Received June 8, 2020, accepted June 18, 2020, date of publication June 22, 2020, date of current version July 20, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3004145

IETCR: An Information Entropy Based Test Case Reduction Strategy for Mutation-Based Fault Localization

HAIFENG WANG[1], BIN DU[1], JIE HE[1], YONG LIU[1], (Member, IEEE), AND XIANG CHEN[2], (Member, IEEE)

[1]College of Information Science and Technology, Beijing University of Chemical Technology, Beijing 100029, China
[2]School of Information Science and Technology, Nantong University, Nantong 226019, China

Corresponding author: Yong Liu (lyong@mail.buct.edu.cn)

This work was supported in part by the National Natural Science Foundation of China under Grant 61902015, Grant 61872026, and Grant 61672085, and in part by the Nantong Application Research Plan under Grant JC2018134 and Grant JC2019106.

ABSTRACT Mutation-based fault localization (MBFL) is a recently proposed technique with the advantage of high fault localization accuracy. However, such a mutation analysis based technique is difficult to be accepted by industry due to its huge computational cost on mutation analysis. There are three ways to improve MBFL's efficiency, which are reducing the number of mutants, optimizing the mutants' execution process, and reducing the number of test cases. The former two ways have been mainly studied and shown promising results, but for the latter way, the related studies are limited since this kind of method will reduce the precision of MBFL. In this paper, we mainly focus on the latter way and propose an information entropy based test case reduction (IETCR) strategy for MBFL. In particular, we first calculate the entropy change of test cases and select a proportion of them according to their value. Then we use a reduced test suite to execute mutants. To show the effectiveness of the IETCR strategy, we choose six real-world programs with 112 faulty versions. In terms of mutation reduction rate, we find MBFL with the IETCR strategy can reduce 56.3%∼88.3% cost while keeping almost the same fault localization accuracy when compared to the original MBFL without test case reduction. Moreover, we use Wilcoxon signed-rank test for statistical analysis, which shows that there is no statistically significant difference between MBFL with IETCR strategy and the original MBFL.

INDEX TERMS Software fault localization, mutation based fault localization, information entropy, test case reduction.

I. INTRODUCTION

Software debugging, which contains fault detecting, fault localization, and bug fixing [1], is one of the essential parts of software development and maintenance. The fault localization process refers to the problem of identifying defective program statements when the execution of some test cases results in failure. It has been recognized as the most complicated and time-consuming activity [2]. Researchers have invested numerous efforts into the area of automatic software fault localization [1], [3], which helps to reduce the debug time and manual cost on software debugging.

Among the automated fault localization techniques, spectrum-based fault localization (SBFL) [4]–[7] is the most studied and evaluated technique. SBFL obtains the coverage information and execution results by executing test cases. Such information will be used for calculating the probability that program entities (such as statements) incur a fault and generating a ranking list of them for developers to inspect code elements. SBFL is simple to implement and previous studies have shown that SBFL is promising to reduce the cost of software debugging [8], [9]. However, when applied to large-scale programs, SBFL still has the problem of poor fault localization accuracy. This leads to more program entities that should be checked before encountering the real faults [10].

Recent studies [11], [12] have shown that mutation-based fault localization (MBFL) techniques outperform state-of-the-art SBFL techniques in terms of the fault localization accuracy. MBFL is based on the hypothesis that the mutation testing [11] performed on the faulty program can

The associate editor coordinating the review of this manuscript and approving it for publication was Porfirio Tramontana.

contribute to the accuracy of fault localization. From different aspects, there are two popular MBFL techniques, named as Metallaxis-FL [13] and MUSE [14]. In particular, Metallaxis-FL applies the idea of mutation analysis [11] to mutate statements, which improves the accuracy of fault localization. MUSE also uses mutation analysis to calculate the suspiciousness value of statements. Different from Metallaxis-FL, MUSE works by checking whether mutants can turn the failed test cases into passed or not. In the empirical studies of Pearson *et al.* [15], they found that Metallaxis-FL can outperform MUSE on fault localization efficiency and accuracy.

MBFL techniques have higher fault localization accuracy, but their efficiency is suffered from an extremely mutant execution cost problem [12], [16], which results in the lower efficiency of fault localization. This problem stems from that MBFL generates a significant number of mutants from the program under test, and each mutant should be executed on all test cases. To reduce the execution cost and improve the efficiency of MBFL, several optimization approaches have been proposed [13], [16]–[21]. From different aspects, such optimization approaches for MBFL can be divided into three categories: (1) reducing the number of mutants [13], [19], (2) the strategy of test case reduction [20], and (3) optimizing the execution process of the mutants [21]. In the first category, the SELECTIVE strategy proposed by Papadakis and Le Traon [11] attempts to select a subset of the original mutation operators, then the number of generated mutants can be reduced. Besides, the SAMPLING strategy [13] randomly samples a certain proportion of mutants from the generated mutant set to decrease the number of mutants. The second category is considered from the test suite aspect. In the work of De Oliveira *et al.* [20], they present FTMES, which only employs the failed test cases to localize the faults but ignores all the passed test cases. Therefore, the execution cost of MBFL will be reduced by executing fewer test cases. The third category focuses on making MBFL more efficient by optimizing the mutant execution process. A dynamic mutation execution strategy (DMES) is proposed by Liu *et al.* [21], which aims to optimize the execution process based on both mutants and test cases.

Although many methods have been proposed for reducing the execution cost of MBFL, there is still room for further improvement. In the work of Yoo *et al.* [22], they found that test cases have different information or effect during the process of program testing. Motivated by this study, in this paper, we utilize information entropy to measure the information of different test cases and propose an information entropy based test case reduction (IETCR) strategy to optimize the mutation execution. IETCR is a strategy of test case reduction, which keeps both passed and failed test cases, while FTMES strategy [20] only employs the failed test cases but ignores all passed test cases. In particular, we first calculate the entropy change of each test case and select a proportion of them according to their value. Therefore, the IETCR strategy can speed up MBFL by decreasing the cost of mutation execution.

To our best knowledge, the contributions of our study can be summarized as follows:

- We propose an information entropy based test case reduction (IETCR) strategy to reduce the execution cost of MBFL from the perspective of reducing test cases.
- To evaluate the effectiveness of our proposed IETCR strategy, we conduct a large-scale empirical study, which chooses 112 faulty versions from 6 real-world programs as empirical subjects and chooses FTMES and SAMPLING strategies as state-of-the-art baselines. In terms of efficiency, final experimental results show that using the IETCR strategy can reduce the mutation execution cost from 56.3% to 88.3% when compared to the original MBFL without any strategy. In terms of accuracy, the final experimental results demonstrate that MBFL with the IETCR strategy can achieve better fault localization accuracy than the chosen two optimization baselines. Moreover, statistical analysis via Wilcoxon signed-rank test also indicates that there is no significant difference in fault localization accuracy between MBFL with IETCR strategy and original MBFL.
- To facilitate other researchers to replicate our study, we share the code and detailed experimental results. In particular, the source code[1] and experimental results[2] of this paper are all available in the GitHub repositories.

The rest of this paper is organized as follows. Section II provides background and related work of our study. Section III illustrates our proposed IETCR strategy and presents its main components. Section IV describes the experimental setup of our study, including research questions, subject programs, and performance metrics. Section VI discusses the experimental results and analyzes potential threats to validity. Section VII describes the related work of our study. Finally, Section VIII summarizes our study and shows potential future work.

II. BACKGROUND

A. MUTATION ANALYSIS

Mutation analysis is a fault-based technique, which works by making syntactic changes into the source code of the program under test [19]. Thus, it can generate a huge number of faulty programs, which are named as mutants [23]. The rules that generate mutants from the original program is named as mutation operators [24].

In the process of mutation analysis, mutants are executed on test cases. By comparing the output of the mutants with the output of the original program, the fault detection ability of the test suite can be assessed. A mutant is *killed* or *distinguished*, if there exists at least a test case, which its output on the mutant is different from the output on the original version. Otherwise, it is *not killed* or *alive*.

Mutation analysis is different from the program spectrum-based technique because it forces candidate test cases not

[1]https://github.com/HJBUCT/TestCaseReduction
[2]https://github.com/HJBUCT/DataOfFL

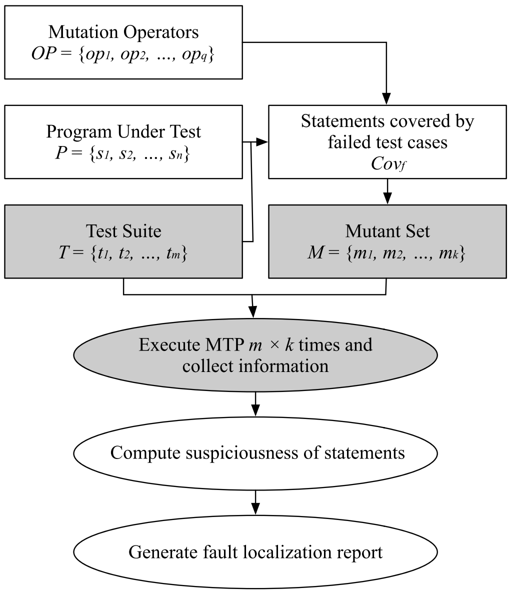

Mutation Operators
$OP = \{op_1, op_2, \ldots, op_q\}$
Program Under Test
$P = \{s_1, s_2, \ldots, s_n\}$
Statements covered by failed test cases
Cov_f
Test Suite
$T = \{t_1, t_2, \ldots, t_m\}$
Mutant Set
$M = \{m_1, m_2, \ldots, m_k\}$
Execute MTP $m \times k$ times and collect information
Compute suspiciousness of statements
Generate fault localization report

FIGURE 1. Framework of mutation-based fault localization.

only to perform special programs to locate but also to find potential triggering faults at these locations based on the output. The requirement of mutations makes it particularly powerful in terms of testing and analysis. Mutants have been proven to be very useful in simulating the behavior of real faults and can find more faults than other traditional test coverage criteria (such as statement coverage) [25].

