Can Higher-Order Mutants Improve the Performance of Mutation-Based Fault Localization?

Haifeng Wang, Zheng Li, Yong Liu, Xiang Chen, *Member, IEEE*, Doyle Paul, Yuxiaoyang Cai, and Luxi Fan

***Abstract*—First-order mutants (FOMs) have been widely used in mutation-based fault localization (MBFL) approaches and have achieved promising results in single-fault localization scenarios (SFL-scenario). Higher-order mutants (HOMs) are proposed to simulate complex faults and can be applied in MBFL theoretically for multiple-fault localization scenarios (MFL-scenario). However, whether HOMs can improve MBFL's performance is not investigated and the effectiveness is not thoroughly evaluated. In this empirical study, we investigate the impact of HOMs on the performance of MBFL in SFL-scenario and MFL-scenario. The experiments on two real-world benchmarks reveal that 1) 2-HOMs can help improve the MBFL performance in SFL-scenarios; 2) in MFL-scenarios, both 2-HOMs and 3-HOMs can achieve better performance than FOMs; and 3) huge computational cost cannot be ignored in the practice of HOMs. Therefore, effective methods to reduce the number of HOMs for future MBFL studies should be considered.**

***Index Terms*—Empirical study, first-order mutants (FOMs), higher-order mutants (HOMs), mutation testing, mutation-based fault localization (MBFL).**

I. Introduction

FAULT localization is the process of identifying faulty program elements (such as statements, functions) that result in failures during program execution [1]. Within the process of software debugging, fault localization is one of the most expensive activities, especially for large-scale and complex software projects in the last decades [2]. To alleviate the cost of human effort in localizing the root cause of faults, automated fault localization methods [3], [4] have been proposed, such as information retrieval (IR)-based techniques [5], [6], machine learning-based techniques [7]–[9], spectrum-based techniques [10], [11], and mutation-based techniques [12], [13]. IR-based techniques treat the bug report as a query and consider source code elements as a document collection [14]. This kind of technique ranks elements according to their textual similarity with the report. Although IR-based techniques are as effective as spectrum-based techniques in previous study [5], there are several differences between these two techniques [15]. One difference is that IR-based techniques are working on bug reports, while spectrum-based techniques are working on programs. Another difference is that most IR-based techniques locate faults at the file level and spectrum-based techniques locate faults at the statement level.

Spectrum-based fault localization (SBFL) [3], [10], [16] is one of the most commonly studied techniques. SBFL considers the binary coverage of the program elements but with inherently limited fault localization accuracy. Recent studies have shown that mutation-based fault localization (MBFL) techniques can help improve the performance of fault localization [17] and achieve a higher fault localization accuracy than the state-of-the-art SBFL techniques [18], [19].

In MBFL, for a program p, a set of faulty programs p' (i.e., mutants) are generated by applying small-scale changes to the original program p. The rules of small-scale changes that generate mutants from the original program are known as mutation operators [20]. The mutants are usually classified into two types, first-order mutants (FOMs) and higher-order mutants (HOMs). In particular, FOMs are generated by applying mutation operators only once, while HOMs are generated by applying mutation operators more than once [21]. It has been presented that HOMs are more like real faults than FOMs [22]. Both FOMs and HOMs were adopted to detect faults within programs [23], [24], but only the FOMs were studied for localizing single program fault [12], [13].

HOMs are more like real multiple faults, but they also have to face the challenges of the huge number of mutants, which tends to grow at an exponential rate of the number of orders [21]. Meanwhile, there exists the issue of equivalent and redundant mutants in HOMs [25] and a lack of clear theory where mutants are valuable [26]. The studies of HOMs mainly focus on measuring the test suite's quality [27], [28] and reducing the cost of mutation testing [29], [30].

Theoretically, HOMs can be applied in MBFL for localizing multiple faults, since the empirical studies [31], [32] indicate that the failures are often triggered by multiple faults. However, Digiuseppe and Jones [33] found that multiple faults have a negligible effect on the effectiveness of the fault localization. Moreover, Xue and Namin [34] presented that localizing

Manuscript received February 7, 2022; accepted March 19, 2022. This work was supported by the National Natural Science Foundation of China under Grant 61902015 and Grant 61872026. Associate Editor: Zheng Zheng. *(Corresponding author: Zheng Li.)*

Haifeng Wang, Zheng Li, Yong Liu, Yuxiaoyang Cai, and Luxi Fan are with the College of Information Science and Technology, Beijing University of Chemical Technology, Beijing 100013, China (e-mail: h.f.wang@hotmail.com; lizheng@mail.buct.edu.cn; lyong@mail.buct.edu.cn; caiyuxiaoyang@163.com; 307945541@qq.com).

Xiang Chen is with the School of Information Science and Technology, Nantong University, Nantong 226007, China (e-mail: xchencs@ntu.edu.cn).

Doyle Paul is with the School of Computer Science, Technological University Dublin, D07 EWV4 Dublin, Ireland (e-mail: paul.doyle@tudublin.ie).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TR.2022.3162039.

Digital Object Identifier 10.1109/TR.2022.3162039

multiple faults have been illustrated to be more difficult, time-consuming, and costly due to the issue of fault interference (i.e., an interaction that multiple faults manifest themselves to cause, or mask, test case failure [35]–[37]), which leads to performance degradation of the existing fault localization techniques. Furthermore, existing studies have not analyzed the relationship between HOMs and multiple faults, while the impact of using HOMs on MBFL is still unknown.

In this article, we aim to investigate the performance of using FOMs and HOMs on single-fault localization and the performance of using FOMs and HOMs on multiple-fault localization. A theoretical analysis of MBFL with FOMs and HOMs is first conducted for both SFL-scenario and MFL-scenario. Then a large-scale empirical study is conducted to investigate whether using HOMs can improve the performance of MBFL.

In the experimental setup, the programs from SIR [38] and Codeflaws [39] are selected as our empirical subjects. Then the mutation operators introduced by Agrawal *et al.* [40] are used to generate FOMs and construct HOMs. Specifically, in the SFL-scenario, we conduct the experiments on 66 versions of six real-world programs from SIR and 1284 real single-fault programs from Codeflaws. In the MFL-scenario, for SIR, we obtain 120 versions of multiple-fault programs by seeding faults to single-fault programs and 124 multiple-fault programs for Codeflaws.

The experimental results show that, in the SFL-scenario, 2-HOMs can help fault localization. In the MFL-scenario, combining 2-HOMs and 3-HOMs can localize more faults in terms of $Top\text{-}N$ and MAP metrics on both benchmarks. We also investigate the performance of different MBFL formulas, and Ochiai [41] can localize faults more precisely in terms of all metrics. Moreover, the high computational cost of using HOMs still exists to such an extent that researchers should consider methods to reduce the cost of using HOMs on fault localization.

The contributions of this study are summarized as follows:

1) The influence of higher-order mutant on MBFL is first investigated and the results show that the higher-order mutant can help for improving the effectiveness of fault localization.
2) We conduct a theoretical analysis of MBFL with FOMs and HOMs when applied in an SFL-scenario and an MFL-scenario. Besides, we have provided a specific example to illustrate how HOMs work in the MBFL process.
3) We demonstrate the impact of HOMs on the performance of MBFL in the SFL-scenario and the MFL-scenario through a large-scale empirical study on two real-world benchmarks, SIR and Codeflaws.
4) To facilitate other researchers in replicating this study, we share both the subjects and scripts[1]. The detailed results of our empirical study are publicly available[2].

The rest of this aticle is organized as follows. Section II presents the background and related work. Section III introduces the theoretical analysis for MBFL and the motivation. Section IV describes the design of the empirical study, while Section V presents the results with respect to research questions.

[1][Online]. Available: https://github.com/paper-results/code
[2][Online]. Available: https://github.com/paper-results/data

TABLE I
THREE POPULAR SUSPICIOUSNESS FORMULAS FOR MBFL

Name	Formula
$Dstar^*$ [44]	$Sus(m) = \frac{a_{kf}^*}{a_{kp}+a_{nf}}$
$Jaccard$ [45]	$Sus(m) = \frac{a_{kf}}{a_{kf}+a_{nf}+a_{kp}}$
$Ochiai$ [41]	$Sus(m) = \frac{a_{kf}}{\sqrt{(a_{kf}+a_{nf})*(a_{kf}+a_{kp})}}$

Section VII discusses threats to the validity of the empirical study. Finally, Section VIII concludes this article.

II. BACKGROUND AND RELATED WORK

A. Mutation-Based Fault Localization

MBFL is a well-studied technique that is based on mutation analysis [19]. Mutation analysis works by making simple syntactic changes to the program under test and then generating many different versions of it. Each program contains artificial faults and these versions are called mutants [42]. The transformation rules that define how to introduce syntactic changes to the program are called mutant operators [20]. In this study, we adopt mutation operators for the C programming language introduced by Agrawal *et al.* [40] (see Table IV).

In mutation analysis, mutants are used to evaluate the quality of test cases based on their ability to distinguish the mutants' behavior from that of the original program [43]. If a test case can distinguish the behavior of a mutant from the original, we say that the mutant has been $killed$ or $detected$. Otherwise, the mutant is $notkilled$ or $live$ [26].

The classical MBFL approach typically includes four steps.

Step 1: Obtain statements covered by failed test cases. Initially, the MBFL approach executes a program P by a test suite T. Next, the coverage information and test results (i.e., pass or fail) are collected. Then, T is divided into two groups: T_p and T_f, where T_p is the set of passed test cases and T_f is the set of failed test cases. Here, all statements covered by T_f are denoted as $Covf$.

Step 2: Generate and execute mutants. The MBFL approach employs mutation operators to artificially inject faults into statements from $Covf$ to generate mutants. Note that a statement s may have a set of mutants, denoted by $M(s)$. After executing a mutant m by T, T is split into two sets: $T_n(m)$ and $T_k(m)$, where $T_n(m)$ is the set of test cases that cannot kill the mutant m and $T_k(m)$ is the set of test cases that can kill the mutant m.

Step 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: $a_{np}(m) = |T_n(m) \cap T_p|$, $a_{kp}(m) = |T_k(m) \cap T_p|$, $a_{nf}(m) = |T_n(m) \cap T_f|$, and $a_{kf}(m) = |T_k(m) \cap T_f|$, 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 $a_{kf}(m)$ is the number of failed test cases which can kill m.

Table I lists three popular MBFL formulas which were used in this article, i.e., $Dstar^*$ [44] ($*$ is set to 3 in this article), $Jaccard$ [45], $Ochiai$ [41].

Next, the suspiciousness of the statement s is assigned the maximum suspiciousness value of the mutants generated by s: $sus(s) = max\{sus(m_1), sus(m_2), \ldots, sus(m_q)\}$, where $m_1, \ldots, m_q$ are mutants in $M(s)$ and the $sus(s)$ is the suspiciousness of the statement s.

Step 4: Generate fault localization report. The MBFL approach sorts all statements in descending order based on their suspiciousness value and returns this ranking list. The developers will use this ranking list to localize the faults in the program and then fix them.

