A Comparison Study of Two Well-known Fault Localization Methods

Nazanin Bayati Chaleshtari
Department of Computer Engineering
Iran University of Science and Technology
Tehran, Iran
n_bayati@comp.iust.ac.ir

Saeed Parsa
Department of Computer Engineering
Iran University of Science and Technology
Tehran, Iran
parsa@iust.ac.ir

***Abstract*—The performance of the fault localization techniques plays a significant role in their practical adoption. This paper reports an empirical study of two states of the art fault localization techniques. The main differences from previous studies are: (1) the specialized focus on two methods to present effective advice, (2) considering different suspiciousness formulae to limit their effect on the fault localization accuracy, and (3) two different test suites containing small and large scale projects to test all aspects of the methods. Important factors of each fault localization methods have defined the accuracy and execution time. Our results provide proper information about spectrum-based and mutation-based fault localization to help other scholars to choose the best method based on their requirements. The empirical results indicate that the mutation-based fault localization is relatively 12 percent slower and 5.2 percent more accurate than the spectrum-based fault localization.**

Keywords—software testing, fault localization, mutation-based, spectrum-based.

I. INTRODUCTION

Fault localization is one of the most important and time-consuming tasks in the debugging process. Several research studies are conducted to automate the fault localization [1][2]. Mutation-based, spectrum-based, and sliced-based fault localization are some of the popular fault localization methods [3]. Mutation-based fault localization, MBFL, uses mutation analysis. For each statement, different versions of the program, named mutants are produced. Then, each mutant is executed by the test suite, and the results are compared with the results of the original program. If the result of the mutant differs from the result of the program, the mutant is named as killed. The same results of the mutant and the program are regarded as a live mutant. Based on the results of executing mutants, the faulty statement is detected [4][5]. Spectrum-based fault localization, SBFL, is a lightweight method that uses the coverage information. The probability of incurring fault in each statement is calculated and based on the scores, the region of the buggy statement estimate. There are various contradictions about the effectiveness of SBFL. Some scholars believe that it is not the most accurate method that can be used in debugging real software [6]. One of the fault localization methods that most of the scientists agree with its efficiency is slice-based fault localization. In slice-based techniques is tried to focus on effective statements of the program. When the search space is restricted to a fewer number of statements, the execution costs also decrease [7][8].

There are several advantages and disadvantages to each fault localization method. Lots of studies have done on comparing the methods, but most of them give a general result of all methods [9][10]. In this paper, two specific methods, MBFL and SBFL, are selected to make a clear comparison. Slice-based fault localization methods are not considered. As these approaches use SBFL suspiciousness formulae, they then are implied as a member of the SBFL category.

In [9], a survey study based on two famous approaches of each method is done to compare some fault localization methods. The correlation between some pairs of techniques and introducing some possible combinations of the methods are explored. Also, it indicates that execution time is one of the most important features. Therefore, a new metric to measure the execution costs of the approaches is proposed. Finally, it is showed that the SBFL method is alternatively more efficient than MBFL. Based on the specialized study, it is not capable of making general decisions. Several studies are held to improve each fault localization technique and applying a new approach probably would change the results.

In another survey study, positive elements of the MBFL are proposed. The accuracy and ability to deal with real programs are mentioned as two important features of the MBFL rather than other methods. It also suggests that one of the difficulties of the mutation regards the finding of the redundant and equivalent mutants [10]. The problem is solved by new studies [11][12], but there is no evidence of execution time comparisons [10].

Empirical evidence implied that SBFL reduces debugging efforts effectively. However, the impractical fault localization accuracy and the unrealistic usage models are criticized, i.e., all statements in the same basic block may have the same spectrum, and this leads to an increased human interaction to check the statements before locating the bug. The MBFL is used as a practical method, and the main effort of the paper is reducing its execution costs [13].

A new method merges the SBFL and MBFL to increase the fault localization accuracy of the SBFL. First, the coverage of the program under test is calculated. Then, based on the suspiciousness area, the MBFL method is applied to find the exact place of the bug. The accuracy of the SBFL increased, but the execution costs of this method are not efficient [14]. Several research studies try to decrease the execution costs of the MBFL [15][16].

