EMBFL: Ensemble of Mutation based techniques for effective Fault Localization

Jitendra Gora [a], Arpita Dutta [b] and Durga Prasad Mohapatra [c]

[a] Computer Science and Engineering, NIT Rourkela, Rourkela, India, E-mail: jitendragora1305@gmail.com; [b] School of Computing, National University of Singapore, Singapore, E-mail: arpitad10j@gmail.com; [c] Professor, Computer Science and Engineering, NIT Rourkela, Rourkela, India, E-mail: durga@nitrkl.ac.in

Abstract

Finding locations of faults in a program is a crucial activity in reliable and effective software development. A large number of fault localization techniques exist, however, none of these techniques outperforms all other techniques in all circumstances for all kinds of faults. Under different circumstances, different fault localization techniques yield different results. In this study, we have proposed Ensemble of Mutation Based techniques for effective Fault Localization (EMBFL). EMBFL classifies statements of a program into Suspicious and Non-Suspicious sets. The model we have used in our research is straightforward and intuitive because it is based solely on information regarding statement coverage and test case execution results. This helps to reduce the search space significantly. Our proposed EMBFL approach, on average, is 31.34% more effective than the techniques for fault localization that currently exist such as DStar (D^*), Tarantula, Back Propagation Neural Network, etc.

Keywords: debugging, ensemble classifier, fault localization, mutation analysis

Introduction

With the continuously growing usage of software in our daily lives, it has become critical to systems in several industries such as healthcare, teaching, marketing, etc. This has resulted in a substantial scale in the complexity and size of software. With so much of complexity and size, software faults are inevitable and these faults often lead to execution failure of the software. Therefore, testing and debugging the software has become highly crucial part of software development process. Fault localization is a vital step of software testing. It is the activity of finding out the faulty locations in a software and has been an expensive task in terms of manual effort, time and money, considering the size and complexity of the software. To overcome these limitations, researchers are trying to develop techniques that partially or fully automate this task and assist developers in the debugging. Many fault localization techniques are being used currently but no technique outperforms all available techniques in all circumstances. For instance, few techniques may perform really well for faults that are related to relational and logical operators whereas few other techniques may perform well for faults that are related to arithmetic operators.

The objective of this study is to develop effective and efficient techniques for fault localization. Our technique is inspired by the two famous domains of fault localization, i.e., Mutation Based Fault Localization (MBFL) and Spectrum Based Fault Localization (SBFL). We named this approach as EMBFL since we combined multiple **M**utation **B**ased **F**ault **L**ocalization techniques using an **E**nsemble classifier.

The rest of the paper is structured in the following way. In the following section, few of the related works are discussed. Our proposed methodology is described in depth in Section 3. The last two sections summarize the experimental studies and conclusion of our work done respectively.

Literature review

In this section, we have attempted to summarize few of the existing works of authors related to fault localization. Weiser (1984) proposed a technique called Program Slicing technique that reduces the search space for inspection. There are some drawbacks of slicing techniques. For instance, slicing techniques are not suitable for assigning ranks to the statements. SBFL techniques as mentioned in the works of Jones et al. (2005), Renieres et al. (2003), and other researchers overcame such limitations by taking different coverage information like statement coverage, branch coverage along with the test case execution results and producing suspiciousness score for each of the program entity. Then they used ranking matrices to assign ranks to all of the executable statements in the program. Later, Wong et al. (2013) improved the results and came up with an effective technique named DStar (D^*) which is currently the state-of-the-art.

Problem with SBFL techniques is that they assign same rank to multiple statements and that increases the inspection domain in order to find the location of faults. To overcome this limitation, MBFL techniques such as Metallaxis and MUSE were introduced by Papadakis et al., (2015) and Moon et al., (2014) respectively. MBFL techniques focus on generating useful mutants to make fault localization more effective. Mutants are copies of the original program under test with the syntax being changed at least once based on mutation operators. Also, MBFL techniques are effective enough to deal with problem of coincidental correctness (Li et al., 2019). Although these MBFL techniques take impact information into consideration, there may be some cases where they perform poorly, e.g., some entities may not have any mutants to simulate their impacts. To eliminates such limitations, researchers currently use machine learning and deep learning-based techniques (Ascari et al., 2009; Wong et al., 2011).

The novelty of our study lies in that machine learning based ensemble classifier has been used in our approach to combine SBFL and MBFL techniques. As per research, ensemble classifiers perform much better than the constituent learning techniques alone (Roychowdhury et al., 2012; Dutta et al., 2021). Besides, the existing

techniques are not effective enough for large sized programs. Our proposed EMBFL approach for fault localization addresses this problem as well.