B. MUTATION-BASED FAULT LOCALIZATION

Mutation-based fault localization (MBFL) technique is based on mutation analysis [12], and its main foundation is that a mutant can be considered as a similar version or a partial fix of real faults [2]. As shown in Figure 1, MBFL has the following four steps:

1) **Obtain statements covered by failed test cases:** given a program P with n statements and a test suite T with m test cases, MBFL will execute P against T and collect the coverage information and test results (i.e., pass or fail). Then, all test cases in T will be divided into two groups: T_p and T_f, which contains all the passed or failed test cases, respectively. All the statements covered by the failed test cases T_f are denoted as Cov_f, where the real faulty statements must belong to them according to the PIE theory [26].
2) **Generate and execute mutants:** MBFL employs a number of mutation operators to artificially inject faults into statements from Cov_f and generates a large number of mutants. It should be noted that there will be more than one mutant generated from the same statement, and the mutants generated from the statement s is denoted by $M(s)$. The test cases are executed on each mutant, and MBFL will record all the execution results. $T_n(m)$ is the set of test cases that can not kill the mutant m and $T_k(m)$ is the set of test cases that can kill the mutant m.

TABLE 1. Five suspiciousness formulas for mutation based fault localization.

Method	Formula
Jaccard [27]	$Sus(m) = \frac{a_{kf}}{a_{kf}+a_{nf}+a_{kp}}$
Ochiai [28]	$Sus(m) = \frac{a_{kf}}{\sqrt{(a_{kf}+a_{nf})*(a_{kf}+a_{kp})}}$
*Op*2 [29]	$Sus(m) = a_{kf} - \frac{a_{kp}}{a_{kp}+a_{np}+1}$
Tarantula [30]	$Sus(m) = \frac{\frac{a_{kf}}{a_{kf}+a_{kp}}}{\frac{a_{kf}}{a_{kf}+a_{nf}}+\frac{a_{kp}}{a_{kp}+a_{np}}}$
*Dstar** [31]	$Sus(m) = \frac{a_{kf}^{*}}{a_{kp}+a_{nf}}$

3) **Compute the suspiciousness:** The suspiciousness of the mutant m can be calculated by different MBFL formulas, which are based on the following four parameters: $a_{np}(m) = |T_n(m) \bigcap T_p|$, $a_{kp}(m) = |T_k(m) \bigcap T_p|$, $a_{nf}(m) = |T_n(m) \bigcap T_f|$, $a_{kf}(m) = |T_k(m) \bigcap T_f|$, where, $a_{np}(m)$ is the number of passed test cases which cannot kill m, $a_{kp}(m)$ is the number of passed test cases which can kill m, $a_{nf}(m)$ is the number of failed test cases which cannot kill m, and at last, $a_{kf}(m)$ is the number of failed test cases which can kill m. Table 1 lists five popular MBFL formulas (i.e., *Jaccard* [27], *Ochiai* [28], *Op*2 [29], *Tarantula* [30], and *Dstar** [31]). It should be noted that the parameter $*$ in *Dstar** formula is set to 3 in our study according to the recommendation of Wong *et al.* [31]. Therefore, we use $Dstar^3$ to indicate *Dstar** formula in the rest of this paper. After calculating the suspiciousness values of all mutants, MBFL techniques will assign the suspiciousness value of the statement s as the maximum suspiciousness value of the mutants generated by s: $sus(s) = max\left\{sus(m_1), sus(m_2), \cdots, sus(m_q)\right\}$, where $m_1, \cdots, m_q$ are mutants in $M(s)$ and the $sus(s)$ is the suspiciousness of the statement s.
4) **Generate fault localization report:** Similar to SBFL, MBFL will also sort all statements in the descending order based on their suspiciousness value. Finally, a ranking list is created, and it will guide the developers to localize the faults in the program and fix them.

C. COST REDUCTION TECHNIQUES FOR MBFL

Benefited from a fine-grained analysis, MBFL can achieve high accuracy of fault localization but suffers from a high execution cost. To improve the efficiency of MBFL, a variety of cost reduction techniques have been proposed. These techniques can be classified into three categories.

1) **Reduce the number of generated mutants:** The techniques in this category aim to reduce the number of mutants since fewer mutants are executed, fewer cost of MBFL technique can have. Papadakis and Le Traon [19] proposed a mutation operator reduction-based strategy called selective mutation SELECTIVE) strategy. SELECTIVE designs the specific sufficient criterion for mutation operators, which assign them with different contribution degree. In the rules, some sufficient operators will be retained

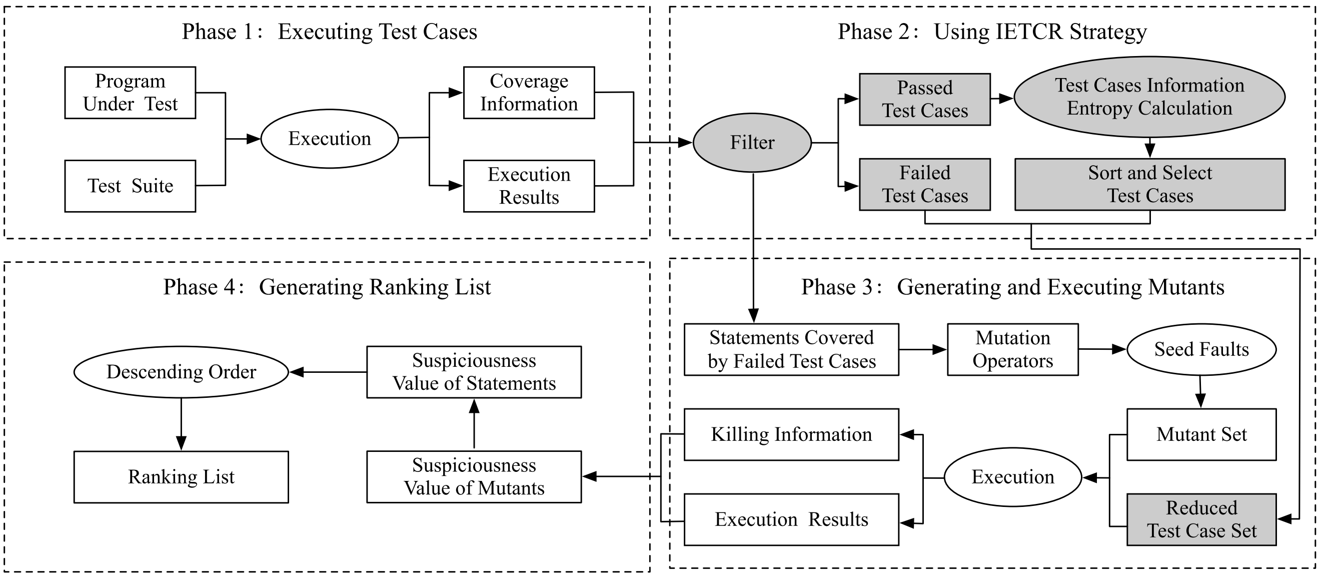

Phase 1 : Executing Test Cases
Program Under Test
Test Suite
Execution
Coverage Information
Execution Results
Phase 2 : Using IETCR Strategy
Filter
Passed Test Cases
Failed Test Cases
Test Cases Information Entropy Calculation
Sort and Select Test Cases
Phase 3 : Generating and Executing Mutants
Statements Covered by Failed Test Cases
Mutation Operators
Seed Faults
Mutant Set
Reduced Test Case Set
Execution
Killing Information
Execution Results
Phase 4 : Generating Ranking List
Descending Order
Suspiciousness Value of Statements
Ranking List
Suspiciousness Value of Mutants

FIGURE 2. Framework of mutation based fault localization with IETCR strategy.

for generating mutants, while other operators with lower contribution degree will be abandoned. Therefore, a large number of mutants will be abandoned.
From the mutant reduction aspect, a mutant sampling (SAMPLING) strategy [13] was presented by Papadakis, and it randomly extracts mutants from the total mutant set. Liu *et al.* [32] proposed a statement-oriented mutant reduction (SOME) strategy, which samples mutants at a specific percentage on statement level.

2) **Reduce the number of executed test cases:** The techniques in this category aim to reduce the test case's execution on mutants. In the work of De Oliveira *et al.* [20], they presented FTMES strategy, which only employs the failed test cases to execute mutants, but ignores all the passed test cases. Therefore the execution cost of MBFL will decrease along with the fewer execution of the test case. Since the passed test cases can also contribute to fault localization, ignoring all of them by FTMES will cause the loss of MBFL's fault localization accuracy.
3) **Optimize the mutant execution process:** The techniques in this category aim to optimize the process of mutation execution. Liu *et al.* [21] proposed a dynamic mutation execution strategy (DMES) for MBFL with two optimization techniques. The first optimization technique focuses on reducing the execution of mutants and is named as mutation execution optimization (MEO). The second optimization technique focuses on reducing the execution of test cases and is named as test case execution optimization (TEO). DMES utilizes the total set of both mutants and test cases, and such a strategy can be combined with other methods to further reduce the cost of MBFL.

In this paper, we propose the IETCR strategy, which aims to reduce the execution of the test cases from the same aspect with FTMES. While FTMES only considers the failed test cases but discards all passed test cases, which will decrease the fault localization accuracy. We propose IETCR to alleviate this issue by keeping both passed and failed test cases. The details of our proposed strategy can be found in Section III.

III. OUR PROPOSED MBFL METHOD

A. FRAMEWORK OF MBFL WITH IETCR STRATEGY

Figure 2 illustrates the framework of MBFL with our proposed IETCR strategy. In particular, it first collects coverage information and execution results of the test cases after executing the program under test. Secondly, the IETCR strategy works by filtering the passed and failed test cases, calculating the information entropy change of each passed test case. Then a reduced test set is generated according to the information entropy values. Next, mutants are generated by seeding faults in statements covered by the failed test cases. After that, it executes the reduced test cases on all mutants to obtain the killing information and execution results for calculating the suspiciousness of mutants. Finally, the suspiciousness of statements is recorded for producing a ranking list to assist the developers in finding faults.

B. DETAILS OF IETCR STRATEGY

The key idea of the IETCR strategy is to reduce the test cases by using information entropy theory. Information entropy is defined by Shannon [33], and it can be used to measure the unpredictability of the state. In this paper, we utilize information entropy to guide the reduction of test cases. Consider a buggy program under test, the probability of whether each statement is faulty is uncertain. This unpredictability is called information entropy. A lower information entropy calculated by a test suite indicates that the test suite can be better to distinguish the difference between faulty statements and other correct statements, and the quality of this test suite is higher than that with larger information entropy.