MBFL works based on the assumption that mutants killed mostly by failed test cases have a connection with the program faults. Recent studies [12], [19] also demonstrated that MBFL could significantly outperform other types of fault localization techniques (such as spectrum-based fault localization techniques [18], [19]).

B. Higher-Order Mutation Testing

Higher-order mutation testing was first introduced by Jia and Harman [21]. According to their study, mutants can be classified into two types: FOMs and HOMs. Both of these types of mutants were first proposed by Offutt, and were named as simple mutants and complex mutants in their study.

Definition 1 (FOM): An FOM of a program p is constructed by making a single syntactic change to p. The rule of producing FOM is called first-order mutation operators FOP.

Definition 2 (HOM): An HOM of a program p is created from p by applying k operators from FOP. This HOM is said to be a kth order mutant of p, recorded as k-HOM.

Note that in this study, the k operators are applied in k different statements, which follows the same idea of "Programs with multiple faults on multiple lines" mentioned in the study of Debroy and Wong [37].

In higher-order mutation testing, HOMs were mainly used to evaluate the quality of test suites [47], and to reduce testing costs [30], [48], [49]. Some researchers use HOMs in test data generation [50] and coupling effect analysis [42], [51]. Also, HOMs can be adopted to alleviate the equivalent mutant problem [52] and estimate mutation coverage of FOMs for reducing the cost on first-order mutation testing [53].

Different from previous HOMs studies, in this study, we adopt HOMs to the field of fault localization, which utilizes HOMs to localize faults in the programs. We conduct an empirical study in this article to determine whether HOMs can help improve the fault localization effectiveness.

III. Research Motivation

In traditional MBFL techniques, most of the studies are based on the single fault hypothesis [12], [13], [17], [54], [55]. However, the empirical studies [31], [32], [56] implied that individual failures are often triggered by multiple faults spread throughout the system. In this study, we try to locate multiple faults using HOMs.

In this section, in order to better illustrate that why HOMs can help in multiple-fault localization, we first provide the basic notations and definitions of the theoretical analysis for MBFL. Then, we present two hypotheses based on these definitions. Finally, we use an example program to demonstrate our hypothesis.

A. Theoretical Analysis for MBFL

To formally represent the notion of the difference between FOMs and HOMs applied in fault localization, we followed the framework proposed by Shin and Bae [2], [57].

In this framework [2], for a correct program p_s, a single-fault program is $p_i \in P$ (i is the faulty line number), a two-fault program is $p_{x,y} \in P$ (x, y are the faulty line number), and mutant m is generated from p_i or $p_{x,y}$. If a test suite T detects a fault in program p_i, it can be represented by

$$\mathbf{d}\left(T, p_i, p_s\right) \neq \mathbf{0} \tag{1}$$

where $\mathbf{0}$ is the zero vector and $\mathbf{d}$ is a vector that is a composite of n elements, i.e.,

$$\mathbf{d}\left(T, p_i, p_s\right) = \left\langle d\left(t_1, p_i, p_s\right), \ldots, d\left(t_n, p_i, p_s\right)\right\rangle.$$

Equation (1) implies that if there exists at least one test $t \in T$ which detects different behavior in p_i and p_s, it says that test suite T has detected the fault in program p_i. In other words, we can say t fails on p_i.

Similarly, a test suite T which kills a mutant m generated from p_i can be represented by

$$\mathbf{d}\left(T, p_i, m\right) \neq \mathbf{0}. \tag{2}$$

This implies that there is at least one test where t's behaviors on p_i are different from the behaviors on m.

When mutants are adopted in the traditional MBFL process, it calculates suspiciousness of mutants using the detect information represented in (1) and killing information represented in (2). Moreover, MUSE [17] has considered the test behaviors of the mutant m and the correct program p_s, which can be represented by

$$d\left(t, m, p_s\right) = 0. \tag{3}$$

Based on (3), the test t has the same behaviors on m and p_s. In other words, mutant m has fixed the fault program p_i on test t, changing p_i to p_s.

Furthermore, if the mutant m has fixed p_i against the test suite T, it can be represented by

$$\mathbf{d}\left(T, m, p_s\right) = \mathbf{0}. \tag{4}$$

This idea is matched to the study of Shin and Bae [2]. They have presented two foundations of MBFL: 1) considering a mutant as a potential fix, noted as MBFL-FIX, and 2) considering a mutant as a fault, noted as MBFL-FLT. The first assumption has been proven to be useful in increasing the suspiciousness of mutants in previous works [2] and the second one is insufficient for calculating the suspiciousness of mutants when the conduct of faulty program is not clearly defined. Hence, we focus on extending the first assumption on single fault and multiple fault localization.

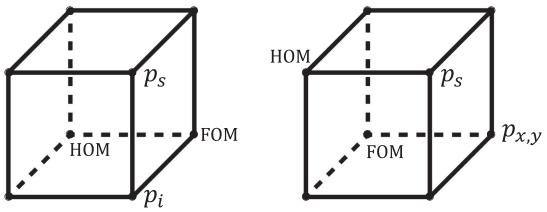


Fig. 1. Foundations of MBFL-fix. (a) SFL-scenario. (b) MFL-scenario.

Definition 3 (MBFL-Fix score): An MBFL-fix score $Fix-score$ is a score, such that

$$Fix\text{-}score\,(T, p_s, m)) = \frac{||\mathbf{d}\,(T, p_s, m)\,||^2}{n} = \frac{||\,\langle d\,(t_1, p_s, m)\,, \ldots, d\,(t_n, p_s, m)\rangle\,||^2}{n}$$

for all $T = \{t_1, \ldots, t_n\} \subseteq T^n, p_s \in P, m \in M$, and $||\cdot||$ is the Euclidean norm of the vector.

In other words, a mutant m returns a $Fix\text{-}score$ on p_s from n tests. A lower $Fix\text{-}score$ indicates m is more similar to the correct program p_s.

Next, we propose two hypotheses with FOM and HOM applied in single-fault and multiple-fault localization.

Hypothesis 1: Within the SFL-scenario, FOMs are more similar to the correct program, introduce to better fault localization effectiveness on single-fault programs.

This assumption considers that FOMs have fewer differences from the correct program p_s than HOMs, on single fault program p_i, such that

$$Fix\text{-}score\,(T, p_s, HOM)) \geq Fix\text{-}score\,(T, p_s, FOM)) \geq Fix\text{-}score\,(T, p_s, p_s)) = 0.$$

Hypothesis 2: Within the MFL-scenario, HOMs are more similar to the correct program, result in better fault localization effectiveness on multiple-fault programs.

This assumption considers that HOMs have fewer differences from the correct program p_s than FOMs on multiple fault program $p_{x,y}$, such that

$$Fix\text{-}score\,(T, p_s, FOM)) \geq Fix\text{-}score\,(T, p_s, HOM)) \geq Fix\text{-}score\,(T, p_s, p_s)) = 0.$$

The foundations of the two hypotheses are depicted in Fig. 1. Each of the cube represents a program space that contains four major points corresponding to the correct program p_s, faulty programs p_i (single-fault), $p_{x,y}$ (multiple-fault), FOM, and HOM. The distance between two points represents the similarity of the two programs. Fig. 1(a) shows the foundation of Hypothesis 1 and Fig. 1(b) shows the foundation of Hypothesis 2.

The study of Shin and Bae [2] implied that MBFL-fix is fundamentally the same as program repair, in which the objective is to move towards p_s from p_i or $p_{x,y}$ using FOM or HOM. In the SFL-scenario, FOM is closer to p_s with only one change from p_i and HOM with several changes is further away from p_s. In the MFL-scenario, since $p_{x,y}$ contains two faults that have a distance from p_s, FOM is further away from p_s and HOM has a higher probability to fix $p_{x,y}$, which brings it closer to p_s. These two hypotheses match our intuition that simple faults can be fixed by simple mutants (FOMs) and complex faults can be fixed by complex mutants (HOMs). In the study of Debory and Wong [37], they found that the reason their proposed strategy cannot fix multiple faults in the same program is because they only considered FOMs. In other words, there is potential for locating or fixing multiple faults in a program by making use of HOMs. We next present an example to describe how FOMs and HOMs work on MBFL.

B. Motivating Example

In traditional mutation-based fault localization, there is a one-to-many relationship between each statement of program under test and the corresponding generated FOMs. The suspiciousness of one statement s is assigned with the maximum suspiciousness value of the FOMs generated by s. However, when HOMs are applied in fault localization, there will be a many-to-many relationship between statements and HOMs. In this article, we consider each k-HOM as composed of k FOMs, and each FOM is related to one statement s. Then we can determine the suspiciousness of the statement s by calculating the mean suspiciousness of related HOMs

$$Sus_{HOMs}(s) = \frac{\sum_{i=1}^{n} Sus(HOM_i)}{n} \tag{5}$$

where n is the number of related HOMs of statement s.

The use of the mean value as the statement suspiciousness is based on the hypothesis: The HOMs related to one statement have similar suspiciousness, and the error in the suspiciousness of one statement can be reduced by calculating the average value. We did not use the traditional method of calculating the maximum value because the maximum value calculation method is prone to the following situation: if an HOM has a high suspiciousness, then the statements associated with it also have a high suspiciousness, and then these statements are ranked high in the list. However, the probability that these statements are faulty is very low. Suppose a program has 50 lines which contain two faults, and each line of statements has one FOM, then the probability that two statements related to one 2-HOM are both faulty is: $\frac{1}{C(50,2)} = \frac{1}{1225}$.

Table II illustrates the detailed process of adopting FOMs and HOMs to MBFL. In Table II, a program P called $mid()$ takes three integers as input parameters and returns the middle value of them. The test suite T is composed of six test cases, and the program P contains 14 program statements where statement s_4 and statement s_{11} are both faulty statements. A statement execution labeled with "1" means that the mutant (FOM or HOM) can be killed by the test case, and is empty otherwise. Test cases (t_2, t_3, t_5) are failed test cases, while test cases (t_1, t_4, t_6) are passed test cases. In this example, the suspiciousness of each mutant is computed by using the Ochiai formula in Table II with two given results, one result is for FOMs and the other result is for HOMs.

