2023

SGS: Mutant Reduction for Higher-order Mutation-based Fault Localization

Luxi Fan
Beijing University of Chemical Technology, China

Zheng Li
Beijing University of Chemical Technology, Beijing, China

Hengyuan Liu
Beijing University of Chemical Technology, China

See next page for additional authors

Funder: National Natural Science Foundation of China

Authors

Luxi Fan, Zheng Li, Hengyuan Liu, Paul Doyle, Haifeng Wang, Xiang Chen, and Yong Liu

2023 IEEE 47th Annual Computers, Software, and Applications Conference (COMPSAC) | 979-8-3503-2697-0/23/$31.00 ©2023 IEEE | DOI: 10.1109/COMPSAC57700.2023.00116

SGS: Mutant Reduction for Higher-order Mutation-based Fault Localization

Luxi Fan[†], Zheng Li[†*], Hengyuan Liu[†*], Doyle Paul[§], Haifeng Wang[¶], Xiang Chen[‡], Yong Liu[†]
[†]*College of Information Science and Technology, Beijing University of Chemical Technology,* Beijing 100029, China
[‡]*School of Information Science and Technology, Nantong University,* Nantong 226019, China
[§]*School of Computer Science, Technological University Dublin,* Dublin, Ireland
[¶]*Center for Advanced Metering Infrastructure, National Institute of Metrology,* Beijing 100029, China
Email: lizheng@mail.buct.edu.cn

***Abstract*—MBFL (Mutation-Based Fault Localization) is one of the most commonly studied fault localization techniques due to its promising fault localization effectiveness. However, MBFL incurs a high execution cost as it needs to execute the test suite on a large number of mutants. While previous studies have proposed mutant reduction methods for FOMs (First-Order Mutants) to help alleviate the cost of MBFL, the reduction of HOMs (Higher-Order Mutants) has not been thoroughly investigated. In this study, we propose SGS (Statement Granularity Sampling), a method which conducts HOMs reduction for HMBFL (Higher-Order Mutation-Based Fault Localization). Considering the relationship between HOMs and statements, we sample HOMs at the statement level to ensure each statement has corresponding HOMs. We empirically evaluate the fault localization effectiveness of HMBFL using SGS on 237 multiple-fault programs taken from the SIR and Codeflaws benchmarks. The experimental results show that (1) The best sampling ratio for HMBFL with SGS is 20%, which preserves the performance and reduces execution costs by 80% ; (2) The fault localization accuracy of HMBFL with SGS outperforms the state-of-the-art SBFL (Spectrum-Based Fault Localization) and MBFL techniques by 20%.**

***Index Terms*—Mutation-based fault localization, Multiple faults, Higher-order-mutants, Mutant reduction**

I. Introduction

Fault localization is essential for identifying faulty program elements [1] and is a time-consuming debugging activity. With larger software projects, various automatic fault localization techniques have been proposed, such as information retrieval-based [2], slice-based [3], machine learning-based [4], spectrum-based [5], and mutation-based strategies [6]. These aim to reduce the human effort required for fault localization.

Mutation-Based Fault Localization (MBFL) techniques have been shown to outperform Spectrum-Based Fault Localization (SBFL) techniques [7]. The MBFL approach utilizes mutant testing, whereby the mutants can be either First-Order-Mutants (FOMs) or Higher-Order-Mutants (HOMs) [8]. Most MBFL studies focus on FOMs, which perform poorly on multiple-fault programs. HOMs can more closely reflect multiple faults [9], so Higher-Order Mutation-Based Fault Localization (HMBFL) can detect faults that MBFL cannot.

Previous research [10] has shown that fault localization techniques based on higher-order mutation are more effective at localization on multiple-fault programs. However, MBFL and HMBFL both suffer from significant computational costs since both require the execution of a large number of mutants against the test suite.

Existing MBFL reduction methods can achieve significant cost reductions on single-fault programs but are unsuitable for reducing HOMs when localizing multiple-fault programs. Xue et al. [11] discovered that individual faults in multiple-fault programs interfere with one another. Therefore, we study the reduction method for the HOMs technique to reduce the execution cost of HMBFL without decreasing accuracy.

In this study, we analyze the reduction in the cost of the fault localization technique using HOMs, while considering their interrelated characteristics with multiple faults. HOMs are collections of FOMs, each corresponding to a program statement. We propose a Statement Granularity Sampling (SGS) method that considers the relationship between HOMs and statements, classifying and sampling HOMs to ensure representation of each statement.