In this study, the execution costs and the accuracy of MBFL and SBFL will be examined using fourteen different test suites. In most of the fault localization techniques, a suspiciousness formula is used to locate the buggy statement.

In this regard, four of the well-known fault suspiciousness formulae are considered, which are shown in Table I.

In the second section, we discuss the empirical study. In the third section, we compare the SBFL and MBFL methods and discuss the results. Finally, conclusion and future work are explained in the fourth section.

and *totalpassed* refers to the total number of test cases that evaluated as passed.

B. Motivation

In Fig. 1 and Fig. 2, an example of running two fault localization methods, SBFL and MBFL, is shown. The first column contains the program code, which has a fault in the first statement, it should be *max = x,* instead of *max = -x*. The

Program code	Mutants	T1	T2	T3	T4	T5	T6	MBFL
		3,1	5,-4	0,-4	0,7	-1,3	6,-2	
max = -x // x	M1: max -= x-1	-	-	P to F	-	-	-	0.25
	M2: max = x	F to P	F to P	-	-	-	F to P	
if (max < y)	M3: max >y	-	-	P to F	P to F	P to F	-	0
	M4: max == y	-	-	-	P to F	P to F	-	
max = y	M5: max = -y	-	-	-	P to F	P to F	-	0
	M6: max = y+1	-	-	-	P to F	P to F	-	
print (max);	M7: print (0);	F to P	F to P	-	P to F	P to F	F to P	0.12
	M8: ;	-	-	P to F	P to F	P to F	-	
Result		F	F	P	P	P	F	

Fig. 1. An example of executing mutation-based fault localization

II. Empirical study

A. Suspiciousness formula

In most of the fault localization techniques a ranked list of suspiciousness scores of the program statements is reported [2][5]. The score is calculated based on the suspiciousness formula. Four well-known suspiciousness formulae are presented in Table I.

TABLE I. Some well-known suspiciousness formulae

Name	Formula
Ochiai	$\frac{failed(e)}{\sqrt{totalfailed * (failed(e) + passed(e))}}$
Tarantula	$\frac{failed(e)/totalfailed}{failed(e)/totalfailed + passed(e)/totalpassed}$
Jaccard	$\frac{failed(e)}{failed(e) + passed(e)}$
Kulczynski2	$\frac{failed(e)}{2 * totalfailed} + \frac{failed(e)}{2 * failed(e) + 2 * passed(e)}$

In proposed formulae *e*, is a code entity, *failed(e)* refers to the number of test cases covering the code entity *e* and failed, *passed(e)* refers to the number of test cases covering *e* and passed, *totalfailed* refers to the total number of failed test cases

In the first-row test cases, and the last row the results of P/F, passed or failed, are mentioned. Failed means the output result by executing the test case was not the expected value. Passed, means the result and the expected value were similar. In Fig. 1, the stars show the statements which are covered by the test case. The fault suspiciousness scores of the statements are calculated using the Tarantula formula.

Program code	T1	T2	T3	T4	T5	T6	SBFL
	3,1	5,-4	0,-4	0,7	-1,3	6,-2	
max = -x //x	*	*	*	*	*	*	0.5
if(max<y)	*	*	*	*	*	*	0.5
max = y	*	*			*	*	0.75
print (max)	*	*	*	*	*	*	0.5
Result	F	F	P	P	P	F	

Fig. 2. An example of executing the spectrum-based fault localization

In Fig. 2, the result of executing the same program by MBFL is presented. Just two samples of the possible mutants are shown in the second column. The result of running the test case on the mutant would be considered as killed or live. Live mutants are placed by dash symbol, and the killed mutants are specifically explained by *F to P* or *P to F*. The *F to P* means the program execution result on the test case was failed, but the result of the mutant by the same test case was changed to pass. The meaning of *P to F* is understood in the same way. Based on the result of locating a fault in the same program, the mutation-based method finds the accurate place of the buggy statement, and a wrong statement is reported by the spectrum-based method. However, based on a small program, any decision cannot be made, and the execution time of both methods is another issue.