First, we formulate test case reduction as an entropy reduction process and present some necessary notations. Let $P = \{s_1, s_2, \cdots, s_m\}$ be the programs under test (PUT) with m statements; let $T = \{t_1, t_2, , t_n\}$ be the set of the test suite with n tests and $T_i = \{t_1, t_2, \cdots, t_i\}$ with i tests. Let $Sus(s|T_i)$ denote the suspiciousness of s calculated by SBFL with test suite T_i. Therefore, approximated probability that statement s_j contains the fault, based on the information observed with T_i, is calculated as the normalized suspiciousness metric for s_j.

$$\mathbf{P}_{T_i(s_j)} = \frac{Sus\left(s_j|T_i\right)}{\sum_{j=1}^{m} Sus\left(s_j|T_i\right)} \tag{1}$$

The information entropy of the test suite T_i can be defined as:

$$\mathbf{E}_{T_i}(P) = -\sum_{j=1}^{m} \mathbf{P}_{T_i}\left(s_j\right) \log_2 \mathbf{P}_{T_i}\left(s_j\right) \tag{2}$$

Secondly, to measure the information contained in an individual test case, we define the entropy of the test case t_i in $T_i = T_{i-1} \cup \{t_i\}$ by subtracting the entropy of T_{i-1} from the entropy of the whole set T_i.

$$\mathbf{EC}(t_i) = \mathbf{E}_{T_i}(P) - \mathbf{E}_{T_{i-1}}(P) \tag{3}$$

From Equation 3, $\mathbf{EC}(t_i)$ measures the entropy change of removing test case t_i from test suite T_i. It should be noted that the value of $\mathbf{EC}(t_i)$ can be negative and non-negative. Suppose **ECSet** is a set contains all the $\mathbf{EC}(t_i)$ values of test cases in T_i. We analyze three situations of how we perform the entropy-based test case selection process:

- All elements in **ECSet** are negative: it indicates that the entropy of T_i will be increased after removing each test case, which leads to the quality of T_i decreases. Then, the test case with a lower $\mathbf{EC}(t_i)$ value will have higher quality since removing this test case can decrease the quality of T to a greater extent than other test cases.
- All elements in **ECSet** are positive: it indicates that the entropy of T_i will be decreased after removing each test case, which leads to the quality of T_i increases. Then, the test case with lower $\mathbf{EC}(t_i)$ value will have higher quality since removing this test case can increase the quality of T_i to a smaller extent than other test cases.
- Elements in **ECSet** have both positive and negative: it indicates that the entropy of T_i decreases or increases after removing different test cases, which leads to the quality of T_i increases or decreases. Then, the test case with lower $\mathbf{EC}(t_i)$ value will have higher quality since test cases with negative $\mathbf{EC}(t_i)$ values will have higher quality than test cases with positive $\mathbf{EC}(t_i)$ values.

In summary, no matter in which situation, the test case with a lower $\mathbf{EC}(t_i)$ value will have higher quality and should be kept during test case selection. Thus, we select test cases with lower $\mathbf{EC}(t_i)$ values to keep the quality of test suite, which can be better to distinguish the difference between faulty statements and other correct statements. Furthermore, the value range of $\mathbf{EC}(t_i)$ is $[-\log_2 |\mathbf{m}|, \log_2 |\mathbf{m}|]$, where $|m|$ is the number of program statements.

C. ALGORITHM OF IETCR STRATEGY

Algorithm 1 IETCR Strategy

```
Input:
    Program under test $P$
    Test suite $T$
Output:
    Reduced tests set $T_{reduced}$
 1: $Cov, R \leftarrow execute(P, T)$
 2: $Sus(s) \leftarrow computeSus(Cov, R)$
 3: $T_p, T_f \leftarrow split(T, R)$
 4: $failedNum \leftarrow size(T_f)$
 5: $ECSet \leftarrow \emptyset$
 6: $E_{T_i} \leftarrow entropy(T)$
 7: for test case $t_i$ in $T_p$ do
 8:     $E_{T_{i-1}} \leftarrow entropy(T_{i-1})$
 9:     $EC(t_i) \leftarrow entropyChange(t_i)$
10:     $ECSet \leftarrow ECSet \cup EC(t_i)$
11: end for
12: $T_{reduced} \leftarrow T_f$
13: $T_{selected} \leftarrow select(failedNum, T_p, ECSet)$
14: $T_{reduced} \leftarrow T_{reduced} \cup T_{selected}$
15: return $T_{reduced}$
```

As introduced in Section III-B, the entropy values of all passed test cases can be calculated and guide the test reduction process. We consider that there is a class imbalanced issue [34] in the field of fault localization, which means that the number of failed test cases are often much smaller than the number of passed test cases in the most of test suite. And researchers found that the fault localization accuracy can be benefited from a class-balanced test suite [35]. Inspired by the work of Gao *et al.* [34] and Gong *et al.* [36], we select the same number of passed test cases as failed test cases to get a balanced test suite for MBFL.

Algorithm 1 provides the pseudo-code of the IETCR strategy. We first obtain coverage information *Cov* and testing results *R* by executing test suite *T* on the program under test *P* (Line 1). Then, we compute the suspiciousness of statements in *P* and split the test suite into two groups (Lines 3 to 4) using coverage information *Cov* and testing results *R*. Next, we calculate the entropy of each test case in T_p in the loops (Lines 7 to 11). Later, we select *failedNum* test cases from T_p according to the value of entropy change from small to large (Line 13). Finally, we return a reduced test set (Line 14).

D. AN ILLUSTRATIVE EXAMPLE

Table 2 presents an illustrative example of how the IETCR strategy works on MBFL. This example is a program segment

TABLE 2. An illustrative example when performing MBFL with IETCR strategy.

Program int mid(int x,int y,int z)		Test suite t_1 3,3,5	 t_2 1,2,3	 t_3 3,2,1	 t_4 5,5,5	 t_5 5,3,4	 t_6 2,1,4	Suspiciousness
1	int m;	●	●	●	●	●	●	2.00
2	m = z;	●	●	●	●	●	●	2.00
3	**if(y <z - 1)//fault**	●	●	●	●	●	●	**2.00**
4	if(x <y)	●					●	0.00
5	m = y;							0.00
6	else if(x <z)	●					●	0.00
7	m = x;	●					●	0.00
8	else		●	●	●	●		4.00
9	if (x >y)		●	●	●	●		4.00
10	m = y;			●		●		0.50
11	else if(x >z)		●		●			0.50
12	m = x;							0.00
13	return m;}	●	●	●	●	●	●	2.00
Results		**P**	**F**	**P**	**P**	**F**	**P**	
Entropy		2.73e-11	-	3.91e-10	3.91e-10	-	2.73e-11	

		Test Suite						Suspiciousness	
Statements	Mutants	t_1	t_2	t_3	t_4	t_5	t_6	Original	IETCR

1 int m;
2 m = z;									

3 **if(y <z- 1)//fault**	M_1:<→<=	0	1	0	0	1	0	+∞	+∞
	M_2:<→>	1	0	1	0	0	1	0.00	0.00
	M_3:<→>=	1	1	1	0	1	0	4.00	8.00
	M_4:<→==	1	1	0	0	1	1	4.00	4.00
	M_5:<→!=	0	0	1	0	0	0	0.00	0.00
	M_6:<→true	0	1	1	0	1	0	8.00	+∞
	M_7:<→false	1	0	0	0	0	1	0.00	0.00
4 if(x <y)	M_8:<→<=	0	0	0	0	0	0	**0.00**	**0.00**
	M_9:<→>	0	0	0	0	0	1	0.00	0.00
	M_{10}:<→>=	0	0	0	0	0	0	0.00	0.00
	M_{11}:<→==	0	0	0	0	0	0	0.00	0.00
	M_{12}:<→!=	0	0	0	0	0	1	0.00	0.00
	M_{13}:<→true	0	0	0	0	0	1	0.00	0.00
	M_{14}:<→false	0	0	0	0	0	0	0.00	0.00
5 m = y;									
6 else if(x >z)	M_{15}:<→<=	0	0	0	0	0	0	**0.00**	**0.00**
	M_{16}:<→>	1	0	0	0	0	1	0.00	0.00
	M_{17}:<→>=	1	0	0	0	0	1	0.00	0.00
	M_{18}:<→==	1	0	0	0	0	1	0.00	0.00
	M_{19}:<→!=	0	0	0	0	0	0	0.00	0.00
	M_{20}:<→true	0	0	0	0	0	0	0.00	0.00
	M_{21}:<→false	1	0	0	0	0	1	0.00	0.00
7 m = x; 8 else									
9 if (x >y)	M_{22}:>→!=	0	1	0	0	0	0	**1.00**	**1.00**
	M_{23}:>→==	0	0	1	0	1	0	0.50	1.00
	M_{24}:>→>=	0	1	0	0	0	0	1.00	1.00
	M_{25}:>→false	0	0	1	0	1	0	0.50	1.00
	M_{26}:>→>	0	1	0	0	0	0	1.00	1.00
	M_{27}:>→true	0	1	0	0	0	0	1.00	1.00
	M_{28}:>→<=	0	0	0	0	0	0	0.00	0.00
10 m = y;									
11 else if(x >z)	M_{29}:>→<=	0	0	0	0	0	0	0.00	0.00
	M_{30}:>→>	0	1	0	0	0	0	**1.00**	**1.00**
	M_{31}:>→>=	0	1	0	0	0	0	1.00	1.00
	M_{32}:>→==	0	0	0	0	0	0	0.00	0.00
	M_{33}:>→!=	0	1	0	0	0	0	1.00	1.00
	M_{34}:>→true	0	1	0	0	0	0	1.00	1.00
	M_{35}:>→false	0	0	0	0	0	0	0.00	0.00

12 m = x;
13 return m;}									

called $mid()$, which takes three integers as parameters and returns the middle value of them. Statement s_3 contains a fault, as it should be $if(y < z)$. The test suite has six test cases, from t_1 to t_6, whose coverage information are marked with black bullets (●). There are two failed test cases (t_2 and t_5), while the rest are four passed test cases.