Proposed approach: EMBFL

Overview

In our approach, the original program is given as input and mutants are obtained for the given program. Then code coverage information is generated using these mutants along with the test cases. After getting code coverage, we give this information with the test case execution results to statement score generator which uses 40 different MBFL techniques to calculate the sequence of suspicious scores for all executable statements of the program. After getting sequence of score, we perform the normalization to bring the scores in a fixed range. Later we apply Ranking algorithms on all the forty MBFL techniques used which results in prioritised list of statements that can contain the faults in the program.

Detailed description of EMBFL

In this subsection, we have presented a detailed discussion on Mutator, Program Spectra Generator, Statement Score Generator and Learning to Rank algorithm. Figure 1 depicts the high-level architecture of our proposed methodology.

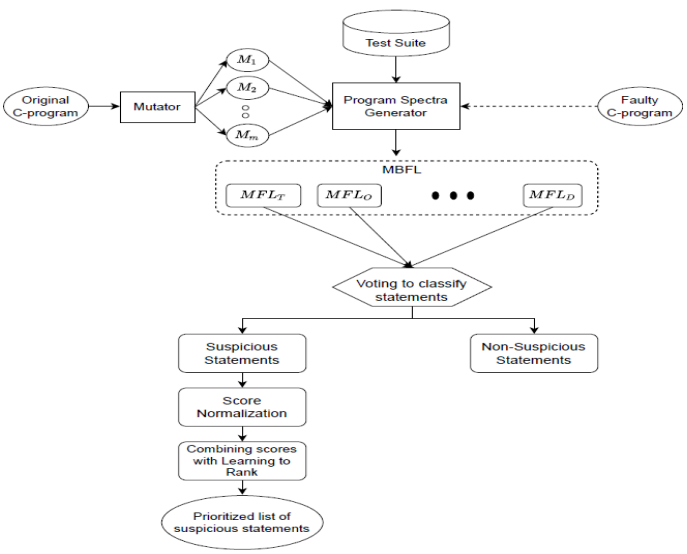


Figure 1. High level architecture of the methodology proposed in this study.

Mutator: Since we are provided with the original program only, we need to generate mutants for that program in order to get the information about code coverage and

results of test case execution. Mutator is a module which we have developed to generate different mutants for the original program. The original program is given as input to the mutator and mutants are obtained as output. Depending upon the available substitutes for a statement, the mutants are categorized as follows: Category-1 and Category-2. Category-1 contains the mutants which have finite number of possible replacements. These mutants can be generated by replacing a operator by one of its substitutes. Few of the operator-substitutes from Category-1 are given below.

- AOR: Arithmetic Operator Replacement (e.g., multiplication (‘*’) in place of division (‘/’))
- (I/D)OR: Increment/ Decrement Operator Replacement (e.g., decrement (‘--’) in place of increment (‘++’))
- AsOR: Assignment Operator Replacement (e.g., plus equals to (‘+=’) in place of minus equals to (‘-=’))
- ROR: Relational Operator Replacement (e.g., less than (‘<’) in place of greater than (‘>’))
- LOR: Logical Operator Replacement (e.g., OR (‘||’) in place of AND (‘&&’))

Category-2 contains the mutants which have numerous numbers of possible replacements. Few of the mutation operations under Category-2 are given below.

- SI: Statement Insertion
- SD: Statement Deletion
- VR: Variable Replacement (e.g., ‘x=y+d’ in place of ‘x=y+z’)
- SIE: Swapping of ‘if' block with 'else’ block statements
- SI: Sign Inversion (e.g., ‘x=-y’ in place of ‘x=y’)

Program Spectra Generator: The next module that we developed is the Program Spectra Generator. It produces information regarding code coverage and results of test case execution by taking original program, its mutant, and all accessible test cases as input. If a test case covers a statement, then we use ‘1’ to represent the statement else we use ‘0’ to represent it. The result of a test case execution indicates whether the particular test case has failed (F) or passed (P). Table 1 depicts a sample program spectra along with the results of test case execution. Suppose that the sample program contains eight executable statements (ST1 to ST8) and our test suite contains five test cases (TC1 to TC5). Table 1 highlights that three of the five test cases (TC1, TC2 and TC5) passed while the other two failed. Let's pick a test case, say TC3, it can be observed that only the statements ST1 and ST8 are executed by TC3 and the remaining statements are not executed by it.