In our experiment, we use 237 programs from SIR [12] and Codeflaws [13] as subjects, generating 2nd order HOMs and FOMs using 15 mutation operators [13]. We find a 20% sampling rate (SGS-20%) to be the most effective and efficient. Comparing HMBFL using SGS-20% to three SBFL and three MBFL techniques, it demonstrates greater fault localization effectiveness while reducing mutation execution costs by 80.0%.

We summarize the main contributions of our study as follows:

- We conduct a detailed theoretical analysis of the relationship between HOMs and statements, determining that a HOM corresponds to several statements and a statement can generate multiple HOMs.
- We propose the SGS mutation reduction strategy for high-order mutant reduction, ensuring each statement has corresponding HOMs.
- We evaluate the effectiveness of the SGS strategy on 237 multiple-fault programs, demonstrating reduced execution cost and preserved fault localization performance.
- To facilitate future study, we share the source code and dataset of our study in a Github repository[1].

[1]https://github.com/lucyVan/SGS

979-8-3503-2697-0/23/$31.00 ©2023 IEEE
DOI 10.1109/COMPSAC57700.2023.00116

II. Background

A. Mutation-Based Fault Localization

Mutation-based fault localization is a technique based on mutation analysis. The MBFL technique generates mutants using mutation operators and executes all mutants against the test suite to obtain information about the execution results. Then, the MBFL techniques calculate the suspiciousness of mutants and program statements based on the collected information and ultimately localize the fault.

If a test case execution behavior of a mutant is different from the original, we say that the mutant is $killed$ or $detected$. Otherwise, we say that the mutant is $notkilled$ or $live$. The MBFL technique first executes a program P by a test suite 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. Then, all mutants 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 killed by T and T_n is the set of mutants not killed by T. Subsequently, 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, a_{kp} denotes the number of pass tests that killed, a_{nf} denotes the number of failed tests not killed, and a_{kf} denotes the number of failed tests killed. Table I lists three popular MBFL formulas.

TABLE I
Suspiciousness formulas for MBFL

Name	Formula
Ochiai	$Sus(m) = \frac{a_{kf}}{\sqrt{(a_{kf}+a_{nf})(a_{kf}+a_{kp})}}$
Dstar	$Sus(m) = \frac{a_{kf}^2}{a_{kp}+a_{nf}}$
GP13	$Sus(m) = a_{kf}\left(1 + \frac{1}{2a_{kp}+a_{kf}}\right)$

The MBFL technique considers the execution difference between faulty programs and correct programs. Many studies [6] have demonstrated that the MBFL technique has the potential to outperform other types of fault localization techniques significantly.

B. Mutant Reduction Methods

In recent years, mutant reduction approaches have been applied to different software engineering tasks such as fault localization and program repair. Offutt et al. [14] determined that the mutation operator was the core of mutation and proposed the SELECTIVE method. However, the SELECTIVE method selects only limited mutation operators, making it impossible to generate specific types of mutants, which in turn results in poor fault localization accuracy [15].

Some mutant reduction methods sample a smaller set of mutants. Papadakis et al. [6] sampling 10-50% of mutants demonstrate that 10% sampling outperforms SBFL in localization, indicating that mutant reduction effectively lowers the computational cost of MBFL techniques.

This issue of how to reduce HOMs has not been thoroughly investigated in previous studies, so in this study we propose a practical approach to address this research gap.

III. Our Method

A. Relationship Between Higher-Order Mutants and Statements

Given a program P, the statement set in the program is $S = \{s_1, s_2, \cdots, s_n\}$, where s_i is the ith line of code in the program. OP is the set of mutation operators. By applying all the mutation operators in OP to each statement in S, we obtain the set of FOMs of program P, denoted as $\text{FOMs}(S) = \bigcup_{i=1}^{n} \text{FOMs}(s_i)$, where $\text{FOMs}(s_i)$ is the set of FOMs with mutation positions in the i-th line of code in the program. The HOMs of the program are composed of FOMs. For the purpose of clarity, we denote the set of kth-order mutants of the program as $k\text{-HOMs}(S^k)$, where $S^k = S \times S \times \cdots \times S$ represents the kth Cartesian product of the set of program statements. Specifically, we have the set of kth-order mutants related to statement s_i, denoted as $k\text{-HOMs}(\{s_i\} \times S^{k-1})$ and abbreviated as $k\text{-HOMs}(s_i \times S^{k-1})$. Clearly, given a kth-order mutant $k\text{-HOM}(\vec{s}) \in k\text{-HOMs}(\vec{s}) \subset k\text{-HOMs}(S^k)$, it can be mapped to a vector of statements $\vec{s} \in S^k$. Similarly, given a statement vector $\vec{s} \in S^k$, we can map it to a set of kth-order mutants $k\text{-HOMs}(\vec{s}) \subset k\text{-HOMs}(S^k)$. Based on the above analysis, we can conclude that there is a many-to-many relationship between the HOMs of a program and the program statements.

