HMBFL: Higher-Order Mutation-Based Fault Localization

Zheng Li[†], Butian Shi[†], Haifeng Wang[†*], Yong Liu[†*], Xiang Chen[‡]
[†]*College of Information Science and Technology*, *Beijing University of Chemical Technology*, China
[‡]*School of Information Science and Technology*, *Nantong University*, China
Email: h.f.wang@hotmail.com, lyong@mail.buct.edu.cn

***Abstract*—Fault localization is one of the most important processes in debugging. Among various automated fault localization techniques, Mutation-Based Fault Localization (MBFL) is one of the commonly studied that could achieve better performance in single-fault localization scenarios. However, the MBFL technique adopted First-Order-Mutants (FOMs) does not perform well in multiple-fault localization. In this paper, we propose an approach HMBFL (Higher-Order Mutation-Based Fault Localization) that first applies Higher-Order-Mutants (HOMs) in fault localization. To utilize HOMs on fault localization, we present three methods for calculating statements' suspiciousness (i.e., Averaging, Maxing, and Frequency). Furthermore, to generate more effective HOMs for fault localization, we propose two strategies that based on the traditional techniques (i.e., SBFL-guided and MBFL-guided). Our empirical results on 112 real-world multiple-fault programs from Codeflaws show that HMBFL outperforms SBFL techniques and traditional MBFL techniques at the metric of EXAM and HMBFL place more faults at the top 1, 3, 5 ranks.**

***Index Terms*—Fault localization, High-order mutants, Multiple faults, High-order mutation-based fault localization**

I. Introduction

Fault localization is one of the most expensive activities in the software debugging process. To alleviate the human effort of fault localization, researchers have proposed different fault localization techniques. Such as Information Retrieval (IR)-based [1]–[4], Program Spectrum-based [5]–[7], and Mutation-based [8]–[10]. IR-based techniques treat the bug report as a query and consider source code elements as a document collection [1]. Then it ranks elements according to their textual similarity with the report. Although IR-based techniques have been shown to be as effective as Spectrum-based techniques [2], there are several issues that avoid them to be widely used [11]. One issue is IR-based techniques are based on the assumption that the bug reports provided by the users can work well as queries, which is satisfied depends on the type of software considered, the characteristics of the bug, and so on. Another problem is that most IR-based techniques locate faults at the file level, leaving developers with a large amount of code to examine.

Spectrum-Based Fault Localization (SBFL) technique was first proposed by Reps et al. [12] and has been continuously researched and improved since then. The core idea is to execute a set of test cases on the program under test and then compare the coverage of failed test cases and pass test cases on the same statement. The possibility of faults in the statement is calculated and the localization of the faulty statements is inferred. Although the SBFL technique has many advantages, simple, automatic, and efficient, the accuracy of localization is affected by factors such as interference between multiple-fault and coincidental correctness test cases [13], [14].

Mutation-Based Fault Localization (MBFL) is a technique based on mutation analysis [15] that works by making syntactic changes on the program under test [16]. Studies have shown that the fault localization effectiveness of MBFL is significantly better than that of SBFL [8]. However, most of the research on the MBFL technique focuses on the first-order mutation, which is applicable on single faults and artificially seeded faults. Zou et al. [17] conducted an empirical study on a dataset containing real faults and found that the MBFL technique performed worse on most real-world programs.

In real-world scenarios, the faults in programs are more complex with multiple ones. Xue et al. [13] found that individual faults interfere with each other in multiple-fault programs, so it is more difficult to locate faults in multiple-fault programs than in single-fault programs using existing fault localization techniques. Debroy and Wong [18] stated that Higher-Order-Mutants (HOMs) can be employed to address the problem of multiple-fault in the program. Moreover, Jia et al [19] found that when they used a search-based approach to HOMs generation, HOMs could simulate the real faults better than First-Order-mutants (FOMs). Hence, we employ the HOMs in this study to improve the fault localization effectiveness in multiple-fault scenarios.

In this paper, we propose an approach HMBFL (Higher-Order Mutation-Based FaultLocalization) that first applies Higher-Order-Mutants (HOMs) in fault localization. To adopt HOMs on fault localization, we present three statements' suspiciousness calculation methods (i.e., Averaging, Maxing, and Frequency) and two strategies for generating effective HOMs (i.e., SBFL-guided and MBFL-guided).

To verify the performance of HMBFL on multiple-fault localization, we do experiments on 112 real-world programs from Codeflaws [20]. Based on the results in our empirical study, we have found HMBFL can achieve better multiple-fault localization performance than SBFL techniques and traditional MBFL techniques. Moreover, our proposed Frequency method for calculating statements' suspiciousness and MBFL-guided generation strategy is effective in fault localization.

The main contributions of this paper are summarized as follows:

2767-6684/21/$31.00 ©2021 IEEE
DOI 10.1109/DSA52907.2021.00016

- We propose a novel fault localization approach, HMBFL (Higher-Order Mutation-Based Fault Localization). To the best of our knowledge, HMMFL is the first to employ HOMs on fault localization.
- To utilize HOMs on fault localization, we propose three methods (i.e., Averaging, Maxing, and Frequency) to calculate the suspiciousness value of statements, and two strategies (i.e., SBFL-guided and MBFL-guided) to generate effective HOMs.
- To evaluate the performance of HMBFL, we conduct the experiment on 112 programs from a real-world benchmark, Codeflaws. The results show that HMBFL can achieve better multiple-fault localization performance than SBFL and traditional MBFL techniques in terms of the EXAM, Top-1, Top-3, Top-5 metrics. Moreover, our proposed Frequency method for calculating statements' suspiciousness and MBFL-guided generation strategy is effective in fault localization.
- To facilitate the replication of our study and evaluation of future work, our source code and dataset used in this paper are all available in the GitHub repository[1].

The rest of this paper is organized as follows. Section II provides the background of multiple-fault localization and related techniques. Section III presents our proposed methods and strategies for statements' suspiciousness calculation and HOMs generation. Section IV first describes the experiment setup, then presents the results of the experiment and discusses threats to the validity of our experiment. Section V overviews the works related to this study. Section VI finally concludes this paper with potential future works.

II. Background

A. Multiple-fault Localization

Statistics In the last decades, a significant number of studies have been proposed for localizing multiple-fault. These approaches identified by Zakari et al. [21] three categories, which are One-bug-at-a-time (OBA) approach, the parallel approach, and multi-bug-at-a-time (MBA) approach.

OBA approach is a simple approach that neutralizes a single fault per debugging iteration and it is performed until all the faults are found and fixed [22], [23]. Various fault localization techniques such as SBFL techniques [5], [24] have utilized the OBA approach in localizing multiple-fault.

Second, the parallel debugging approach is basically dividing the debugging task into small units so as to allow multiple developers to work on different units [25], such as cluster the failed test cases into different fault-focused cluster [26]. The fault-focused clusters which composed of both failed and passed test cases will be given to separate individual developers to debug in parallel.

MBA approach localizes multiple-fault in a single debugging iteration, unlike the OBA approach needs several iterations [27]. MBA has the advantage of reducing the debugging time and reducing the computation overhead of utilizing clustering algorithms in identifying failure-to-fault relationships as in the case with parallel debugging approach [28]. Follow the most multiple-fault studies, in this paper, we utilize the OBA approach in identifying multiple-fault using the HOMs.

[1]https://github.com/chajishaji/HMBFL

TABLE I
TYPICAL MUTATION OPERATORS

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	Increase/Decrease 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

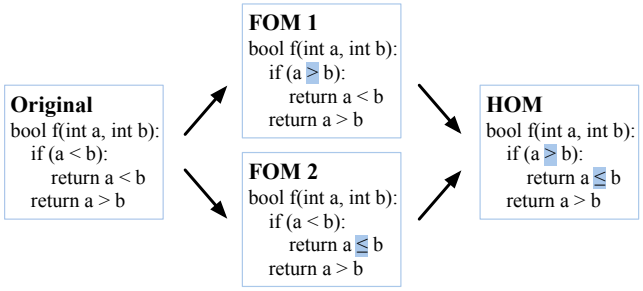


Fig. 1. Example of the FOM and HOM

B. Mutation-based Fault Localization

Mutation-based fault localization is a widely researched method of fault localization based on mutation analysis. Mutation analysis is performed by making simple semantic changes to the program under test and the program with artificially injected faults are called mutants [15]. The rule that generates the mutant is known as mutation operator [10], [29]. Table I lists ten typical C mutation operators proposed by Agrawal et al. [30]. According to the usage of times of the mutation operator, the mutants can be classified into two groups: First-Order-Mutants (FOMs) and Higher-Order-Mutants (HOMs). A FOM is generated by applying the mutation operator only once and a HOM is generated by applying the mutation operator more than once [19]. Noted that a k-HOM is a k^{th} order mutant that applying k mutation operators on k different statements.