C. Approch

Scientists and scholars paid attention to MBFL and mostly SBFL more than the other methods [18][19]. In section A, with an example, the higher accuracy of MBFL is shown, but the number of research studies that are conducted on SBFL is more than MBFL. In this study, the fault localization accuracy and execution time are considered as potential factors in defining an effective method.

The original version of the MBFL [20] and SBFL [21] are examined. In this paper for a fair comparison, only the original version of SBFL and MBFL will be tested not only on large programs but also on two different groups of test suites. The detailed information of the test suites is available in Table II and Table III. The information contains the project name, program size per line of source code, and the test file size per line of code. Table II includes four large java programs that are collected from the three repositories of SIR [22], Jaxen [23], and Defect4J [24]. Table III contains seven small java programs provided by the SEAR repository [25].

TABLE II. LARGE SCALE TEST SUITE

Project name	Program size (LOC)	Test file size (LOC)
Commons-lang	24,289	41,758
Jaxen	12,449	8,371
Joda-time	28,479	54,645
Jtopas	2,031	3,185

TABLE III. SMALL SCALE TEST SUITE

Project name	Program size (LOC)	Test file size (LOC)
Cal	135	54
Calculation	100	51
CheckPalindorme	37	89
countPositive	52	85
findLast	83	95
findVal	79	105
Gaussian	77	78

Let look at the first row of Table II, Commons-lang is a java program with about 24k line of code, and the size of the test file is about 42k.

The accuracy of a fault localization approach is a significant element in suggesting an effective method. Whether the execution costs of the method can change the result, it means if the method deals with high execution cost, it won't be labeled as an effective one. In the evaluation section, both sides of being efficient for SBFL and MBFL are checked.

III. EVALUATION

In most of the research studies, test suites are only some large programs, but in some actions, like the example in Fig. 1, small programs highlight a hidden feature of a method. In this regard, the using test suite is containing two separate groups of programs. One includes small, and the other includes large programs to detect all available features of the method. Their ability to locate the buggy statement using SCORE is measured. The SCORE criterion is widely used in several research studies, and minimize human interaction in localizing the fault by computing the percentage of statements that should be verified before finding the faulty statement in the program under test [13][14]. The lower value of the SCORE means higher accuracy because it needs fewer statements to be inspected. The SCORE formula is as below:

$$SCORE = \frac{rank\_of\_faulty\_statement}{number\_of\_executed\_statements} * 100 \quad (1)$$

The execution costs include the time needed to complete the fault localization process and the efforts to report the exact location of the bug, especially in SBFL.

A. Accuracy

The results of executing SBFL and MBFL methods on large and small scale programs are respectively shown in Table IV and Table V. In Table IV, the first column indicates the SCORE value that is the percentage of the code examined. In the other columns, the percentage of the localized faults of two fault localization methods present. Relatively, MBFL has higher accuracy than SBFL in large scale projects.

TABLE IV. THE ACCURACY OF METHODS ON LARGE SCALE PROJECTS

Code examined (%)	Percentage of fault located	
	MBFL	SBFL
1	15.10	10.68
5	39.97	31.35
10	50.85	42.23
15	59.39	50.30
20	62.95	58.55
30	73.52	69.36
40	84.92	75.77
50	92.45	83.98
60	95.87	88.19
70	98.29	91.16

80	99.37	95.87
90	100	98.29
100	100	100

In another examination, the accuracy of the MBFL and SBFL on the small dataset is available in Table V. The value of the rows is conceded the same as Table IV. Based on the percentage of the localized faults, MBFL earns higher accuracy even in small scale programs. Overall, MBFL has higher accuracy than SBFL. The suspicious formula is Tarantula, but to recognize the effect of the different formulae on the accuracy of these two methods, we directed other evaluations using different suspicious formulae from Table I. The results are shown in Fig. 3 and Fig. 4.