B. Higher-Order Mutant Reduction Method Based on Statement Granularity Sampling

Sampling HOMs for multiple-fault programs requires considering the relationship between HOMs and program statements. We propose a HOM reduction method based on Statement Granularity Sampling (SGS) by sampling HOMs associated with a statement. Sampling HOMs at the statement granularity level ensures that each statement has a suspiciousness value.

The SGS method generates HOMs by mutating the program using mutation operators and classifying the generated mutants at the statement level. Fig. 1 shows the framework of the SGS method. First, we generate $\text{FOMs}(S)$ by applying mutation operators OP on the program's statements S. Then by repeatedly combining k mutants in $\text{FOMs}(S)$, we can get kth-order mutants $k\text{-HOMs}(S^k)$. According to the relationship between HOMs and statements, we divided $k\text{-HOMs}(S^k)$ into n set of mutants, each set per code statement (e.g. $k\text{-HOMs}(s_i \times S^{k-1})$ is the set of HOMs related to code statement s_i). By randomly selecting $x\%$ mutants from $k\text{-HOMs}(s_i \times S^{k-1})$, we can get a subset $k\text{-HOMs}_i^{'}$ of $k\text{-HOMs}(s_i \times S^{k-1})$. Finally, the reduced HOM set ($k\text{-HOMs}^{'}$) is the union of each reduced subset of HOMs ($k\text{-HOMs}_i^{'}$).

SGS classifies HOMs corresponding to the same statement, then samples to ensure each statement has corresponding

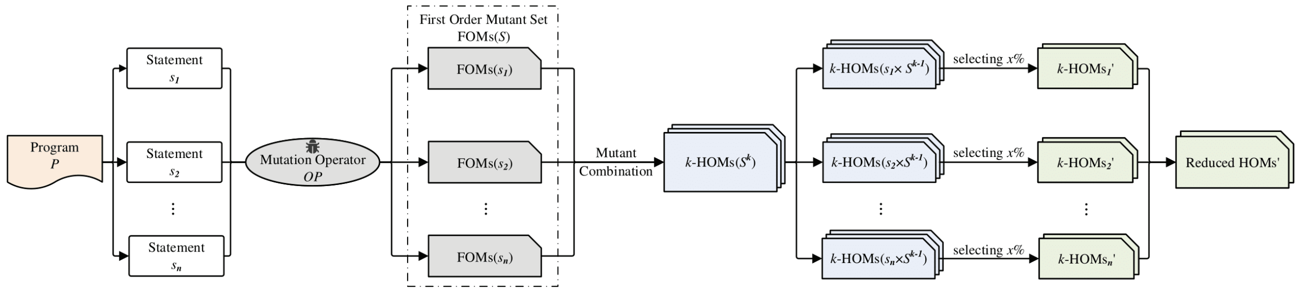


Fig. 1. The framework of SGS.

HOMs. The sampling of HOMs at the statement granularity level ensures that each statement has a suspiciousness value.

The SGS method is comparable to the conventional MBFL reduction technique, with the following advantages:

- Sampling the complete set of mutation operators to avoid missing essential mutation operators.
- Sampling the generated HOMs from the granularity level of statements prevents some statements from being incapable of calculating a suspiciousness value.
- Sampling the generated HOMs of statements equally maintains the distribution of the extracted HOMs and avoids statement suspiciousness bias caused by distribution difference.

IV. Experimental Design

A. Benchmark

Subject programs from two benchmark suites are used to measure the effectiveness of our proposed method. Table II presents statistics for all subject programs.