As shown in Fig. 1, the program is mutated into two FOMs, which are later combined to form a HOM. If a test case has the execution behavior of a FOM or HOM different from the original we say that the FOM or HOM is $killed$ or $detected$. Otherwise, we say that the FOM or HOM is $notkilled$ or $live$ [18].

The traditional MBFL technique consists of four main steps:

(1) Get the statements covered by failed test cases: MBFL technique first executes a program under test P by a test suite

TABLE II
SUSPICIOUSNESS FORMULAS FOR MBFL

Name	Formula
Tarantula [24]	$Sus(m) = \frac{\frac{a_{kf}}{a_{kf}+a_{kp}}}{\frac{a_{kf}}{a_{kf}+a_{kf}}+\frac{a_{kp}}{a_{kp}+a_{np}}}$
Op2 [31]	$Sus(m) = a_{\text{kf}} - \frac{a_{kp}}{a_{kp}+a_{np}+1}$
Jaccard [32]	$Sus(m) = \frac{a_{kf}}{a_{kf}+a_{nf}+a_{np}}$
Ochiai [33]	$Sus(m) = \frac{a_{\text{kf}}}{\sqrt{(a_{kf}+a_{nf})(a_{kf}+a_{kp})}}$
Dstar [7]	$Sus(m) = \frac{a_{kf}^{*}}{a_{kp}+a_{nf}}$
GP13 [6]	$Sus(m) = a_{kf}\left(1+\frac{1}{2a_{kp}+a_{kf}}\right)$
Naish1 [31]	$Sus(m) = \begin{cases} -1 & \text{if } a_{kf} < \lvert T_f\rvert \\ \lvert T_p\rvert - a_{kp} & \text{if } a_{kf} = \lvert T_f\rvert \end{cases}$

T. Next, the coverage information and test results are obtained for classifying T into pass tests set T_p and fail tests set T_f. Besides, all statements covered by the failed tests are recorded as COV_f.

(2) Generate and execute mutants: The MBFL employs mutation operators to seed faults into the statement s in COV_f. The set of mutants mutated from the statement s is denoted as $M(s)$. Then, all mutants in $M(s)$ are executed against the tests in T. The results can be divided into T_k and T_n, where T_k is the set of mutants are killed by T and T_n is the set of mutants do not killed by T.

(3) Calculating the suspiciousness of program statements: The suspiciousness of the mutant m can be calculated using different MBFL formulas, which are based on the following four parameters: $a_{np} = |T_n \cap T_p|$, $a_{kp} = |T_k \cap T_p|$, $a_{nf} = |T_n \cap T_f|$, and $a_{kf} = |T_k \cap T_f|$, where a_{np} denotes the number of pass tests that cannot killed m, a_{kp} denotes the number of pass tests that killed m, a_{np} denotes the number of fail tests that cannot killed m, and a_{nf} denotes the number of fail tests that killed m. Table II lists seven popular MBFL formulas (i.e., Tarantula [24], Op2 [31], Jaccard [32], Ochiai [33], Dstar [7], GP13 [6], Naish1 [31]). These formulas are proposed based on the different rule of statistical methods and studies [34], [35] have shown that there does not exist one formula outperform the other formulas in any scenario.

(4) Generate statements' ranking list: The MBFL technique aggregates the mutant value to the suspiciousness of the statement s given by the maximum value: $Sus(s) = Maxing(Sus(m_1), \ldots, Sus(m_n))$, where $m_1, \ldots, m_n$ are all mutants in $M(s)$. Then, MBFL sorts all statements in descending order based on their suspiciousness value and returns a ranking list. Developers can find and fix program faults from top to bottom according to the ranking list.

Based on the description of the above process, we can find that MBFL works based on the assumption that mutants killed mostly by failed test cases have a connection with the program faults. Recent studies [9], [36] also demonstrated that MBFL could significantly outperform other types of fault localization techniques (such as spectrum-based fault localization techniques [8], [36]).

III. OUR APPROACH

A. Motivation

In previous studies, most MBFL techniques were based on the single-fault assumption. However, empirical studies have shown that single program failures are often triggered by multiple-fault in the system. Digiuseppe and Jones [37] found that multiple-fault negatively affect the accuracy of fault localization. In addition, Offutt's findings concluded that it remains to be determined whether killing nth-order mutants can detect complex faults [38]. In Debory and Wong's study, they found that the reason why their proposed strategy could not fix multiple-fault in the same program was that they only considered first-order mutation [18].

Therefore, in this study, we employ HOMs to improve the effectiveness of multiple-fault localization. Since we are the first to apply HOMs on fault localization, we first need to solve the problem of how to calculate statements' suspiciousness using HOMs (see Phase 1 of Fig 2). In addition, the space of HOMs rises exponentially with the order, we are trying to propose strategies to find more effective HOMs for fault localization (see Phase 2 of Fig 2). In Phase 3 of Fig 2, we adopt HOMs in fault localization to produce a ranking list (see Phase 1 of Fig 2).

B. Statements' Suspiciousness Calculation Method

In the traditional MBFL process, the suspiciousness of a statement s is assigned by the maximal value of mutants' suspiciousness. It can be represented by:

$$Sus(s) = Maxing(Sus(FOM_1), \ldots, Sus(FOM_n)),$$

where $FOM_1, \ldots, FOM_n$ are the FOMs generated from s.

From the generation rule of HOMs, a k-HOM is associated with k statements that applying mutation operators in different lines, which introduces the cases of multiple mapping relationships between the statement s and HOMs associated with s [2].

In this section, we present three methods to handle the multiple mapping cases to apply HOMs on fault localization:

(1) Averaging: To reduce the impact of outliers, for each statement, the suspiciousness of the statement s is assigned by the averaging value of HOMs' suspiciousness associated with s. It can be represented by:

$$Sus(s) = Averaging(Sus(HOM_1), \ldots, Sus(HOM_m)),$$

where $HOM_1, \ldots, HOM_n$ are all HOMs associated with s.

(2) Maxing: Inspired by the idea of the traditional MBFL technique, a HOM with higher suspiciousness indicates the statements mutated to have a higher possibility to be faulty. The statements' suspiciousness assigned by the maximal value of HOMs' suspiciousness. It can be formulated as:

$$Sus(s) = Maxing(Sus(HOM_1), \ldots, Sus(HOM_m)),$$

[2]In this paper, we say a k-HOM is associated with the statement s is k-HOM generated from mutating s once and mutating other $k-1$ statements each time.

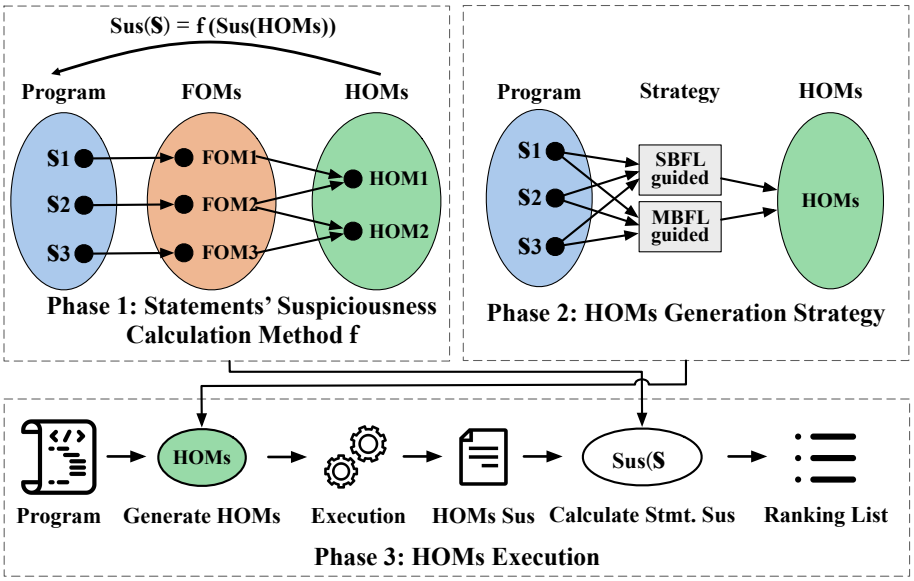


Fig. 2. Framework Of HMBFL

where $HOM_1, \ldots, HOM_m$ are all HOMs associated with s.