IETCR strategy first employs the SBFL technique to calculate the suspiciousness values of all statements. It should be noted that the SBFL used in this paper is $Dstar^3$ [31], which was proved to be a state-of-the-art SBFL formula. Next, the entropy of the whole test suite can be calculated by Equation 2 as follows.

$$\mathbf{E}_{T_6}(P) = -\sum_{j=1}^{13} \mathbf{P}_{T_6}\left(s_j\right) \log_2 \mathbf{P}_{T_6}\left(s_j\right)$$

When removing the passed test case t_1, the entropy of the rest five test cases can be calculated as follows.

$$\mathbf{E}_{T_5}(P) = -\sum_{j=1}^{13} \mathbf{P}_{T_5}\left(s_j\right) \log_2 \mathbf{P}_{T_5}\left(s_j\right)$$

Therefore, the entropy change of t_1 is calculated by Equation 3 as follows.

$$\mathbf{EC}(t_1) = \mathbf{E}_{T_6}(P) - \mathbf{E}_{T_5}(P) = 2.73\text{e-}11$$

Similarly, the entropy change values of other three passed test cases (i.e., t_3, t_4 and t_6) are 3.91e-10, 3.91e-10, 2.73e-11 respectively. Hence, IETCR strategy selects the two passed test cases (t_1 and t_6) with lower EC values and ignored other passed test cases, which can reduce the execution cost of MBFL.

As shown in Table 2, for the original MBFL, it utilizes the killing information of all test cases (from t_1 to t_6) to calculate suspiciousness for mutants, which is listed under the "Original" column. The maximum suspiciousness in the group of mutants is assigned to the suspiciousness of statements, which are highlighted in boldface. And for MBFL with the IETCR strategy, MBFL ignores t_3 and t_4, and the corresponding execution cost is reduced, which is highlighted with red color.

From the results of the "Original" and "IETCR" columns, we can find that MBFL with the IETCR strategy has the same fault localization accuracy when compared with original MBFL, which returns the same suspiciousness of statements. For the fault localization accuracy, both MBFL with IETCR strategy and the original MBFL can rank the exact faulty statement s_3 at the top of the ranking list. Therefore, in this illustrative example, using the IETCR strategy can reduce the number of execution costs under the premise of losing fault localization accuracy.

IV. EXPERIMENTAL SETUP

A. RESEARCH QUESTIONS

To evaluate the effectiveness of our proposed IETCR strategy, we design empirical studies from two aspects: fault localization accuracy (effectiveness) and mutation execution cost (efficiency). In our study, we mainly investigate the following two research questions (RQs):

- **RQ1:** Compared with original MBFL and two state-of-the-art optimization strategies, how does the IETCR strategy perform in terms of mutation execution cost reduction rate?
- **RQ2:** Compared with original MBFL and two state-of-the-art optimization strategies, how does the IETCR strategy perform in terms of fault localization accuracy?

In **RQ1**, we attempt to quantitatively analyze the performance of the IETCR strategy in terms of efficiency. In **RQ2**, we want to examine the performance of the IETCR strategy in terms of effectiveness.

In our experiments, we first choose the original MBFL without any strategy as the first baseline. Then we choose MBFL with two optimization strategies (i.e., FTMES [20] strategy and SAMP(30%) strategy) as the second baseline and the third baseline. In particular, FTMES is the latest test case reduction strategy of MBFL, and SAMP(30%) is SAMPLING strategy [13], which randomly selects 30% test cases from the test suite. Previously studies [13], [32] showed that randomly choosing 30% mutants can keep the loss of MBFL's fault localization accuracy at a low level; therefore we also use the same SAMPLING ratio in our study. Besides, we use $Dstar^3$ as the SBFL formula when calculating the entropy of test cases in our proposed IETCR strategy, because it has been proved as a state-of-the-art SBFL formula [31].

All experiments were performed on Linux system (version 3.10.0-957.el7.x86-64) with 18 cores CPU (Intel(R) Xeon(R) Gold 6240 CPU@2.60GHz).

B. SUBJECT PROGRAMS

We evaluate the effectiveness of the IETCR strategy on six subject programs from dataset SIR [37], which has been widely adopted by recent studies on fault localization and program repair [21], [32], [38]. The first four programs are relatively small-scale programs with hundreds of lines of code from Siemens Suite, while the last two programs (sed and grep) are large-scale real-world programs. All of these are real-world C programs with faulty versions and test suites provided by SIR. Table 3 presents the information of all subject programs. There are 112 faulty versions from six programs; The reasons of excluding some versions are summarized as follows: (1) the related test suite of some versions cannot defect failures on the faulty versions; (2) the failures of some versions lead to segment faults or other runtime errors, and it is hard to collect full coverage information for these faulty versions.

Therefore, a total of 112 faulty versions are used in our experiments. In our study, we use the GNU gcov tool [39] to collect coverage information and Proteum tool [40] to generate mutants, both of these tools are popular and have been widely adopted by many researchers [12], [21], [32]. Besides, we employ mutation operators suggested by the

TABLE 3. The characteristics of subject programs.

Program	#Versions(used)	#LOC	#Mutants	#Test Cases
printtokens	7 (7)	342	4235	4130
schedule	10 (10)	296	2223	2650
totinfo	33 (28)	273	6314	1052
tcas	48 (48)	139	5115	1608
sed	9 (9)	11470	78543	360
grep	10 (10)	13826	86151	668

work of Agrawal *et al.* [41]. The suspiciousness formulas used in our study are listed in Table 1.

C. PERFORMANCE METRICS

In our study, we use a Mutant-Test-Pair metric to evaluate the efficiency of MBFL and use the EXAM score metric to evaluate the accuracy of MBFL, which will be introduced as follows.

1) MUTANT-TEST-PAIR METRIC

A Mutant-Test-Pair (*MTP*) counts the number of mutant executions on the test cases. *MTP* measures the mutant execution cost of MBFL and has been used in previous studies [32], [42], [43]. The idea of *MTP* is that the number of mutation execution is linked to the computational cost required to obtain the rank of statements [32]. Compared with the actual run-time cost, *MTP* metric has the advantage of avoiding the influence of the run-time environment. A lower *MTP* value means the corresponding MBFL technique has better efficiency. Assume that a mutant set with n mutants and they are executed by a test suite with m test cases, *MTP* can be calculated by the following formula:

$$MTP = m \times n \tag{4}$$

2) EXAM SCORE METRIC

EXAM score [44] ($EXAM$) is the percentage of program elements that have to be inspected until finding the exact faulty element. It is a commonly used metric for fault localization techniques, and a lower *EXAM* indicates a better fault localization technique [45], [46]. The *EXAM* measures the relative position of the faulty element in the ranking list, and the formula of *EXAM* can be defined as follows.

$$EXAM = \frac{rank}{Number\ of\ executable\ statements} \tag{5}$$

The numerator in Equation 5 represents the rank of the faulty statement in the ranking list. And the denominator is the total number of statements that need to be checked. More specifically, *rank* can be calculated by:

$$rank = \frac{(i+1)+(i+j)}{2} \tag{6}$$

In Equation 6, i is the number of non-faulty statements whose suspiciousness value is higher than the faulty statement, and j is the number of statements that share the same suspiciousness value with the faulty statement. To break the tie, we take the average of the first $(i+1)$ and last $(i+j)$ ranks to determine the rank of the faulty statement.

TABLE 4. Average reduction rates of IETCR, FTMES, and SAMP(30%) strategies when compared to the original MBFL without using any strategy in terms of MTP metric.

Programs	IETCR	FTMES	SAMP(30%)
printtokens	64.1%	80.8%	70.0%
schedule	56.3%	73.1%	70.0%
totinfo	71.2%	85.0%	70.0%
tcas	88.3%	94.1%	70.0%
sed	67.3%	87.3%	70.0%
grep	76.8%	87.3%	70.0%
Average	71.0%	83.5%	70.0%

V. RESULTS ANALYSIS

A. ANSWER FOR RQ1

To answer RQ1, we measure the mutation execution cost of MBFL in terms of the MTP metric. We calculate the executed mutant number (i.e., *MTP*) of three optimization strategies (i.e., our proposed IETCR strategy, FTMES strategy, and SAMP(30%) strategy) and the original MBFL without using any strategy. Notice, to reduce the bias caused by the random process in SAMP(30%) strategy, we repeat the experiments 50 times with different random seeds and use the average *MTP* value.

The results are shown in Figure 3. In this figure, we choose the program version as the independent variable and the MTP percentage of original MBFL as the dependent variable, which shows the percentage of the MTP calculated by each method to the original MBFL. A lower MTP percentage of original MBFL demonstrates a better mutation execution cost reduction. Each sub-figure has three broken lines, which represent three optimization strategies IETCR, FTMES, and SAMP(30%), respectively. The results show that IETCR can significantly reduce the MTP execution cost, but in most cases, FTMES has a lower MTP percentage of original MBFL than IETCR. The reason is that IETCR selects all failed test cases and the same number of passed test cases, while FTMES only keeps the failed test cases, which leads to the double cost of IETCR.

Table 4 further investigates the exact reduction rate of different optimization strategies when compared to the original MBFL without any strategy. The average reduction rate of the IETCR strategy (71.0%) is larger than that of SAMP(30%) strategy (70.0%) and smaller than that of the FTMES strategy (83.5%). As shown in this table, FTMES has the highest reduction rate. The reason is that at the same number of mutants, the FTMES strategy can only execute the failed test cases and ignores all the passed test cases, while the IETCR strategy and SAMP(30%) strategy also execute some passed test cases.