TABLE II
MOTIVATING EXAMPLE OF OUR STUDY

	Program	FOMs	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	FOM Sus	Stmt. Sus	Rank
s_1	int mid(int x,int y,int z){	FOM_1: x → x - 1	0	0	1	0	0	1	0.41	1.00 (s_1)	2
		FOM_2: y → -y	0	1	1	0	1	0	1.00		
s_2	int m;										
s_3	m = z;	FOM_3: z → z + 1	0	1	0	1	0	0	0.41	0.41 (s_3)	5
		FOM_4: z → -z	0	1	0	1	0	0	0.41		
s_4	**if(y <z - 1) //fault1. Correct: if(y <z)**	FOM_5: <→<=	0	1	0	0	1	0	0.82	0.82 (s_4)	3
		FOM_6: <→>	1	0	1	0	0	1	0.33		
s_5	if(x <y)										
s_6	m = y;										
s_7	else if(x <z)										
s_8	m = x;										
s_9	else										
s_{10}	if (x >y)	FOM_7: >→>=	0	0	0	1	0	0	0.41	1.00 (s_{10})	2
		FOM_8: >→ ==	0	0	1	1	1	0	1.00		
s_{11}	**m = -y; //fault2. Correct: m = y**	FOM_9: = → +=	0	0	1	0	1	0	0.82	0.82 (s_{11})	3
		FOM_{10}: -y → y	0	0	1	0	1	0	0.82		
s_{12}	else if(x >z)	FOM_{11}: >→>=	0	0	0	0	0	0	+∞	+∞ (s_{12})	1
		FOM_{12}: >→<	0	1	0	0	0	0	0.58		
s_{13}	m = x;										
s_{14}	return m;}	FOM_{13}: m → -m	1	1	1	1	1	1	0.71	0.71 (s_{14})	4
		FOM_{14}: m → 0	1	1	1	1	1	1	0.71		
		Results	**P**	**F**	**F**	**P**	**F**	**P**			

	HOMs	Test Suite t_1	t_2	t_3	t_4	t_5	t_6	HOM Sus	Stmt. Sus	Rank
	HOM_1: FOM_1 + FOM_{11}	0	0	1	0	0	1	0.41	0.80 (s_1)	2
	HOM_2: FOM_2 + FOM_8	0	1	1	0	1	0	1.00		
	HOM_3: FOM_2 + FOM_{10}	0	1	1	0	1	0	1.00	0.72 (s_3)	5
	HOM_4: FOM_3 + FOM_{11}	0	1	0	0	0	0	0.58	0.75 (s_4)	3
	HOM_5: FOM_4 + FOM_9	0	1	1	1	1	0	0.87		
	HOM_6: FOM_4 + FOM_{14}	1	1	1	1	1	1	0.71		
	HOM_7: FOM_5 + FOM_7	0	1	0	1	1	0	0.67		
	HOM_8: FOM_5 + FOM_9	0	1	1	0	1	0	1.00		
	HOM_9: FOM_5 + FOM_{10}	0	1	1	0	1	0	1.00		
	HOM_{10}: FOM_6 + FOM_9	1	0	1	0	1	1	0.58	0.69 (s_{10})	6
	HOM_{11}: FOM_6 + FOM_{10}	1	0	1	0	1	1	0.58	0.84 (s_{11})	1
	HOM_{12}: FOM_6 + FOM_{13}	1	1	1	1	1	1	0.71	0.46 (s_{12})	7
	HOM_{13}: FOM_7 + FOM_{12}	0	1	0	1	0	0	0.41		
	HOM_{14}: FOM_7 + FOM_{13}	1	1	1	0	1	1	0.77	0.73 (s_{14})	4

The bold entities to highlight the faulty statements followed by corrected versions (s4 and s11).

For fault localization using FOMs, we assume that MBFL generates only two FOMs per statement covered by the failed test cases, resulting in a total of 14 FOMs (listed under column "FOMs"). For MBFL, it first utilizes the killing information of test cases (from t_1 to t_6) to calculate suspiciousness of FOMs (listed under column "FOM sus"). Next, the maximum suspiciousness of mutants generated from the same statement is assigned to the statement suspiciousness (listed under column "Stmt. Sus"). Finally, in the column "Rank," MBFL ranks the faulty statements s_4 and s_{11} jointly in the third place.

For fault localization using HOMs, we first construct HOMs from only two different FOMs for three categories, resulting in a total of 14 HOMs (listed under column "HOMs"). Then, the suspiciousness of HOMs is computed in the column "HOM sus". Next, to guarantee fairness, we obtain the statement suspiciousness by calculating the mean of corresponding HOMs suspiciousness. More specifically, using statement s_1 as an example, the corresponding HOMs of s_1 are three HOMs (i.e., HOM_1, HOM_2, and HOM_3) and the suspiciousness are 0.41, 1.00, and 1.00 respectively. Therefore, the suspiciousness of s_1 is: $Sus(s_1) = \frac{1.00+0.41+1.00}{3} = 0.80$. Finally, the statement suspiciousness using HOMs is listed in the column "Stmt. Sus" and it ranks the faulty statements s_{11} and s_4 in the first place and third place, respectively.

C. Motivation

As analyzed in Section III-A, we consider that HOMs are more similar to the correct program, which results in better fault localization effectiveness on multiple-fault programs. Further motivating example indicates that FOMs rank the faulty statements in the top 5, while HOMs ranks the faulty statements in the top 3, which shows that HOMs have a better fault localization accuracy in the example. This result matches the Hypothesis 2 that motivates our study in this article. In recent years, the most of the MBFL techniques are based on the single fault hypothesis [12], [13], [17], [54], [55], empirical studies [31], [32], [56] have implied that individual failures are often triggered by multiple bugs spread throughout the system. Thus, in this study, to our best knowledge, we are the first to investigate the

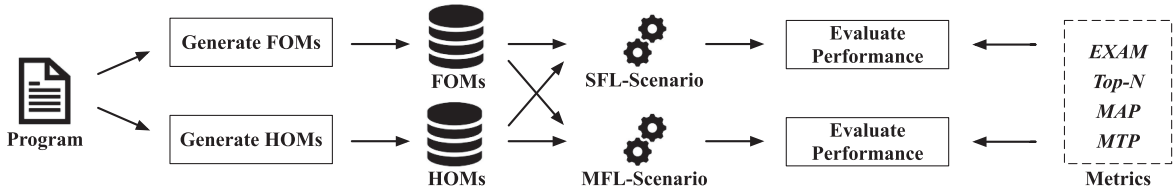


Fig. 2. Overview of experimental design.

TABLE III
STATISTICS OF SUBJECT PROGRAMS

Benchmark	Program	Versions (used)	LOC	Test Cases	FOMs	2-HOMs	3-HOMs	4-HOMs	5-HOMs
SIR	printtokens	7 (3)	563	4,130	32,956	29,912	23,719	18,704	14,935
	printtokens2	9 (6)	508	4,115	50,716	34,319	21,261	12,851	8,171
	schedule2	10 (4)	307	2,710	32,402	23,424	14,417	8,955	5,495
	totinfo	23 (18)	406	1,080	49,417	39,307	25,371	16,252	10,608
	tcas	41 (30)	173	1,608	27,592	19,129	12,246	7,627	4,762
	sed	9 (5)	7,125	360	154,025	101,400	58,565	33,891	19,662
Codeflaws		3,902 (1,408)	43	58	339,621	246,049	164,610	110,682	74,975

impact of FOMs and HOMs on the performance of MBFL in both the MFL-scenario and the SFL-scenario.

IV. EMPIRICAL STUDY

In this section, we present the experimental subjects and design proposed to address our research questions. Fig. 2 shows an overview of the experimental design and followed by the details of each part of the design.

A. Subject Programs

We conducted the experimental studies on two benchmarks of software-artifact infrastructure repository (SIR) [38] and Codeflaws [39].

1) SIR[3] is a collection of artificially injected faults of real-world open-source programs, which have been widely used by previous studies on fault localization [3], [13], [54], [58]. SIR also provides adequate test cases to measure the fitness of HOMs more precisely [21]. We include six subject programs from SIR. The first five programs are relatively small-scale programs with hundreds of lines of code from Siemens Suite, while the last program (sed) is a large-scale real-world program.
2) Codeflaws [39] is a large benchmark of real faults on C programs which are diverse and relatively hard to expose [24], [59]. Codeflaws consists of 3902 real fault programs in 7436 programs selected from the Codeforces[4] online database. Each fault in this benchmark has a unique rejected "faulty" submission and the accepted "corrected" submission. It is noted that every faulty program in the benchmark is unique, meaning that every program is different from the others. We excluded the programs where the failures cannot be detected and contain runtime errors. Overall, we considered 1408 different programs out of 3902.

TABLE IV
TYPICAL MUTATION OPERATORS FOR MBFL

Mutation Operator	Description	Example
CRCR	Required constant replacement	a=b + ∗p → a=0 + ∗p
OAAN	Arithmetic operator mutation	a + b → a * b
OAAA	Arithmetic assignment mutation	a += b → a -= b
OCNG	Logical context negation	if(a) → if(!a)
OIDO	Increment/decrement mutation	++a → a++
OLLN	Logical operator mutation	a && b → a || b
OLNG	Logical negation	a && b → !(a && b)
ORRN	Relational operator mutation	a < b → a <= b
OBBA	Bitwise assignment mutation	a &= b → a |= b
OBBN	Bitwise operator mutation	a & b → a | b
OCOR	Cast operator replacement	int a → float a
SRSR	Return statement replacement	return 0 → return 1
VTWD	Twiddle Mutations	a = b → a = b + 1
VDTR	Domain Trap	c = a → c = a ∗0
SSDL	Statement deletion	a = 1 → <no-op>

Table III presents statistics for all subject programs, including the program name, the total number of versions, used versions, average lines of code, the number of test cases, and the number of FOMs and HOMs used in the study.

In the experiments, 66 versions from SIR and 1408 versions from Codeflaws are selected, while some versions are excluded because of the following: 1) the related test suite of some versions cannot detect failures on the faulty versions; and 2) the failures of some versions lead to runtime errors, and it is hard to collect full coverage information for these faulty versions.

B. Generate Mutants

FOMs and HOMs are first generated for the empirical study. In this phase, we collect statements covered by the failed test cases and insert faults by using mutation operators to these statements. We adopt 15 types of C mutation operators (see the Table IV)

[3][Online]. Available: https://sir.csc.ncsu.edu/
[4][Online]. Available: https://codeforces.com/

proposed by Agrawal *et al.* [40], for a total of 199 mutation operators, and Proteum [60] is used to generate mutants.

The FOMs are generated using Metallaxis-FL [12] in the study since previous empirical studies [61], [62] have shown that Metallaxis-FL could outperform MUSE [17] on the effectiveness and efficiency of fault localization. The HOMs from the second order to fifth order are generated based on FOMs. Considering the huge computation cost of MBFL, we only generate the same number of HOMs on each order as FOMs. In the experiments, we first randomly choose k statements covered by the failed tests as the mutation statements. Then, FOM is generated for each statement in k, and the total k FOMs forms a k-HOM. In summary, we generate 686 729 FOMs and 1 161 299 HOMs in total for all of the programs, and the number of FOMs and each order of HOMs are presented in Table III.

It is important to note that we did not consider higher orders (e.g., 6-HOMs, 7-HOMs, etc.) because: 1) the survey of Silva *et al.* [63] observes that 71% of the studies use 2-HOMs, 33% of studies consider 3-HOMs, and 27% of studies consider 4-HOMs; 2) Wong *et al.* [64] found that the HOMs of up to the fourth order are more effective for mutation testing; 3) Nguyen *et al.* [28] found that the fifth order can be a relevant highest order in higher-order mutation testing, which indicates that HOMs from the second order to fifth order are sufficient for finding high-quality and reasonable HOMs in mutation testing.