(3) Frequency: Another method to calculate statements' suspiciousness is to aggregate the frequency of maximum value in HOMs' suspiciousness. This method designed for breaking the tie of statements with the same maximum value. It can be represented as:

$$Sus(s) = Frequency(||\{HOM_i | Sus(HOM_i) = Maxing\}||),$$

where $Maxing$ is the suspiciousness of s computed by Maxing method, $HOM_i \in \{HOM_1, \ldots, HOM_m\}$ and $||\cdot||$ is the number of elements in the set.

To illustrate how to use these methods to calculate the suspiciousness of statements, we present an example in Table III. Table III consists of three statements and each one has three associated HOMs with their suspiciousness.

For Averaging method, the suspiciousness of s_1 is $Sus(s_1) = \frac{1.0+0.9+0.5}{3} = 0.8$, and using Maxing method: $Sus(s_1) = Maxing(1.0, 0.9, 0.5) = 1.0$. To break the tie of ranking computed by Maxing method, s_1's suspiciousness is updated by $Sus(s_1) = Frequency(||HOM_1||) = 1.0$.

Another method to improve the statements' suspiciousness when adopting HOMs, we combined the suspiciousness assigned by FOMs with that assigned by HOMs. The **Combined** method can be represented by:

$$Sus(s)_{Combined} = Maxing(Sus(s)_{FOMs}, Sus(s)_{HOMs}),$$

where $Sus(s)_{FOMs}$ and $Sus(s)_{HOMs}$ are the statements' suspiciousness computed by FOMs and HOMs.

C. HOMs Generation Strategy

As mentioned in Section II, a k-HOM is produced by applying k mutation operators on k different statements. To generate 2-HOMs for a program with n lines and each line have one operator (i.e., one FOM), the total number of 2-HOMs is $C(n, 2)$, which is a combination number. Assume that $n = 50$, and the number of 2-HOMs is $C(50, 2) = 1225$. In real programs, one line can be applied to mutation operators in several locations, which leads to a huge space of HOMs. It is a high computational cost to execute all HOMs against the tests. To reduce the cost of mutation testing and generate more effective HOMs, we propose two strategies that make use of the test information of SBFL technique and traditional MBFL technique, as shown in Phase 2 of Fig 2.

The idea of these two strategies is a statement with higher rank indicates higher probability to be faulty. Follow this intuition, for a program with n statements $P = \{s_1, \ldots, s_n\}$, we define $weight$ of a statement s to measure this probability:

$$weight(s) = \frac{length(Ranking\ list) - rank(s) + 1}{\sum_{i=1}^{n} rank(s_i)},$$

where $length(Ranking\ list)$ and $rank(s)$ are the length of the ranking list and the rank of statements produced by the SBFL technique or MBFL technique.

We use the $weight$ to guide the generation of HOMs. Assume to generate m k-$HOMs$, the specific steps are as follows:

Step 1: For each statement s_i in P, the number of k-$HOMs$ associated to statement s_i is $Num(k\text{-}HOMs(s_i)) = m \times weight(s_i)$.

Step 2: Then, follow the order of programs, we first randomly generate $Num(k\text{-}HOMs(s_1))$ k-$HOMs$ associated with s_1. By the same way, we generate $Num(k\text{-}HOMs(s_2))$ k-$HOMs$ for next statement s_2.

Step 3: The rest of k-$HOMs$ are produced according to **Step 2**). Finally, we have generated m k-$HOMs$ for program P.

TABLE III
WORKING EXAMPLE OF THREE STATEMENTS' SUSPICIOUSNESS CALCULATION METHODS

Statements	HOMs	HOMs Sus	Stmt.Sus Averaging	Rank	Stmt.Sus Maxing	Rank	Stmt.Sus Frequency	Rank
s_1	HOM_1	1.0	0.8	1	1.0	1	1	2
	HOM_2	0.9						
	HOM_3	0.5						
s_2	HOM_1	1.0	0.8	1	1.0	1	2	1
	HOM_4	1.0						
	HOM_5	0.4						
s_3	HOM_2	0.9	0.7	2	0.9	2		3
	HOM_6	0.5						
	HOM_7	0.6						

For simplify, we named the strategy using SBFL techniques to calculate $weight$ as **SBFL-guided** and the strategy using MBFL techniques to calculate $weight$ as **MBFL-guided**.

D. HOMs Execution

In the last phase of our approach, we apply the statements' suspiciousness calculation method and the HOMs generation strategy in the HOMs execution. In detail, we first generate the HOMs follow the strategy of **SBFL-guided** and **MBFL-guided** (see Section III-C). Then, all generated HOMs are executed against the tests to collect killing information. With the killing information, we calculate the HOMs' suspiciousness using MBFL formulas. Next, we obtain the statements' suspiciousness by adopting the three calculation methods (see Section III-B). Finally, we produce a ranking list of statements by sorting the statements' suspiciousness in descending order.

IV. EXPERIMENTAL STDUY

A. Research Questions

To evaluate the effectiveness of our proposed approach that adopted HOMs, we investigate the following four research questions:

- **RQ1:** What is the fault localization effectiveness of MBFL with HOMs when applying different statements' suspiciousness calculation methods?
- **RQ2:** How does MBFL with HOMs perform when combining the traditional MBFL's results in terms of the fault localization effectiveness?
- **RQ3:** What is the fault localization effectiveness of MBFL with HOMs when applying different generation strategies?
- **RQ4:** Compared with SBFL and traditional MBFL techniques, how does our approach perform about the fault localization effectiveness?

RQ1 examines the performance of different calculation methods of statements' suspiciousness. **RQ2** further studies if combining FOMs will impact the fault localization of MBFL with HOMs. **RQ3** focuses on evaluating the performance of two generation strategies for HOMs. **RQ4** finally examines the fault localization effectiveness of our approach.

In our experiments, we use four suspiciousness formulas, i.e., Ochiai [33], Dstar [7], GP13 [6], Naish1 [31], as SBFL techniques and MBFL formulas. Of these formulas, Ochiai is publicly studied in the previous works [9], [16] and Wong et al. empirically shown that Dstar is optimal to localize single and multiple faults. The other two formulas of GP13 and Naish1 are proven to be maximal in theory [39].

B. Experimental Setup

1) Subject Programs: We evaluate the effectiveness of our approach on the real-world benchmark of Codeflaws [20]. Codeflaws [20] consists of 3902 real fault programs out of 7,436 programs selected from the Codeforces[3] online database. Each fault in this benchmark program has rejected 'faulty' submission and the accepted 'corrected' submission. The programs with a single fault and cannot detect failures or runtime errors are excluded. Overall, we consider 112 multiple-fault programs out of 3,902 ones.

2) Configuration: In our study, we use the GNU gcov tool [40] to collect coverage information and we develop a tool to generate FOMs and HOMs, which is publicly available in Github repositories[4]. In our tool, we employ mutation operators suggested by the work of Agrawal et al. [30]. Table I lists ten typical mutation operators.

We choose to generate HOMs with 2-order since the study of Nguyen et al. [41] and Wong et al. [42] indicated that the lower order of mutants has more effective on mutation testing. Moreover, 2-HOMs have been studies in various works [43], [44].