TABLE V. THE ACCURACY OF METHODS ON SMALL SCALE PROJECTS

Code examined (%)	Percentage of fault located	
	MBFL	SBFL
1	11.65	9.27
5	25.32	20.54
10	41.64	36.51
15	49.72	41.42
20	57.84	53.24
30	68.61	62.78
40	79.83	70.63
50	88.35	81.15
60	93.76	89.61
70	97.33	92.98
80	100	97.33
90	100	100
100	100	100

The accuracy of the SBFL method using four suspicious metrics, Tarantula, Ochiai, Jaccard, and kulczynski2, are presented in Fig. 3. The variation between the bars suggests the influence of different metrics on the accuracy of the SBFL, which is not a strong point. However, in Fig. 4, the fluctuate of the MBFL bars are relatively negligible.

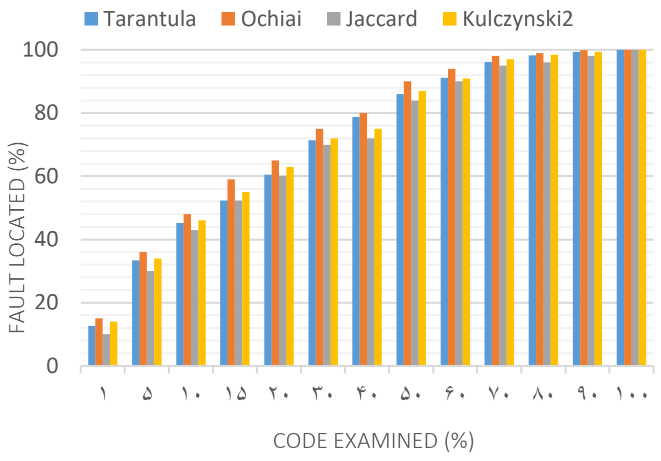

Fig. 3. SBFL accuracy using four different suspiciousness formulae

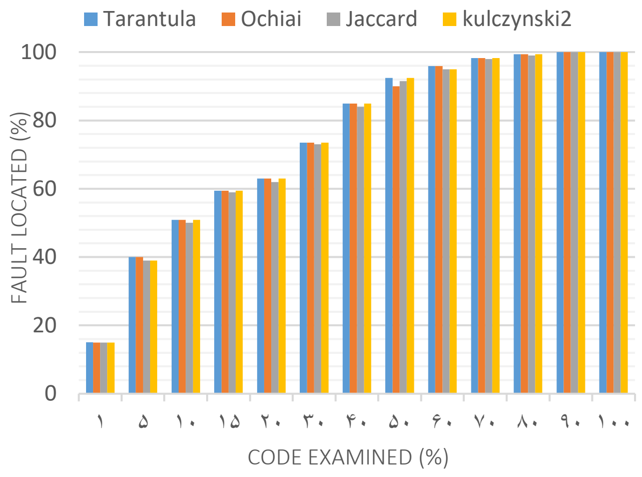

Fig. 4. MBFL accuracy using four different suspiciousness formulae

On average, the accuracy of the MBFL method is 5.2 percent higher than the SBFL on all projects. Also, it is indicated that the MBFL is less sensitive to different suspiciousness formulae rather than SBFL.

B. Execution cost

The execution costs have a significant role in choosing a fault localization approach. Real programs are mostly large, and the larger the program leads to higher execution time. In this section, the execution time of MBFL and SBFL is measured, the results are in Fig. 5, and Fig. 6. Respectively, Fig. 5 and Fig. 6 indicate the execution time on a large scale and the small scale datasets. As MBFL creates different mutants for each statement of the program and applies the mutation analysis to locate the fault, it is expected to get higher execution time.