Table 1: Sample program spectra with test case execution results.

S. No.	Test Cases	ST1	ST2	ST3	ST4	ST5	ST6	ST7	ST8	Result
1	TC1	1	1	1	1	1	1	1	0	P
2	TC2	0	0	1	0	0	0	1	1	P
3	TC3	1	0	0	0	0	0	0	1	F
4	TC4	1	0	1	1	1	1	1	0	F

5	TC5	1	0	1	0	0	0	0	1	P

Statement Score Generator: Statement Score Generator (SSG) is a module which we have developed to obtain statement score sequences. It takes program spectra and the results of test case execution as input and produces sequence of statement scores depending upon multiple MBFL techniques as output. Since we have used forty MBFL techniques in our approach, SSG produces forty score sequences. Table 2 enlists the formulas of few of the SBFL techniques which we have used in our approach. N is the total number of test cases available; N_p is the number of passed test cases among those available and N_f is the number of failed test cases among those available. The number of succeeded test cases that executed a statement is N_{ep}, whereas the number of failed test cases that executed a statement is N_{ef}. Similarly, N_{np} represents the number of succeeded test cases that did not execute a specific statement, while N_{nf} represents the number of failed test cases that did not execute a specific statement.

Table 2: Definitions of some SBFL Techniques

S. No.	Name of Technique	Formula
1	Tarantula	$(N_{ef}/(N_{ef}+N_{nf})) / (N_{ef}/(N_{ef}+N_{nf}) + N_{ep}/(N_{ep}+N_{np}))$
2	Ample1	$\lvert (N_{ef}/N_{ef}*N_{ep}) - (N_{ep}/N_{ep}*N_{np}) \rvert$
3	Ample2	$N_{ef}/N_f - N_{ep}/N_p$
4	Lee	$N_{ef} + N_{np}$
5	Goodman	$(2*N_{ef}-N_{nf}-N_{ep})/2*N_{ef}+N_{nf}+N_{ep}$
6	Wong1	N_{ef}
7	Wong2	$N_{ef}-N_{ep}$
8	DStar (D^*)	$(N_{ef})^*/(N_{ep}*N_{nf})$
9	Jaccard	$N_{ef}/(N_f+N_{ep})$
10	Euclid	$\sqrt{(N_{ef} + N_{np})}$

Learning to Rank: This algorithm considers a collection of objects and assigns a suitable rank to them. We are using forty MBFL techniques in our approach and each technique assigns a different score to all the statements of the input program for each mutant. Therefore, we have used Learning to Rank algorithm to all of the techniques and this results in assigning higher rank to the right technique in every different scenario. After applying this Learning to Rank algorithm we get a prioritised list of statements.

Experimental Studies

This section presents the data set, evaluation metrics used and the empirical results obtained in our approach.

Data Set Used

We have used Siemens suite as our input data set which contains 7 different programs named "printtokens", "printtoken2", "replace", "schedule", "schedule2", "tcas" and "totinfo". Table 3 shows the characteristics of Siemens suite 7 programs. We have generated different category-1 mutants for each program in the Siemens suite.

Table 3: Characteristics of programs available in Siemen suite

S. No.	Name of Program	Count of Faulty Versions	Count of Executable LOC	Count of Test Cases	Count of Mutants
1	Print_Tokens	7	195	4130	285
2	Print_Tokens2	10	200	4115	314
3	Replace	32	244	5542	508
4	Tcas	41	65	1608	216
5	Tot_info	23	122	1052	571
6	Schedule	9	152	2650	406
7	Schedule2	10	128	2710	350

Evaluation Metrics

For evaluating the performance of our approach EMBFL and comparing it with that of the existing fault localization techniques, Exam-Score evaluation metric has been used. Exam-Score is the percentage of statements that must be examined in order to locate faults in a program. It is computed using Equation 1.

$$Exam - score = (|\, S_{examined} \,| \div |\, S_{total} \,|) * 100 \quad (1)$$

where, $|S_{examined}|$ denotes the count of statements required to be examined to find the faulty locations. $|S_{total}|$ denotes the total number of statements in the program. The fault localization technique that has Exam-Score less than that of all others is the most effective technique.