For the huge space of HOMs, we have tested the number of HOMs from one to twenty times that of FOMs in our pre-experiments. The results showed that HOMs has a better fault localization effectiveness at ten times with acceptable cost on our experiment environment. Therefore, we set the number of HOMs generated as ten times that of FOMs. Overall, for 112 multiple-fault programs, we first generate 26,003 FOMs applied the mutation operators presented by Agrawal et al. [30], and then we generate 26,0030 HOMs (ten times to the number of FOMs) for each HOMs generation strategy (i.e., random, SBFL-guided, and MBFL-guided), leading to $26,0030 \times 3 = 780,090$ 2-HOMs in total.

3) Evaluation Metrics: To evaluate the performance of our proposed approach in this paper on multiple-fault localization, we use the evaluation metrics EXAM and Top-N. These two

[3]https://codeforces.com

[4]https://github.com/chajishaji/HMBFL

metrics have been widely used in the previous studies [17], [45].

EXAM: EXAM is the percentage of program elements that have to inspected until finding the exact faulty element. A lower EXAM indicates a better fault localization technique [17]. The formula of EXAM can be represented by Equation 1.

$$EXAM = \frac{rank}{number\ of\ executable\ statement} \quad (1)$$

In Equation 1, $rank$ is the rank of the faulty statement in the ranking list and $number\ of\ executable\ statement$ is the total number of statements should be checked. More detail, $rank$ is formulaed by:

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

where 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. We take the average of the first $(i+1)$ and last $(i+j)$ ranks to determine the rank of the faulty statement to break the tie.

Top-N: Top-N measures how many faults can be located within the top N program elements among all candidates [46]. Apparently, a fault localization technique with higher Top-N is better than others. In the work of Kochhar et al. [47], they found that 73.58% developers only inspect Top-5 program elements. Follow the previous study [45], [46], [48], we set N to 1, 3, 5 to make comparisons.

C. Results Analysis

1) ***RQ1 (The effectiveness of different calculation methods):*** To answer this question, we use EXAM and Top-N (N is set to 1, 3, 5) to evaluate the performance of three statements' suspiciousness calculation methods (denoted by Averaging, Maxing, and Frequency) proposed in Section III-B. We generate HOMs by random strategy, i.e., generating HOMs without any guiding strategy. Besides, we display the results with four MBFL formulas (i.e., Ochiai, Dstar, GP13, and Naish1).

In terms of the metric EXAM, the Frequency method performs better than the other two methods. We compute the EXAM of three methods for 112 programs and display the results via a box-plot diagram (see Fig. 3). From the bottom to the top, each column of these diagrams presents the minimum, the 1st quartile, the medium, the 3rd quartile, and the maximum of the EXAM of the respective program. From Fig. 3, we can see that the Frequency method performs better than Averaging and Maxing in four formulas, with a distribution of EXAM closer to the $X\text{-}axis$. Moreover, the performance of the Averaging method varies with the formula and Averaging performs better in the formula of Ochiai. Besides, Maxing is more stable between these formulas with the worse results.

In terms of the metric Top-1, Top-3, and Top-5, the Frequency method again outperform the Averaging method and Maxing method. Table IV presents the results of three methods employing four formulas and the best value are marked by the background color of a gray color. As shown in Table IV, Frequency also outperforms Averaging and Maxing in the metric of Top-1, Top-3, and Top-5, when using the four formulas. Besides, the Averaging method ranks more faults than the Maxing method in most cases, but the Maxing method ranks 2 more faults than the Averaging method in Dstar. The Maxing method also performs poorly at Top-1, Top-3, and Top-5.

As mentioned in Section III, the Averaging method takes the average value of associated HOMs' suspiciousness as statements' suspiciousness. The average of these suspiciousness measures the central tendency of one formula and different suspiciousness formulas produce different values. Therefore, the statements' suspiciousness with the Averaging method various in different MBFL formulas. The Maxing method takes the maximal value of associated HOMs' suspiciousness as statements' suspiciousness. If one HOMs with maximal value, the corresponding statements are all assigned by the maximum, which leads to the non-faulty statements have a higher rank and has the worst fault localization performance. However, the Frequency method breaks the tie of statements with the same maximal value and alleviates the problem that exists in the Maxing method, which achieves a better performance than the Averaging method and the Maxing method.

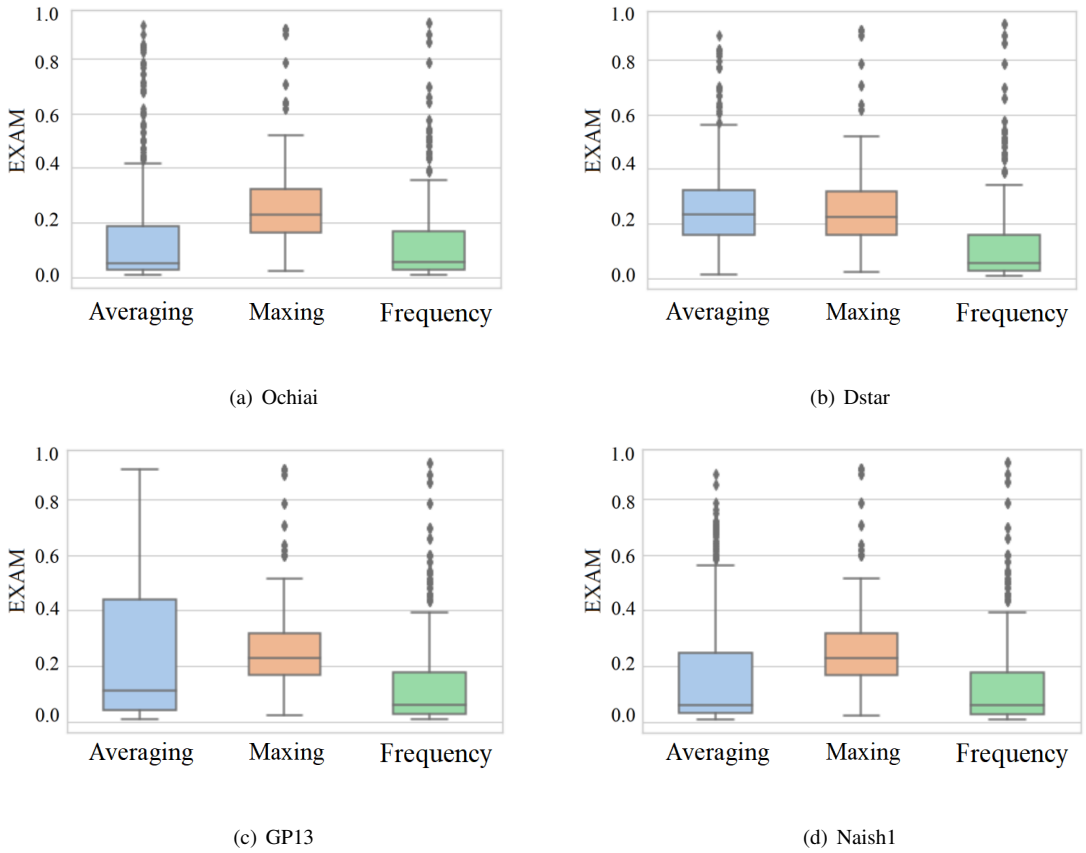
In summary, the statements' suspiciousness calculation method of Frequency performs better than the Averaging method and the Maxing method at the metric of EXAM and Top-N.

2) ***RQ2 (The effectiveness of Combined method):*** To answer this question, we compute the EXAM and Top-N for the Combined method, as mentioned in Section III-B. We compare the fault localization effectiveness of the Combined method with the three suspiciousness calculation methods. In this RQ, we randomly generate HOMs as RQ1 and use four MBFL formulas.

In terms of the metric EXAM, the Combined method improves the Averaging method and has no significant effect on Maxing and Frequency. Fig. 4 presents the EXAM distribution with box-plots. In each box plot of a method, it is divided into two types: the method without combining the FOMs (None) and the method with combining the FOMs (Combined). It can be seen from Fig. 4 that, in all formulas, the Combined method had no significant effect on the results of Maxing and Frequency. However, the Combined method improves the Averaging method in the formula of Ochiai, GP13, and Naish1. Moreover, the Combined method performs better than the Frequency method when using Ochiai as the MBFL formula.

In terms of the metrics Top-1, Top-3, and Top-5, the Combined method also performs better on the Averaging method and has no improvements than Maxing and Frequency. Table V also shows that the Combined method is large improved the Averaging method at Top-1 under the Ochiai, GP13, and Naish1, which are the best results.