TABLE II
Statistics of subject programs

Benchmark	Program	#Versions	#LOC	#Test Cases	#FOMs	#2-HOMs
SIR	printtokens	20	563	4,130	21,705	21,161
	printtokens2	20	510	4,115	47,215	48,699
	tcas	20	173	1,608	13,317	15,180
	sed	20	7,125	360	59,571	60,247
Codeflaws		157	51	58	20,631	24,092
Total		237	-	-	162,439	169,379

SIR (Software-artifact Infrastructure Repository) [12] is a repository of open-source programs for program analysis and software testing. We selected four programs: three small-scale (printtoken, printtoken2, tcas) and one large-scale (sed) for their comprehensive test suites and use in previous fault localization studies [16]. We formed 60 multiple-fault programs from single-fault SIR programs and 20 versions of the large-scale program.

Codeflaws [13] is a benchmark collection containing real faults from Codeforces[2]. We selected 157 multiple-fault programs, which have been widely used in prior fault localization studies [16].

[2]https://codeforces.com/

B. Experimental setup

We use 15 types of C mutation operators (see Table III) provided by Agrawal et al. [13] for a total of 199 mutation operators.

TABLE III
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
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>

Compared to traditional MBFL, HOM execution amounts are similar to FOMs, ensuring equivalent execution costs. In our initial experiment, we adapted three SBFL formulas (Dstar, GP13, Ochiai) into MBFL formulas to eliminate formula effects on results. Since results were comparable, we report only Dstar. The mutant generation strategy and statement suspiciousness measure follow Li et al. [17].

For the experiments, 2-order HOMs were generated for the following reasons: (1) Nguyen et al. [18] discovered that lower-order HOMs had better mutation testing results. (2) Wong et al. [19] also discovered that HOMs of lower order could detect program faults. (3) 2-HOM can effectively reduce the number of equivalent mutants in mutation testing [10]. (4) 2-HOM has been widely used in previous mutation testing and fault localization studies [19].

C. Evaluation Metrics

The performance of our proposed method was evaluated in terms of the following five evaluation metrics.

(1) $EXAM$ [20] is used to determine the percentage of program statements that need to be manually checked to find the faulty statement.

(2) $Top\text{-}N$ [21] represents the number of faults discovered in the top N most suspicious statements. Based on the previous studies [4], we set N to 1, 3, and 5 for comparison purposes.

(3) MAP ($Mean\ Average\ Precision$) [21] is the average position of all fault statements in the ranking list. The higher the value of MAP, the better the performance of the corresponding technique.

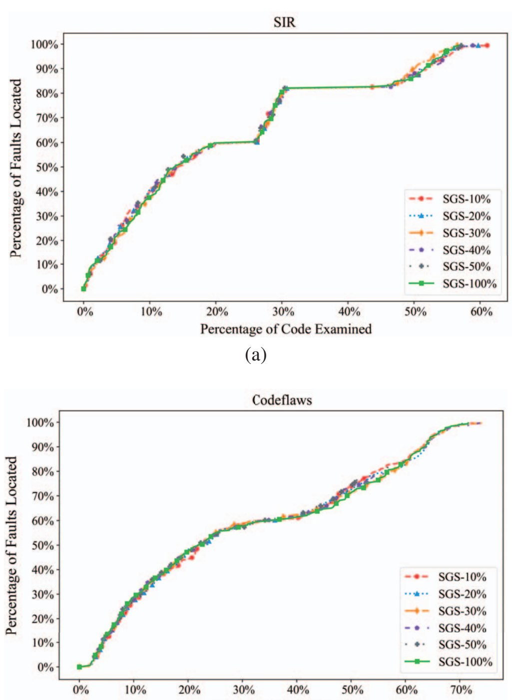
(4) Wilcoxon signed-rank test [22] is a non-parametric method for determining whether the difference in localization results is statistically significant.

(5) MTP ($Mutant\text{-}Test\text{-}Pair$) [23] is used to quantify the mutant execution cost of MBFL techniques. A smaller MTP value for an MBFL technique indicates a lower execution cost and higher level of efficiency.

V. Results Analysis

A. RQ1: How does the effectiveness of HMBFL with the SGS method at different sampling ratios compare?

In RQ1, we evaluate SGS's fault localization effectiveness at various sampling ratios (10%-100%) using $EXAM$, $Top\text{-}N$ and MAP metrics.



