keywords:
Software fault localization
Mutation-based fault localization
PageRank algorithm
Mutation analysis

ABSTRACT

Mutation-based fault localization (MBFL) is a popular method based on mutation testing. MBFL applies a variety of operators to generate mutants and calculates the statement's suspiciousness by counting the execution results of the test cases on the mutants. However, the tie problem of MBFL creates obstacles to accurate fault localization. The tie problem refers to that many statements have the same suspiciousness. To solve the tie problem, we propose a fault localization approach based on the PageRank algorithm and mutation analysis (PRMA). We first apply the PageRank algorithm to calculate the faultiness scores of the statements. Then, we weight the suspicious value of the statements with faultiness scores to solve the tie problem. Finally, the weighted suspicious values are sorted in descending order to generate a list, which is provided to developers for fault localization. To evaluate our approach, we conduct experiments on the real fault benchmark Defects4J and the artificial fault dataset Siemens. We compare PRMA with the traditional MBFL techniques (Metallaxis and MUSE) and recently proposed MBFL methods (MCBFL-hybrid-avg, SMFL and SMBFL). The experimental results show that our approach outperforms above comparison methods and improves the effect of fault localization in both quantity and accuracy.

1. Introduction

Software testing is a critical stage in the software development process. The testing stage can improve the quality of the product and further meet the needs of target users (Wong et al., 2016). Software fault prediction, localization, and repair are essential parts in software testing. Fault localization refers to finding the element that caused the program to fail, and it is considered a costly part of the testing process (Pearson et al., 2017). Early fault localization mainly relied on the experience of developers or inserting breakpoints in the program and printing logs (Vessey, 1985). However, as the requirements for software functions become more and more diverse, software's scale gradually increases. It leads to time-consuming manual fault localization. Therefore, many researchers are exploring efficient semi-automated or automated software fault localization techniques.

Recently, many fault localization techniques have been continuously proposed and verified (Pearson et al., 2017). Mutation-based Fault localization (MBFL) is a widely studied technology. MBFL uses the mutation test to measure the effect of elements on program execution outcomes (Papadakis and Le Traon, 2012, 2015). By comparing mutation testing results to distinguish faults from normal elements, MBFL can find unknown faults more effectively (Papadakis and Le Traon, 2012). Specifically, mutation testing is a fault-based testing method that utilizes mutation operators to inject artificial faults into programs. Programs created by artificial faults are called mutants, which are killed if the results of the test case executed on the original program and the mutant are different. The researchers count the killed mutants to calculate the suspiciousness of the mutant (Jia and Harman, 2011), which is further used to calculate the suspicious values of elements. MBFL assists developers in fault localization by providing a list of program elements sorted by suspicious values.

Metallaxis is the most popular method (Papadakis and Le Traon, 2015) that applies mutation analysis to fault localization. It sets the maximum suspicious value of mutants as the suspiciousness of the statement. The MUSE (Moon et al., 2014) method averages the scores of the mutants as the suspicious value of the statement. Although the traditional MBFL methods use different strategies to calculate the suspiciousness of statements, they all only consider how to utilize single-dimensional mutation information for fault localization. At the same time, when the test suite is not comprehensive enough, the low number of killed mutants may lead to

☆ Editor: Laurence Duchien.

* Corresponding author at: Engineering Research Center of Mine Digitalization (China University of Mining and Technology), Ministry of Education, Xuzhou 221116, China.
E-mail address: shjjiang@cumt.edu.cn (S. Jiang).

https://doi.org/10.1016/j.jss.2023.111799
0164-1212/

statements with the same suspicious values since the Metallaxis, or MUSE methods use the number of killed or passed mutants to calculate the suspicious values of statements. Subsequently, some techniques have emerged that combine spectral coverage information and mutation information to improve the performance of fault localization (Pearson et al., 2017; Chaleshtari and Parsa, 2020; Cui et al., 2020). For instance, MCBFL-hybrid-avg (Pearson et al., 2017) averages the suspicious scores calculated by MBFL and Spectrum-Based Fault Localization (SBFL) (Abreu et al., 2006) as the new suspiciousness of the statement. However, there is also a statement binding problem in the SBFL method since the coverage matrix (spectrum) of the same statement block may be the same (Pearson et al., 2017). Therefore, it is difficult to solve the tie problem by combining SBFL and MBFL technologies for fault localization. It is necessary to fundamentally consider how to alleviate the tie problem in MBFL to improve the accuracy of fault localization. Research has shown that the maximum acceptable range for developers is to check the top five statements of the list for fault localization (Xie et al., 2013). When the suspicious values of multiple statements are the same, the probability of ranking the faulty statement at the front is random, which will greatly reduce the effect of fault localization.

To solve the tie problem of existing MBFL techniques, we propose PRMA, a fault localization approach that combines the PageRank algorithm (Page et al., 1998) and mutation analysis. The first application of the PageRank algorithm for fault localization was to improve SBFL (Zhang et al., 2017). Zhang et al. proposed to use the coverage graph in the PageRank algorithm to represent the relationship between test cases and methods and use faultiness scores to improve the effectiveness of SBFL. Inspired by the principle of the webpage ranking algorithm and previous works, we calculate the correlation between the statement and the execution result by simulating the test cases and statements as nodes in the webpage graph model. Unlike previous work, since mutation analysis is performed at the statement level, we perform fault localization at statement granularity. Webpage ranking algorithms mainly contain query-dependent and query-dependent algorithms. The query-dependent algorithm is based on the semantic information of keywords to recommend and rank the webpage. MBFL only consider the executive relationship between the statement and the test case instead of using textual or code similarity. Therefore, the query-dependent algorithm is unsuitable for combining with the MBFL method. We finally choose the widely used query-independent PageRank algorithm (Page et al., 1998) to further improve the effectiveness of fault localization. Specifically, we first exploit the coverage information in the spectrum to capture the connections between failed tests and statements, generating a coverage graph. Then, we construct the transition matrix between the test cases and the statements according to the coverage graph. The transition matrix is used to generate the statement's faultiness score in the PageRank algorithm. The faultiness score represents the association between the statement and the failed test case. Finally, we solve the tie problem by weighting the statements with faultiness scores and sorting them according to the weighted scores for fault localization.

To evaluate the performance of PRMA, we first conduct experiments on the real-world benchmark Defects4J (Just et al., 2014). Experimental results show that our approach outperforms the existing MBFL methods. For example, our method can locate 88 faults in *Top-1*, which is 27 more than Metallaxis. To investigate the generalizability of PRMA, we further evaluate our approach on the artificial fault benchmark Siemens (Do et al., 2005). The results show that at least 30% of artificial faults can be located in *Top-5*. Our contributions are summarized as follows:

(1) We propose an approach that combines PageRank and mutation analysis for fault localization. The tie problem is solved by weighting the suspicious value of the statement with the faultiness score calculated by the PageRank algorithm. Our approach improves the performance of fault location in terms of quantity and accuracy.
(2) We evaluate our approach using the real fault Defects4J dataset. Experimental results show that our method outperforms the traditional MBFL methods (Metallaxis and MUSE) and recent proposed MBFL techniques (MCBFL-hybrid-avg, SMFL and SMBFL).
(3) To evaluate the scalability of our approach, we further conduct experiments on the artificial fault benchmark Siemens. The experimental results show that the localization effect of PRMA on artificial faults is better than the traditional MBFL method.

2. Background

In this section, we first introduce the principle and development trend of mutation-based fault location technology. Then, we illustrate the implementation and application of the PageRank algorithm.

2.1. Mutation-based fault localization

Mutational analysis (Jia and Harman, 2011; Just et al., 2012) runs different mutants in a specific test suite and tries to identify how the mutants behave differently from the original program. Mutation-based fault localization methods (MBFL) use mutation analysis to identify suspicious mutants and use their locations for fault localization (Papadakis and Le Traon, 2015). MBFL assigns suspicious scores to mutants based on the assumption that test cases that kill mutants contribute more to fault localization accuracy. When the results of the test case produced by executing the original program and the mutant are different, the mutant is killed by the test case. Although there are differences between the existing MBFL techniques (Papadakis and Le Traon, 2015; Moon et al., 2014), they can be implemented in four steps:

(1) Collecting statement coverage information and test case execution results.
(2) Generating mutants with mutation operators and executing mutants using the test suite.
(3) Suspicious values for mutants and statements are calculated by counting the execution results of the test suite.
(4) Producing a list of statements sorted by suspicious value in descending order.

Developers can find the location of the fault by observing the list of suspicious statements provided by MBFL. The detailed description of the above four steps is as follows.

Firstly, We denote Program Under Test (PUT) and test suite by $PUT = \{e_1, e_2, \ldots, e_n\}$ and $T = \{t_1, t_2, \ldots, t_m\}$, respectively. We collect coverage information of the program and test execution results. T_f and T_p represent the sets of failed test cases and successful test cases, respectively.

Secondly, the statements covered by failed test cases are injected with artificial faults to form mutants by mutation operators. The mutation operators (Just et al., 2012; Siami Namin et al., 2008) are synthesized based on simple rules, such as $a < b$ will be mutated to $a > b$. Generally, a statement has at least one mutant and $M(e_i)$ represents the mutant set of e_i. The test cases are re-executed on each mutant $m \in M(e_i)$. If the results of the test case in the mutant and the original program are different, the test case kills the mutant. Test cases can be divided into two categories T_n

Table 1
The parameters of $Susp(m)$.