As mentioned in Section III-B, the Combined method



(a) Ochiai (b) Dstar

(c) GP13 (d) Naish1

Fig. 3. EXAM of three statements' suspiciousness calculation methods with four formulas

TABLE IV
TOP-N OF THREE STATEMENTS' SUSPICIOUSNESS CALCULATION METHODS WITH FOUR FORMULAS

Method	Ochiai Top-			Dstar Top-			GP13 Top-			Naish1 Top-		
	1	3	5	1	3	5	1	3	5	1	3	5
Averaging	127	173	191	24	114	167	61	101	131	114	160	181
Maxing	23	106	162	26	109	165	24	107	165	24	107	165
Frequency	129	192	204	129	194	206	127	192	205	127	192	205

assigned the statements' suspiciousness are based on the maximum value of FOMs' suspiciousness and HOMs' suspiciousness. If there exists some bias in the Averaging method, the Combined method can alleviate this behaves and improves the fault localization effectiveness. While the Maxing and Frequency methods are based on the maximum statistics, so when combined, no significant improvement occurs.

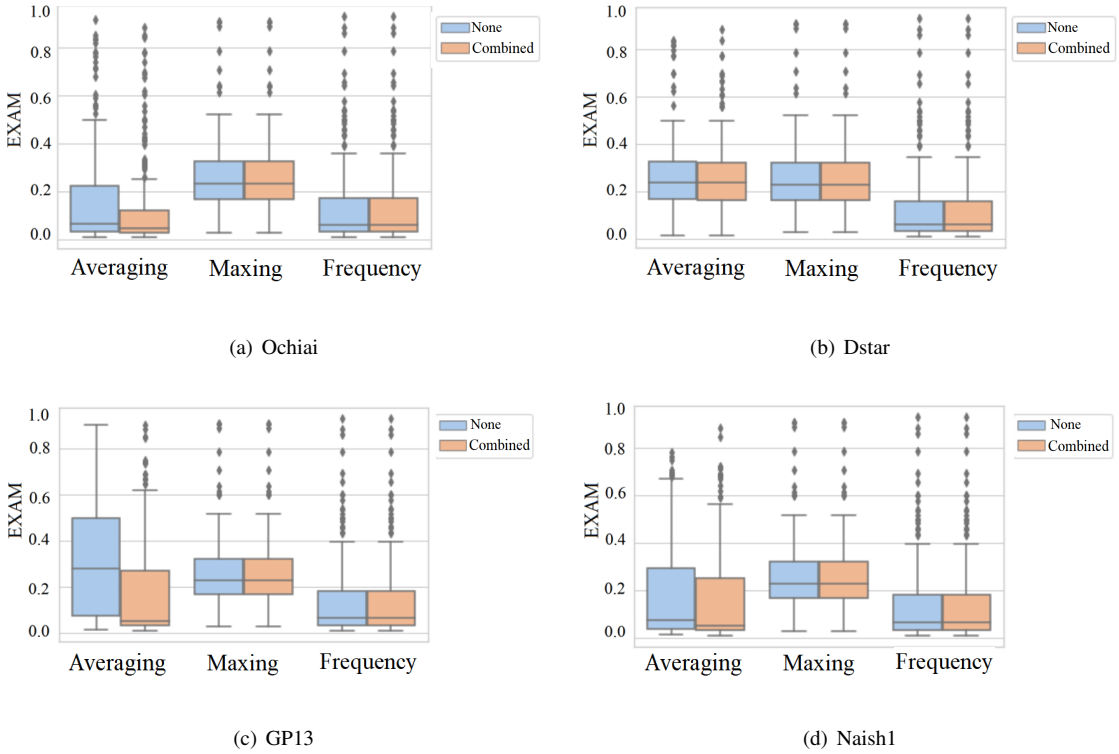
In summary, the Combined method improves the Averaging method and has no significant improvements in the Maxing and Frequency method.

*3) **RQ3 (The effectiveness of generation strategies)**:* To answer this question, we explore the performance of three HOMs generation strategies (i.e., Random, SBFL-guided, and MBFL-guided) at the metric of EXAM and Top-N.

In this question, we use the Frequency method to calculate the statements' suspiciousness based on the results of RQ1. Besides, we choose the $Ochiai$ formula to calculate the $weight$ for SBFL and MBFL techniques since the previous study showed that it performs better [49].

In terms of the metric EXAM, MBFL-guided strategy outperforms the other two generation strategies. Fig. 5 shows the performance of different generation strategies for HOMs. From four formulas in Fig. 5, the EXAM distribution of MBFL-guided strategy are closer to $X\text{-}axis$ that MBFL-guided outperforms Random and SBFL-guided. Besides, the SBFL-guided performs better than Random with lower 3rd quartile, while the Random strategy performs worst in these cases.

In terms of the metric Top-1, Top-3, Top-5, the MBFL-guided strategy again outperforms the strategy of Random and SBFL-guided. Table VI shows that, in four formulas, the MBFL-guided strategy can rank more faults at the top 1, top 3, and top 5 ranks. Moreover, the SBFL-guided is second best that locate more faults than the Random strategy. The results of



(a) Ochiai (b) Dstar

(c) GP13 (d) Naish1

Fig. 4. Comparison of the Combined method with three calculation methods with four formulas at EXAM

TABLE V
COMPARISON OF THE COMBINED METHOD WITH THREE CALCULATION METHODS WITH FOUR FORMULAS AT TOP-N

Method		Ochiai Top-			Dstar Top-			GP13 Top-			Naish1 Top-		
		1	3	5	1	3	5	1	3	5	1	3	5
Averaging	None	127	173	191	24	114	167	61	101	131	114	160	181
	Combined	175	189	195	32	112	166	161	175	188	161	174	188
Maxing	None	23	106	162	26	109	165	24	107	165	24	107	165
	Combined	23	106	162	26	109	165	24	107	165	24	107	165
Frequency	None	129	192	204	129	194	206	127	192	205	127	192	205
	Combined	129	192	204	129	194	206	127	192	205	127	192	205

MBFL-guided and SBFL-guided indicate that the generation strategy of HOMs for fault localization is effective.

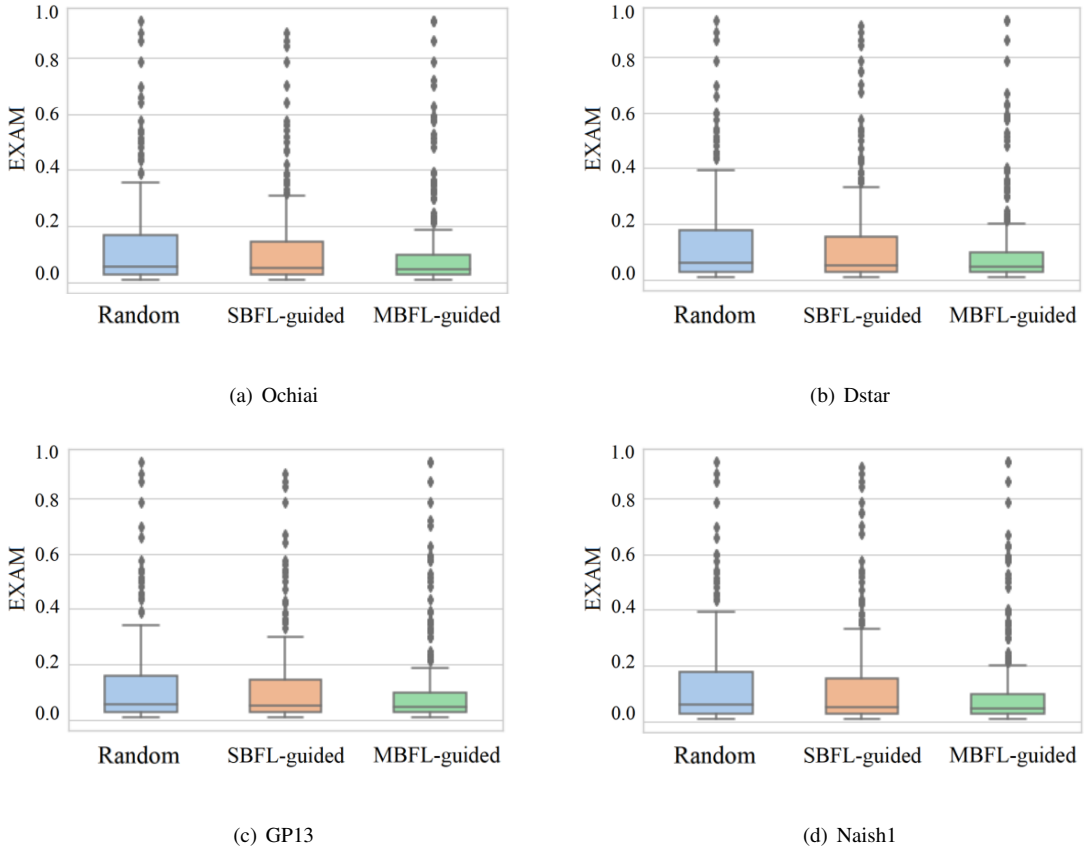
As mentioned in Section III-C, a statement with higher SBFL or MBFL suspiciousness are generated more associated HOMs. And in previous studies [8], MBFL techniques has more promising fault localization effectiveness than SBFL techniqus, which indicates that MBFL-guided generation strategy can produce more effective HOMs for fault localization.