(b)

Fig. 2. The fault localization effectiveness of HMBFL with the SGS method with different sampling ratios

In terms of $EXAM$, different SGS sampling ratios show insignificant differences. As shown in Fig. 2(a), HMBFL with the SGS method with different sampling ratios can examine the same proportion of defects as HMBFL with the unsampled (SGS-100%) method. On Codeflaws (Fig. 2(b)), , SGS-10% detects 44.4% of faults examining 20% of codes, while SGS-100% detects 48.1%, showing a decline in effectiveness at higher ratios.

TABLE IV
THE TOP-N AND MAP OF HMBFL WITH THE SGS METHOD WITH DIFFERENT SAMPLING RATIOS

Benchmark	Sampling Ratio	Top-1	Top-3	Top-5	MAP
SIR	10%	0	1	3	0.0834
	20%	0	8	12	0.1042
	30%	0	**11**	20	0.0951
	40%	0	10	20	0.1036
	50%	0	10	18	0.1059
	100%	0	10	**41**	**0.1217**
Codeflaws	10%	45	96	130	0.5673
	20%	46	97	130	0.5682
	30%	45	100	131	0.5699
	40%	47	102	131	0.5781
	50%	**48**	102	**132**	**0.5811**
	100%	**48**	**104**	130	0.5803

In terms of $Top\text{-}N$, MAP, SGS localizes more faults in top 1, 3, and 5 as sampling ratio increases. However, the SGS-20% method is practical when the fault localization effectiveness and mutation execution cost are considered. Table IV shows the $Top\text{-}N$ and MAP of the SGS method with different sampling ratios for two benchmarks. The result indicates that the SGS-20% method can remove a higher percentage of mutants while maintaining fault localization effectiveness.

TABLE V
THE P-VALUES OF THE SGS METHOD WITH DIFFERENT SAMPLING RATIOS

Benchmark	Sampling Ratio	p-value
SIR	10%	0.0548
	20%	0.0537
	30%	0.0435
	40%	0.2174
	50%	0.1689
Codeflaws	10%	0.8566
	20%	0.8702
	30%	0.6498
	40%	0.4265
	50%	0.4548

Table V details the fault localization performance differences for SGS with various sampling ratios. The p-values are often larger than 0.05, indicating no statistically significant differences. Importantly, the SGS-20% results are not significantly different from SGS-100%, confirming statistical significance.

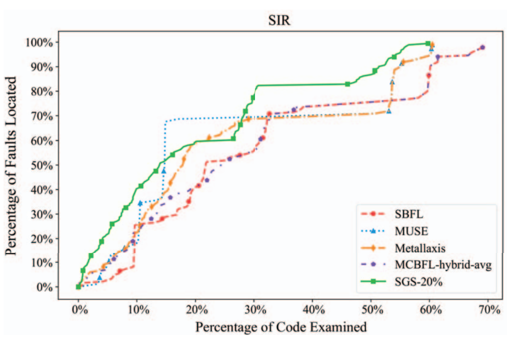
In terms of MTP, the SGS-20% method substantially reduces the mutation execution cost by around 80%. As the sampling ratio grows, so does the related mutation cost. On SIR, the MTP for the SGS-20% method reduces the HOMs execution cost by 80.4%. On Codeflaws, the SGS-20% method can reduce the execution cost by approximately 79.7%.

Summary for RQ1: The effectiveness of HMBFL with the SGS-20% method is statistically comparable to that of unsampled while reducing the execution cost by about 80%. In terms of the $Top\text{-}N$ and MAP metrics, HMBFL with the SGS-20% method provides a higher fault localization effectiveness.

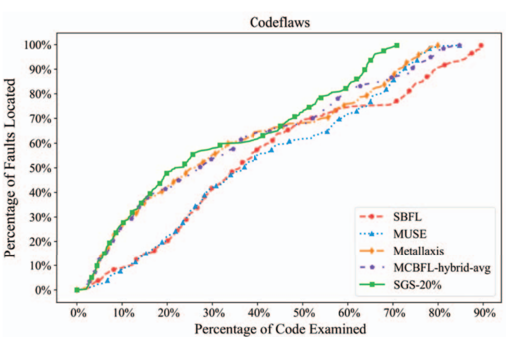
B. RQ2: How does the effectiveness and efficiency of HMBFL with the SGS-20% method compare to SBFL and MBFL techniques?