Parameter	Description
$a_{np} = \left\|T_n \cap T_p\right\|$	The number of passed test cases that without killing the mutant.
$a_{kp} = \left\|T_k \cap T_p\right\|$	The number of passed test cases that killed the mutant.
$a_{nf} = \left\|T_n \cap T_f\right\|$	The number of failed test cases that without killing the mutant.
$a_{kf} = \left\|T_k \cap T_f\right\|$	The number of failed test cases that killed the mutant.

and T_k, where T_n represents the set of test cases that do not kill mutants and T_k represents the set of test cases that kill mutants.

Thirdly, the suspicious value of mutant is calculated with formulas in SBFL, such as Ochiai (Abreu et al., 2006). The calculation is related to four parameters, as shown in Table 1. Taking the Ochiai formula as an example, the suspicious value of m is shown in Eq. (1), where $a_{kf} + a_{nf}$ can also be denoted as $\left|T_f\right|$. The suspiciousness of the statement is based on $Susp(m)$ in traditional MBFL approaches. For example, the suspicious value of statement e_i calculated by the Metallaxis (Papadakis and Le Traon, 2015) and MUSE (Moon et al., 2014) methods are shown in Eqs. (2) and (3), respectively.

$$Susp(m) = \frac{a_{kf}}{\sqrt{\left(a_{kf} + a_{nf}\right)\left(a_{kf} + a_{kp}\right)}} \tag{1}$$

$$Susp\_Metallaxis(e_i) = Max\left(Susp\left(m_1\right), \ldots, Susp\left(m_k\right)\right) \tag{2}$$

$$Susp\_MUSE(e_i) = \sum_{i=1}^{k} Susp\left(m_i\right)/k \tag{3}$$

Finally, MBFL sorts the statements in descending order based on suspicious values and generates a list of statements for developers.

2.2. PageRank algorithm

Due to the rapid development of the World Wide Web, to save users' time and effort in finding target information, an efficient search engine is expected to rank and recommend web pages according to users' preferences (Borodin et al., 2005). Accurate ranking and recommendation need to use the textual and structural information of the Web. In the last two decades, several link ranking algorithms have been proposed to efficiently extract the structural information of the Web (Farahat et al., 2006). In general, link ranking algorithms can be divided into query-independent algorithms (such as PageRank Page et al., 1998; F, 2015) and query-dependent algorithms (such as HITS Kleinberg, 1999 and SALSA Lempel and Moran, 2000). The query-dependent algorithm mainly focuses on textual similarity between web pages and keywords. In contrast, the MBFL does not use textual or code similarity between statement and test case for fault localization. So, the query-dependent algorithm is not considered in our approach. We choose the popular query-independent PageRank algorithm to improve the traditional MBFL techniques and the accuracy of fault localization.

PageRank (Page et al., 1998) was proposed by Larry Page and Sergey Brin to measure how important a particular webpage is relative to other web pages in search engines. It improves search quality and speed by optimizing the ranking of webpages in search engines. PageRank views the World Wide Web as a collection of web page nodes and ranks them according to their importance. The idea of PageRank is that for each node, if important nodes link it, then it should be more critical than nodes connected by non-influential nodes. The judgment of important nodes mainly relies on the following two assumptions.

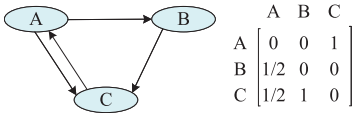


Fig. 1. A simple network and transition matrix.

(1) Quantity: In the network graph model, if a page receives more incoming links from other web pages, then the page is more important.
(2) Quality: The quality of incoming links pointing to web pages is different, and pages with high quality will pass more weight to other pages through links. Therefore, the higher the quality of the page points to this page, the more important this page is.

We use a simple example to introduce PageRank more intuitively. Fig. 1 presents a directed graph describing a small network with three web pages, represented by nodes A, B, C. An edge between two nodes indicates that the start node contains a link to the end node. We can observe that more edges point to C, so C should be more important than other nodes. Also, A is pointed by C, which should be assigned a high score to show its importance according to PageRank's assumptions. Formally, these pages are described by a directed graph $G = (V, E)$, which contains r nodes and l edges. As shown in Fig. 1, $P_{r\times r}$ represents the transition matrix of nodes. Each matrix element P_{ij} represents the probability of linking from node j to i, and its value is $1/(Outbound\ Link\ Number\ of\ Node\ j)$. For example, the outbound link number of node A is 2, and the P_{BA} is 1/2.

Then, we use the transition matrix to calculate each node's score, representing the importance of node. The PageRank score of node i depends on the PageRank scores of the nodes pointed to i. Therefore, the PageRank score of i is shown in Eq. (4). To make the equation more compact, we denote the PageRank score of each node by a vector $\vec{x}$ in Eq. (5), where P represents the transition matrix.

$$PR_i = \sum_{\forall j, j\to i} \frac{PR_j}{\text{Outbound Link Number of Node j}} \tag{4}$$

$$\vec{x} = P \cdot \vec{x} \tag{5}$$

Sometimes, a node may have no outbound links, and its PageRank score cannot be assigned to other nodes. Therefore, a teleportation vector $\vec{v}$ weighted by the damping coefficient d is added to Eq. (5) and $\vec{v}$ is a positive vector in Eq. (6). The meaning of d is the probability that a user reaches a specific page and continues to browse backwards at any time. Its value is estimated based on users' average frequency of browser bookmarks, which usually equals 0.7 (Zhang et al., 2017). When the network size increases, finding an exact solution for Eq. (6) in a reasonable time is not easy. Therefore Page et al. (1998) introduced an iterative method to obtain approximate solutions. The PageRank score for the kth iteration is shown in Eq. (7).

$$\vec{x} = d \cdot P\vec{x} + (1-d) \cdot \vec{v} \tag{6}$$

$$\vec{x}^{(k)} = d \cdot P\vec{x}^{(k-1)} + (1-d) \cdot \vec{v} \tag{7}$$

For the example network in Fig. 1, we use the default damping coefficient $d = 0.7$, the vector $\vec{v} = \left[\frac{1}{r}, \frac{1}{r}, \ldots, \frac{1}{r}\right]^T$, and the initial

Table 2
The program fragment and spectrum of motivating example.

Statement	t_1	t_2	t_3	t_4
mid (int x, int y, int z){	–	–	–	–
e_1: int m;	1	1	1	1
e_2: m=z;	1	1	1	1
e_3: if(y<z*(−1))	1	1	1	1
e_4: if (x<y)	1	0	0	0
e_5: m=y;	1	0	0	0
e_6: else if (x<z)	0	0	0	0
e_7: m=x;	0	0	0	0
e_8: else	0	1	1	1
e_9: if (x>y)	0	1	1	1
e_{10}: m=y;	0	1	0	0
e_{11}: else if (x>z)	0	0	1	1
e_{12}: m=x;	0	0	0	1
e_{13}: return m;	1	1	1	1
}	1	1	1	1

value of the $\vec{x}$ is $\vec{x}^{(0)} = \vec{v}$. After three iterations, the PageRank scores of nodes A, B, C are 0.304, 0.36 and 0.51, respectively. The PageRank score is consistent with node importance by observing. All other nodes point to C, which becomes the most critical node. A is the only node that C points to, so it is the second most important node in the network.

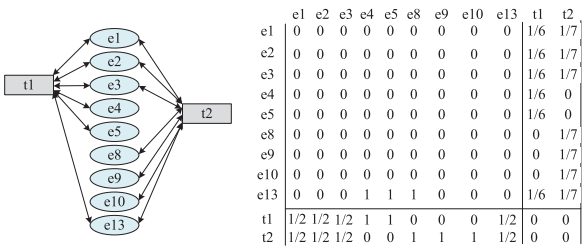
PageRank is also widely used in other fields. PageRank-based techniques have recently been proposed for analyzing software systems (F, 2015). For example, Bhattacharya et al. (2012) proposed the concept of NodeRank based on PageRank to measure the importance of nodes on static graphs for software analysis and fault prediction. Therefore, based on the idea and application of the PageRank algorithm, we apply the PageRank algorithm to solve the tie problem (the same suspicious value of multiple statements) in traditional MBFL technique.

3. Motivating example

In this section, we illustrate our research motivation with an example. Metallaxis (Papadakis and Le Traon, 2015) is the most widely used method of applying mutation analysis to fault localization. For this reason, we use the Metallaxis method to represent the traditional MBFL technique. As shown in Eq. (1), we use Ochiai to calculate the suspicious value of the mutant, which is the better-performing formula in SBFL (Xie et al., 2013). We use a simple example to illustrate how to use the faultiness scores generated by PageRank to improve the performance of fault localization.

As shown in Table 2, the function of the sample program is to calculate the median (column 1). We execute the program with four test cases: $t_1(-1, 0, -2)$, $t_2(2, 1, 7)$, $t_3(0, 0, 0)$ and $t_4(1, 2, 0)$ (columns 2–5). The failed test cases include t_1 and t_2, and the successful test cases contain t_3 and t_4. Table 2 shows the coverage information of the statement, "1" indicates that the corresponding test case covers the statement, and "0" indicates the test case does not cover the statement. As can be observed from the program, e_3 is the faulty statement because its conditional expression should be $if(y < z)$, but it was mistakenly written as $if(y < z * (-1))$.