C. Simulate Fault Localization Scenarios

To simulate the SFL-scenario and the MFL-scenario in the experiments, we use the original single fault programs from SIR and Codeflaws as our SFL-scenario. Then, the multiple-fault programs of SIR are generated by a random combination of single fault versions for each subject. The number of faults in each program in the MFL-scenario ranges from 2 to 5. Later we generate 120 versions of multiple faults programs for SIR and 124 multifault versions from Codeflaws.

It is noted that we determine the fault locations of SIR using the fault seeding location [33], [36], [65], [66]. For Codeflaws, we compare the rejected "faulty" submission and the accepted "corrected" submission to determine the locations. However, some "corrected" submissions in Codeflaws inserted new code rather than modifying the existing code. This kind of fault is regarded as the omission fault [62], [67]. To address this issue, we follow the methodology proposed by Pearson *et al.* [62]. Specifically, we identify the statements in the "faulty" submission that need to be inserted as the locations of fault. A fault localization technique communicates with the programmer using statements of source code [62] and a more useful technique can rank the statements needed to insert in a higher place of the ranking list.

Finally, we run the mutants generated from these single-fault and multiple fault programs to collect the testing results for performance evaluation.

D. Evaluation Metrics of MBFL

To evaluate the effectiveness of applying FOMs and HOMs in MBFL, we utilize three performance metrics, i.e., $EXAM$, $Top\text{-}N$, and MAP, which have been widely used in previous studies [4], [68]. Moreover, we use a mutant-test-pair metric to evaluate the efficiency of MBFL.

1) EXAM [4]: is used to measure the percentage of program elements that need to be inspected by developers until finding the faulty element. $EXAM$ is a commonly used metric for fault localization techniques, and a lower $EXAM$ value indicates a better fault localization technique [4]. The formula of $EXAM$ is defined as follows:

$$EXAM = \frac{rank}{Number\ of\ executable\ statements}. \quad (6)$$

The numerator in (6) is the ranking number of the faulty statement, and the denominator is the total number of statements that need to be checked. More specifically, $rank$ is calculated as

$$rank = \frac{(i+1)+(i+j)}{2}. \quad (7)$$

In (7), i is the number of nonfaulty 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.

2) Top-N: is used to measure how many faults can be located within the top N program elements among all candidates [69]. In the survey of Kochhar *et al.* [70], 73.58 % of developers only inspect Top-5 program elements and almost all developers agree that Top-10 elements are the upper bound for inspection within their acceptability level. Therefore, following the previous study [68], [69], we set N to 1, 3, 5 to make comparisons. A fault localization technique with higher $Top\text{-}N$ is better.

3) Mean Average Precision (MAP): evaluates ranking statements in information retrieval [71]; it is the mean of the average precision of all faults. AP (Average precision) can be calculated as follows:

$$AP = \sum_{i=1}^{M} \frac{P(i) \times pos(i)}{\text{number of faulty statements}}. \quad (8)$$

In (8), i is a rank of the statement, M is total number of statements in the ranked list, and $pos(i)$ is a Boolean function, while $pos(i) = 1$ indicates the ith statement is faulty, otherwise $pos(i) = 0$. $P(i)$ is the precision of localization at each rank i

$$P(i) = \frac{\text{number of faulty statements in top } i \text{ ranks}}{i}. \quad (9)$$

MAP is the mean of AP (Average Precision) values computed for a set of faults. We calculate MAP for faults belonging to the same project. A higher MAP value demonstrates a better technique.

4) Mutant-Test-Pair (MTP): measures the mutant execution cost of MBFL and has been used in previous studies [54], [72], [73]. An MTP counts the number of mutant executions on the test cases. The idea of MTP is that the number of mutation execution is linked to the computational cost required to obtain the rank of statements [74]. Compared with the actual run-time cost, MTP metric has the advantage of avoiding the influence of

TABLE V
Top-N AND AVERAGE *MAP* OF MBFL USING FOMS AND HOMS IN THE SFL-SCENARIO WITH THREE FORMULAS

Benchmark	Technique	Dstar				Jaccard				Ochiai			
		Top-1	Top-3	Top-5	MAP	Top-1	Top-3	Top-5	MAP	Top-1	Top-3	Top-5	MAP
SIR	SBFL	6	15	20	0.2906	6	15	20	0.2914	5	15	20	0.2842
	FOMs	27	41	43	0.4199	41	51	51	0.6216	41	54	55	0.6324
	2-HOMs	29	42	45	0.4964	39	49	50	0.6106	38	50	51	0.6123
	3-HOMs	27	37	41	0.4218	40	49	49	0.6158	40	49	50	0.6170
	4-HOMs	27	35	38	0.4087	41	48	50	0.6140	41	49	51	0.6208
	5-HOMs	26	39	40	0.4147	41	50	51	0.6192	41	51	52	0.6247
Codeflaws	SBFL	183	383	619	0.3253	182	381	616	0.4108	182	379	614	0.4093
	FOMs	821	1,043	1,143	0.7664	821	1,042	1,142	0.7649	825	1,048	1,144	0.7649
	2-HOMs	594	968	1,110	0.6541	821	1,044	1,145	0.7654	825	1,050	1,144	0.7654
	3-HOMs	650	970	1,117	0.6873	820	1,044	1,145	0.7646	825	1,049	1,144	0.7646
	4-HOMs	781	1,004	1,112	0.7385	820	1,038	1,138	0.7638	824	1,043	1,139	0.7638
	5-HOMs	810	1,023	1,126	0.7546	820	1,038	1,137	0.7640	824	1,043	1,139	0.7640

the run-time environment. A mutant set with m mutants executed by a test suite with n test cases, MTP can be calculated by the following formula:

$$MTP = m \times n. \tag{10}$$

A lower MTP value means the corresponding MBFL technique has better efficiency.

V. EXPERIMENTAL RESULTS

In this section, we present our experimental results to address the proposed research questions.

A. Research Questions

RQ1: Can HOMs help in improving fault localization accuracy in the SFL-scenario?
RQ2: Can HOMs help in improving fault localization accuracy in the MFL-scenario?
RQ3: Which MBFL formula is better for MBFL techniques when doing fault localization?
RQ4: What is the execution cost of HOMs when applied to fault localization?

RQ1 and RQ2 review the performance of HOMs in SFL-scenario and MFL-scenario. RQ3 compares the fault localization effectiveness of different MBFL formulas. RQ4 measures the cost of adopting HOMs.

B. Answer for RQ1 (Fault Localization Effectiveness on SFL-Scenario)

To answer RQ1, we use $Top\text{-}N$ (N is set to 1, 3, 5) and MAP to evaluate the fault localization effectiveness of different fault localization techniques (i.e., SBFL techniques and MBFL techniques with FOMs and HOMs).

In terms of the metrics $Top\text{-}1$, $Top\text{-}3$, $Top\text{-}5$, FOMs and HOMs again outperform SBFL techniques on three formulas (see Table V). For SIR, FOMs locate more faults than HOMs (from the second order to fifth order) on $Jaccard$, $Ochiai$ formulas at $Top\text{-}1$, $Top\text{-}3$, $Top\text{-}5$, while 2-HOMs on $Dstar$ formula have the best performance. For Codeflaws, 3-HOMs on $Jaccard$ and 2-HOMs on $Ochiai$ performs better than other techniques, which shows that in these two formulas, HOMs can help improve fault localization effectiveness.

In terms of the metrics MAP, FOMs localize faults more precisely than HOMs for SIR, and 2-HOMs localize faults more precisely than FOMs for Codeflaws on $Jaccard$, $Ochiai$. In these two benchmarks, MAP values for SBFL techniques do not exceed 0.5, which is the worst outcome.

To further determine the statistical significance between FOMs and HOMs, we collect the $EXAM$ of all program versions for different techniques and then employ the $wilcoxon\ signed\text{-}rank\ test$ [75] at a confidence level of 95%. Table VI summarizes the testing results on $EXAM$ of FOMs with SBFL and different HOM orders. In Table VI, the $p\text{-}value$ (highlighted background) being less than 0.05 means there is a statistical difference between FOMs and other techniques. We can find that the $EXAM$ of 2-HOMs have significant differences with FOMs on $Jaccard$, $Ochiai$ for SIR and $Dstar$, $Ochiai$ for Codeflaws.

In traditional MBFL, a mutant (FOM) is regarded as a partial fix or a similar version for faulty programs that can achieve better performance in the SFL-scenario [2], which matches Hypothesis 1. FOMs have a higher probability of fixing the fault and are more similar to the faulty programs than HOMs. When HOMs are applied in the SFL-scenario, they will inject more faults and increasing distance from the original one, which produces HOMs that cannot help improve in seeded fault programs (SIR). While a real single fault is more complex than the seeded fault, FOMs have a lower probability to fix or similar to the faulty version, but some HOMs (2-HOMs) mutates multiple lines that can be more similar to the real single fault, which improves the fault localization effectiveness.

> Summary for RQ1: FOMs can localize more faults on SIR and 2-HOMs can localize more faults on Codeflaws using the metrics $Top\text{-}N$ and MAP within the SFL-scenario. Further statistical testing shows that there is significant difference between FOMs and 2-HOMs. It suggests that 2-HOMs can help improve the fault localization effectiveness on real-fault programs in the SFL-scenario.

C. Answer for RQ2 (Fault Localization Effectiveness on MFL-Scenario)

To answer RQ2, we adopt $Top\text{-}N$ and MAP to evaluate the fault localization effectiveness of SBFL techniques and MBFL techniques.

TABLE VI
P-value OF MBFL USING FOMS AND HOMS IN THE SFL-SCENARIO WITH THREE FORMULAS

Technique	SIR			Codeflaws		
	Dstar	Jaccard	Ochiai	Dstar	Jaccard	Ochiai
SBFL	9.94E-10	1.12E-10	4.78E-09	7.05E-140	1.34E-139	1.12E-139
2-HOMs	0.3897	0.0129	0.0098	6.78E-74	0.1778	3.31E-42
3-HOMs	0.5970	0.2489	0.4859	1.03E-21	0.0356	5.30E-36
4-HOMs	0.0394	0.0060	0.0040	0.6492	0.0188	2.10E-38
5-HOMs	0.2719	0.0745	0.1538	1.63E-06	0.0004	4.27E-33

TABLE VII
Top-N AND AVERAGE *MAP* OF MBFL USING FOMS AND HOMS IN THE MFL-SCENARIO WITH THREE FORMULAS

Benchmark	Technique	Dstar				Jaccard				Ochiai			
		Top-1	Top-3	Top-5	MAP	Top-1	Top-3	Top-5	MAP	Top-1	Top-3	Top-5	MAP
SIR	SBFL	24	41	63	0.1188	17	38	62	0.1139	23	43	73	0.1282
	FOMs	49	87	101	0.2256	62	96	116	0.2666	66	105	118	0.2790
	2-HOMs	53	87	99	0.2317	63	97	120	0.2724	66	107	122	0.2792
	3-HOMs	50	94	108	0.2365	61	91	118	0.2686	64	105	123	0.2804
	4-HOMs	48	85	97	0.2212	61	90	109	0.2591	63	96	111	0.2659
	5-HOMs	46	86	99	0.2213	61	92	114	0.2633	62	101	115	0.2703
Codeflaws	SBFL	106	125	162	0.5383	106	125	162	0.5386	107	127	165	0.5415
	FOMs	182	195	208	0.7166	187	200	208	0.7669	188	201	209	0.7771
	2-HOMs	79	195	214	0.5970	187	203	212	0.7734	188	203	214	0.7826
	3-HOMs	105	180	205	0.5995	186	199	211	0.7739	188	202	211	0.7772
	4-HOMs	156	191	212	0.6510	185	199	209	0.7669	186	200	211	0.7753
	5-HOMs	175	191	210	0.7006	187	200	210	0.7689	188	201	212	0.7761