In summary, MBFL-guided generation strategy outperforms SBFL-guided strategy and Random strategy in terms of the EXAM and Top-N metric.

4) ***RQ4 (Compared with other fault localization techniques)****:* Based on the results of RQ1, RQ2, and RQ3, we choose the MBFL-guided strategy to generate HOMs and the Frequency method to calculate statements' suspiciousness. We denote this approach as HMBFL, i.e., Higher-Order-Mutation-Based Fault Localization. We compare HMBFL with four SBFL techniques (use four SBFL formulas) and four traditional MBFL techniques (use four MBFL techniques) at the metric of EXAM and Top-N. For simplify, we denote the SBFL technique as SBFL and traditional MBFL technique as MBFL.

In terms of the metric EXAM, HMBFL outperforms SBFL techniques and traditional MBFL techniques. We display the EXAM via the violin plots shown in Fig. 6. In the violin plot, the $X\text{-}axis$ represents different techniques, while the $Y\text{-}axis$ indicates the EXAM. Each block in the violin plot indicates the distribution of EXAM metric and the corresponding formula. The breadth of the block represents the data density of the corresponding value of the $Y\text{-}axis$ for all multiple-fault program. So the wider in the bottom of the block and thinner in the up of the block indicates that the corresponding technique has a better fault localization effectiveness.

As shown in Fig. 6, the EXAM distribution of HMBFL are more gather around $X\text{-}axis$, more concentrated than SBFL techniques and traditional MBFL techniques, indicating



(a) Ochiai (b) Dstar

(c) GP13 (d) Naish1

Fig. 5. EXAM of two HOMs generation strategies with four formulas

TABLE VI
TOP-N OF TWO HOMS GENERATION STRATEGIES WITH FOUR FORMULAS

Method	Ochiai Top-			Dstar Top-			GP13 Top-			Naish1 Top-		
	1	3	5	1	3	5	1	3	5	1	3	5
Random	129	192	204	129	194	206	127	192	205	127	192	205
SBFL-guided	142	191	206	142	190	206	137	186	202	137	186	202
MBFL-guided	166	196	206	166	196	206	164	194	206	164	194	206

that the fault localization effectiveness of HMBFL is much better than these two kind of techniques. Moreover, traditional MBFL techniques perform better than SBFL techniques with more lower EXAM values.

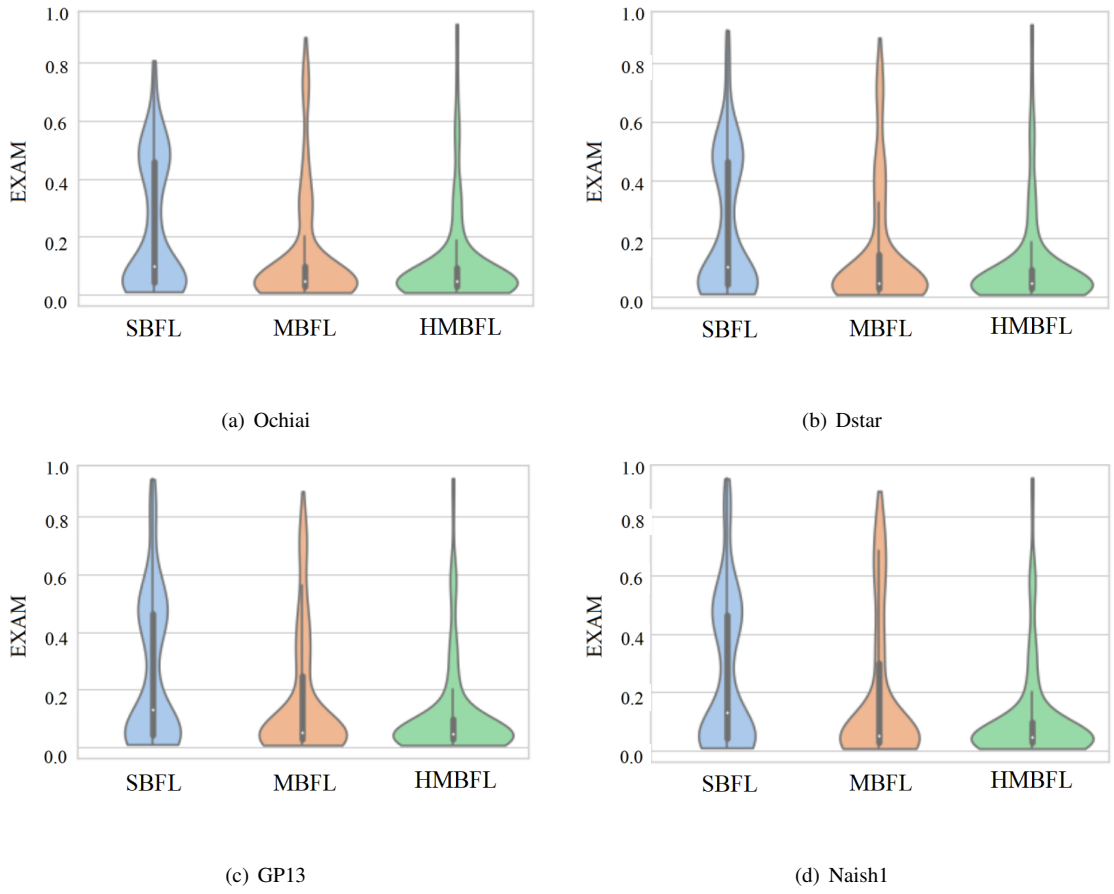
In terms of the metric Top-N, HMBFL again outperforms SBFL techniques and traditional MBFL techniques. Table VII shows that HMFL localize more faults with GP13 and Naish1 formulas than other two kind of techniques. In Ochiai and Dstar formulas, traditional MBFL technique can place more faults at the top 1 rank (173 and 169 faults especially), while HMBFL rank more faults in terms of Top-3 and Top-5. In all these cases, SBFL localize fewer faults compared to HMBFL and traditional MBFL techniques.

Traditional MBFL techniques treat the mutant (FOM) as a partial fix or a similar version of faulty programs that can achieve better performance on single-fault localization [50], while the real faults (multiple-fault) are more complex and FOMs are hard to fix these programs or have a large distance to the correct ones. However, HOMs (2-HOMs) mutates multiple lines that have a higher probability to fix these programs and are more similar to the real faults, which improves the multiple-fault localization effectiveness.

In summary, our proposed approach HMBFL, with the MBFL-guided generation strategy and the suspiciousness calculation method of Frequency, has the better fault localization effectiveness than SBFL techniques and traditional MBFL techniques at the metric of EXAM and Top-N.