As shown in the program, the main execution statements of the *mid* method are the branch statements, so we mainly perform mutation analysis on the branch statements. As shown in Table 3, we mutated e_3, e_4, e_6, e_9, e_{11} by mutation operators (column 1). Each statement generated four mutants, denoted by m_i (column 2). We represent mutants that are killed by K and those that are not killed by N (columns 3–6). a_{kp} and a_{kf} denote the number of passed and failed test cases that killed the mutant (columns 7–8), respectively. Finally, we calculate the suspiciousness of the



	e1	e2	e3	e4	e5	e8	e9	e10	e13	t1	t2
e1	0	0	0	0	0	0	0	0	0	1/6	1/7
e2	0	0	0	0	0	0	0	0	0	1/6	1/7
e3	0	0	0	0	0	0	0	0	0	1/6	1/7
e4	0	0	0	0	0	0	0	0	0	1/6	0
e5	0	0	0	0	0	0	0	0	0	1/6	0
e8	0	0	0	0	0	0	0	0	0	0	1/7
e9	0	0	0	0	0	0	0	0	0	0	1/7
e10	0	0	0	0	0	0	0	0	0	0	1/7
e13	0	0	0	1	1	1	0	0	0	1/6	1/7
t1	1/2	1/2	1/2	1	1	0	0	0	1/2	0	0
t2	1/2	1/2	1/2	0	0	1	1	1	1/2	0	0

Fig. 2. The coverage graph and transition matrix of program.

mutant (column 9) with Eq. (1), which is denoted as $Susp(m_i)$. Then, we calculate the suspiciousness of the statement by Eq. (2), which is indicated as $Me(e_i)$ (column 10). The faultiness score and weighted suspicious value of a statement are denoted by *PR* (column 11) and *PR-Me* (column 12), respectively. We observe that the $Me(e_i)$ values of e_3, e_4 and e_9 are the same. The faulty statements (e_3) cannot be sorted at the top. After weighting by the *PR*, the *PR-Me* of e_3 is 0.743, which is higher than other statements. Therefore, we can rank the faulty statements at the top by applying the PageRank algorithm.

For the convenience of observation, we show the scores of the tied statements in Table 4. From Table 4, we can observe that e_3 cannot be directly found by Metallaxis method. Because the suspicious values for e_3, e_4 and e_9 are all equal to 0.71 (column 2), which is no better than random guessing. Therefore, we solve this tie problem by weighting $Me(e_i)$ with faultiness scores calculated by the PageRank algorithm. The higher the faultiness score, the more likely the statement causes the test case to fail. Then, we will use PageRank to calculate the faultiness scores of the statements.

Firstly, according to the coverage information in Table 2, we generate a bidirectional coverage graph between the failed test cases and the statements in Fig. 2. For brevity, when building the sample program coverage graph, we only focus on the suspicious statements covered by failed tests t_1 and t_2. From Fig. 2, we can see that t_1 covers six statements (e_1, e_2, e_3, e_4, e_5 and e_{13}), and t_2 covers seven statements (e_1, e_2, e_3, e_8, e_9, e_{10} and e_{13}). Our intuition is that e_3 is more likely to be the faulty statement. The first basis is that both test cases cover e_3. Furthermore, it can be observed from Table 3 that the $Me(e_i)$ value of e_3 is higher. According to the PageRank algorithm, we generate the transition matrix of test cases and statements according to the coverage graph, as shown in Fig. 2. Since there is no connection between the statements, the upper left of the matrix is all 0. Similarly, the bottom right of the matrix is also 0. The values in the matrix are the probability of connecting from node j to i. For example, 1/6 in the first row represents the probability that t_1 points to e_1.

Then, we use the calculation formula of the PageRank algorithm in Section 2 to calculate the faultiness scores of statements, which are denoted by the vector $\vec{x} = [Fe_1, Fe_2, Fe_3, Fe_4, Fe_5, Fe_8, Fe_9, Fe_{10}, Fe_{13}, Wt_1, Wt_2]^T$. The Fe_i in the vector represents the faultiness score of the corresponding statement. The Wt_i indicates the test ability of the test case, denoted by the vector $\vec{v} = [0, 0, 0, 0, 0, 0, 0, 0, 0, Wt_1, Wt_2]^T$. They can be calculated with the formula $Wt_i = c_i^{-1} / \sum c_j^{-1}$, where c_i is defined as the number of statements covered by the test case. For this example, $c_1 = 6$ and $c_2 = 7$, so $\vec{v} = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0.54, 0.46]^T$ according to the formula. We use the default $d = 0.7$ and $\vec{x}^{(0)} = \vec{v}$. After three iterations based on Eq. (7), we can get the vector $\vec{x} = [0.033, 0.033, 0.033, 0.019, 0.019, 0.014, 0.014, 0.014, 0.033, 0.26, 0.24]^T$. We can observe that the faultiness score of e_3 is higher than e_4 and e_9, indicating that e_3 is more closely related to the failed test, which is consistent with our intuition.

Table 3
Mutation analysis of motivating example.

Statement	Mutant	t_1	t_2	t_3	t_4	a_{kp}	a_{kf}	$Susp(m_i)$	$Me(e_i)$	PR	PR-Me
mid (int x, int y, int z){	–	–	–	–	–	0	0	0	0	0	0
e_1: int m;	–	–	–	–	–	0	0	0	0	0.033	0.033
e_2: m=z;	–	–	–	–	–	0	0	0	0	0.033	0.033
e_3: if(y<z*(−1))	m_1 :<→≥	N	K	N	K	1	1	0.5	0.71	0.033	0.743
	m_2 :<→==	K	N	N	N	0	1	0.71			
	m_3 :<→ *true*	N	K	N	K	1	1	0.5			
	m_4 :<→ *false*	N	N	N	N	0	0	0			
e_4: if (x<y)	m_5 :<→>	K	N	N	N	0	1	0.71	0.71	0.019	0.729
	m_6 :<→≥	K	N	N	N	0	1	0.71			
	m_7 :<→ *true*	N	K	N	N	0	0	0			
	m_8 :<→ *false*	K	N	N	N	0	1	0.71			
e_5: m=y;	–	–	–	–	–	0	0	0	0	0.019	0.019
e_6: else if (x<z)	m_9 :<→>	N	N	N	N	0	0	0	0	0	0
	m_{10} :<→≥	N	N	N	N	0	0	0			
	m_{11} :<→ *true*	N	N	N	N	0	0	0			
	m_{12} :<→ *false*	N	N	N	N	0	0	0			
e_7: m=x;	–	–	–	–	–	0	0	0	0	0	0
e_8: else	–	–	–	–	–	0	0	0	0	0.014	0.014
e_9: if (x>y)	m_{13} :>→<	N	N	N	K	1	0	0	0.71	0.014	0.724
	m_{14} :>→≤	N	N	N	K	1	0	0			
	m_{15} :>→ *true*	N	N	N	K	1	0	0			
	m_{16} :>→ *false*	N	K	N	N	0	1	0.71			
e_{10}: m=y;	–	–	–	–	–	0	0	0	0	0.014	0.014
e_{11}: else if (x>z)	m_{17} :>→<	N	N	N	K	1	0	0	0	0	0
	m_{18} :>→≤	N	N	N	K	1	0	0			
	m_{19} :>→ *true*	N	N	N	N	0	0	0			
	m_{20} :>→ *false*	N	N	N	K	1	0	0			
e_{12}: m=x;	–	–	–	–	–	0	0	0	0	0	0
e_{13}: return m;	–	–	–	–	–	0	0	0	0	0.033	0.033
}	–	–	–	–	–	0	0	0	0	0	0

Table 4
The suspicious scores of statements.

Statement	$Me(e_i)$	*PR*	*PR-Me*	*PR-Me**	Rank
e_3	0.71	0.033	0.743	1	1
e_4	0.71	0.019	0.729	0.26	2
e_9	0.71	0.014	0.724	0	3

$Me(e_i)$ denotes the suspiciousness calculated by Metallaxis.
PR represents the faultiness score of the statement.
PR-Me indicates the weighted score.
*PR-Me** represents the normalized value of *PR-Me*.

Finally, we weighted the suspicious values calculated by the Metallaxis of the statements with faultiness scores generated by PageRank. Specifically, as shown in Table 4, we add the $Me(e_i)$ (column 2) and *PR* (column 3) values to get the weighted suspiciousness, *PR-Me* (column 4). In order to facilitate the ranking of statements, we apply the Min–Max method (Rabinowitz, 1978) to normalize the weighted suspicious values to between 0 and 1. The table shows that the normalized suspicious value of e_3 is 1 (column 5). The fault statement can be ranked first for developers to check (column 6). This motivating example further illustrates that combining PageRank with mutation analysis can effectively improve the accuracy of fault localization.

4. Approach

This section illustrates our approach, which utilizes PageRank and mutation analysis for fault localization. The general framework of our approach is shown in Fig. 3. We apply the faultiness scores generated by the PageRank algorithm to break the tie problem in MBFL and promote the effectiveness of fault localization. Specifically, PRMA mainly includes three stages: mutation analysis, PageRank analysis and ranking. In the mutation analysis phase (Section 4.1), we first calculate the suspicious values of statements according to the formula of Metallaxis and MUSE methods. In the PageRank analysis phase (Section 4.2), PRMA collects the coverage information of statements by running test cases and builds a coverage graph. Based on the coverage graph, PRMA conducts a transition matrix to generate faultiness scores for statements. In the ranking phase (Section 4.3), PRMA utilizes the faultiness score to weight the suspicious values generated by Metallaxis and MUSE. The statements are sorted according to the weighted suspicious value and provided to the developer for fault localization.