In RQ2, we compare the effectiveness and efficiency of HMBFL with the SGS-20% method to that of traditional fault localization techniques (i.e., SBFL and MBFL). We select a SBFL (Dstar) and three MBFL techniques (MUSE, Metallaxis, and MCBFL-hybrid-avg) as the baselines and choose $EXAM$, $Top\text{-}N$, MAP, and MTP as the evaluation metrics.

As shown in Fig. 3, in terms of $EXAM$, HMBFL with the SGS-20% method localizes more faults than SBFL and MBFL techniques while examining the same amount of code in most cases.



(a)



(b)

Fig. 3. The fault localization effectiveness of different fault localization techniques

In terms of $Top\text{-}N$ and MAP, HMBFL with the SGS-20% method ranks more faults in the top 1,3,5 and has more accurate fault localization results than the SBFL and MBFL techniques. Table VI shows the $Top\text{-}N$ and MAP of HMBFL with the SGS-20% , SBFL, and MBFL techniques. SBFL and MUSE techniques have a $Top\text{-}N$ of 0 on SIR using the Dstar formula, while Metallaxis and MCBFL-hybrid-avg

TABLE VI
THE TOP-N AND MAP OF DIFFERENT FAULT LOCALIZATION TECHNIQUES

Benchmark	**Method**	**Top**			**MAP**
		1	**3**	**5**	
SIR	SBFL	0	0	0	0.0425
	MUSE	0	0	0	0.0494
	Metallaxis	0	5	**15**	0.0692
	MCBFL-hybrid-avg	0	5	**15**	0.0726
	SGS-20%	0	**8**	12	**0.1042**
Codeflaws	SBFL	11	35	69	0.3176
	MUSE	8	34	69	0.2992
	Metallaxis	33	91	120	0.5295
	MCBFL-hybrid-avg	35	91	120	0.5309
	SGS-20%	**46**	**97**	**130**	**0.5682**

techniques rank 5 and 15 faults in the top 3 and 5, and HMBFL with the SGS-20% method has a maximum Top-3 of 8 and a highest MAP (0.1217). On Codeflaws, HMBFL with the SGS-20% method has the highest Top-1 (46), Top-3 (97), Top-5 (130), and MAP (0.5682), although the MCBFL-hybrid-avg technique can perform better than all other fault localization techniques.

TABLE VII
THE P-VALUES OF DIFFERENT FAULT LOCALIZATION TECHNIQUES

Benchmark	Method	p-value
SIR	SBFL	2.70E-06
	MUSE	0.0782
	Metallaxis	0.0127
	MCBFL-hybrid-avg	3.10E-05
Codeflaws	SBFL	3.30E-08
	MUSE	6.80E-23
	Metallaxis	8.60E-08
	MCBFL-hybrid-avg	0.1887

Table VII presents p-values of the Wilcoxon signed-rank test comparing HMBFL with SGS-20% to other fault localization techniques. Most p-values are below 0.05, indicating significant differences in fault localization.

In terms of MTP, SGS-20% reduces cost by 80.0% compared to MBFL. On SIR, SGS-20% performed 65,358,134 times while MBFL required 326,790,674. SGS-20% reduces execution cost on Codeflaws by 78.8%. Overall, SGS-20% reduces mutation execution cost by approximately 79.9%.

Summary for RQ2: The results demonstrate that HMBFL with the SGS-20% method can effectively minimize mutation execution overhead, while its fault localization effectiveness is superior to SBFL and MBFL.

VI. THREATS TO VALIDITY

Internal Validity. The first internal threat to our experiment is the mutant generation and sampling randomness. Different mutant sets and mutant samples will affect the fault localization result. The second internal threat is the order of HOMs used in our study. Previous studies indicates that higher-order mutants aren't always effective in mutation analysis, and 2-HOMs are commonly used in previous studies. Therefore, we select 2-HOMs in this study and will consider HOMs in the future.

Construct Validity. The formulas used in the SBFL and MBFL techniques are the first construct validity threat. Different formulas may affect the experimental results, so we choose three formulas for our experiment. The second construct validity threat is the evaluation metrics. We evaluate the effectiveness of fault localization using $EXAM$, $Top\text{-}N$, and MAP. Moreover, we used Wilcoxon signed-rank test [22] to verify the statistical difference between different methods.