D. Threats of Validity

1) ***Internal Validity****:* The first threat is the order of HOMs we used in this paper. Previous high order mutation testing have applied various orders [38], [41] (from 2-order to 10-order and more). We limited the 2-HOMs and did not consider the higher order mutants since Nguyen et al. [41] and Wong



(a) Ochiai (b) Dstar

(c) GP13 (d) Naish1

Fig. 6. Comparison of HMBFL with SBFL techniques and traditional MBFL techniques with four formulas at EXAM

TABLE VII
COMPARISON OF HMBFL WITH SBFL TECHNIQUES AND TRADITIONAL MBFL TECHNIQUES WITH FOUR FORMULAS AT EXAM

Method	Ochiai Top-			Dstar Top-			GP13 Top-			Naish1 Top-		
	1	3	5	1	3	5	1	3	5	1	3	5
SBFL	112	129	157	110	127	154	110	126	150	110	126	150
MBFL	173	185	193	169	180	192	158	173	187	158	169	184
HMBFL	166	196	206	166	196	206	164	194	206	164	194	206

et al. [42] have found that the lower order of mutants has more effective on mutation testing. Moreover, 2-HOMs are studies in various works [43], [44]. Our results are consistent with these studies that 2-HOMs are more effective for fault localization, and we will include higher order mutants in the future work.

The second threat is the number of HOMs we generated. We only maintain the number of HOMs ten times to FOMs since in our pre-experiment showed that it has a better fault localization effectiveness at this setting. In the future, we will enlarge the search space of HOMs for fault localization.

*2) **External Validity**:* One external validity of our experiment is the representatives of the subject programs we used. We consider the Codeflaws [20] as our benchmark that are all written in C language. Although these programs are real-world, these programs only represent limited classes of programs. Other programs in the experiment may produce different results. We will conduct our experiments on other languages in the future.

Another threat is the implementation correctness of MBFL. In our experiment, we implemented MBFL strictly based on the description of the original studies [8], [9] and the actual fault localization performance is very close to the results in these studies.

*3) **Construct Validity**:* In this experiment, we include four formulas (i.e., Ochiai, Dstar, GP13 and Naish1) as SBFL techniques and MBFL formulas. There exist other formulas and fault localization techniques not included. Different formulas and techniques may have different results. We will include more formulas and techniques to further validity our approach.

We use metrics of EXAM and Top-N to evaluate the fault localization effectiveness of a technique. EXAM is the

popular evaluating the performance of fault localization techniques [17] and Top-N (other studies [46], [51] refer to acc@n) is the metric using absolute ranks rather than percentages of program inspected [45], [46], [51]. 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 [46] and MAP [45]).

V. Related Work

A. Fault Localization Techniques

In recent years, fault localization techniques have been proposed for reducing the cost of software debugging, such as Machine-learning based [52], IR-based [1], [4], [53], Spectrum-based [5], Mutation-based [8], [16], and so on. Among these techniques, IR-based and Spectrum-based techniques are the commonly studied techniques by their lightweight and effective [4], [6]. Le et al. [54] proposed AML (Adaptive Multi-modal bug Localization) that utilizes the technique of IR-based and Spectrum-based. Their results show that AML performs better than the individual techniques. An Ngoc et al. [53] developed DNNLOC, combining deep neural network and information retrieval Moreover, Mutation-based techniques, MUSE and Metallaxis are two MBFL pioneer techniques. Both these two techniques are based on mutation analysis [15], which relies on the assumption that most of the mutations from "realistic" faults, even if artificially seeded [9]. Our proposed HMBFL is based on the MBFL techniques that first utilizes the HOMs on fault localization.

B. Higher Order Mutation Testing

Mutant testing is one of the research hotspots in recent years and has been applied in many fields, Demillo et al. [55] pioneered a constraint-based test case generation method, which is based on control flow analysis and conforming execution. The concept of HOMs was first introduced by Harman et al. [19]. For the application of higher-order mutation testing, Harman et al. [56] used HOMs to generate test data, and Gopinath et al. [57] used HOMs to analyze coupling effects. In terms of applying HOMs on fault localization, there are no published studies. To fill the gap in this area of research, we employ HOMs on fault localization and propose an effective approach in this paper.

VI. Conclusion

In this paper, we propose an approach HMBFL which uses HOMs for fault localization. To utilize HOMs on fault localization, we present three statements' suspiciousness calculation methods (i.e., Averaging, Maxing, and Frequency) and two generation strategies (i.e., SBFL-guided and MBFL-guided). We conduct our experiments on 112 real-world multiple-fault programs. The results show that: 1) The Frequency method performs better than the Averaging method and Maxing method; 2) The Combined method can improve the Averaging method but has no significant effect on Maxing and Frequency; 3) The MBFL-guided strategy outperforms Random and SBFL-guided strategy that can generate more effective HOMs for fault localization; 4) Our approach HMBFL, consist of Frequency method for statements' suspiciousness calculation and MBFL-guided strategy for HOMs generation, performs better than SBFL techniques and traditional MBFL techniques for multiple-fault localization. In the future, we first want to further investigate the methodology of HOMs that have better multiple-fault localization effectiveness. We second want to include other higher order of mutants (such as 3-HOMs and 4-HOMs) to extend our experimental studies.

Acknowledgment

The work is supported by the National Natural Science Foundation of China (Grant nos. 61902015, 61872026 and 61672085).

References

[1] S. K. Lukins, N. A. Kraft, and L. H. Etzkorn, "Bug localization using latent dirichlet allocation," *Information and Software Technology*, vol. 52, no. 9, pp. 972–990, 2010.

[2] S. Rao and A. Kak, "Retrieval from software libraries for bug localization: a comparative study of generic and composite text models," in *Proceedings of the 8th Working Conference on Mining Software Repositories*, 2011, pp. 43–52.

[3] J. Zhou, H. Zhang, and D. Lo, "Where should the bugs be fixed? more accurate information retrieval-based bug localization based on bug reports," in *2012 34th International Conference on Software Engineering (ICSE)*. IEEE, 2012, pp. 14–24.

[4] R. K. Saha, M. Lease, S. Khurshid, and D. E. Perry, "Improving bug localization using structured information retrieval," in *2013 28th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2013, pp. 345–355.

[5] V. Debroy and W. E. Wong, "Insights on fault interference for programs with multiple bugs," in *2009 20th International Symposium on Software Reliability Engineering*. IEEE, 2009, pp. 165–174.

[6] S. Yoo, "Evolving human competitive spectra-based fault localisation techniques," in *International Symposium on Search Based Software Engineering*. Springer, 2012, pp. 244–258.

[7] W. E. Wong, V. Debroy, R. Gao, and Y. Li, "The dstar method for effective software fault localization," *IEEE Transactions on Reliability*, vol. 63, no. 1, pp. 290–308, 2013.

[8] M. Papadakis and Y. Le Traon, "Using mutants to locate" unknown" faults," in *2012 IEEE Fifth International Conference on Software Testing, Verification and Validation*. IEEE, 2012, pp. 691–700.

[9] ——, "Metallaxis-fl: mutation-based fault localization," *Software Testing, Verification and Reliability*, vol. 25, no. 5-7, pp. 605–628, 2015.

[10] Z. Li, H. Wang, and Y. Liu, "Hmer: A hybrid mutation execution reduction approach for mutation-based fault localization," *Journal of Systems and Software*, p. 110661, 2020.

[11] Q. Wang, C. Parnin, and A. Orso, "Evaluating the usefulness of ir-based fault localization techniques," in *Proceedings of the 2015 International Symposium on Software Testing and Analysis*, 2015, pp. 1–11.

[12] T. Reps, T. Ball, M. Das, and J. Larus, "The use of program profiling for software maintenance with applications to the year 2000 problem," in *Software Engineering—Esec/Fse'97*. Springer, 1997, pp. 432–449.

[13] X. Xue and A. S. Namin, "How significant is the effect of fault interactions on coverage-based fault localizations?" in *2013 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*. IEEE, 2013, pp. 113–122.

[14] 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," *Journal of Systems and Software*, vol. 151, pp. 20–37, 2019.

[15] 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.

[16] S. Moon, Y. Kim, M. Kim, and S. Yoo, "Ask the mutants: Mutating faulty programs for fault localization," in *2014 IEEE Seventh International Conference on Software Testing, Verification and Validation*. IEEE, 2014, pp. 153–162.

[17] D. Zou, J. Liang, Y. Xiong, M. D. Ernst, and L. Zhang, "An empirical study of fault localization families and their combinations," *IEEE Transactions on Software Engineering*, 2019.