4.1. Mutation analysis phase

This phase aims to perform a preliminary ranking of statements according to suspicious values. We implement two traditional MBFL methods at this stage: Metallaxis (Papadakis and Le Traon, 2015) and MUSE (Moon et al., 2014). Firstly, we mutate the statements to generate one or more mutants with operators. We re-execute the test suite on the mutant to obtain the killing information. Then, we calculate the suspicious value for each mutant using the Ochiai formula in the SBFL technique (Abreu et al., 2006). Finally, the suspiciousness of the statement is generated by the Metallaxis or MUSE methods. For each mutant, re-executing the test suite incurs high computational and time costs. Research has proved that only considering the statements covered by the failed test cases does not affect the performance of fault localization (Pearson et al., 2017; G et al., 2007). Therefore, we only use the statements covered by the failing test cases for mutation analysis. By applying Metallaxis (Eq. (2)) and MUSE (Eq. (3)) methods, we obtain preliminary suspicious values for statements. PRMA will weight the preliminary suspicious values

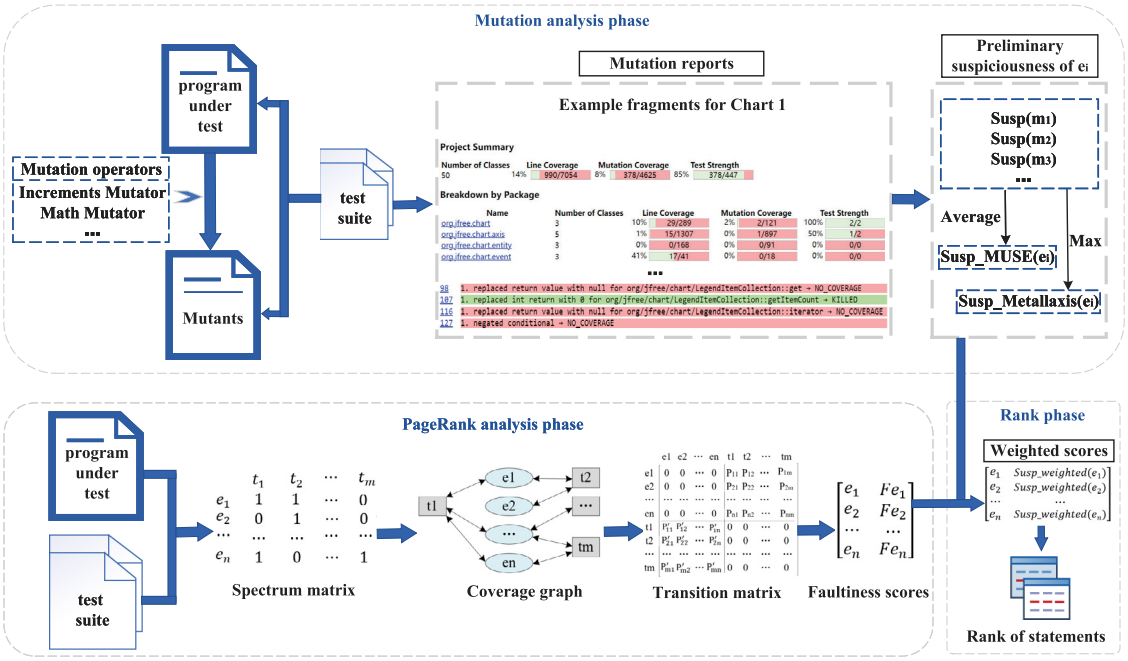


Fig. 3. The overall framework of our approach.

in the ranking phase to solve the tie problem in the traditional MBFL technique.

4.2. PageRank analysis phase

As shown in Fig. 3, the PageRank analysis phase aims to collect the execution results of the test cases to calculate the statement's faultiness score. We apply the faultiness score as another basis for judging the suspiciousness of a statement. We weight the suspicious statements provide by mutation analysis with the faultiness scores. Firstly, PRMA uses SBFL technology to generate coverage information of test cases and statements. Then, by mining the coverage relationship in the spectrum, PRMA further conducts the coverage graph of test cases and statements. The coverage graph is bidirectional and contains connections between test cases and statements. As shown in the example in Section 3, the failing test case t_1 covers the statement e_3, so a bidirectional edge is constructed between nodes t_1 and e_3. Next, we will analyze the constituent elements of the PageRank calculation formula (Eq. (7)). The average frequency of bookmarks by users determine the damping factor d, which is usually set as $d = 0.7$. We mainly describe the construction of the transition matrix P and the design of vector $\vec{v}$.

According to the coverage information between test case and statement, the transition matrix can be conducted as Eq. (8). P_{ET} and P_{TE} represent the transition between statements and tests. Since we do not consider the calling relationship between statements, P_{EE} is a zero matrix. Similarly, there is no connection relationship between test cases, so P_{TT} is also a zero matrix. P_{ET} and P_{TE} are constructed from the coverage graph of statements and test cases. The element P_{ij} in the matrix P_{ET} represents the probability that test case j points to statement i, and its value is $1/(Outbound\ Link\ Number\ of\ Node\ j)$. The element in the matrix P_{TE} denotes as P'_{ij} which indicates the probability that statement j points to test case i. P'_{ij} is calculated in the same way as P_{ij}.

$$P = \begin{bmatrix} P_{EE} & P_{TE} \\ P_{ET} & P_{TT} \end{bmatrix} \tag{8}$$

The teleportation vector $\vec{v}$ consists of two sub-vectors: $\vec{v} = \left[\vec{v}_e^T, \vec{v}_t^T\right]^T$, where $\vec{v}_e^T$ and $\vec{v}_t^T$ represent the teleportation vectors of the statement and the test case, respectively. Statements initially have the same probability of fault, so we set $\vec{v}_e$ to $\vec{0}$. Research shows that the weight of failed test cases with small coverage should be larger (Zhang et al., 2017). Therefore, the value of $\vec{v}_t$ is set according to the ability of the test case. $\vec{v}_t$ can be expressed as $[Wt_1, Wt_2, \ldots, Wt_m]^T$, where $Wt_i = c_i^{-1} / \sum c_j^{-1}$ and c_i is defined as the number of statements covered by the test case t_i. Furthermore, the initial value of $\vec{x}$ in the PageRank equation is set to $\vec{v}$.

Finally, PRMA substitutes the above parameters into Eq. (7) to generate the faultiness score of the statement. The faultiness score indicates the association of the statement with the failed test case. A higher score indicates that the statement is more likely to be faulty.

4.3. Ranking phase

The statements' preliminary suspicious values and faultiness scores are obtained in the first and second phases, respectively. We use the Metallaxis method to illustrate how to weight the preliminary suspicious values. As shown in Eq. (2), the preliminary suspicious value calculated by the Metallaxis method is expressed as $Susp\_Metallaxis(e_i)$. The faultiness scores are denoted as Fe_i. We add the preliminary suspicious value and faultiness score as the weighted suspicious value of the statement, as shown in Eq. (9). Then, we normalize the weighted suspicious values by the Min–Max method (Rabinowitz, 1978) before ranking. PRMA sorts the statements in descending order according to the normalized $Susp\_weighted(e_i)$ and provides the list to developers for fault localization.

$$Susp\_weighted(e_i) = Susp\_Metallaxis(e_i) + Fe_i \tag{9}$$

5. Evaluation

This section mainly describes the experimental settings and results analysis of PRMA. Firstly, we design five research questions to demonstrate the effectiveness of our method. Secondly, we describe the experimental subjects and implementation details of PRMA in detail in order to reproduce the experiments. Finally, we evaluate PRMA through four evaluation metrics and analyze the experimental results of PRMA.

5.1. Research questions

We evaluate the performance of PRMA for fault localization through the following five research questions. We illustrate the details of the RQs as follows.

RQ1: Compared with the traditional MBFL method, how effective is PRMA for fault localization?

PRMA aims to improve the performance of fault localization based on the traditional MBFL method. The statement's preliminary suspicious values (calculated by Metallaxis and MUSE) are weighted by the faultiness scores calculated by the PageRank algorithm. Therefore, we first compare PRMA with the Metallaxis and MUSE methods.

RQ2: How does PRMA compare with mutation-based fault localization methods proposed in recent years?

Mutation-based fault location methods are continuously proposed and validated. Many hybrid fault localization methods combining spectrum and mutation have been proposed, one of which is MCBFL-hybrid-avg (Pearson et al., 2017). MCBFL-hybrid-avg addresses the limitations of immutable statements in the MBFL approach using the mutant killed information and spectrum coverage information. Following their ideas, SMFL (Cui et al., 2020) is proposed, which combines the essence of SBFL and MBFL techniques for fault localization. In addition, SMBFL (Chaleshtari and Parsa, 2020) further explores program dynamic slicing technique to improve MBFL. In order to evaluate the effect of PRMA, we compare it with the above three mutation-based fault localization methods.

RQ3: What is the performance of PRMA in artificial fault localization?

To objectively evaluate our approach, we further explore the fault localization effect of PRMA on artificial faults to reduce the threat of external validity.

RQ4: How much does PRMA improve the effectiveness of fault localization?

We deeply analyze the improvement of PRMA in the quantity and accuracy of fault localization through this research question.

RQ5: What about the time cost of our approach?

We compare the time costs of PRMA and traditional MBLF methods (Metallaxis and MUSE). We also evaluated the time cost spent on the mutation and PageRank analysis phases.

5.2. Experimental subjects

To answer these research questions, we conduct experiment using the real fault dataset Defects4J (Just et al., 2014) and the artificial fault benchmark Siemens Suite (G et al., 2007). Defects4J is a mature dataset that has been widely used in software testing research (Cui et al., 2020; Zou et al., 2021; Li et al., 2019). For each fault, Defects4J provides a program for the fault, a repair program for code changes, and the test suite. Siemens Suite is considered a benchmark for evaluating the effectiveness of fault location techniques (Dutta and Godboley, 2021; Liu et al., 2017), providing a faulty version and a test suite.