Empirical Evaluation

We have compared the effectiveness of EMBFL with four other prominent existing fault localization techniques. Among these four techniques, Tarantula and DStar (D*) are taken from SBFL family. On the other hand, BPNN and RBFNN belong to the neural network family of fault localization techniques. Figures 2 to 5 depict the effectiveness comparison of EMBFL with other techniques for fault localization using the line graphs. Best effectiveness means the statement containing fault is examined "first" among those with the same score of suspiciousness. Worst effectiveness, on the other hand, means that the statement containing fault gets examined "last" among those with the same score of suspiciousness.

Figure 2 compares the effectiveness of Tarantula and EMBFL for Siemens suite. The figure suggests that by inspecting only 3% of program statements, EMBFL localizes faults in 25.77% of faulty versions. Tarantula (Worst) and Tarantula (Best) respectively localize faults in 10.31% and 20.61% of faulty versions only. On average, Tarantula (Worst) and Tarantula (Best) require 31.71% and 19.98% of program statement examination to localize bugs. On the other hand, EMBFL requires only 16.08% of code examination, on average. Our proposed EMBFL method is 49.29% and 19.54% more effective than Tarantula (Worst) and Tarantula (Best) respectively.

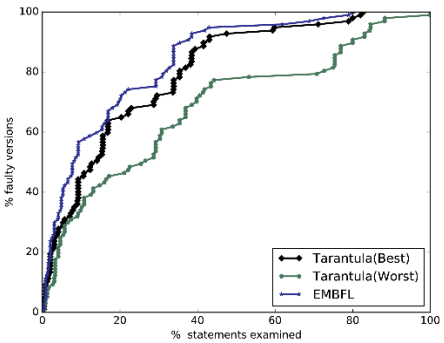


Figure 2. Effectiveness of EMBFL, and Tarantula for the Siemens suite

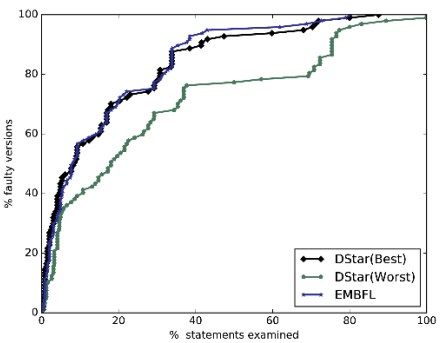
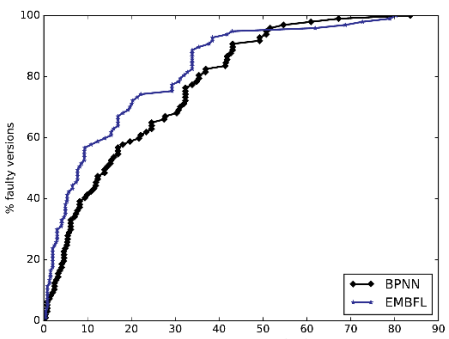


Figure 3. Effectiveness of EMBFL, and and D* for the Siemens suite

Figure 3 presents the effectiveness comparison of DStar and EMBFL for Siemens suite. It is clearly evident from the figure that EMBFL is performing equally well as Dstar(Best) for most of the program points. However, there are several points present on which EMBFL is performing much better than DStar (Best). We observe that there is a huge gap between the effectiveness of DStar (Worst) and EMBFL. EMBFL requires 20% and 7.5% less code inspection than DStar(Worst) and DStar(Best), respectively, in the worst scenario. On average, EMBFL outperforms DStar (Worst) and DStar (Best) by 43.53% and 4.38% respectively.



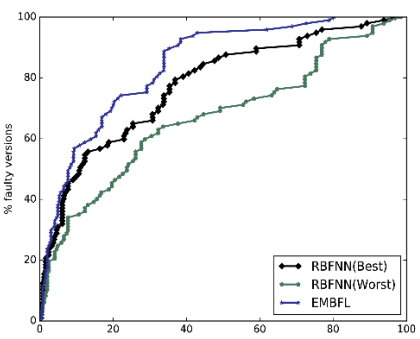


Figure 4. Effectiveness of EMBFL, and BPNN for the Siemens suite

Figure 5. Effectiveness of EMBFL, and and RBFNN for the Siemens suite

Figure 4 compares the results of BPNN and EMBFL for Siemens suite. According to the graph, EMBFL only requires 8% code inspection to locate defects in 50% of the faulty versions, whereas BPNN requires at least 14.47% code inspection. In the worst-case scenario, EMBFL requires 3% less code inspections than BPNN, and it is 21.15% more effective on average.