To measure the cost of IETCR more precisely, we have gathered the time of test case entropy change calculation on each program version. Figure 4 shows the calculation time in seconds with box plots. In Figure 4, the range of calculation time is from 21 seconds to 832 seconds, and program printtokens and grep require more time for entropy change calculation while totinfo and tcas require less time for entropy change calculation. Therefore, the calculation time of

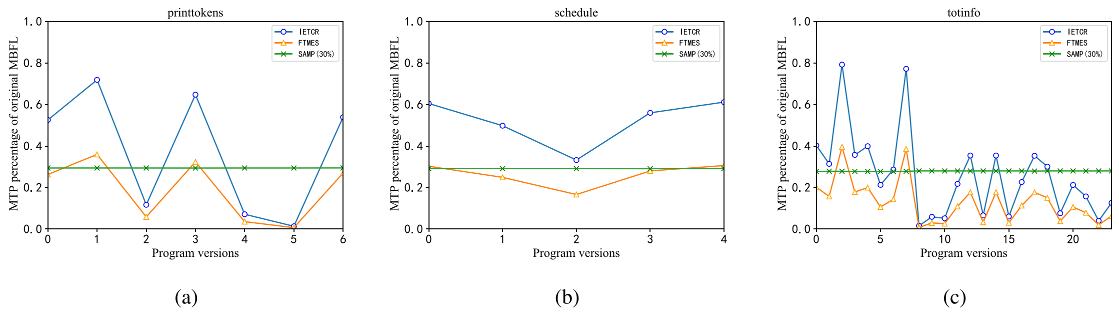
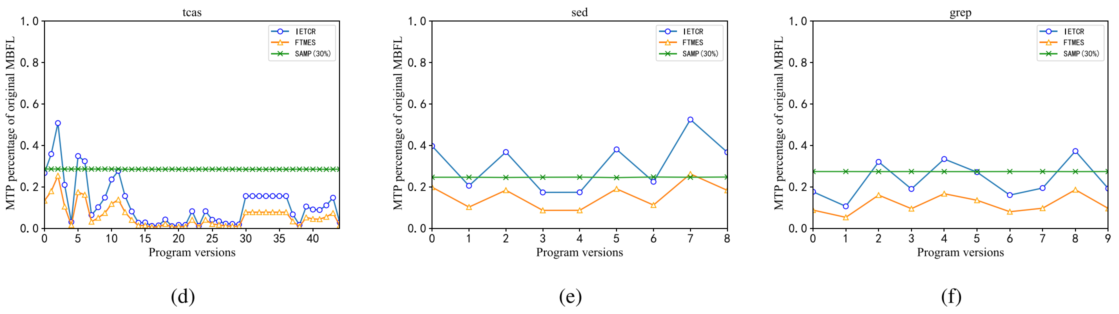



FIGURE 3. Cost reduction performance of using different strategies.

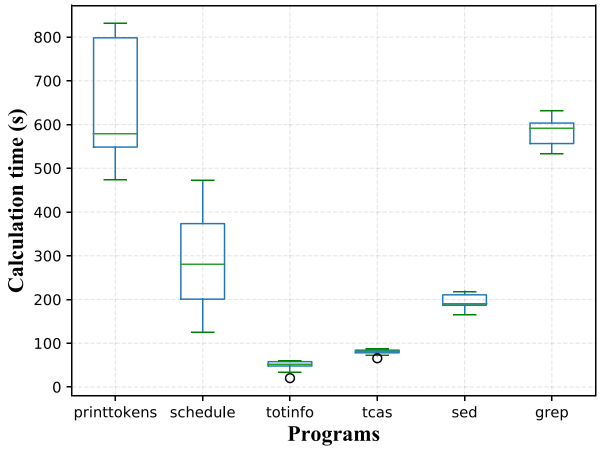


FIGURE 4. Calculation time of test case entropy change on different programs.

test case entropy change is worth considering the significant savings in test case reduction.

Summary for RQ1: IETCR strategy has a more tremendous mutation execution reduction rate than SAMP(30%) strategy, but smaller than FTMES strategy on average, since IETCR strategy can execute some passed test cases than FTMES strategy. Additionally, the calculation time of IETCR is worth considering the significant savings in test case reduction.

B. ANSWER FOR RQ2

In RQ2, we focus on the fault localization accuracy and compare our proposed IETCR strategy with two optimization strategies and the original MBFL without any strategy. To answer RQ2, we use the EXAM score metric to show the fault localization accuracy. The comparison results of using five different suspiciousness formulas of our proposed strategy and three baselines via the violin plots are shown in Figure 5. In these violin plots, the x-axis represents different MBFL methods, while the y-axis indicates the fault localization accuracy. Each block in the violin plot suggests the distribution of the EXAM score metric when considering the corresponding suspiciousness formula. The breadth of the block represents the data density of the corresponding value of the y-axis for all subject program versions. Therefore, the wider in the bottom of the block and the thinner in the up of the block indicate that the corresponding MBFL technique has a better fault localization accuracy.

From Figure 5, we can find that no matter which suspiciousness formula is used to calculate the suspiciousness of mutants, our proposed IETCR strategy can achieve the similar performance of the original MBFL without any strategy. Moreover, we also find the distribution of the EXAM score metric when using the IETCR strategy is almost around 0, which is more concentrated than two optimization strategies (i.e., FTMES and SAMP(30%)). This indicates the fault localization accuracy of the IETCR strategy is higher than two baseline optimization strategies.

More detailed comparison results can be found in Table 5. In this table, we consider the various thresholds of the EXAM score, which is shown in the first column. Then the other columns are the percentage of 112 faulty versions whose EXAM score is smaller than the corresponding threshold. In Table 5, the results highlighted with bold are the best performance when given the suspiciousness formula and EXAM score threshold. Notice, JA denotes *Jaccard*, OC denotes

TABLE 5. Comparison of fault localization accuracy of different MBFL techniques when considering different EXAM thresholds.

EXAM	Original MBFL					IETCR					FTMES					SAMP(30%)				
threshold	JA	OC	OP	TA	DS	JA	OC	OP	TA	DS	JA	OC	OP	TA	DS	JA	OC	OP	TA	DS
1%	**0.31**	**0.31**	**0.27**	**0.05**	0.30	0.29	0.29	**0.27**	0.02	0.28	0.03	0.03	0.03	0.01	0.03	0.22	0.22	0.20	**0.05**	0.22
5%	**0.72**	**0.74**	**0.70**	**0.37**	**0.74**	0.70	0.70	0.64	0.21	0.69	0.23	0.23	0.27	0.19	0.24	0.55	0.57	0.56	0.28	0.58
10%	0.79	**0.81**	**0.76**	**0.55**	**0.80**	**0.80**	0.80	0.73	0.33	0.77	0.36	0.36	0.39	0.27	0.36	0.63	0.65	0.63	0.42	0.66
15%	0.80	**0.82**	**0.79**	**0.61**	**0.82**	**0.82**	**0.82**	0.78	0.41	0.81	0.50	0.50	0.57	0.34	0.50	0.68	0.68	0.69	0.48	0.69
20%	**0.84**	**0.84**	**0.81**	**0.69**	**0.84**	0.82	0.82	0.78	0.42	0.81	0.52	0.52	0.58	0.35	0.55	0.71	0.74	0.71	0.55	0.73
30%	**0.88**	**0.89**	**0.86**	**0.80**	**0.88**	**0.88**	0.88	**0.86**	0.58	**0.88**	0.64	0.64	0.74	0.49	0.66	0.75	0.78	0.77	0.62	0.78
40%	**0.88**	**0.89**	**0.88**	**0.81**	**0.89**	**0.88**	0.88	**0.88**	0.78	0.88	0.73	0.73	0.83	0.62	0.74	0.81	0.83	0.82	0.67	0.83
50%	0.89	**0.92**	**0.90**	0.83	**0.91**	**0.90**	0.90	**0.90**	**0.90**	0.81	0.90	0.75	0.85	0.67	0.78	0.87	0.86	0.85	0.81	0.86
60%	0.94	0.94	0.92	**0.90**	0.93	**0.95**	**0.95**	**0.93**	**0.90**	**0.95**	0.85	0.86	0.92	0.80	0.89	0.89	0.89	0.89	0.88	0.89
70%	**0.95**	**0.95**	**0.93**	**0.94**	0.94	**0.95**	**0.95**	**0.93**	0.90	**0.95**	0.90	0.90	0.92	0.86	0.92	0.90	0.89	0.89	0.89	0.89
80%	**0.97**	**0.97**	**0.96**	**0.97**	**0.97**	**0.97**	**0.97**	**0.96**	0.96	**0.97**	0.96	0.96	0.94	0.94	0.96	0.93	0.93	0.93	0.92	0.92
90%	**0.98**	**0.98**	**0.98**	**0.98**	**0.98**	**0.98**	**0.98**	**0.98**	**0.98**	**0.98**	0.97	0.97	0.96	0.97	0.97	**0.98**	**0.98**	**0.98**	**0.98**	**0.98**
100%	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**

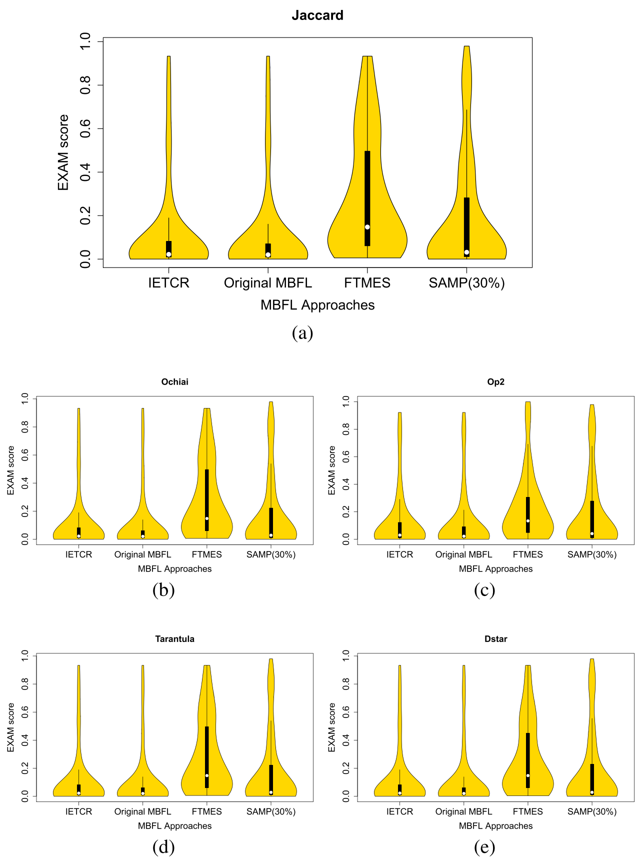


FIGURE 5. Comparison of fault localization accuracy of different MBFL approaches with different formulas.

Ochiai, OP denotes *Op*2, TA denotes *Tarantula*, and DS denotes *Dstar*^{3}.