Table 5 shows the statistics of Defects4J and Siemens datasets. In Table 5, column 1 presents the subjects, column 2 shows the number of faults in each subject, columns 3 and 4 show the number of lines of code and test cases, respectively. We can find that the number of test cases in the Defects4J benchmark is huge. As shown in Table 5, the Closure subject contains nearly eight thousand test cases. If we consider failed and passed test cases simultaneously, the transition matrix dimension in the PageRank algorithm will be huge, and the substantial computational and time costs will be large. Therefore, we only consider the transition matrix between failed test cases and statements in the PageRank analysis phase. We apply 395 faults (lines 2–8) from six commonly used subjects in Defects4J (v1.2.0), all of which are java programs. Furthermore, we applied three artificial fault C programs, which belong to Siemens suite and downloaded from the SIR repository (G et al., 2007) (lines 9–12).

Table 5
Subjects of Defects4J benchmark and Siemens Suite.

Subject	♯Bugs	Lines of code	♯Tests
Chart	26	86K	2180
Closure	133	90K	7926
Lang	65	21K	2239
Math	106	85K	3597
Mockito	38	22K	1326
Time	27	28K	4130
Total	395	332K	21 398
Tcas	36	173	1608
Tot_info	19	406	1052
Schedule	7	412	2650
Total	62	991	5310

5.3. Implementation and supporting platform

All the experiments are performed on a server with Tesla T4 GPU, the running environment is Ubuntu 20.04. We first used the Gzoltar (Campos et al., 2012) tool to analyze the coverage information of the program to obtain the program spectrum and test case execution results. Then, we applied Proteum (Papadakis et al., 2013) and PIT (Laurent et al., 2017; Laurent and Ventresque, 2019) mutation analysis tools to implement the traditional MBFL methods. Proteum is a well-known tool and is widely used in many mutation testing studies (G et al., 2007). For the three C programs in the Siemens dataset, we used the Proteum tool to generate mutants for the statements covered by failing test case. PIT is a widely used mutation testing system that provides standard test coverage for Java programs (Laurent et al., 2017). We generated mutants using the 11 default mutation operators provided by PIT (v1.6.4), as shown in Table 6.

5.4. Evaluation metrics

To evaluate the performance of PRMA, we use four commonly used metrics. The specific definitions and descriptions are as follows.

Top-n. *Top-n* is widely used to counts whether the faulty element appears in the top n positions of the sorted list provided by the fault localization technology (Li et al., 2019). Multiple faulty elements with the same fault in the top n positions are counted only once. Research (Xie et al., 2013) shows that about 70% of the developers only check the first five elements, and it is beyond acceptable to check more than ten program elements after using the fault localization method. Therefore, we set the value of n to 1, 3, 5.

MAR. The average Rank (AR) is a commonly used ranking evaluation index in the information retrieval, which represents the average ranking of all faulty elements of each fault in the list. The mean average rank (MAR) of the subject is the average value of *AR* calculated by a set of faults, which refers to the faults in the same subject.

MFR. For faults with multiple faulty elements, it is critical to determine the location of the first faulty element. The first rank (FR) represents the ranking of the first faulty element of each fault in the list. The mean first rank (MFR) of each subject refers to the average of all faults' *FR* values.

Improvement. *Improvement* is used to evaluate the promotion of the method in fault localization performance. *Improvement* is obtained by subtracting the values of metrics (*Top-n*, *MAR*, *MFR*) between PRMA and the other methods.

Table 6
Function descriptions of mutators in PIT.

Mutator	Function description
Conditionals boundary	Replacing the relational operators $<$, $\leq$, $>$, $\geq$.
Increments	Mutating and assignment increments or decrements of local variables.
Invert negatives	Inverting negation of integer and floating-point numbers.
Math	Replacing binary arithmetic operations with another operation.
Negate conditionals	Mutating all conditionals found.
Void method call	Removing method calls to void methods.
Empty returns	Replacing return values with an 'empty' value.
False returns	Replacing primitive and boxed boolean return values with false.
True returns	Replacing primitive and boxed boolean return values with true.
Null returns	Replacing return values with null.
Primitive returns	Replacing int, short, long, char, float and double return values with 0.

5.5. Results analysis

This section analyzes the experimental results around four research questions. Firstly, we answer RQ1 by comparing the effects of PRMA with Metallaxis and MUSE methods. We also evaluate the effectiveness of fault localization by only considering PageRank's faultiness scores and compare it with our approach in RQ1. Secondly, the performance of PRMA is compared with MCBFL-hybrid-avg (Pearson et al., 2017), SMFL (Cui et al., 2020), and SMBFL (Chaleshtari and Parsa, 2020) to illustrate RQ2. Thirdly, we explore the localization effect of PRMA on artificial faults as the answer to RQ3. Finally, we analyze the improvement of PRMA to answer RQ4 and evaluate the time cost of our approach in RQ5.

RQ1: Effectiveness of PRMA compared with traditional MBFL methods.

To answer this question, we present the experimental results of PRMA on six real fault subjects. Table 7 shows the experiment subjects (column 1) and the experiment results of techniques (columns 2–7). From Table 7, we can get the following observations. Firstly, in terms of the traditional MBFL methods, Metallaxis has a better fault localization performance than the MUSE. For Math subject, Metallaxis can locate 14 faults in *Top-1* and 29 faults in *Top-3*. Secondly, PRMA can improve traditional MBFL techniques. The overall performance of the *Top-n*, *MFR* and *MAR* values of PRMA-Me and PRMA-MU is better than the corresponding Metallaxis and MUSE methods. Specially, for the Chart subject, the MAR of PRMA-Me is 10.45, which is obvious lower than that of Metallaxis. It means that PRMA-Me can improve the ranking of faulty statements. Thirdly, the PageRank performs poorly in fault localization compared to PRMA and the MBFL methods. For example, the PageRank method finds two faulty statements in *Top-5* for the Chart subject. According to the first quantity assumption of the PageRank algorithm, the higher the in-degree of a node, the more suspicious the node is. Therefore, for subjects with a large amount of code, the basis of faultiness scores calculated by the PageRank algorithm is too weak to sort the faulty statements at the front of the list. Finally, the improvement of the fault localization effectiveness of PRMA is most obvious in the Closure subject, PRMA-Me can find 31 faults in *Top-1*, 12 more than Metallaxis. The reason may be that the Closure subject's amount of code is very large, which leads to a serious tie problem. PRMA effectively solves the tie problem, so the fault localization performance is significantly improved.

We also show each technique's overall fault localization effect (sum of *Top-n* and average of *MFR*, *MAR*). Table 8 shows that PRMA-Me and PRMA-MU are better than Metallaxis and MUSE methods, respectively. Our approach can also find more faults than the PageRank method in *Top-n*. Specially, PRMA-Me locates 27 more faults than Metallaxis in *Top-1* and 195 more than PageRank in *Top-5*. The *MAR* and *MFR* values of PRMA are lower than those of PageRank, Metallaxis and MUSE. For example, the *MFR* of PRMA-MU is 16.39, which is 3.77 lower than MUSE. The experimental results can show that PRMA improves the fault localization accuracy of Metallaxis and MUSE.

Table 7
Effectiveness of PageRank, PRMA, Metallaxis (Papadakis and Le Traon, 2015) and MUSE (Moon et al., 2014) for every subject.

Subject	Technique	Top-1	Top-3	Top-5	MAR	MFR
Chart	PageRank	0	1	2	52.07	51.90
	Metallaxis	4	10	12	13.04	12.30
	PRMA-Me	**7**	**14**	**17**	**10.45**	**9.01**
	MUSE	4	11	14	14.85	12.77
	PRMA-MU	**7**	**13**	**15**	**11.08**	**9.82**
Closure	PageRank	0	0	3	108.62	103.22
	Metallaxis	19	41	50	27.19	22.50
	PRMA-Me	**31**	**49**	**68**	**17.55**	**15.92**
	MUSE	17	39	48	29.32	26.31
	PRMA-MU	**28**	**46**	**57**	**19.13**	**16.88**
Lang	PageRank	0	0	1	78.54	77.90
	Metallaxis	12	23	27	12.54	11.19
	PRMA-Me	**14**	**26**	**32**	**11.94**	**10.86**
	MUSE	10	20	26	14.92	13.73
	PRMA-MU	**13**	**22**	**28**	**13.90**	**11.62**
Math	PageRank	0	2	3	65.31	62.45
	Metallaxis	14	29	34	13.59	12.06
	PRMA-Me	**19**	**38**	**53**	**12.01**	**10.42**
	MUSE	14	28	33	14.68	11.95
	PRMA-MU	**18**	**35**	**44**	**13.25**	**11.08**
Mockito	PageRank	0	1	2	115.70	112.36
	Metallaxis	9	14	18	25.81	23.24
	PRMA-Me	**12**	**16**	**24**	**21.29**	**18.33**
	MUSE	8	11	15	31.20	25.63
	PRMA-MU	**11**	**13**	**19**	**23.51**	**20.79**
Time	PageRank	0	0	1	85.30	82.57
	Metallaxis	3	7	8	31.64	28.82
	PRMA-Me	**5**	**10**	**13**	**28.53**	**26.08**
	MUSE	2	5	6	32.59	30.55
	PRMA-MU	**3**	**9**	**8**	**30.22**	**28.17**

Table 8
Overall effectiveness of PageRank, PRMA, Metallaxis (Papadakis and Le Traon, 2015) and MUSE (Moon et al., 2014).