TABLE VIII
P-value OF MBFL USING FOMS AND HOMS IN THE MFL-SCENARIO WITH THREE FORMULAS

Technique	SIR			Codeflaws		
	Dstar	Jaccard	Ochiai	Dstar	Jaccard	Ochiai
SBFL	2.50E-47	2.08E-48	1.85E-50	8.39E-10	1.70E-11	4.47E-09
2-HOMs	0.1057	7.70E-07	3.13E-07	8.93E-14	0.2531	6.50E-09
3-HOMs	0.9766	0.0454	1.06E-05	8.49E-11	0.6536	2.73E-05
4-HOMs	0.3362	0.0006	5.09E-07	2.46E-01	0.6567	5.98E-06
5-HOMs	0.0228	0.2424	0.0086	0.0417	0.9785	8.14E-07

In terms of the metrics *Top-1*, *Top-3*, *Top-5*, FOMs and HOMs again outperform SBFL techniques on three formulas (see Table VII). For SIR, 3-HOMs performs best on *Dstar*, *Ochiai* formulas, while 2-HOMs performs best on *Jaccard* formula. For Codeflaws, FOMs can rank more faults at the top and 3rd place with *Dstar* formula, while 3-HOMs can place more faults in the top 1, 3, and 5 ranks than other techniques at the metric of *Top-1*, *Top-3*, and *Top-5* on the *Jaccard* and *Ochiai* formula.

In terms of the metric *MAP*, for SIR, 3-HOMs again localizes faults more precisely than other techniques with *Dstar*, *Ochiai* formulas, while 2-HOMs performs better on the *Jaccard* formula. For Codeflaws, FOMs have a higher *MAP* on average with the *Dstar* formula, while using the *Jaccard* and *Ochiai* formulas, 3-HOMs and 2-HOMs, respectively, perform better.

Table VIII shows the result of *wilcoxon signed-rank test* on the *EXAM* scores between FOMs and HOMs in the MFL-scenario. We can find that, in most cases, the *EXAM* of FOMs have significant differences when applying SBFL and HOMs techniques (except the case of 2-HOMs, 3-HOMs, 4-HOMs on *Dstar* and 5-HOMs on *Jaccard* for SIR and the case of 2- to 5-HOMs on *Jaccard* for Codeflaws). It is noted that there is a statistical difference between FOMs and 2-HOMs, 3-HOMs in the formulas of *Jaccard* and *Ochiai*. Also, the *EXAM* of 2-HOMs, 3-HOMs have significant differences on *Dstar* and *Ochiai* for Codeflaws.

The results match Hypothesis 2 that HOMs are closer to the correct program. FOM only changes one line that is hard to fix multiple fault programs, while a k-HOM mutates k lines has a higher probability of fixing the faults than FOMs. Also, HOMs with higher orders (e.g., 4-HOMs and 5-HOMs) will introduce more faults that are different from the original programs.

> Summary for RQ2: Using 2-HOMs and 3-HOMs in MBFL can localize more faults in terms of *Top-N* metric and *MAP* on both benchmarks. Statistical testing indicates that there is significant difference between FOMs, 2-HOMs and 3-HOMs. This finding suggests that 2-HOMs and 3-HOMs can help improve the effectiveness of fault localization in MFL-scenario than SFL-scenario.

D. Answer for RQ3 (Impacts of Different MBFL Formulas)

To answer RQ3, we use *Top-N* and *MAP* metrics to compare the fault localization effectiveness of different formulas. For the SFL-senario, in terms of the *Top-N* metric, the results in

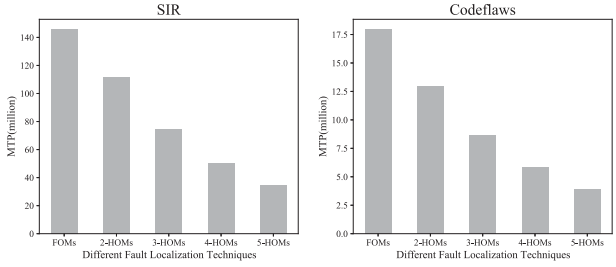


Fig. 3. MTP of MBFL using FOMs and HOMs in SFL-scenario. (a) SIR. (b) Codeflaws.

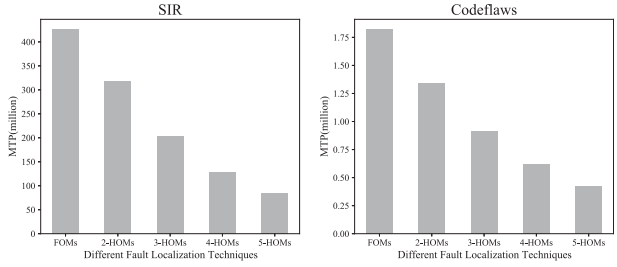


Fig. 4. MTP of MBFL using FOMs and HOMs in MFL-scenario.

Table V show that the $Ochiai$ formula can rank more faults at the top rank on SIR and Codeflaws, with 41 and 825, respectively. Furthermore, FOMs and 2-HOMs using $Ochiai$ find more faults at the Top-3 ranks on SIR and Codeflaws, with 54 and 1050, respectively. The $Jaccard$ formula can localize one more fault than $Ochiai$ on Codeflaws at the Top-5. In terms of the MAP metric, using the $Ochiai$ formula in MBFL techniques have a higher average MAP value compared to the other two formulas in most cases (see the MAP column of $Ochiai$ section). However FOMs have the highest MAP when applied $Dstar$ in Codeflaws.

For the MFL-scenario, in Table VII, in terms of $Top\text{-}N$, MBFL techniques can place more faults at the Top-1, Top-3, and Top-5 rank when using $Ochiai$ as the MBFL formula (see the column of $Top\text{-}1$, $Top\text{-}3$, and $Top\text{-}5$ in Section $Ochiai$). In terms of the MAP metric, MBFL techniques using the $Ochiai$ formula again outperform $Dstar$ and $Jaccard$ formulas in both SIR and Codeflaws (see the MAP column of $Ochiai$ section in Table VII).

From the above analysis, in the MFL-scenario, using $Ochiai$ as the MBFL formula, can localize faults more precisely than when using the $Dstar$ and $Jaccard$ formulas. This result is consistent with previous work [13].

Summary for RQ3: In both SFL-scenario and MFL-scenario, MBFL with $Ochiai$ formula has better fault localization performance than MBFL with the $Dstar$ and $Jaccard$ formulas.

E. Answer for RQ4 (Comparison of Mutation Execution Cost)

To address RQ4, we used the MTP metric to measure the mutation execution cost of each MBFL technique. Figs. 3 and 4 show the cost of FOMs and HOMs in terms of MTP values in the SFL-scenario and the MFL-scenario. The *X*-axis shows different MBFL techniques and the *Y*-axis is the sum of all versions' MTP values in one benchmark. From Fig. 3(a), FOMs have the highest cost because of the huge amount of mutants executed. Next, the 2-HOMs method presents 111.7 million MTPs of cost in SIR. The cost of 3-HOMs, 4-HOMs, and 5-HOMs decreases because the corresponding executed HOMs are decreasing while the test suite has no change. Similar cases can be concluded from Codeflaws and the MFL-scenario in Fig. 4. This indicates that combining HOMs on fault localization still incurs a huge execution cost. This suggests researchers should propose methods to reduce the number of HOMs.

TABLE IX
NUMBER OF PROGRAMS IN DIFFERENT FAULT COMPLEXITY

Benchmark	Program	2-fault	3-fault	4-fault	5-fault
SIR	printtokens	5	5	5	5
	printtokens2	5	5	5	5
	schedules	5	5	5	5
	totinfo	5	5	5	5
	tcas	5	5	5	5
	sed	5	5	5	5
Codeflaws		118	6	0	0

Summary for RQ4: In both SFL-scenario and MFL-scenario, the execution cost of using HOMs in MBFL is lower than that of using FOMs. However we still have to execute a large number of mutants on test cases. In future work, researchers should pay attention to reducing the cost of MBFL with HOMs.

VI. DISCUSSIONS

A. Fault Complexity and Mutant Order

Our study conducts an empirical study that applies HOMs on fault localization. The results indicate that both in single-fault localization scenarios and in multiple-fault localization scenarios, HOMs (i.e., 2-HOMs and 3-HOMs) can help improve fault localization effectiveness. In this section, we further investigate the relationship between fault complexity (i.e., the number of faults in the program) and mutant orders. Table IX lists the fault complexity of the program used in our experiments and we provide the additional results in the MFL-scenario (the results of the SFL-scenario are shown in Section ??). However, in the practical fault localization scenario, the complexity of faults is unknown for programmers.

Table X reports the results on SIR and Codeflaws benchmarks with different fault complexity. We use $Top\text{-}N$ and MAP to evaluate the results and $Ochiai$ as the MBFL formula in the table. From the results, we find that there is no strong correlation between the complexity of faults and mutant order. For SIR, 2-HOMs rank more faults at the top 5 than FOMs

TABLE X
Top-N AND AVERAGE *MAP* OF MBFL USING DIFFERENT ORDERS OF HOMS WITH OCHIAI FORMULA

Benchmark		Metric	SBFL	FOMs	2-HOMs	3-HOMs	4-HOMs	5-HOMs
SIR	2-fault	Top-1	4	21	21	21	21	21
		Top-3	11	34	34	33	33	33
		Top-5	14	36	36	35	35	35
		MAP	0.1779	0.5032	0.5020	0.5002	0.4852	0.4932
	3-fault	Top-1	5	20	20	20	20	20
		Top-3	10	27	28	28	27	27
		Top-5	17	29	30	31	30	29
		MAP	0.1219	0.2886	0.2940	0.2951	0.2897	0.2916
	4-fault	Top-1	8	13	13	13	13	12
		Top-3	11	24	25	24	20	21
		Top-5	19	27	27	27	23	25
		MAP	0.1165	0.1801	0.1786	0.1794	0.1668	0.1670
	5-fault	Top-1	6	12	12	10	9	9
		Top-3	11	20	20	20	16	20
		Top-5	23	26	29	30	23	26
		MAP	0.0965	0.1442	0.1425	0.1469	0.1220	0.1295
Codeflaws	2-fault	Top-1	99	175	175	175	173	175
		Top-3	119	188	189	189	187	188
		Top-5	152	194	198	196	196	196
		MAP	0.5287	0.7955	0.7967	0.7957	0.7926	0.7961
	3-fault	Top-1	8	13	13	13	13	13
		Top-3	8	13	14	13	13	13
		Top-5	13	15	16	15	15	16
		MAP	0.5544	0.7586	0.7685	0.7587	0.7580	0.7562

in Table X. Besides, 3-HOMs have a higher MAP value on programs with 3 faults and 5 faults, while 4-HOMs and 5-HOMs perform worse in all fault complexity. Similar findings can be summarized from the results of Codeflaws that 2-HOMs perform best, while 4-HOMs and 5-HOMs have the worse performance in that case. This is mainly attributed to the fact that some HOMs are closed to the correct programs and these HOMs are effective for fault localization. Besides, other HOMs different from the correct programs may have a negative influence on the fault localization effectiveness, which perform worse with more higher orders (e.g., 4-HOMs and 5-HOMs). Also, our findings are consistent with the study of Nguyen *et al.* [28] that high quality and reasonable HOMs (HOMs are more realistic complex faults [76]) can be found from 2-HOMs and 3-HOMs. Besides, the study of Wong *et al.* [64] shows that lower-order HOMs (e.g., 2-HOMs) are more likely to be SSHOMs [21] (strongly subsuming HOMs), which is a kind of HOMs with higher test effectiveness.