For example, when we use the original MBFL with the suspiciousness formula *Jaccard*, there are 31% faulty versions whose EXAM score is lower than 1%, which performs best when compared with MBFL with IETCR strategy (0.28), FTMES strategy (0.03), and SAMP(30%) strategy (0.22). For the different formulas with the same MBFL approach, both *Ochiai* and *Dstar*^{3} have the best performance in original MBFL, and *Dstar*^{3} has the best performance in IETCR strategy and FTMES strategy. Besides, for the same suspiciousness formulas, IETCR strategy can achieve better fault localization accuracy than FTMES strategy and SAMP(30%) strategy in all the formulas.

In addition, we also compare the fault localization accuracy of different MBFL techniques when considering different programs, and the results are shown in Table 6. Due to the horizontal space limitation for the table, we only show two digits after the decimal point. In Table 6, we can find that in most cases, the EXAM Scores of IETCR are close to original MBFL and lower than FTMES and SAMP(30%). Only in a few highlighted cases, the corresponding EXAM Scores of IETCR are higher than or equal to that of FTMES or SAMP(30%).

To analyze the statistical significance between our proposed IETCR strategy and other baselines, we employ the Wilcoxon signed-rank test [47] at a confidence level of 95%, since Wilcoxon signed-rank test does not have any distribution requirement for the underlying data. Table 7 summarizes average value and statistical analysis results on the fault localization accuracy of IETCR with three baselines (Original MBFL, FTMES, and SAMP(30%)). These results are presented in five different suspicious formulas. In Table 7, the values in brackets are the *p*-Values, and the *p*-value (in bold) greater than 0.05 represent that there is no statistically significant difference between MBFL with IETCR strategy and the corresponding MBFL baseline technique. In Table 7, we first find that MBFL with IETCR has a better fault localization accuracy than FTMES and SAMP(30%) on average in terms of EXAM score when using different formulas, while almost the same to the original MBFL (such as *Jaccard* and *Dstar*^{3}). Second, according to the *p*-Values in brackets, there is no statistically significant difference between MBFL with IETCR strategy and the original MBFL on the fault localization accuracy when using *Jaccard*, *Ochiai*, *Op*2, *Tarantula* and *Dstar*^{3}. But the fault localization accuracy of MBFL with FTMES and SAMP(30%) strategies have significant differences with the IETCR strategy.

Summary for RQ2: MBFL with the IETCR strategy has better fault localization accuracy than MBFL with the other two optimization strategies. Further statistical testing results

TABLE 6. Comparison of fault localization accuracy of different MBFL techniques when considering different programs.

Program	Original MBFL					IETCR					FTMES					SAMP(30%)				
	JA	OC	OP	TA	DS	JA	OC	OP	TA	DS	JA	OC	OP	TA	DS	JA	OC	OP	TA	DS
printtokens	0.19	0.18	0.19	0.25	0.19	0.19	0.18	0.19	0.34	0.19	0.32	0.32	0.22	0.35	0.28	0.34	0.34	0.33	0.32	0.34
schedule	0.28	0.28	0.40	0.20	0.35	0.32	0.32	0.41	0.25	0.36	0.46	0.46	0.47	0.33	0.45	0.41	0.41	0.41	0.39	0.41
totinfo	0.09	0.07	0.11	0.23	0.08	0.07	0.07	0.13	0.25	0.07	0.37	0.37	0.32	0.53	0.36	0.21	0.20	0.20	0.29	0.20
tcas	0.11	0.10	0.11	0.18	0.10	0.11	0.11	0.12	0.34	0.11	0.27	0.27	0.22	0.33	0.26	0.28	0.27	0.29	0.34	0.28
sed	0.00	0.00	0.01	0.22	0.00	0.00	0.00	0.01	0.28	0.00	0.09	0.09	0.03	0.35	0.06	0.34	0.34	0.34	0.39	0.34
grep	0.00	0.00	0.01	0.02	0.01	0.01	0.01	0.01	0.06	0.01	0.04	0.04	0.03	0.08	0.04	0.28	0.28	0.28	0.28	0.28

show that there is no significant difference between original MBFL and MBFL with the IETCR strategy.

VI. THREATS TO VALIDITY

A. THREATS TO INTERNAL VALIDITY

The main threat is the mutation tools used in our experiments. We choose Proteum for mutation analysis, which is a popular mutation analysis tool and is widely used in previous studies [12], [13], [21], [32]. Since using different mutation tools may affect our empirical results, we will use other mutation tools in our future work (such as Major [48]).

B. THREATS TO EXTERNAL VALIDITY

The first threat is the choice of baselines. We choose FTMES and SAMP (30%) as our baselines since FTMES is the latest test case reduction strategy for MBFL, and SAMP(30%) is SAMPLING strategy [13], which randomly selects 30% test cases from the test suite. Previously studies [13], [32] showed that randomly choosing 30% mutants can keep the loss of MBFL's fault localization at a low level; therefore, we use the same SAMPLING ratio in our study. The second threat is the implementation correctness of baselines, we implemented these baseline methods strictly based on the description of the corresponding papers, and the actual performance of these baselines is very close to the results in the corresponding papers.

C. THREATS TO CONSTRUCT VALIDITY

In our study, we use *MTP* as the mutation reduction performance metric and EXAM score as the fault localization accuracy metric. The former metric is popularly used in evaluating the cost of MBFL [12], [21], [32], and the latter one is also a commonly used fault localization metric in previous studies [45], [46]. Therefore, the performance metrics used in our study can reflect the real situation in evaluating MFL methods. In the future, we also want to evaluate our method in terms of other performance metrics (such as acc@n [31], [45]).

D. THREATS TO CONCLUSION VALIDITY

To show whether there exists a statistical difference between our proposed IETCR strategy with baselines in terms of the EXAM score metric, we use the Wilcoxon signed-rank test. Since this kind of statistical test method has been widely used in previous studies [21], [49]–[52].

TABLE 7. Average EXAM Score and *p*-value of different MBFL techniques.

Formula	IETCR	Original	FTMES	SAMP(30%)
Jaccard	0.10(−)	0.10(**0.47**)	0.27(0.00)	0.28(0.00)
Ochiai	0.10(−)	0.09(**0.36**)	0.27(0.00)	0.27(0.00)
Op2	0.12(−)	0.11(**0.36**)	0.22(0.00)	0.28(0.00)
Tarantula	0.28(−)	0.19(**0.09**)	0.35(0.04)	**0.33(0.70)**
Dstar^3	0.10(−)	0.10(**0.29**)	0.25(0.00)	0.28(0.00)

VII. RELATED WORK

A. FAULT LOCALIZATION TECHNIQUES

Spectrum-Based Fault Localization (SBFL) is one of the most studied techniques among the fault localization techniques, such as slice-based techniques [53], mutation-based techniques [13], [19] and machine learning-based techniques [54]. In addition to the studies in suspicious formulas [29]–[31], researchers have reported the approaches with various covered information such as call sequences [55], du-pairs [56], statement frequency [57], and so on. Moreover, considering that test suite influences on the effectiveness of fault localization, Dandan *et al.* [58] and Vidács *et al.* [59] presented strategies for prioritizing and reducing test suite.

In Mutation-Based Fault Localization (MBFL) context, MUSE (MUtation-baSEd FL) [14] and Metallaxis-FL [13] are two MBFL pioneer techniques. Both these two techniques are based on mutation analysis [19], which relies on the assumption that most of the mutations from "realistic" faults, even if artificially seeded [13]. In particular, MUSE aims at producing diverse program behaviors by mutation operators, and some metrics that reflect these behaviors according to the test result are used to measure the probabilities of statements containing faults. Previous works indicate that Metallaxis-FL outperforms MUSE on efficiency and accuracy [15].

B. COST REDUCTION ON MUTATION ANALYSIS

The mutation-based fault localization (MBFL) technique is based on mutation analysis, which is a high computational cost technique. There are three categories in cost reduction: 1) reduce the number of generated mutants; 2) reduce the number of executed test cases; 3)Optimize the mutant execution process.

For reducing created mutants, mutant sampling is a simple way to randomly select a subset of mutants from the set of all generated mutants. Papadakis and Le Traon [13] proposed a mutant sampling strategy that extracts mutants from the whole mutant set. Liu *et al.* [32] presented a statement-oriented mutant reduction strategy, which samples mutants at a specific percentage on statement level. Another alternative way is selective mutation. The underlying idea is that certain types of mutants may be more important than others. Namin *et al.* [60] used a statistical analysis procedure to identify a small set of mutation operators, which is still sufficient for measuring test effectiveness. Their results showed that it is possible to reduce the number of mutants by approximately 93%. Papadakis and Traon proposed SELECTIVE [19] to identify the specific sufficient mutant operators for generating mutants. Besides, higher-order mutants are used for reducing the number of mutants. In this case, mutants are generated by inserting two or more faults at the same time. Polo *et al.* [61] presented three strategies to combine first-order mutants and generate second-order mutants. They found that mutant combination can achieve significant cost reductions without any effectiveness loss. Later, higher-order combination strategies have been shown that they can achieve a relatively good trade-off between cost and effectiveness [62]–[64]. In particular, Papadakis and Malevris [63] found that second-order strategies can accomplish reductions of 80% to 90% of the equivalent mutants and roughly 30% of the test cases, with approximately 10% or less of fault localization effectiveness loss. Furthermore, Parsai *et al.* [65] built a prediction model that estimates the first-order mutation coverage based on higher-order mutation coverage and the model can describe the real behavior of the vast majority of higher-order mutants.

For reducing executed test cases, De Oliveira *et al.* [20] presented FTMES strategy that only employs the failed test cases to execute mutants, but ignores all the passed test cases. Therefore the execution cost of MBFL will decrease along with the fewer execution of the test case. Since the passed test cases can also contribute to fault localization, ignoring all of them by FTMES will cause the loss of MBFL's fault localization accuracy.

The idea of employing test case prioritization is first proposed by Just *et al.* [66] to optimize the mutant execution process, and Zhang *et al.* [42] optimized the mutation testing process by using the former execution information to guide the latter executions. Kapoor and Bowen [67] presented a method to improve mutation testing by identifying the fault and mutant hierarchies, where a strategy is given to suggest the ordering of the mutants such that if a mutant is stronger than another, then test case killing the stronger will automatically kill the weaker. Besides, Liu *et al.* [21] proposed a dynamic mutation execution strategy (DMES) for MBFL with mutation execution optimization and test case execution optimization. DMES utilizes the total set of both mutants and test cases, and such a strategy can be combined with other methods to further MBFL cost reduction.