Technique	Top-1	Top-3	Top-5	MAR	MFR
PageRank	0	4	12	84.26	81.73
Metallaxis	61	124	149	20.64	18.35
PRMA-Me	**88**	**153**	**207**	**16.96**	**15.10**
MUSE	55	114	142	22.93	20.16
PRMA-MU	**80**	**138**	**171**	**18.52**	**16.39**

RQ2: Effectiveness of PRMA compared with recent proposed MBFL techniques.

In the exploration of RQ2, we choose PRMA-Me as the representative of PRMA, which performs better than the PRMA-MU. We compare PRMA-Me with three fault localization methods proposed in recent years: MCBFL-hybrid-avg (Pearson et al., 2017), SMFL (Cui et al., 2020) and SMBFL (Chaleshtari and Parsa, 2020). MCBFL-hybrid-avg (Pearson et al., 2017) averages the MBFL suspicious value of each statement with the suspiciousness calculated

Table 9
Effectiveness of PRMA, MCBFL-hybrid-avg (Pearson et al., 2017), SMFL (Cui et al., 2020) and SMBFL (Chaleshtari and Parsa, 2020) for every subject.

Subject	Technique	Top-1	Top-3	Top-5	MAR	MFR
Chart	SMFL	4	11	14	11.87	11.01
	SMBFL	4	10	13	12.94	11.35
	MCBFL-hybrid-avg	5	11	15	11.26	10.52
	PRMA-Me	**7**	**14**	**17**	**10.45**	**9.01**
Closure	SMFL	21	42	55	24.95	21.28
	SMBFL	19	40	46	28.36	24.71
	MCBFL-hybrid-avg	24	43	57	23.51	21.65
	PRMA-Me	**31**	**49**	**68**	**17.55**	**15.92**
Lang	SMFL	12	24	27	12.38	11.15
	SMBFL	11	23	25	13.16	12.97
	MCBFL-hybrid-avg	14	25	27	12.09	11.02
	PRMA-Me	**14**	**26**	**32**	**11.94**	**10.86**
Math	SMFL	15	32	49	12.84	11.53
	SMBFL	14	30	39	13.27	11.88
	MCBFL-hybrid-avg	17	34	50	12.33	10.95
	PRMA-Me	**19**	**38**	**53**	**12.01**	**10.42**
Mockito	SMFL	9	13	19	24.63	22.89
	SMBFL	9	14	18	25.59	23.30
	MCBFL-hybrid-avg	10	14	20	23.18	20.96
	PRMA-Me	**12**	**16**	**24**	**21.29**	**18.33**
Time	SMFL	4	7	11	29.72	27.34
	SMBFL	3	5	8	32.69	31.89
	MCBFL-hybrid-avg	4	8	11	29.38	27.10
	PRMA-Me	**5**	**10**	**13**	**28.53**	**26.08**

Table 10
Overall effectiveness of PRMA, MCBFL-hybrid-avg (Pearson et al., 2017), SMFL (Cui et al., 2020) and SMBFL (Chaleshtari and Parsa, 2020).

Technique	Top-1	Top-3	Top-5	MAR	MFR
SMFL	65	129	175	19.40	17.53
SMBFL	60	122	149	21.00	19.35
MCBFL-hybrid-avg	74	135	180	18.63	17.03
PRMA-Me	**88**	**153**	**207**	**16.96**	**15.10**

by the SBFL technique, and it outperforms other hybrid techniques. After that, the method of combining SBFL and MBFL for fault localization has been proposed continuously. SMFL (Cui et al., 2020) reorders the top n statements ranked by SBFL according to their mutation scores, and achieves better results in the localization of real faults. In addition, SMBFL (Chaleshtari and Parsa, 2020) combines program slicing technology to improve the fault localization effect of MBFL.

We present the experimental results of PRMA-Me, MCBFL-hybrid-avg, SMFL and SMBFL in Table 9. Overall, PRMA-Me outperforms other techniques on all subjects. Specifically, for the largest Closure subject, PRMA-Me locates the 31 faults in *Top-1*, which finds 7 more faults than MCBFL-hybrid-avg. For the Chart project, the *MAR* value of PRMA-Me is 1.42 lower than that of SMFL, which can save the inspection costs.

Table 10 shows the overall performance of PRMA-Me, MCBFL-hybrid-avg, SMFL and SMBFL. In *Top-1*, PRMA-Me locates 23 and 28 more faults than SMFL and SMBFL methods, respectively. The *MAR* values of PRMA-Me are lower than those of the MCBFL-hybrid-avg, SMFL and SMBFL methods. By analyzing the experimental results, we can conclude that the PRMA method can improve the accuracy of the real fault localization effectiveness.

RQ3: Effectiveness of PRMA on artificial faults.

So far, we have investigated the effectiveness of the PRMA method for locating real faults from the Defects4J dataset. In this section, we further investigate the effectiveness of PRMA on artificial faults to reduce the threat of external validity. Table 11 shows the experimental results of 62 manual fault in three Siemens subjects. In the Table, we can observe that the most effective technique on artificial faults is PRMA-MU, which is different from

Table 11
Effectiveness of PRMA on artificial faults.

Subject	Technique	Top-1	Top-3	Top-5	MAR	MFR
Tcas	Metallaxis	2	5	7	27.18	24.95
	PRMA-Me	**3**	**7**	**10**	**23.56**	**22.97**
	MUSE	2	6	7	26.74	24.45
	PRMA-MU	**3**	**8**	**11**	**22.13**	**21.64**
Tot_info	Metallaxis	1	1	3	17.28	15.76
	PRMA-Me	**1**	**2**	**5**	**16.32**	**14.20**
	MUSE	2	3	5	15.91	14.08
	PRMA-MU	**3**	**6**	**8**	**14.17**	**13.65**
Schedule	Metallaxis	0	1	2	12.53	10.80
	PRMA-Me	**1**	**2**	**4**	**11.03**	**9.57**
	MUSE	0	2	2	11.74	10.01
	PRMA-MU	**1**	**2**	**4**	**10.92**	**9.12**

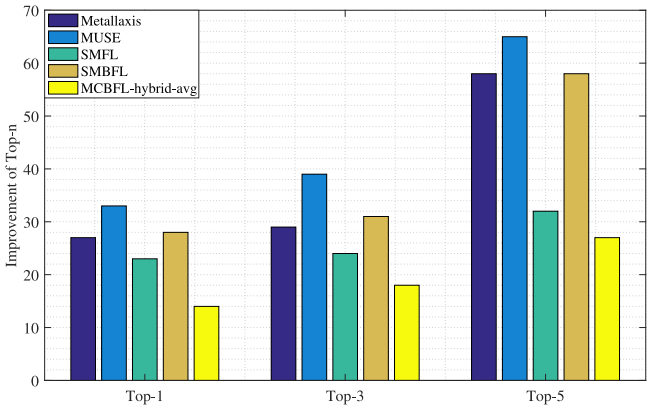


Fig. 4. Improvement of PRMA on Top-n.

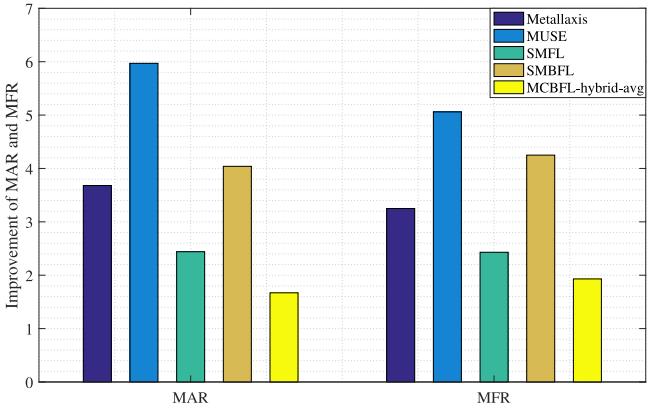


Fig. 5. Improvement of PRMA on MAR and MFR.

the technique on the real fault Defects4J dataset. The reason may be that the amount of code for artificial faults is small, and there are fewer tie cases than real faults, which can highlight the superiority of the MUSE method. Therefore, for projects with different code structures, the performance of fault localization methods will also be different. Compared with Metallaxis and MUSE methods, PRMA also improves the effectiveness of fault localization on artificial faults. Specially, for the Tcas subject, PRMA-MU can locate 11 faults in the *Top-5*, which is 4 more faults than MUSE. The *MAR* and *MFR* values of PRMA-MU are also lower than the MUSE. For Tcas subject, PRMA-MU reduces the *MFR* value of MUSE from 24.45 to 21.64.

RQ4: Improvement of PRMA.

Furthermore, we analyze the improvement of PRMA for the real faults localization, as shown in Figs. 4 and 5. The value of *Top-n* in Fig. 4 is obtained by subtracting the value of the comparison

Table 12
The time cost of our approach.

Techniques	Phases	Chart	Closure	Lang	Math	Mockito	Time
Metallaxis	–	104.8	256.7	91.1	129.2	113.9	76.5
MUSE	–	104.8	256.7	91.1	129.2	113.9	76.5
PRMA-Me	Mutation	104.8	256.7	91.1	129.2	113.9	76.5
	PageRank	2.3	7.4	3.3	5.1	4.2	2.1
	Total	107.1	264.1	94.4	134.3	118.1	78.6
PRMA-MU	Mutation	104.8	256.7	91.1	129.2	113.9	76.5
	PageRank	2.3	7.4	3.3	5.1	4.2	2.1
	Total	107.1	264.1	94.4	134.3	118.1	78.6

method from PRMA-Me. In Fig. 5, the values of *MAR* and *MFR* are calculated by subtracting the corresponding value of PRMA-Me by the comparison method. Firstly, we can observe that the data of all histograms are greater than 0, indicating that PRMA-Me has improved the Metallaxis, MUSE, MCBFL-hybrid-avg, SMFL and SMBFL methods on each evaluation metric. Secondly, by observing Fig. 4, we can observe that PRMA-Me has the greatest improvement on the MUSE method in *Top-n*, and can find 65 more faults in *Top-5*. Finally, we can find that the *MAR* and *MFR* values are all reduced by PRMA from Fig. 5. The experimental results can further verify the high efficiency of our approach for fault localization.