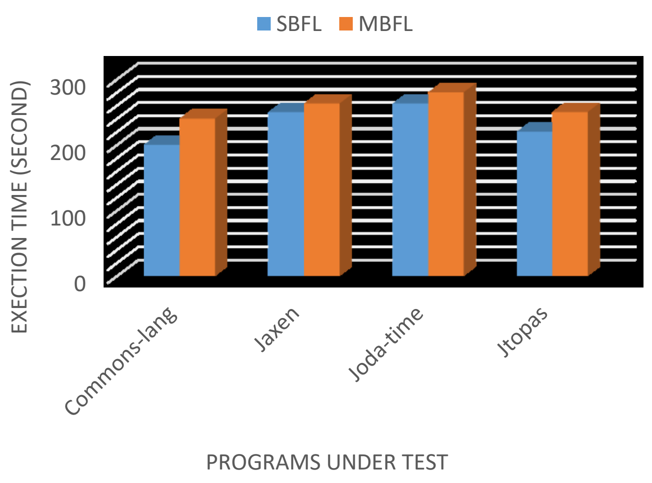

Fig. 5. The execution time of MBFL and SBFL on large scale projects

The red bars in Fig. 5 indicate the MBFL and the blue bars are related to SBFL. It is crystal clear that in all four projects, SBFL attained lower execution time. Also, in Fig. 6, the average value of the execution time of the MBFL is higher than the SBFL. On average, SBFL is 12 percent faster than MBFL.

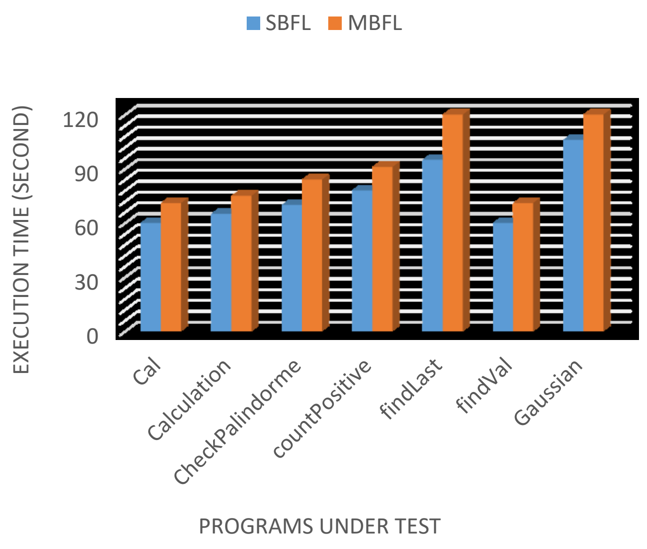

Fig. 6. The execution time of MBFL and SBFL on small scale projects

IV. Conclusion and Future Works

There are several research studies conducted to locate the exact place of the fault in programs. The two most popular methods are MBFL and SBFL. In this research, we extracted the benefits of each approach, which we aim to help other scientists to select the proper method based on their requirements. Two separate datasets are used to make the results more reliable. The empirical evaluation showed the MBFL has relatively higher accuracy in locating the exact place of the faults in programs than SBFL. Besides the accuracy that is an important factor in determining an effective fault localization technique, the execution cost is checked. As MBFL not only executes the program statements but also execute the mutants, the MBFL has higher execution time rather than SBFL. Some reports reveal the high execution cost of the MBFL avoids scholars to benefit from its great accuracy. Reducing the execution time of MBFL can be one of our future works. Some studies improve the execution time by limiting the number of mutants or the mutation operators, but no one claims that completely solves the problem. Another future work could be combining MBFL and SBFL to utilize both strength points.

References

[1] Y. Zheng, Z. Wang, X. Fan, X. Chen, and Z. Yang, “Localizing multiple software faults based on evolution algorithm,” *J. Syst. Softw.*, vol. 139, pp. 107–123, 2018.

[2] S. Chakraborty, Y. Li, M. Irvine, R. Saha, and B. Ray, “entropy guided spectrum based bug localization using statistical language model,” arXiv, inpress.

[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, 2016.

[4] N. Gupta, A. Sharma, and M. K. Pachariya, “A novel approach for mutant diversity-based fault localization: DAM-FL,” *Int. J. Comput. Appl.*, in press.

[5] J. Kim, G. An, R. Feldt, and S. Yoo, “Amortising the Cost of Mutation Based Fault Localisation using Statistical Inference,” arXiv, inpress.

[6] Y. Liu, Z. Li, L. Wang, Z. Hu, and R. Zhao, “Statement-oriented mutant reduction strategy for mutation based fault localization,” *Proc. - IEEE Int. Conf. Softw. Qual. Reliab. and Secur., QRS*, Prague, Czech Republic, 2017, pp. 126–137.

[7] F. Feyzi and S. Parsa, “A program slicing-based method for effective detection of coincidentally correct test cases,” *Computing*, vol. 100, no. 9, pp. 927–969, 2018.