TABLE XI
DISTRIBUTION OF THE FAULTS IN MUTATION OPERATORS

Mutation Operator	SIR	Codeflaws
CRCR	213(43.8%)	280(18.2%)
OAAN	0	0
OAAA	0	0
OCNG	14(2.9%)	64(4.2%)
OIDO	14(2.9%)	3(0.2%)
OLLN	50(10.3%)	167(10.9%)
OLNG	0	0
ORRN	24(4.9%)	129(8.4%)
OBBA	0	0
OBBN	0	0
OCOR	0	4(0.3%)
SRSR	1(0.2%)	5(0.3%)
VTVD	2(0.4%)	27(1.8%)
VDTR	53(10.9%)	108(7.0%)
SSDL	15(3.1%)	413(26.9%)
Other	100(20.6%)	336(21.9%)
All	486(100.0%)	1536(100.0%)

B. Fault Type and Mutation Operator

In this section, we aim to investigate the relationship between faults and mutation operators. For each fault in the program, we classify it into a specific kind of mutation operator according to the comparison of "faulty" and "corrected" program versions. However, some faults cannot be classified following the rule of mutation operators, such as omission faults. As a result, the faults are classified into 16 categories, which are shown in Table XI. The top 15 categories are grouped by the mutation operators (see Table IV) and the "Other" category includes faults that cannot be classified.

Table XI shows the distribution of different fault types in two benchmarks. For SIR, 43.8% of the faults are grouped into the "CRCR" category, which accounts for the largest proportion of the total. Followed by 20.6% of the faults which cannot be classified in SIR. For Codeflaws, the most faults (26.9%) are the "SSDL" type, meaning these faults can be fixed by deleting the program statement. The unclassified faults in Codeflaws also accounted for 21.9% of all faults. Besides, we can find that the distribution of faults are similar in SIR and Codeflaws, with high proportion of 'CRCR,' 'OLLN,' 'Other' types of faults and without any "OAAN," "OAAA," "OLNG," "OBBA," "OBBN" types of faults. These findings may help the programmers to avoid these kind of faults in the real-world debugging process.

Furthermore, we collect the localization results of each fault classified in Table XI. We also use $Top\text{-}N$ and MAP to evaluate the performance of different techniques on each fault type. Note

TABLE XII
Top-N AND AVERAGE *MAP* OF DIFFERENT FAULT TYPES ON SIR WITH *Ochiai* FORMULA

Mutation Opeartor	Metric	SBFL	FOMs	2-HOMs	3-HOMs	4-HOMs	5-HOMs
CRCR	Top-1	9	60	58	59	60	59
	Top-3	24	85	83	86	82	83
	Top-5	54	93	92	97	91	92
	MAP	0.1307	0.3686	0.3682	0.3818	0.3701	0.3708
OCNG	Top-1	0	0	0	0	0	0
	Top-3	0	0	1	1	0	0
	Top-5	0	0	1	1	0	0
	MAP	0.0035	0.0036	0.0320	0.0511	0.0129	0.0110
OIDO	Top-1	1	4	5	4	3	3
	Top-3	2	7	6	5	4	6
	Top-5	2	7	7	7	5	6
	MAP	0.1236	0.4033	0.4254	0.3979	0.3046	0.3441
OLLN	Top-1	2	7	8	7	7	7
	Top-3	5	15	18	12	12	13
	Top-5	7	18	19	14	14	14
	MAP	0.1023	0.2557	0.2766	0.2497	0.2366	0.2410
ORRN	Top-1	0	11	10	10	10	10
	Top-3	1	13	12	13	11	12
	Top-5	1	13	12	13	13	13
	MAP	0.0748	0.5175	0.4964	0.5140	0.4696	0.4961
SRSR	Top-1	0	1	1	1	1	1
	Top-3	1	1	1	1	1	1
	Top-5	1	1	1	1	1	1
	MAP	0.5000	1.0000	1.0000	1.0000	1.0000	1.0000
VTVD	Top-1	0	0	0	0	0	0
	Top-3	1	1	1	0	0	1
	Top-5	1	1	1	1	0	2
	MAP	0.3088	0.2769	0.2121	0.1909	0.1242	0.3111
VDTR	Top-1	4	9	8	9	9	9
	Top-3	8	12	11	12	12	11
	Top-5	8	12	12	12	12	12
	MAP	0.1290	0.2104	0.2109	0.2105	0.2102	0.2035
SSDL	Top-1	0	4	4	4	4	4
	Top-3	1	5	5	5	5	5
	Top-5	1	5	5	5	5	5
	MAP	0.0756	0.3090	0.3086	0.3093	0.2965	0.3070
Other	Top-1	12	11	10	10	10	10
	Top-3	15	20	20	20	18	20
	Top-5	18	23	24	23	21	22
	MAP	0.1711	0.1796	0.1795	0.1797	0.1627	0.1714

that we only show the result of fault types with more than one fault. Tables XII and XIII present the detailed results in two benchmarks. For SIR, 3-HOMs performs better on "CRCR," "OCNG," "SSDL," and "Other" fault types (with higher MAP values). In most cases, FOMs and HOMs can localize the top nine kinds of faults more precisely than faults in the "Other" category (with higher MAP values). For Codeflaws, 2-HOMs have higher MAP values on "OLLN," "ORRN," "VTVD," "VDTR" "SSDL," and "Other" fault types. Also, the faults can be classified into specific mutation operators that have better fault localization effectiveness when using FOMs and HOMs. This suggests that mutation-based techniques place the faults that can be classified exactly by the mutation operators in a higher rank than the unclassified faults. This finding is consistent with the competent programmer hypothesis [46] that the programs that are close to being correct and these faults can be revealed by mutating. Moreover, Debroy and Wong [37] found that 20.70% of the faults can be fixed using some specific mutant operators in their study. Therefore, some mutation operators are better than others and the choice of mutation operators is important for the mutation-based techniques. Selecting and discovering more useful mutation operators for detecting faults becomes a future research problem.

VII. THREATS TO VALIDITY

A. Internal Validity

The first threat is the order of HOMs used in our study. Previous higher-order mutation testing has applied various orders [21], [22], [46], [77]–[79] (from 2-order to 70-order). We limited the HOMs to be between the second-order to fifth-order and did not consider higher-order mutants since previous studies [28], [63], [64] have shown that HOMs from the second order to fifth order are enough for finding valuable and reasonable HOMs in mutation testing. Moreover, the number of mutants grows exponentially with the order [21], which results in the higher computational cost in adopting HOMs with higher orders.

The second threat is the number of HOMs we generated and the test cases we used in our study. The number of HOMs generated for each order was the same as the number of FOMs, but the number of mutants compiled varies in different order of HOMs, which means the actual number executed for each order

TABLE XIII
Top-N AND AVERAGE *MAP* OF DIFFERENT FAULT TYPES ON CODEFLAWSWITH *Ochiai* FORMULA

Mutation Opeartor	Metric	SBFL	FOMs	2-HOMs	3-HOMs	4-HOMs	5-HOMs
CRCR	Top-1	48	178	178	180	178	177
	Top-3	88	235	236	237	233	234
	Top-5	137	256	259	259	251	253
	MAP	0.4590	0.7764	0.7790	0.7825	0.7742	0.7728
OCNG	Top-1	8	52	52	52	52	52
	Top-3	23	57	57	57	57	57
	Top-5	35	60	61	61	61	60
	MAP	0.3674	0.8814	0.8819	0.8820	0.8819	0.8811
OIDO	Top-1	0	1	1	1	1	1
	Top-3	1	2	2	2	2	2
	Top-5	1	3	3	3	3	3
	MAP	0.3434	0.5952	0.5952	0.5952	0.5952	0.5952
OLLN	Top-1	24	138	138	138	138	138
	Top-3	49	158	158	158	158	158
	Top-5	83	162	162	162	162	162
	MAP	0.4249	0.9019	0.9054	0.9023	0.9018	0.9017
ORRN	Top-1	22	101	101	101	101	101
	Top-3	40	112	112	112	112	111
	Top-5	57	120	120	120	120	120
	MAP	0.4476	0.8467	0.8491	0.8484	0.8465	0.8459
OCOR	Top-1	1	1	1	1	1	1
	Top-3	1	2	2	2	2	2
	Top-5	2	2	2	2	2	2
	MAP	0.3504	0.4884	0.4884	0.4884	0.4884	0.4884
SRSR	Top-1	0	3	3	3	3	3
	Top-3	1	4	4	4	4	4
	Top-5	2	5	5	5	5	5
	MAP	0.2606	0.7371	0.7371	0.7371	0.7371	0.7371
VTVD	Top-1	7	13	13	13	13	13
	Top-3	8	20	20	20	20	20
	Top-5	13	24	24	24	24	24
	MAP	0.6016	0.6898	0.7013	0.6920	0.6919	0.6859
VDTR	Top-1	16	58	58	58	58	58
	Top-3	25	80	81	80	80	80
	Top-5	53	92	92	91	92	91
	MAP	0.3651	0.6862	0.6870	0.6850	0.6851	0.6843
SSDL	Top-1	117	314	314	312	311	314
	Top-3	168	350	352	351	348	349
	Top-5	232	365	366	365	366	365
	MAP	0.4560	0.8315	0.8336	0.8308	0.8282	0.8313
Other	Top-1	46	154	154	154	154	154
	Top-3	102	229	229	229	227	227
	Top-5	164	264	264	263	264	266
	MAP	0.3907	0.6244	0.6256	0.6239	0.6232	0.6228

of HOMs is different. Moreover, we use all test cases provided by the benchmarks and different number of test cases may lead to different results. Therefore, the settings of mutants and test cases will influence the fault localization effectiveness. In the future, we will increase the search space of HOMs according to their order and find more efficient HOMs for fault localization. We will also investigate the impact of different numbers of mutants and test cases in our future work.