RQ5: Time cost of PRMA.

Finally, we evaluate the time costs of the PRMA method on the six subjects of Defects4J. Since we only consider failed test cases during the mutation and PageRank analysis phases, the time of PRMA will be greatly reduced. The running time of PRMA is compared with traditional MBFL methods (Metallaxis and MUSE). The experimental results are shown in Table 12, and we count the time cost by the hour. We can observe that the time cost of our approach is acceptable. Compared with the Metallaxis and MUSE, the extra cost is due to our method adding PageRank analysis based on mutation analysis. It can be seen that PageRank analysis consumes less time. As shown in Table 12, the PRMA-Me method took 2.3 h more than the Metallaxis method but located 27 more faults in Top-1. Therefore, the extra time cost of our method is worth it. In addition, we can find from Table 12 that the time costs spent by Metallaxis and MUSE are the same. Similarly, the time costs of PRMA-Me and PRMA-MU are also the same. This phenomenon is because the difference between the two MBFL methods mainly focuses on calculating suspicious values, and their previous mutation analysis phases are the same. In future work, we aim to reduce the test suite and the computational cost of PRMA.

6. Threats to validity

Threat to internal validity. The main threat to internal validity is uncontrolled factors that may affect the results and reduce their credibility. In our approach, the main threat to internal validity is a potential flaw in the implementation of our own technique and the reimplementation of other approaches. To reduce this threat, we leverage mature tools and frameworks to build our experiment, such as ASM bytecode manipulation framework, Gzoltar, and PIT. We carefully check all code and experimental scripts to ensure their correctness. However, there is always a small possibility of flaws that pose a risk to the correctness of the results.

Threat to external validity. The threat to external validity primarily concerns to whether our technique still holds up in other experimental settings. In our work, tests and faults may threaten to external validity. To reduce these threats, we evaluate our method using real faults from Defects4J. We further leverage artificial faults in the Siemens dataset to evaluate the effectiveness of our method. The quality of the test suite has a large impact on mutation analysis. To reduce the computational cost, we only use failed test cases for PageRank analysis in our experiments, which poses a threat to the effectiveness of the method. Our future work plans to reduce the threat by exploring the impact of using both passed and failed test cases on the effectiveness of fault localization. Moreover, the faults in the Defects4J dataset are multi-faults. That is, each fault version contains at least one buggy location. Therefore, there may be a threat to validity in judging whether we have found the root faulty statement. To mitigate this threat, we manually confirmed the root buggy statements for the fault version containing multiple faulty locations to ensure the robustness of the experiment. Afterwards, we will also consider improving the accuracy of multi-fault localization.

Threat to construct validity. The threat to construct validity is primarily the suitability of our evaluation metrics. To mitigate risks, we evaluate our method using four metrics commonly used in fault location techniques (Liu et al., 2020; Wong et al., 2012) (*Top-n*, *MAR*, *MFR*, and *Improvement*).

7. Related works

In this section, we briefly review related works on mutation-based fault localization. MBFL is an important fault location technology with high location accuracy (Pearson et al., 2017). MBFL calculates the mutation score for each mutant and uses it to calculate the suspicious value of the statement. Metallaxis proposed by Papadakis and Le Traon (2015) is the extensive used method to apply mutation analysis to fault localization. Metallaxis took the maximum score of all mutant as the suspicious value of the statement. Zhang et al. (2013) were the first to apply mutation testing in regression testing for faults localization. Moon et al. (2014) proposed MUSE, which takes the average of the mutation scores of the mutants as the suspicious value of the statement. In general, a statement is suspect if it affects failing test cases more frequently and passing test cases infrequently. However, since the traditional MBFL method only considers the suspiciousness of statements by the scores of mutants, it ignores the problem of the same suspicious values of multiple statements due to the limited number of mutants. Considering this problem, we use the faultiness scores generated by the PageRank algorithm to weight the statements, thereby improving the accuracy of fault localization.

In recent years, many methods have been proposed to improve traditional MBFL in combination with other techniques. Pearson et al. (2017) proposed an improved technique MCBFL using coverage and mutation information. It considered dimensional information and combined SBFL and MBFL techniques to calculate the suspicious value of the statement. Cui et al. (2020) also proposed a method combining SBFL's sentence ranking list and mutation score for fault localization. Chaleshtari and Parsa (2020) combined the dynamic slicing technique with the MBFL technique in order to reduce the execution cost of MBFL. Zou et al. (2021) proposed CombineFL to explore the combination of a wide range of techniques that rely on different information sources. It combined seven techniques including SBFL (Abreu et al., 2006), MBFL (Papadakis and Le Traon, 2015), program slicing, stack trace, predicate switching, information retrieval and history for fault localization. Although it performs better in fault localization, it takes a lot of time to perform all the techniques. In contrast, we are not limited to improving the performance of traditional MBFL by combining techniques in the field of fault localization. Based on the analysis of the web page ranking algorithm, we simulate test cases and statements as nodes in the network topology graph. Furthermore, according to the actual scenario of fault localization,

we apply the PageRank algorithm to calculate the correlation (faultiness score) between statements and failed test cases. We perform fault localization by combining faultiness scores with mutant scores to generate the suspicious values of statements.

8. Conclusion

The ever-increasing size of software has made manual software testing expensive and painful, so researchers have developed various automated fault localization techniques. Mutation-based fault localization (MBFL) is a popular fault localization technique that helps developers infer the location of faulty program elements. However, current MBFL techniques are not precise enough and the costs are expensive. In this paper, we propose PRMA, which is a new method to improve the accuracy of MBFL. PRMA uses the PageRank algorithm to analyze the correlation of statements with failed test cases, generating a faultiness score for the statement. Then, the suspicious value generated by the traditional MBFL method is weighted by the faultiness score. Finally, a new sorted list of statements is generated and provided to developers for fault localization. We compare the performance of PRMA with traditional MBFL methods (Metallaxis and MUSE) and other mutation-based fault localization methods (MCBFL-hybrid-avg Pearson et al., 2017, SMFL Cui et al., 2020 and SMBFL Chaleshtari and Parsa, 2020). The experiment is conducted on 395 real faults and 62 artificial faults. Experimental results show that PRMA outperforms the above MBFL techniques. However, during our experiments with the PIT tool (Laurent et al., 2017), we found that some statements only had one mutant. Fewer mutants may cause some faults that cannot be tested, which negatively affects fault localization. Therefore, we are trying to improve the PIT tool to generate more mutants for statements and promote the effect of mutation testing. On this basis, we plan to explore the PageRank algorithm to measure the importance between test cases or statements with mutants in the future and further apply it to fault localization or program repair scenarios.

CRediT authorship contribution statement

Yue Yan: Conceptualization, Methodology, Software, Formal analysis, Writing – original draft. **Shujuan Jiang:** Formal analysis, Investigation, Writing – review & editing. **Yanmei Zhang:** Supervision, Investigation, Writing – review & editing. **Cheng Zhang:** Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request

Acknowledgments

This research was supported in part supported by National Natural Science Foundation of China (61673384); Natural Science Foundation of Jiangsu Province, China (BK20181353).

References

Abreu, R., Zoeteweij, P., Van Gemund Arjan, J.C., 2006. An evaluation of similarity coefficients for software fault localization. In: 2006 12th Pacific Rim International Symposium on Dependable Computing (PRDC'06). pp. 39–46. http://dx.doi.org/10.1109/PRDC.2006.18.

Bhattacharya, P., Iliofotou, M., Neamtiu, I., Faloutsos, M., 2012. Graph-based analysis and prediction for software evolution. In: 2012 34th International Conference on Software Engineering (ICSE). pp. 419–429. http://dx.doi.org/10.1109/ICSE.2012.6227173.

Borodin, A., Roberts, G.O., Rosenthal, J.S., Tsaparas, P., 2005. Link analysis ranking: Algorithms, theory, and experiments. ACM Trans. Int. Technol. 5 (1), 231–297. http://dx.doi.org/10.1145/1052934.1052942.

Campos, J., Riboira, A., Perez, A., Abreu, R., 2012. GZoltar: an eclipse plug-in for testing and debugging. In: ACM International Conference on Automated Software Engineerin. ACM, pp. 378–381. http://dx.doi.org/10.1145/2351676.2351752.

Chaleshtari, N.B., Parsa, S., 2020. SMBFL: slice-based cost reduction of mutation-based fault localization. Empir. Softw. Eng. 25 (5), 1–33.

Cui, Z., Jia, M., Chen, X., Zheng, L., Liu, X., 2020. Improving software fault localization by combining spectrum and mutation. IEEE Access 8, 172296–172307.

Do, H., Elbaum, S., Rothermel, G., 2005. Supporting controlled experimentation with testing techniques: An infrastructure and its potential impact. Empir. Softw. Eng. 10 (4), 405–435.

Dutta, A., Godboley, S., 2021. MSFL: a model for fault localization using mutation-spectra technique. In: Lean and Agile Software Development: 5th International Conference, LASD 2021, Virtual Event, January 23, 2021, Proceedings 5. Springer, pp. 156–173.