VIII. CONCLUSIONS AND FUTURE WORK

In this paper, we present a novel information entropy based test case reduction (IETCR) strategy for mutation-based fault localization. IETCR adopts information theory to calculate the entropy change of the test cases, then generates a rank list of test cases by sorting them by the entropy in the ascending order. IETCR is a strategy of considering both passed and failed test cases, and it keeps all failed test cases and selects some of the passed test cases according to the entropy. We evaluate the effectiveness of IETCR on 112 faulty versions from six real-world programs, and the results show that using the IETCR strategy can effectively reduce the execution cost and maintain almost the same fault localization accuracy with original MBFL. For fault localization accuracy, our evaluation results also indicate that the IETCR strategy outperforms the two state-of-the-art baseline strategies on MBFL (i.e., FTMES and SAMP(30%)).

In the future, we first want to improve the IETCR strategy by combining other techniques [21], [32] for further reducing the execution cost of MBFL. We second want to verify the effectiveness of our proposed IETCR strategy on MBFL by considering more larger-scale real-word programs (such as programs from *Defects4J* [68]).

REFERENCES

[1] W. E. Howden, "Theoretical and empirical studies of program testing," *IEEE Trans. Softw. Eng.*, vol. SE-4, no. 4, pp. 293–298, Jul. 1978.
[2] D. Shin and D.-H. Bae, "A theoretical framework for understanding mutation-based testing methods," in *Proc. IEEE Int. Conf. Softw. Test., Verification Validation (ICST)*, Apr. 2016, pp. 299–308.
[3] P. Li, M. Jiang, and Z. Ding, "Fault localization with weighted test model in model transformations," *IEEE Access*, vol. 8, pp. 14054–14064, 2020.
[4] W. E. Wong, R. Gao, Y. Li, R. Abreu, and F. Wotawa, "A survey on software fault localization," *IEEE Trans. Softw. Eng.*, vol. 42, no. 8, pp. 707–740, Aug. 2016.
[5] Y. Wang, Z. Huang, B. Fang, and Y. Li, "Spectrum-based fault localization via enlarging non-fault region to improve fault absolute ranking," *IEEE Access*, vol. 6, pp. 8925–8933, 2018.
[6] Y. Xiaobo, B. Liu, and W. Shihai, "An analysis on the negative effect of multiple-faults for spectrum-based fault localization," *IEEE Access*, vol. 7, pp. 2327–2347, 2019.
[7] H. He, J. Ren, G. Zhao, and H. He, "Enhancing spectrum-based fault localization using fault influence propagation," *IEEE Access*, vol. 8, pp. 18497–18513, 2020.
[8] F. Keller, L. Grunske, S. Heiden, A. Filieri, A. van Hoorn, and D. Lo, "A critical evaluation of spectrum-based fault localization techniques on a large-scale software system," in *Proc. IEEE Int. Conf. Softw. Qual., Rel. Secur. (QRS)*, Jul. 2017, pp. 114–125.
[9] A. Arrieta, S. Segura, U. Markiegi, G. Sagardui, and L. Etxeberria, "Spectrum-based fault localization in software product lines," *Inf. Softw. Technol.*, vol. 100, pp. 18–31, Aug. 2018.
[10] S. Moon, Y. Kim, M. Kim, and S. Yoo, "Hybrid-MUSE: Mutating faulty programs for precise fault localization," KAIST, Daejeon, South Korea, Tech. Rep., 2014.
[11] M. Papadakis and Y. Le Traon, "Using mutants to locate 'unknown' faults," in *Proc. IEEE 5th Int. Conf. Softw. Test., Verification Validation*, Apr. 2012, pp. 691–700.
[12] M. Kooli, F. Kaddachi, G. D. Natale, A. Bosio, P. Benoit, and L. Torres, "Computing reliability: On the differences between software testing and software fault injection techniques," *Microprocessors Microsyst.*, vol. 50, pp. 102–112, May 2017.
[13] M. Papadakis and Y. Le Traon, "Metallaxis-FL: Mutation-based fault localization," *Softw. Test., Verification Rel.*, vol. 25, nos. 5–7, pp. 605–628, Aug. 2015.
[14] S. Moon, Y. Kim, M. Kim, and S. Yoo, "Ask the mutants: Mutating faulty programs for fault localization," in *Proc. IEEE 7th Int. Conf. Softw. Test., Verification Validation*, Mar. 2014, pp. 153–162.
[15] S. Pearson, J. Campos, R. Just, G. Fraser, R. Abreu, M. D. Ernst, D. Pang, and B. Keller, "Evaluating and improving fault localization," in *Proc. IEEE/ACM 39th Int. Conf. Softw. Eng. (ICSE)*, May 2017, pp. 609–620.
[16] X.-Y. Zhang, Z. Zheng, and K.-Y. Cai, "Exploring the usefulness of unlabelled test cases in software fault localization," *J. Syst. Softw.*, vol. 136, pp. 278–290, Feb. 2018.

[17] J. Tu, X. Xie, and B. Xu, "Code coverage-based failure proximity without test oracles," in *Proc. IEEE 40th Annu. Comput. Softw. Appl. Conf. (COMPSAC)*, Jun. 2016, pp. 133–142.

[18] T. B. Noor and H. Hemmati, "Studying test case failure prediction for test case prioritization," in *Proc. 13th Int. Conf. Predictive Models Data Anal. Softw. Eng. (PROMISE)*, 2017, pp. 2–11.

[19] M. Papadakis and Y. Le Traon, "Effective fault localization via mutation analysis: A selective mutation approach," in *Proc. 29th Annu. ACM Symp. Appl. Comput. (SAC)*, 2014, pp. 1293–1300.

[20] A. A. L. de Oliveira, C. G. Camilo-Junior, E. N. de Andrade Freitas, and A. M. R. Vincenzi, "FTMES: A failed-test-oriented mutant execution strategy for mutation-based fault localization," in *Proc. IEEE 29th Int. Symp. Softw. Rel. Eng. (ISSRE)*, Oct. 2018, pp. 155–165.

[21] Y. Liu, Z. Li, R. Zhao, and P. Gong, "An optimal mutation execution strategy for cost reduction of mutation-based fault localization," *Inf. Sci.*, vol. 422, pp. 572–596, Jan. 2018.

[22] S. Yoo, M. Harman, and D. Clark, "Fault localization prioritization: Comparing information-theoretic and coverage-based approaches," *ACM Trans. Softw. Eng. Methodol.*, vol. 22, no. 3, pp. 1–29, Jul. 2013.

[23] R. A. DeMillo, R. J. Lipton, and F. G. Sayward, "Hints on test data selection: Help for the practicing programmer," *Computer*, vol. 11, no. 4, pp. 34–41, Apr. 1978.

[24] A. J. Offutt and R. H. Untch, "Mutation 2000: Uniting the orthogonal," in *Mutation Testing for the New Century*. Cham, Switzerland: Springer, 2001, pp. 34–44.

[25] V. Debroy and W. E. Wong, "Combining mutation and fault localization for automated program debugging," *J. Syst. Softw.*, vol. 90, pp. 45–60, Apr. 2014.

[26] J. M. Voas, "PIE: A dynamic failure-based technique," *IEEE Trans. Softw. Eng.*, vol. 18, no. 8, pp. 717–727, Aug. 1992.

[27] M. Y. Chen, E. Kiciman, E. Fratkin, A. Fox, and E. Brewer, "Pinpoint: Problem determination in large, dynamic Internet services," in *Proc. Int. Conf. Dependable Syst. Netw.*, 2002, pp. 595–604.

[28] R. Abreu, P. Zoeteweij, and A. Van Gemund, "An evaluation of similarity coefficients for software fault localization," in *Proc. 12th Pacific Rim Int. Symp. Dependable Comput. (PRDC)*, 2006, pp. 39–46.

[29] X. Xie, T. Y. Chen, F.-C. Kuo, and B. Xu, "A model for spectra-based software diagnosis," *ACM Trans. Softw. Eng. Methodol.*, vol. 20, no. 3, pp. 11:1–11:32, Aug. 2011.

[30] J. A. Jones, M. J. Harrold, and J. Stasko, "Visualization of test information to assist fault localization," in *Proc. 24th Int. Conf. Softw. Eng. (ICSE)*, 2002, pp. 467–477.

[31] W. E. Wong, V. Debroy, R. Gao, and Y. Li, "The DStar method for effective software fault localization," *IEEE Trans. Rel.*, vol. 63, no. 1, pp. 290–308, Mar. 2014.

[32] Y. Liu, Z. Li, L. Wang, Z. Hu, and R. Zhao, "Statement-oriented mutant reduction strategy for mutation based fault localization," in *Proc. IEEE Int. Conf. Softw. Qual., Rel. Secur. (QRS)*, Jul. 2017, pp. 126–137.

[33] C. E. Shannon, "A mathematical theory of communication," *Bell Syst. Tech. J.*, vol. 27, no. 3, pp. 379–423, 1948.

[34] Y. Gao, Z. Zhang, L. Zhang, C. Gong, and Z. Zheng, "A theoretical study: The impact of cloning failed test cases on the effectiveness of fault localization," in *Proc. 13th Int. Conf. Qual. Softw.*, Jul. 2013, pp. 288–291.

[35] L. Zhang, L. Yan, Z. Zhang, J. Zhang, W. K. Chan, and Z. Zheng, "A theoretical analysis on cloning the failed test cases to improve spectrum-based fault localization," *J. Syst. Softw.*, vol. 129, pp. 35–57, Jul. 2017.

[36] C. Gong, Z. Zheng, W. Li, and P. Hao, "Effects of class imbalance in test suites: An empirical study of spectrum-based fault localization," in *Proc. IEEE 36th Annu. Comput. Softw. Appl. Conf. Workshops*, Jul. 2012, pp. 470–475.

[37] H. Do, S. Elbaum, and G. Rothermel, "Supporting controlled experimentation with testing techniques: An infrastructure and its potential impact," *Empirical Softw. Eng.*, vol. 10, no. 4, pp. 405–435, Oct. 2005.