Figure 5 compares the effectiveness of RBFNN and EMBFL for Siemens suite. The figure indicates that EMBFL locates errors in 38.14% of faulty programs by evaluating only 5% of statements in the program, but RBFNN(Worst) and RBFNN(Best) locate faults in only 24.74% and 30% of faulty versions, respectively. On average, EMBFL outperforms RBFNN(Worst) and RBFNN(Best) by 52.01% and 29.49%, respectively. EMBFL also has to examine 18.14% and 16.60% less code in the worst scenario than RBFNN(Worst) and RBFNN(Best).

Table 4: Pairwise comparison between EMBFL and existing fault localization techniques with respect to Exam-Score.

	EMBFL v/s Tarantula (Best)	EMBFL v/s Tarantula (Worst)	EMBFL v/s DStar (Best)	EMBFL v/s DStar (Worst)	EMBFL v/s RBFNN (Best)	EMBFL v/s RBFNN (Worst)	EMBFL v/s BPNN
More Effective	49.48	81.44	18.56	68.04	43.3	69.07	58.76
Equally Effective	32.99	5.15	39.18	10.31	8.25	7.22	6.19
Less Effective	17.53	13.4	42.27	21.65	48.45	23.71	35.05

Table 4 presents the pairwise comparison of EMBFL with Tarantula, DStar, RBFNN, and BPNN. The table illustrates the percentage of faulty versions on which EMBFL performs more effectively, equally effectively and less effectively than the respective fault localization techniques in rows 2, 3, and 4 respectively. It can be observed that EMBFL performs at least as effective than all the other techniques in more than 55% of faulty versions. There are substantially a smaller number of faulty versions present on which EMBFL is lesser effective than the existing methods of fault localization.

Conclusion and Future Scope

Fault localization is a crucial part of developing reliable and effective software. To make fault localization easier and more effective, in this study we have proposed an ensemble classifier-based technique. We have combined different mutation-based fault localization techniques. Our method is able to effectively identify locations of common as well as intrinsic faults present in the program. From our empirical evaluation we have observed that, on average, EMBFL performs 31.34% more

effectively in terms of less code examination than the related fault localization techniques such as Tarantula, DStar, BPNN, and RBFNN.

As part of future scope of our work, we intend to apply EMBFL on multiple faulty programs and also attempt to improve its performance. We also intend to further improve our approach such that it makes use of the individual fault exposing capabilities of a test case.

References

Weiser, M. (1984). Program slicing. IEEE Transactions on software engineering. 4: 352-357.

Jones, James, A., and Harrold, M. J. (2005). Empirical evaluation of the tarantula automatic fault-localization technique, In Proceedings of the 20th IEEE/ACM International Conference on Automated software engineering (pp. 273-282).

Renieres, M. and Reiss, S. P. (2003). Fault localization with nearest neighbor queries, In 18th IEEE International Conference on Automated Software Engineering, 2003. Proceedings. (pp. 30-39). IEEE.

Wong, W. E., Debroy, V., Gao, R. and Li, Y. (2013). The DStar method for effective software fault localization. IEEE Transactions on Reliability. 63(1): 290-308.

Papadakis, M. and Traon, Y. L. (2015). Metallaxis-FL: mutation-based fault localization. Software Testing, Verification and Reliability. 25(5-7):605-628.

Moon, S., Kim, Y., Kim, M. and Yoo, S. (2014). Ask the mutants: Mutating faulty programs for fault localization, In 2014 IEEE Seventh International Conference on Software Testing, Verification and Validation (pp. 153-162). IEEE.

Ascari, L. C., Araki, L. Y., Pozo, A. R. and Vergilio, S. R. (2009). Exploring machine learning techniques for fault localization, In 2009 10th Latin American Test Workshop (pp. 1-6). IEEE.

Roychowdhury, S. (2012). Ensemble of feature selectors for software fault localization, In 2012 IEEE International Conference on Systems, Man, and Cybernetics (SMC) (pp. 1351-1356). IEEE.

Wong, W. E., Debroy, V., Golden, R., Xu, X. and Thuraisingham, B. (2011). Effective software fault localization using an RBF neural network. IEEE Transactions on Reliability. 61(1): 149-169.

Li, X., Li, W., Zhang, Y. and Zhang, L. (2019). Deepfl: Integrating multiple fault diagnosis dimensions for deep fault localization, In Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis (pp. 169-180).

Dutta, A., Srivastava, S. S., Godboley, S. and Mohapatra, D. P. (2021). Combi-FL: Neural network and SBFL based fault localization using mutation analysis. Journal of Computer Languages. 66: 101064.