External Validity. The first external validity threat to our experiment is the practicability of the methods. In the experiments, the benchmark contains both artifact faults and real faults, which empirically evaluate the effectiveness of our methods in practice. The programming language of the benchmarks used in our experiment is the second external validity threat. We only use C language datasets. Our proposed mutant reduction method is irrelevant to program languages so that it can be easily applied to other program languages. In the future, we will apply our methods to other program languages and verify the generalization of our proposed method.

VII. Conclusion

Considering the relationship between HOMs and statements, we propose a HOMs reduction method based on Statement Granularity Sampling (SGS). We apply our method to two benchmarks with 237 multiple-fault programs. The experiment results show that the SGS method with a sampling ratio of 20% is similar to the method of using all the mutants when considering fault localization performance and reduces the execution cost by around 80%. Finally, we compare HMBFL with the SGS-20% method to traditional fault localization techniques (i.e., three SBFL and three MBFL techniques). The results of this comparison show that HMBFL with the SGS-20% method has a higher fault localization effectiveness and efficiency than state-of-the-art SBFL and MBFL techniques.

In the future, we plan to apply our methods to more benchmarks (such as Defects4J) to verify the generalization of its effectiveness further. Moreover, we plan to perform mutant sampling by considering the HOM value in fault localization.

Acknowledgment

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

References

[1] W. E. Howden, "Theoretical and empirical studies of program testing," *IEEE Transactions on Software Engineering*, no. 4, pp. 293–298, 1978.

[2] 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.

[3] X. Zhang, H. He, N. Gupta, and R. Gupta, "Experimental evaluation of using dynamic slices for fault location," in *Proceedings of the sixth international symposium on Automated analysis-driven debugging*, 2005, pp. 33–42.

[4] 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.

[5] 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.

[6] M. Papadakis and Y. Le Traon, "Metallaxis-fl: mutation-based fault localization," *Software Testing, Verification and Reliability*, vol. 25, no. 5-7, pp. 605–628, 2015.

[7] 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.

[8] Y. Jia and M. Harman, "Higher order mutation testing," *Information and Software Technology*, vol. 51, no. 10, pp. 1379–1393, 2009.

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

[10] J. Liu and L. Song, "Second-order mutation testing cost reduction based on mutant clustering using som neural network model," in *2021 IEEE 45th Annual Computers, Software, and Applications Conference (COMPSAC)*. IEEE, 2021, pp. 974–979.

[11] 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.

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

[13] H. Agrawal, R. A. 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.

[14] A. J. Offutt, A. Lee, G. Rothermel, R. H. Untch, and C. Zapf, "An experimental determination of sufficient mutant operators," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 5, no. 2, pp. 99–118, 1996.

[15] L. Zhang, S.-S. Hou, J.-J. Hu, T. Xie, and H. Mei, "Is operator-based mutant selection superior to random mutant selection?" in *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering-Volume 1*, 2010, pp. 435–444.

[16] 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.

[17] Z. Li, B. Shi, H. Wang, Y. Liu, and X. Chen, "Hmbfl: Higher-order mutation-based fault localization," in *2021 8th International Conference on Dependable Systems and Their Applications (DSA)*. IEEE, 2021, pp. 66–77.

[18] Q. V. Nguyen and L. Madeyski, "On the relationship between the order of mutation testing and the properties of generated higher order mutants," in *Asian Conference on Intelligent Information and Database Systems*. Springer, 2016, pp. 245–254.

[19] C.-P. Wong, J. Meinicke, L. Chen, J. P. Diniz, C. Kästner, and E. Figueiredo, "Efficiently finding higher-order mutants," in *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2020, pp. 1165–1177.

[20] 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*, vol. 47, no. 2, pp. 332–347, 2019.

[21] B. Du, Y. Cai, H. Wang, Y. Liu, and X. Chen, "Improving the Performance of Mutation-based Fault Localization via Mutant Bias Practical Experience Report," in *2022 IEEE 33rd International Symposium on Software Reliability Engineering (ISSRE)*, Oct. 2022, pp. 309–320.

[22] Z. Zhang, Y. Lei, X. Mao, M. Yan, L. Xu, and J. Wen, "Improving deep-learning-based fault localization with resampling," *Journal of Software: Evolution and Process*, vol. 33, no. 3, p. e2312, 2021.

[23] P. Gong, R. Zhao, and Z. Li, "Faster mutation-based fault localization with a novel mutation execution strategy," in *2015 IEEE Eighth International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*. IEEE, 2015, pp. 1–10.