[38] Z. Li, Y. Wu, and Y. Liu, "An empirical study of bug isolation on the effectiveness of multiple fault localization," in *Proc. IEEE 19th Int. Conf. Softw. Qual., Rel. Secur. (QRS)*, Jul. 2019, pp. 18–25.

[39] Q. Yang, J. J. Li, and D. M. Weiss, "A survey of coverage-based testing tools," *Comput. J.*, vol. 52, no. 5, pp. 589–597, Aug. 2009.

[40] M. E. Delamaro, J. C. Maldonado, and A. M. R. Vincenzi, "Proteum/im 2.0: An integrated mutation testing environment," in *Mutation Testing for the New Century*. Cham, Switzerland: Springer, 2001, pp. 91–101.

[41] H. Agrawal, R. DeMillo, R. Hathaway, W. Hsu, W. Hsu, E. W. Krauser, R. J. Martin, A. P. Mathur, and E. Spafford, "Design of mutant operators for the C programming language," Softw. Eng. Res. Center, Purdue Univ., West Lafayette, IN, USA, Tech. Rep. SERC-TR-41-P, 1989.

[42] L. Zhang, D. Marinov, and S. Khurshid, "Faster mutation testing inspired by test prioritization and reduction," in *Proc. Int. Symp. Softw. Test. Anal. (ISSTA)*, 2013, pp. 235–245.

[43] R. Just, M. D. Ernst, and G. Fraser, "Efficient mutation analysis by propagating and partitioning infected execution states," in *Proc. Int. Symp. Softw. Test. Anal. (ISSTA)*, 2014, pp. 315–326.

[44] M. Renieres and S. P. Reiss, "Fault localization with nearest neighbor queries," in *Proc. 18th IEEE Int. Conf. Automated Softw. Eng.*, Oct. 2003, pp. 30–39.

[45] D. Zou, J. Liang, Y. Xiong, M. D. Ernst, and L. Zhang, "An empirical study of fault localization families and their combinations," *IEEE Trans. Softw. Eng.*, early access, Jan. 10, 2019, doi: 10.1109/TSE.2019.2892102.

[46] Y. Liu, M. Li, Y. Wu, and Z. Li, "A weighted fuzzy classification approach to identify and manipulate coincidental correct test cases for fault localization," *J. Syst. Softw.*, vol. 151, pp. 20–37, May 2019.

[47] F. Wilcoxon, "Individual comparisons by ranking methods," in *Breakthroughs in Statistics*. Biometrics Bulletin: JSTOR, 1945, pp. 80–83.

[48] R. Just, "The major mutation framework: Efficient and scalable mutation analysis for Java," in *Proc. Int. Symp. Softw. Test. Anal. (ISSTA)*, 2014, pp. 433–436.

[49] X. Chen, Y. Zhao, Q. Wang, and Z. Yuan, "MULTI: Multi-objective effort-aware just-in-time software defect prediction," *Inf. Softw. Technol.*, vol. 93, pp. 1–13, Jan. 2018.

[50] X. Chen, D. Zhang, Y. Zhao, Z. Cui, and C. Ni, "Software defect number prediction: Unsupervised vs supervised methods," *Inf. Softw. Technol.*, vol. 106, pp. 161–181, Feb. 2019.

[51] X. Chen, Y. Zhao, Z. Cui, G. Meng, Y. Liu, and Z. Wang, "Large-scale empirical studies on effort-aware security vulnerability prediction methods," *IEEE Trans. Rel.*, vol. 69, no. 1, pp. 70–87, Mar. 2020.

[52] X. Chen, Y. Mu, Y. Qu, C. Ni, M. Liu, T. He, and S. Liu, "Do different cross-project defect prediction methods identify the same defective modules?" *J. Softw., Evol. Process*, vol. 32, no. 5, pp. 70–87, May 2020.

[53] X. Zhang, H. He, N. Gupta, and R. Gupta, "Experimental evaluation of using dynamic slices for fault location," in *Proc. 6th Int. Symp. Automated Anal.-Driven Debugging (AADEBUG)*, 2005, pp. 33–42.

[54] W. E. Wong, Y. Shi, Y. Qi, and R. Golden, "Using an RBF neural network to locate program bugs," in *Proc. 19th Int. Symp. Softw. Rel. Eng. (ISSRE)*, Nov. 2008, pp. 27–36.

[55] V. Dallmeier, C. Lindig, and A. Zeller, "Lightweight bug localization with AMPLE," in *Proc. 6th Int. Symp. Automated Anal.-Driven Debugging (AADEBUG)*, 2005, pp. 99–104.

[56] W. Masri, "Fault localization based on information flow coverage," *Softw. Test., Verification Rel.*, vol. 20, no. 2, pp. 121–147, May 2009.

[57] T. Shu, T. Ye, Z. Ding, and J. Xia, "Fault localization based on statement frequency," *Inf. Sci.*, vol. 360, pp. 43–56, Sep. 2016.

[58] G. Dandan, W. Tiantian, S. Xiaohong, and M. Peijun, "A test-suite reduction approach to improving fault-localization effectiveness," *Comput. Lang., Syst. Struct.*, vol. 39, no. 3, pp. 95–108, Oct. 2013.

[59] L. Vidacs, A. Beszedes, D. Tengeri, I. Siket, and T. Gyimothy, "Test suite reduction for fault detection and localization: A combined approach," in *Proc. Softw. Evol. Week IEEE Conf. Softw. Maintenance, Reengineering, Reverse Eng. (CSMR-WCRE)*, Feb. 2014, pp. 204–213.

[60] A. Siami Namin, J. H. Andrews, and D. J. Murdoch, "Sufficient mutation operators for measuring test effectiveness," in *Proc. 13th Int. Conf. Softw. Eng. (ICSE)*, 2008, pp. 351–360.

[61] M. Polo, M. Piattini, and I. García-Rodríguez, "Decreasing the cost of mutation testing with second-order mutants," *Softw. Test., Verification Rel.*, vol. 19, no. 2, pp. 111–131, Jun. 2009.

[62] M. Kintis, M. Papadakis, and N. Malevris, "Evaluating mutation testing alternatives: A collateral experiment," in *Proc. Asia Pacific Softw. Eng. Conf.*, Nov. 2010, pp. 300–309.

[63] M. Papadakis and N. Malevris, "An empirical evaluation of the first and second order mutation testing strategies," in *Proc. 3rd Int. Conf. Softw. Test., Verification, Validation Workshops*, Apr. 2010, pp. 90–99.

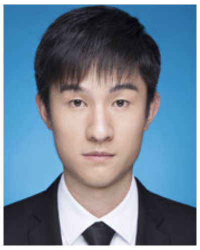
[64] M. Papadakis, N. Malevris, and M. Kintis, "Mutation testing strategies-a collateral approach," in *Proc. ICSOFT*, 2010, pp. 325–328.

[65] A. Parsai, A. Murgia, and S. Demeyer, "A model to estimate first-order mutation coverage from higher-order mutation coverage," in *Proc. IEEE Int. Conf. Softw. Qual., Rel. Secur. (QRS)*, Aug. 2016, pp. 365–373.

[66] R. Just, G. M. Kapfhammer, and F. Schweiggert, "Using non-redundant mutation operators and test suite prioritization to achieve efficient and scalable mutation analysis," in *Proc. IEEE 23rd Int. Symp. Softw. Rel. Eng.*, Nov. 2012, pp. 11–20.
[67] K. Kapoor and J. P. Bowen, "Ordering mutants to minimise test effort in mutation testing," in *Proc. Int. Workshop Formal Approaches Softw. Test.* Cham, Switzerland: Springer, 2004, pp. 195–209.
[68] R. Just, D. Jalali, and M. D. Ernst, "Defects4J: A database of existing faults to enable controlled testing studies for Java programs," in *Proc. Int. Symp. Softw. Test. Anal. (ISSTA)*, 2014, pp. 437–440.

HAIFENG WANG received the B.S. degree in mathematics from the Beijing University of Chemical Technology, China, in 2017, where he is currently pursuing the Ph.D. degree in control theory and control engineering. His research interests are software testing, fault localization, and software defect prediction.

BIN DU received the B.S. degree in automatic instrument from the Hebei Institute of Technology, and the M.S. and Ph.D. degrees in control science and engineering from the Beijing University of Chemical Technology, in 2000 and 2011, respectively. She is currently an Associate Professor with the College of Information Science and Technology, Beijing University of Chemical Technology. Her research interests include software analysis, fault localization, and program repair.



JIE HE received the B.S. degree from the Beijing University of Chemical Technology, Beijing, China, in 2017, where he is currently pursuing the master's degree in computer science and technology. His research interests are fault localization and software testing.

YONG LIU (Member, IEEE) received the B.Sc. and M.Sc. degrees in computer science and technology, and the Ph.D. degree in control science and engineering from the Beijing University of Chemical Technology, China, in 2008, 2011, and 2015, respectively. He is currently an Assistant Professor with the College of Information Science and Technology, Beijing University of Chemical Technology. In his research areas, he has published about ten papers in referred journals or conferences such as the *Journal of Systems and Software*, *Information Sciences*, QRS, SATE, and COMPSAC. His research interest is mainly in software engineering. Particularly, he is interested in software debugging and software testing such as source code analysis, mutation testing, and fault localization. He is a member of the CCF, China, and ACM.

XIANG CHEN (Member, IEEE) received the B.Sc. degree from the School of Management, Xi'an Jiaotong University, China, in 2002, and the M.Sc. and Ph.D. degrees in computer software and theory from Nanjing University, China, in 2008 and 2011, respectively. He is an Associate Professor with the Department of Information Science and Technology, Nantong University. His research interests are mainly in software engineering. Particularly, he is interested in software maintenance and software testing such as software defect prediction, combinatorial testing, regression testing, and fault localization. In his research areas, he has published over 60 papers in refereed journals or conferences such as the IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, IEEE ACCESS, *Information and Software Technology*, the *Journal of Systems and Software*, the IEEE TRANSACTIONS ON RELIABILITY, the *Journal of Software: Evolution and Process*, *Software Quality Journal*, the *Journal of Computer Science and Technology*, ASE, ICSME, SANER, and COMPSAC. He is a Senior Member of the CCF, China, and a member of the ACM. He is currently serving as an Associate Editor for IEEE ACCESS.