The third threat is the choice of mutation tools used in our empirical experiments. To alleviate this threat, we choose Proteum [60] as our mutation tool, which is popular and has been adopted in previous studies [12], [13], [19]. We will explore more mutation tools (such as MiLu [80]) to investigate the generalization of our empirical findings.

The fourth threat is the generation method of multiple-fault programs. For SIR, we combined the single real faults to generate multiple fault versions. These artificially generated multiple faults may be different from the real faults, which will lead to different results in our experiments. Therefore, we have selected the benchmark Codeflaws due to the fact that if contains real-world programs. In the future studies, we intend to choose alternative real-world multiple faults programs (such as CoREBench [81]) to investigate whether these findings are still valid.

B. External Validity

One external validity concerning the effectiveness of HOMs is the representatives of the subject programs we used. We have considered SIR [38] and Codeflaws [39] as our benchmarks which are all written in the C language. Although some of the programs are real-world and large-scale, these programs only represent limited classes of programs. In the future, we aim to conduct experiments on more large-scale programs (such as

CoREBench [81]) to investigate whether different results may be generated.

Another threat is the implementation correctness of MBFL. In our experiment, we implemented MBFL strictly based on the description of the original studies [12], [18] and the actual fault localization performance is very close to the results in these studies.

C. Construct Validity

In this experiment, we include three formulas ($Dstar$ [44], $Jaccard$ [45], and $Ochiai$ [41]) as the SBFL techniques and MBFL formulas. There exist other formulas and fault localization techniques which are not included. Different formulas and techniques may have different results. For example, both DeepFL [68] and PRINCE [58] are representative machine learning-based fault localization techniques and are used to compare with new machine learning-based fault localization techniques. Also, there are various mutation-based techniques [82]–[84] and the result may be different using these techniques.

We use metrics $EXAM$, $Top\text{-}N$, MAP to evaluate the performance of a technique. We also extend the analysis of MBFL execution cost using the metric MTP. $EXAM$ is popular, and used in evaluating the performance of fault localization techniques [4], [11]. $Top\text{-}N$ (other studies [58], [69] refer to $acc@n$) is a metric that uses absolute ranks rather than percentages of program inspected [58], [68], [69]. MAP has been widely used in previous fault localization studies [58], [69]. MTP counts the number of mutant runs for test cases rather than the run-time of test cases [54], [73]. The use of other metrics may produce different results. In the future, we also want to evaluate the performance of the techniques in terms of other performance metrics (such as $wef@n$ [69] and MAR [68]).

VIII. Conclusion

In this article, we conducted a large-scale empirical study to investigate fault localization effectiveness when using FOMs and HOMs in the SFL-scenario and the MFL-scenario. The experiments on two public benchmarks, i.e., SIR and Codeflaws, showed that in the SFL-scenario, FOMs and 2-HOMs have better fault localization effectiveness. In the MFL-scenario, both 2-HOMs and 3-HOMs performed better than SBFL techniques and FOMs. Further statistical tests showed that there was a significant difference between FOMs and 2-HOMs in both the SFL-scenario and the MFL-scenario. Moreover, MBFL techniques using $Ochiai$ had better performance than other formulas. We also evaluated the cost for HOMs on fault localization. HOMs had lower cost than FOMs and the high computational problem still existed such that researchers should propose methods to reduce the cost of using HOMs on fault localization.

In the future, we first want to further investigate the methodology of HOMs that can help improve the effectiveness of fault localization. And then we will investigate the mutant reduction techniques with ML-based and search-based algorithms [24].

References

[1] W. E. Howden, "Theoretical and empirical studies of program testing," *IEEE Trans. Softw. Eng.*, vol. 22, 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. Testing*, Verification Validation, 2016, pp. 299–308.

[3] 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.

[4] 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.*, vol. 47, no. 2, pp. 332–347, Feb. 2021.

[5] S. Rao and A. Kak, "Retrieval from software libraries for bug localization: A comparative study of generic and composite text models," in *Proc. 8th Work. Conf. Mining Softw. Repositories*, 2011, pp. 43–52.

[6] R. K. Saha, M. Lease, S. Khurshid, and D. E. Perry, "Improving bug localization using structured information retrieval," in *Proc. 28th IEEE/ACM Int. Conf. Autom. Softw. Eng.*, 2013, pp. 345–355.

[7] W. E. Wong, V. Debroy, R. Golden, X. Xu, and B. Thuraisingham, "Effective software fault localization using an RBF neural network," *IEEE Trans. Rel.*, vol. 61, no. 1, pp. 149–169, Mar. 2012.

[8] R. Gao, W. E. Wong, Z. Chen, and Y. Wang, "Effective software fault localization using predicted execution results," *Softw. Qual. J.*, vol. 25, no. 1, pp. 131–169, 2017.

[9] Z. Zhang, Y. Lei, X. Mao, and P. Li, "CNN-FL: An effective approach for localizing faults using convolutional neural networks," in *Proc. IEEE 26th Int. Conf. Softw. Anal., Evol. Reengineering*, 2019, pp. 445–455.

[10] 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, 2018.

[11] 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, 2019.

[12] M. Papadakis and Y. L. Traon, "Metallaxis-FL: Mutation-based fault localization," *Softw. Testing, Verification Rel.*, vol. 25, no. 5/7, pp. 605–628, 2015.

[13] 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, 2018.

[14] S. K. Lukins, N. A. Kraft, and L. H. Etzkorn, "Bug localization using latent dirichlet allocation," *Inf. Softw. Technol.*, vol. 52, no. 9, pp. 972–990, 2010.

[15] Q. Wang, C. Parnin, and A. Orso, "Evaluating the usefulness of IR-based fault localization techniques," in *Proc. Int. Symp. Softw. Testing Anal.*, 2015, pp. 1–11.

[16] 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.*, 2017, pp. 114–125.

[17] 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. Testing, Verification Validation*, 2014, pp. 153–162.

[18] M. Papadakis and Y. Le Traon, "Using mutants to locate "unknown" faults," in *Proc. IEEE 5th Int. Conf. Softw. Testing, Verification Validation*, 2012, pp. 691–700.

[19] M. Kooli, F. Kaddachi, G. Di Natale, A. Bosio, P. Benoit, and L. Torres, "Computing reliability: On the differences between software testing and software fault injection techniques," *Microprocessors Microsystems*, vol. 50, pp. 102–112, 2017.

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

[21] Y. Jia and M. Harman, "Higher order mutation testing," *Inf. Softw. Technol.*, vol. 51, no. 10, pp. 1379–1393, 2009.

[22] W. B. Langdon, M. Harman, and Y. Jia, "Efficient multi-objective higher order mutation testing with genetic programming," *J. Syst. Softw.*, vol. 83, no. 12, pp. 2416–2430, 2010.

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

[24] T. T. Chekam, M. Papadakis, T. F. Bissyandé, Y. L. Traon, and K. Sen, "Selecting fault revealing mutants," *Empir. Softw. Eng.*, vol. 25, no. 1, pp. 434–487, 2020.

[25] A. S. Ghiduk, M. R. Girgis, and M. H. Shehata, "Higher order mutation testing: A systematic literature review," *Comput. Sci. Rev.*, vol. 25, pp. 29–48, 2017.

[26] M. Papadakis, M. Kintis, J. Zhang, Y. Jia, Y. L. Traon, and M. Harman, "Mutation testing advances: An analysis and survey," in *Advances in Computers*, vol. 112. Amsterdam, Netherlands:Elsevier, 2019, pp. 275–378.

[27] E. Omar and S. Ghosh, "An exploratory study of higher order mutation testing in aspect-oriented programming," in *Proc. IEEE 23rd Int. Symp. Softw. Rel. Eng.*, 2012, pp. 1–10.

[28] Q. V. Nguyen and L. Madeyski, "On the relationship between the order of mutation testing and the properties of generated higher order mutants," in *Proc. Asian Conf. Intell. Inf. Database Syst.*, 2016, pp. 245–254.

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

[30] J. A. P. Lima, G. Guizzo, S. R. Vergilio, A. P. Silva, H. L. J. Filho, and H. V. Ehrenfried, "Evaluating different strategies for reduction of mutation testing costs," in *Proc. 1st Braz. Symp. Systematic Autom. Softw. Testing*, 2016, pp. 1–10.

[31] Lucia, F. Thung, D. Lo, and L. Jiang, "Are faults localizable?," in *Proc. 9th IEEE Work. Conf. Mining Softw. Repositories*, 2012, pp. 74–77.

[32] A. Zakari, S. P. Lee, R. Abreu, B. H. Ahmed, and R. A. Rasheed, "Multiple fault localization of software programs: A systematic literature review," *Inf. Softw. Technol.*, vol. 124, 2020, Art. no. 106312.

[33] N. DiGiuseppe and J. A. Jones, "On the influence of multiple faults on coverage-based fault localization," in *Proc. Int. Symp. Softw. Testing Anal.*, 2011, pp. 210–220.

[34] X. Xue and A. S. Namin, "How significant is the effect of fault interactions on coverage-based fault localizations?," in *Proc. ACM/IEEE Int. Symp. Empir. Softw. Eng. Meas.*, 2013, pp. 113–122.

[35] V. Debroy and W. E. Wong, "Insights on fault interference for programs with multiple bugs," in *Proc. 20th Int. Symp. Softw. Rel. Eng.*, 2009, pp. 165–174.

[36] N. DiGiuseppe and J. A. Jones, "Fault density, fault types, and spectra-based fault localization," *Empir. Softw. Eng.*, vol. 20, no. 4, pp. 928–967, 2015.

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

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

[39] S. H. Tan, J. Yi, Yulis, S. Mechtaev, and A. Roychoudhury, "Codeflaws: A programming competition benchmark for evaluating automated program repair tools," in *Proc. IEEE/ACM 39th Int. Conf. Softw. Eng. Companion*, 2017, pp. 180–182.

[40] H. Agrawal *et al.*, "Design of mutant operators for the c programming language," Software Eng. Res. Center, Purdue, W. Lafayette, IN, USA, Tech. Rep. SERC-TR-41-P, 1989.

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

[42] 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, 1978.

[43] M. Jimenez *et al.*, "Are mutants really natural? A study on how "naturalness" helps mutant selection," in *Proc. 12th ACM/IEEE Int. Symp. Empir. Softw. Eng. Meas.*, 2018, pp. 1–10.

[44] 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.

[45] 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.

[46] A. J. Offutt, "Investigations of the software testing coupling effect," *ACM Trans. Softw. Eng. Methodol.*, vol. 1, no. 1, pp. 5–20, 1992.

[47] Q. V. Nguyen and L. Madeyski, "Higher order mutation testing to drive development of new test cases: An empirical comparison of three strategies," in *Proc. Asian Conf. Intell. Inf. Database Syst.*, 2016, pp. 235–244.

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

[49] Q. V. Nguyen and L. Madeyski, "Addressing mutation testing problems by applying multi-objective optimization algorithms and higher order mutation," *J. Intell. Fuzzy Syst.*, vol. 32, no. 2, pp. 1173–1182, 2017.