[18] V. Debroy and W. E. Wong, "Combining mutation and fault localization for automated program debugging," *Journal of Systems and Software*, vol. 90, pp. 45–60, 2014.

[19] M. Harman, Y. Jia, and W. B. Langdon, "A manifesto for higher order mutation testing," in *2010 Third International Conference on Software Testing, Verification, and Validation Workshops*. IEEE, 2010, pp. 80–89.

[20] S. H. Tan, J. Yi, S. Mechtaev, A. Roychoudhury *et al.*, "Codeflaws: a programming competition benchmark for evaluating automated program repair tools," in *2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C)*. IEEE, 2017, pp. 180–182.

[21] A. Zakari, S. Abdullahi, N. M. Shagari, A. B. Tambawal, N. M. Shanono, J. Z. Maitama, R. A. Rasheed, A. Adamu, and S. M. Abdulrahman, "Spectrum-based fault localization techniques application on multiple-fault programs: A review," *Global Journal of Computer Science and Technology*, 2020.

[22] N. Aribi, N. Lazaar, Y. Lebbah, S. Loudni, and M. Maamar, "A multiple fault localization approach based on multicriteria analytical hierarchy process," in *2019 IEEE International Conference On Artificial Intelligence Testing (AITest)*. IEEE, 2019, pp. 1–8.

[23] Z. Li, X. Bai, H. Wang, and Y. Liu, "Irbfl: An information retrieval based fault localization approach," in *2020 IEEE 44th Annual Computers, Software, and Applications Conference (COMPSAC)*. IEEE, 2020, pp. 991–996.

[24] J. A. Jones, M. J. Harrold, and J. Stasko, "Visualization of test information to assist fault localization," in *Proceedings of the 24th International Conference on Software Engineering. ICSE 2002*. IEEE, 2002, pp. 467–477.

[25] J. A. Jones, J. F. Bowring, and M. J. Harrold, "Debugging in parallel," in *Proceedings of the 2007 international symposium on Software testing and analysis*, 2007, pp. 16–26.

[26] R. Gao and W. E. Wong, "Mseer—an advanced technique for locating multiple bugs in parallel," *IEEE Transactions on Software Engineering*, vol. 45, no. 3, pp. 301–318, 2017.

[27] R. Abreu, P. Zoeteweij, and A. J. Van Gemund, "Spectrum-based multiple fault localization," in *2009 IEEE/ACM International Conference on Automated Software Engineering*. IEEE, 2009, pp. 88–99.

[28] ——, "Simultaneous debugging of software faults," *Journal of Systems and Software*, vol. 84, no. 4, pp. 573–586, 2011.

[29] 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. 124 297–124 310, 2020.

[30] 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," Citeseer, Tech. Rep., 1989.

[31] L. Naish, H. J. Lee, and K. Ramamohanarao, "A model for spectra-based software diagnosis," *ACM Transactions on software engineering and methodology (TOSEM)*, vol. 20, no. 3, pp. 1–32, 2011.

[32] M. Y. Chen, E. Kiciman, E. Fratkin, A. Fox, and E. Brewer, "Pinpoint: Problem determination in large, dynamic internet services," in *Proceedings International Conference on Dependable Systems and Networks*. IEEE, 2002, pp. 595–604.

[33] R. Abreu, P. Zoeteweij, and A. J. Van Gemund, "An evaluation of similarity coefficients for software fault localization," in *2006 12th Pacific Rim International Symposium on Dependable Computing (PRDC'06)*. IEEE, 2006, pp. 39–46.

[34] R. Abreu, W. Mayer, M. Stumptner, and A. J. van Gemund, "Refining spectrum-based fault localization rankings," in *Proceedings of the 2009 ACM symposium on Applied Computing*, 2009, pp. 409–414.

[35] T.-D. B. Le, F. Thung, and D. Lo, "Theory and practice, do they match? a case with spectrum-based fault localization," in *2013 IEEE International Conference on Software Maintenance*. IEEE, 2013, pp. 380–383.

[36] 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 and Microsystems*, vol. 50, pp. 102–112, 2017.

[37] N. DiGiuseppe and J. A. Jones, "On the influence of multiple faults on coverage-based fault localization," in *Proceedings of the 2011 international symposium on software testing and analysis*, 2011, pp. 210–220.

[38] A. J. Offutt, "Investigations of the software testing coupling effect," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 1, no. 1, pp. 5–20, 1992.

[39] S. Yoo, X. Xie, F.-C. Kuo, T. Y. Chen, and M. Harman, "Human competitiveness of genetic programming in spectrum-based fault localisation: Theoretical and empirical analysis," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 26, no. 1, pp. 1–30, 2017.

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

[41] Q.-V. Nguyen *et al.*, "Is higher order mutant harder to kill than first order mutant? an experimental study," in *Asian Conference on Intelligent Information and Database Systems*. Springer, 2018, pp. 664–673.

[42] W. E. Wong and A. P. Mathur, "Reducing the cost of mutation testing: An empirical study," *Journal of Systems and Software*, vol. 31, no. 3, pp. 185–196, 1995.

[43] P. G. Frankl, S. N. Weiss, and C. Hu, "All-uses vs mutation testing: an experimental comparison of effectiveness," *Journal of Systems and Software*, vol. 38, no. 3, pp. 235–253, 1997.

[44] M. Papadakis and N. Malevris, "An empirical evaluation of the first and second order mutation testing strategies," in *2010 Third International Conference on Software Testing, Verification, and Validation Workshops*. IEEE, 2010, pp. 90–99.

[45] X. Li, W. Li, Y. Zhang, and L. Zhang, "Deepfl: Integrating multiple fault diagnosis dimensions for deep fault localization," in *Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis*, 2019, pp. 169–180.

[46] T.-D. B Le, D. Lo, C. Le Goues, and L. Grunske, "A learning-to-rank based fault localization approach using likely invariants," in *Proceedings of the 25th International Symposium on Software Testing and Analysis*. ACM, 2016, pp. 177–188.

[47] P. S. Kochhar, X. Xia, D. Lo, and S. Li, "Practitioners' expectations on automated fault localization," in *Proceedings of the 25th International Symposium on Software Testing and Analysis*, 2016, pp. 165–176.

[48] J. Sohn and S. Yoo, "Fluccs: Using code and change metrics to improve fault localization," in *Proceedings of the 26th ACM SIGSOFT International Symposium on Software Testing and Analysis*. ACM, 2017, pp. 273–283.

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

[50] D. Shin and D.-H. Bae, "A theoretical framework for understanding mutation-based testing methods," in *2016 IEEE International Conference on Software Testing, Verification and Validation (ICST)*. IEEE, 2016, pp. 299–308.

[51] Y. Kim, S. Mun, S. Yoo, and M. Kim, "Precise learn-to-rank fault localization using dynamic and static features of target programs," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 28, no. 4, pp. 1–34, 2019.

[52] W. E. Wong, Y. Shi, Y. Qi, and R. Golden, "Using an rbf neural network to locate program bugs," in *2008 19th International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 2008, pp. 27–36.

[53] A. N. Lam, A. T. Nguyen, H. A. Nguyen, and T. N. Nguyen, "Bug localization with combination of deep learning and information retrieval," in *2017 IEEE/ACM 25th International Conference on Program Comprehension (ICPC)*. IEEE, 2017, pp. 218–229.

[54] T.-D. B. Le, R. J. Oentaryo, and D. Lo, "Information retrieval and spectrum based bug localization: Better together," in *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, 2015, pp. 579–590.

[55] R. A. DeMillo, A. J. Offutt *et al.*, "Constraint-based automatic test data generation," *IEEE Transactions on Software Engineering*, vol. 17, no. 9, pp. 900–910, 1991.

[56] M. Harman, Y. Jia, P. Reales Mateo, and M. Polo, "Angels and monsters: An empirical investigation of potential test effectiveness and efficiency improvement from strongly subsuming higher order mutation," in *Proceedings of the 29th ACM/IEEE international conference on Automated software engineering*, 2014, pp. 397–408.

[57] R. Gopinath, C. Jensen, and A. Groce, "The theory of composite faults," in *2017 IEEE International Conference on Software Testing, Verification and Validation (ICST)*. IEEE, 2017, pp. 47–57.