[8] Y. Zhang and R. Santelices, “Prioritized static slicing and its application to fault localization,” *J. Syst. Softw.*, vol. 114, pp. 38–53, 2016.

[9] 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.*, pp. 1–16, 2019.

[10] M. Papadakis, M. Kintis, J. Zhang, Y. Jia, Y. Le Traon, and M. Harman, “Mutation Testing Advances: An Analysis and Survey,” *Adv. Comput.*, vol. 112, pp. 275–378, 2019.

[11] P. R. Mateo and M. P. Usaola, “Reducing mutation costs through uncovered mutants,” *Softw. Testing, Verif. Reliab.*, vol. 25, pp. 464–489, 2014.

[12] A. S. Namin, J. H. Andrews, and D. J. Murdoch, “Sufficient mutation operators for measuring test effectiveness,” *Proc. - Int. Conf. Softw. Eng.*, Leipzig, Germany, 2008, pp. 351–360.

[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. (Ny).*, vol. 422, pp. 572–596, 2018.

[14] Z. Li, L. Yan, Y. Liu, Z. Zhang, and B. Jiang, “MURE: Making Use of MUtations to REfine Spectrum-Based Fault Localization,” *Proc. - IEEE 18th Int. Conf. Softw. Qual. Reliab. Secur. Companion, QRS-C*, Lisbon, Portugal, 2018, pp. 56–63.

[15] P. Gong, R. Zhao, and Z. Li, “Faster mutation-based fault localization with a novel mutation execution strategy,” *Proc. - IEEE 8th Int. Conf. Softw. Testing, Verif. Valid. Work., ICSTW*, Graz, Austria, 2015, pp. 1–10.

[16] D. Schuler, V. Dallmeier, and A. Zeller, “Efficient mutation testing by checking invariant violations,” *Proc. - 18th Int. Symp. Softw. Testing, Anal., ISSTA*, Chicago, USA, 2009, pp. 69–79.

[17] S. Moon, Y. Kim, M. Kim, and S. Yoo, “Ask the Mutants: Mutating faulty programs for fault localization,” *Proc. - IEEE 7th Int.* Conf. Softw. Testing, Verif. Valid., ICST, Cleveland, USA, 2014, pp. 153–162.

[18] A. Perez, R. Abreu, and A. Van Deursen, “A Test-Suite Diagnosability Metric for Spectrum-Based Fault Localization Approaches,” *Proc. - IEEE Int. Conf. on Softw. Eng. ICSE*, vol. 2, Buenos Aires, Argentina, 2017, pp. 654–664.

[19] S. Pearson, J. Campos, R. Just, G. Fraser, R. Abreu, M. D. Ernst, D. Pang, and B. Keller, “Evaluating and Improving Fault Localization techniques,” *Proc. - IEEE/ACM 39th Int. Conf. on Softw. Eng., ICSE*, Buenos Aires, Argentina, 2017, pp. 609–620.

[20] M. Papadakis and Y. Taron, “Metallaxis-FL: mutation-based fault localization,” *Softw. Testing. Verif. Reliab.*, vol. 25, no. 6, pp. 605–628, 2015.

[21] L. Naish, H. J. Lee, and K. Ramamohanarao, “A model for spectra-based software diagnosis,” *ACM Trans. Softw. Eng. Methodol.*, vol. 20, no. 3, pp. 1–32, 2011.

[22] 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, 2005.

[23] Jaxen, Maven repository, https://mvnrepository.com/artifact/jaxen/jaxen/1.1.1. .

[24] R. Just, D. Jalali, and M. D. Ernst, “Defects4J: A database of existing faults to enable controlled testing studies for Java programs,” *Proc. - Int. Symp. Softw. Testing, Anal, ISSTA,* San Jose, USA, 2014, pp. 437–440.

[25] L. Deng, J. Offutt, and N. Li, “Empirical evaluation of the statement deletion mutation operator,” *Proc. - IEEE 6th Int. Conf. Softw. Testing, Verif. Valid., ICST,* Luxembourg, 2013, pp. 84–93.