[50] M. Harman, Y. Jia, and W. B. Langdon, "Strong higher order mutation-based test data generation," in *Proc. 19th ACM SIGSOFT Symp. 13th Eur. Conf. Found. Softw. Eng.*, 2011, pp. 212–222.

[51] R. Gopinath, C. Jensen, and A. Groce, "The theory of composite faults," in *Proc. IEEE Int. Conf. Softw. Testing, Verification Validation*, 2017, pp. 47–57.

[52] M. Kintis, M. Papadakis, and N. Malevris, "Employing second-order mutation for isolating first-order equivalent mutants," *Softw. Testing, Verification Rel.*, vol. 25, no. 5/7, pp. 508–535, 2015.

[53] 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.*, 2016, pp. 365–373.

[54] Z. Li, H. Wang, and Y. Liu, "HMER: A hybrid mutation execution reduction approach for mutation-based fault localization," *J. Syst. Softw.*, vol. 168, 2020, Art. no. 110661.

[55] H. Wang, B. Du, J. He, Y. Liu, and X. Chen, "IETCR: An information entropy based test case reduction strategy for mutation-based fault localization," *IEEE Access*, vol. 8, pp. 124297–124310, 2020.

[56] M. Hamill and K. Goseva-Popstojanova, "Common trends in software fault and failure data," *IEEE Trans. Softw. Eng.*, vol. 35, no. 4, pp. 484–496, Jul./Aug. 2009.

[57] D. Shin, S. Yoo, and D.-H. Bae, "A theoretical and empirical study of diversity-aware mutation adequacy criterion," *IEEE Trans. Softw. Eng.*, vol. 44, no. 10, pp. 914–931, Oct. 2018.

[58] Y. Kim, S. Mun, S. Yoo, and M. Kim, "Precise learn-to-rank fault localization using dynamic and static features of target programs," *ACM Trans. Softw. Eng. Methodol.*, vol. 28, no. 4, pp. 1–34, 2019.

[59] M. Papadakis, T. T. Chekam, and Y. L. Traon, "Mutant quality indicators," in *Proc. IEEE Int. Conf. Softw. Testing, Verification Validation Workshops*, 2018, pp. 32–39.

[60] 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*. Berlin, Germany:Springer, 2001, pp. 91–101.

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

[62] S. Pearson *et al.*, "Evaluating and improving fault localization," in *Proc. 39th Int. Conf. Softw. Eng.*, 2017, pp. 609–620.

[63] R. A. Silva, S. d. R. S. de Souza, and P. S. L. de Souza, "A systematic review on search based mutation testing," *Inf. Softw. Technol.*, vol. 81, pp. 19–35, 2017.

[64] C.-P. Wong, J. Meinicke, L. Chen, J. P. Diniz, C. Kästner, and E. Figueiredo, "Efficiently finding higher-order mutants," in *Proc. 28th ACM Joint Meeting Eur. Softw. Eng. Conf. Symp. Found. Softw. Eng.*, 2020, pp. 1165–1177.

[65] R. Gao and W. E. Wong, "MSeer–An advanced technique for locating multiple bugs in parallel," *IEEE Trans. Softw. Eng.*, vol. 45, no. 3, pp. 301–318, Mar. 2019.

[66] A. Zakari and S. P. Lee, "Parallel debugging: An investigative study," *J. Softw.: Evol. Process*, vol. 31, no. 11, 2019, Art. no. e2178.

[67] W. E. Wong, V. Debroy, and B. Choi, "A family of code coverage-based heuristics for effective fault localization," *J. Syst. Softw.*, vol. 83, no. 2, pp. 188–208, 2010.

[68] X. Li, W. Li, Y. Zhang, and L. Zhang, "DeepFL: Integrating multiple fault diagnosis dimensions for deep fault localization," in *Proc. 28th ACM SIGSOFT Int. Symp. Softw. Testing Anal.*, 2019, pp. 169–180.

[69] T.-D. B. Le, D. Lo, C. Le Goues, and L. Grunske, "A learning-to-rank based fault localization approach using likely invariants," in *Proc. 25th Int. Symp. Softw. Testing Anal.*, 2016, pp. 177–188.

[70] P. S. Kochhar, X. Xia, D. Lo, and S. Li, "Practitioners' expectations on automated fault localization," in *Proc. 25th Int. Symp. Softw. Testing Anal.*, 2016, pp. 165–176.

[71] A. Moffat and J. Zobel, "Rank-biased precision for measurement of retrieval effectiveness," *ACM Trans. Inf. Syst.*, vol. 27, no. 1, 2008, Art. no. 2.

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

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

[74] 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.*, 2017, pp. 126–137.

[75] F. Wilcoxon, "Individual comparisons by ranking methods," *Biometrics Bull.*, vol. 1, pp. 80–83, 1945.
[76] Q. V. Nguyen and L. Madeyski, "Empirical evaluation of multiobjective optimization algorithms searching for higher order mutants," *Cybern. Syst.*, vol. 47, no. 1/2, pp. 48–68, 2016.
[77] E. Omar, S. Ghosh, and D. Whitley, "Subtle higher order mutants," *Inf. Softw. Technol.*, vol. 81, pp. 3–18, 2017.
[78] Q.-V. Nguyen *et al.*, "Is higher order mutant harder to kill than first order mutant? An experimental study," in *Proc. Asian Conf. Intell. Inf. Database Syst.*, 2018, pp. 664–673.
[79] J. A. do Prado Lima and S. R. Vergilio, "A systematic mapping study on higher order mutation testing," *J. Syst. Softw.*, vol. 154, pp. 92–109, 2019.
[80] Y. Jia and M. Harman, "MILU: A customizable, runtime-optimized higher order mutation testing tool for the full C language," in *Proc. Testing: Academic Ind. Conf.- Pract. Res. Techn.*, 2008, pp. 94–98.
[81] M. Böhme and A. Roychoudhury, "CoREBench: Studying complexity of regression errors," in *Proc. Int. Symp. Softw. Testing Anal.*, 2014, pp. 105–115.
[82] S. Hong *et al.*, "Mutation-based fault localization for real-world multilingual programs (t)," in *Proc. 30th IEEE/ACM Int. Conf. Autom. Softw. Eng.*, 2015, pp. 464–475.
[83] X. Li and L. Zhang, "Transforming programs and tests in tandem for fault localization," *Proc. ACM Program. Lang.*, vol. 1, no. OOPSLA, pp. 1–30, 2017.
[84] J. Holmes and A. Groce, "Using mutants to help developers distinguish and debug (compiler) faults," *Softw. Testing, Verification Rel.*, vol. 30, no. 2, 2020, Art. no. e1727.

Haifeng Wang received the B.Sc. degree in information and computing sciences in 2017 from the Beijing University of Chemical Technology, Beijing, China, where he is currently working toward the Ph.D. degree in control science and engineering.

His current research interests include software testing and fault localization.

Zheng Li received the B.Sc. degree in the computer science and technology from the Beijing University of Chemical Technology, Beijing, China, in 1996, and the Ph.D. degree in computer science from the CREST Centre, King's College London, London, U.K., in 2009.

He is currently a Professor with the College of Information Science and Technology, Beijing University of Chemical Technology. He was a Research Associate with King's College London and University College London, London, U.K.. He has authored or coauthored more than 60 papers in referred journals or conferences, such as IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, International Conference on Software Engineering, *Journal of Software: Evolution and Process*, *Information and Software Technology, Journal of Systems and Software*, International Conference on Software Maintenance, IEEE International Conference on Software Maintenance and Evolution, International Computer Software and Applications Conference, IEEE International Working Conference on Source Code Analysis and Manipulation, and IEEE International Conference on Software Quality, Reliability and Security. His research interests include software engineering, in particular program testing, source code analysis, and manipulation.

Yong Liu received the B.Sc. and M.Sc. degrees in the computer science and technology, and the Ph.D. degree in control science and engineering from the Beijing University of Chemical Technology, Beijing, China, in 2008, 2011, and 2018, respectively.

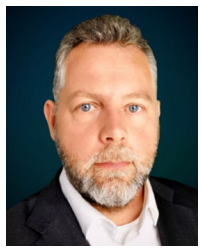
He is an Associate Professor with the College of Information Science and Technology, Beijing University of Chemical Technology. In the areas of his research interests, he has authored or coauthored more than 20 papers in referred journals or conferences, such as *Journal of Systems and Software, Information Sciences*, *Journal of Computer Science and Technology*, IEEE International Conference on Software Quality, Reliability and Security, and IEEE International Computer Software and Application Conference. His research interest includes software engineering. Particularly, he is interested in software debugging and software testing, such as source code analysis, mutation testing, and fault localization.

Dr. Liu is a member of the China Computer Federation and the Association for Computing Machinery.

Xiang Chen (Member, IEEE) received the B.Sc. degree in information management and system from the School of Management, Xi'an Jiaotong University, Xi'an, China, in 2002, the M.Sc. and Ph.D. degrees in computer software and theory from Nanjing University, Nanjing, China, in 2008 and 2011, respectively.

He is currently an Associate Professor with the Department of Information Science and Technology, Nantong University, Nantong, China. He has authored or coauthored more than 60 papers in referred journals or conferences, such as IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, *Information and Software Technology*, *Journal of Systems and Software*, IEEE TRANSACTIONS ON RELIABILITY, *Journal of Software: Evolution and Process*, *Software Quality Journal*, *Journal of Computer Science and Technology*, International Conference on Software Engineering, IEEE/ACM International Conference Automated Software Engineering, IEEE International Conference on Software Maintenance and Evolution, IEEE International Conference on Software Analysis, Evolution and Reengineering, and International Computer Software and Applications Conference. His research interests include software engineering, particularly software maintenance and software testing, such as software defect prediction, combinatorial testing, regression testing, and fault localization.

Dr. Chen is a Senior Member of the China Computer Federation and a member of the Association for Computing Machinery.



Paul Doyle received the Ph.D. degree in astronomical distributed data processing from the Dublin Institute of Technology, Dublin, Ireland, in 2015, building a national and globally distributed data processing network infrastructure for astronomical image processing.

He is currently the Head of the School of Computer Science, Technological University Dublin, Dublin, Ireland. He has spent more than 20 years in industry in Silicon Valley, CA, USA and Dublin, Ireland. He was a Product and Quality Director of CR2, a banking software company, Dublin, Ireland; a Senior Manager with Sun Microsystems, Menlo Park, CA, USA; and a Senior Developer with BlueStar Financial Investment. His research interests include Big Data processing of astronomical images, distributed systems, systems infrastructure, and educational pedagogy.

Yuxiaoyang Cai received the B.Sc. degree in information and computing sciences in 2020 from the Beijing University of Chemical Technology, Beijing, China, where he is working toward the master's degree in software engineering.

His research interests include fault localization and fault understanding.

Luxi Fan received the B.Eng. degree in software engineering from Tiangong University, Tianjin, China, in 2021. He is currently working toward the master's degree in software engineering with the Beijing University of Chemical Technology, Beijing, China.

His research interests include fault localization and fault prediction.