F, G.D., 2015. PageRank beyond the web. Siam Rev. 57 (4), 321–363.

Farahat, A., LoFaro, T., Miller, J.C., Rae, G., Ward, L.A., 2006. Authority rankings from HITS, PageRank, and SALSA: Existence, uniqueness, and effect of initialization. SIAM J. Sci. Comput. 27 (4), 1181–1201. http://dx.doi.org/10.1137/S1064827502412875.

G, A., S., S., B., R., 2007. Automated fault localization for C programs. Electron. Notes Theor. Comput. Sci. 174 (4), 95–111. http://dx.doi.org/10.1016/j.entcs.2006.12.032.

Jia, Y., Harman, M., 2011. An analysis and survey of the development of mutation testing. IEEE Trans. Softw. Eng. 37 (5), 649–678. http://dx.doi.org/10.1109/TSE.2010.62.

Just, R., Jalali, D., E., M.D., 2014. Defects4J: a database of existing faults to enable controlled testing studies for Java programs. In: Proceedings of the International Symposium on Software Testing and Analysis. pp. 437–440. http://dx.doi.org/10.1145/2610384.2628055.

Just, R., Kapfhammer, G.M., Schweiggert, F., 2012. Using non-redundant mutation operators and test suite prioritization to achieve efficient and scalable mutation analysis. In: 2012 IEEE 23rd International Symposium on Software Reliability Engineering. pp. 11–20. http://dx.doi.org/10.1109/ISSRE.2012.31.

Kleinberg, J.M., 1999. Authoritative sources in a hyperlinked environment. J. ACM 46 (5), 604–632. http://dx.doi.org/10.1145/324133.324140.

Laurent, T., Papadakis, M., Kintis, M., Henard, C., Le Traon, Y., Ventresque, A., 2017. Assessing and improving the mutation testing practice of PIT. In: 2017 IEEE International Conference on Software Testing, Verification and Validation (ICST). pp. 430–435. http://dx.doi.org/10.1109/ICST.2017.47.

Laurent, T., Ventresque, A., 2019. PIT-HOM: an extension of pitest for higher order mutation analysis. In: 2019 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW). pp. 83–89. http://dx.doi.org/10.1109/ICSTW.2019.00036.

Lempel, R., Moran, S., 2000. The stochastic approach for link-structure analysis (SALSA) and the TKC effect. In: Proceedings of the 9th International World Wide Web Conference on Computer Networks: The International Journal of Computer and Telecommunications Netowrking. pp. 387–401.

Li, X., Li, W., Zhang, Y., Zhang, L., 2019. DeepFL: integrating multiple fault diagnosis dimensions for deep fault localization. In: Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis. ACM, pp. 169–180.

Liu, Y., Li, Z., Wang, L., Hu, Z., Zhao, R., 2017. Statement-oriented mutant reduction strategy for mutation based fault localization. In: 2017 IEEE International Conference on Software Quality, Reliability and Security (QRS). pp. 126–137. http://dx.doi.org/10.1109/QRS.2017.23.

Liu, C., Ma, C., Zhang, T., 2020. Improving Spectrum-Based Fault Localization using quality assessment and optimization of a test suite. In: 20th IEEE International Conference on Software Qualit. IEEE, pp. 72–78. http://dx.doi.org/10.1109/QRS-C51114.2020.00023.

Moon, S., Kim, Y., Kim, M., Yoo, S., 2014. Ask the mutants: Mutating faulty programs for fault localization. In: 2014 IEEE Seventh International Conference on Software Testing, Verification and Validation. pp. 153–162. http://dx.doi.org/10.1109/ICST.2014.28.

Page, L., Brin, S., Motwani, R., Winograd, T., 1998. The PageRank Citation Ranking: Bringing Order to the Web. Stanford Digital Libraries Working Paper.

Papadakis, M., Delamaro, M.E., Le Traon, Y., 2013. Proteum/FL: A tool for localizing faults using mutation analysis. In: 2013 IEEE 13th International Working Conference on Source Code Analysis and Manipulation (SCAM). pp. 94–99. http://dx.doi.org/10.1109/SCAM.2013.6648189.

Papadakis, M., Le Traon, Y., 2012. Using mutants to locate "unknown" faults. In: 2012 IEEE Fifth International Conference on Software Testing, Verification and Validation. pp. 691–700. http://dx.doi.org/10.1109/ICST.2012.159.

Papadakis, M., Le Traon, Y., 2015. Metallaxis-FL: mutation-based fault localization. J. Softw. Test., Verication, Reliab. 25 (5–7), 605–628. http://dx.doi.org/10.1002/stvr.1509.

Pearson, S., Campos, J., Just, R., Fraser, G., Abreu, R., E., M.D., Pang, D., Keller, B., 2017. Evaluating and improving fault localization. In: Proceedings of the 39th International Conference on Software Engineering. pp. 609–620. http://dx.doi.org/10.1109/ICSE.2017.62.

Rabinowitz, P.H., 1978. Some minimax theorems and applications to nonlinear partial differential equations. In: Nonlinear Analysis. Academic Press, pp. 161–177. http://dx.doi.org/10.1016/B978-0-12-165550-1.50016-1.

Siami Namin, A., Andrews, J., Murdoch, D., 2008. Sufficient mutation operators for measuring test effectiveness. In: 2008 ACM/IEEE 30th International Conference on Software Engineering. pp. 351–360. http://dx.doi.org/10.1145/1368088.1368136.

Vessey, I., 1985. Expertise in debugging computer programs: A process analysis. Int. J. Man-Mach. Stud. 23 (5), 459–494.

Wong, W.E., Debroy, V., Golden, R., Xu, X., Thuraisingham, B., 2012. Effective software fault localization using an RBF neural network. IEEE Trans. Reliab. 61 (1), 149–169. http://dx.doi.org/10.1109/TR.2011.2172031.

Wong, W.E., Gao, R., Li, Y., Abreu, R., Wotawa, F., 2016. A survey on software fault localization. IEEE Trans. Softw. Eng. 42 (8), 707–740. http://dx.doi.org/10.1109/TSE.2016.2521368.

Xie, X., Chen, T.Y., Kuo, F., Xu, B., 2013. A theoretical analysis of the risk evaluation formulas for spectrum-based fault localization. ACM Trans. Softw. Eng. Methodol. 22 (4), 31:1–31:40. http://dx.doi.org/10.1145/2522920.2522924.

Zhang, M., Li, X., Zhang, L., Khurshid, S., 2017. Boosting spectrum-based fault localization using PageRank. In: Proceedings of the 26th ACM SIGSOFT International Symposium on Software Testing and Analysis. In: ISSTA 2017, Association for Computing Machinery, pp. 261–272. http://dx.doi.org/10.1145/3092703.3092731.

Zhang, L., Zhang, L., Khurshid, S., 2013. Injecting mechanical faults to localize developer faults for evolving software. In: Conference on Object-Oriented Programming Systems, Languages, and Applications. pp. 765–784.

Zou, D., Liang, J., Xiong, Y., Ernst, M.D., Zhang, L., 2021. An empirical study of fault localization families and their combinations. IEEE Trans. Softw. Eng. 47 (2), 332–347. http://dx.doi.org/10.1109/TSE.2019.2892102.

Yue Yan was born in DeZhou City, Shandong Province, in 1995. She received both B.S. and M.S. degree in Computer Science Department from Qufu Normal University, RiZhao, ShanDong, in 2017 and 2020 respectively.

From 2020 to totay, she has been working toward the Ph.D. degree under the supervision of Professor Shujuan Jiang at China University of Mining and Technology, XuZhou, JiangSu. Her research interests include software fault localization, program debugging and analysis, etc.

Shujuan Jiang was born in Laiyang City, Shandong Province, in 1966. She received the B.S. degree in Computer Science of Computer Science Department from East China Normal University, Shanghai, in 1990 and the M.S. degree in Computer Science from China University of Mining and Technology, Xuzhou, Jiangsu, in 2000. She received the Ph.D. degree in Computer Science from Southeast University, Nanjing, Jiangsu, in 2006.

From 1995 to today, she has been a teaching assistant, lecturer, associate professor and professor in the Computer Science Department, China University of Mining and Technology, Xuzhou, Jiangsu. She is the author of more than 80 articles. Her research interests include software engineering, program analysis and testing, software maintenance, etc. She is a Member of the IEEE.

Yanmei Zhang received the B.S. degree in Computer Science of Computer Science Department from the Nanjing Tech University, in 2007 and the Ph.D. degree in Computer Science from China University of Mining and Technology, Xuzhou, Jiangsu, in 2012. From 2007 to 2009, she was an M.S. candidate in Computer Science from the China University of Mining and Technology, Xuzhou, Jiangsu. From 2012 to 2017, she has been a Lecturer, and since 2018, she has been an Associate Professor at the Computer Science Department, China University of Mining and Technology, Xuzhou, Jiangsu. She is the author of one book, more than 10 articles, and more than 2 inventions. Her research interests include software engineering, program analysis, and testing, software quality, reinforcement learning, intelligent evolutionary algorithm, etc.

Cheng Zhang was born in Xuzhou City, Jiangsu Province, in 1995. He received the B.B.A degree from Yuan Ze University, TaoYuan, Taiwan, in 2017 and the M.S. degree in information technology from the University of Melbourne, VIC, Australia, in 2020. After graduated in 2020, he was a research assistant with the Hainan Acoustic Laboratory, Chinese Academy of Sciences, Hainan, China. He is currently a Ph.D. candidate in China University of Mining and Technology. His research interest includes steganography, infectious disease, decision support systems, computational modeling and neural networks.