Sādhanā (2020)45:148
https://doi.org/10.1007/s12046-020-01382-6

Testing and debugging: an empirical evaluation of integrated approaches

NEHA GUPTA[1,*], ARUN SHARMA[1] and MANOJ KUMAR PACHARIYA[2]

[1]Department of Information Technology, Indira Gandhi Delhi Technical University for Women, New Delhi 110006, India
[2]Department of Computer Science and Applications, Makhanlal Chaturvedi National University of Journalism and Communication, Bhopal 462011, India
e-mail: nehagupta_4321@yahoo.co.in; arunsharma@igdtuw.ac.in; m_pachariya1@yahoo.com

MS received 11 December 2019; revised 21 April 2020; accepted 3 May 2020

Abstract. Software development is a continuous process. Among all phases of software development, testing and debugging are the most essential phases. The main intention of testing is to detect maximum faults as soon as possible. After a fault is detected, it must be removed through appropriate debugging approach. Both phases are performed one after another and require different information. Hence, it is arduous to merge them. In fault detection, such test cases are required that are able to detect all the faults with less effort whereas in fault localization such test cases are required that are able to reproduce faults and locate them with accuracy. For enhancing the performance of testing, the integration of these two phases with consideration of multi-objective selection of test cases is to be conducted before regression testing. In the current work, an approach for multi-objective test case selection based on statement and diversified mutant coverage has been proposed and compared with existing approaches. For empirical study, SPEA-2, NSGA-2 and VEGA algorithms have been used and experiments were conducted on three applications of the Defects4j database. Outcomes of the study specify that the proposed approach has the ability to detect more faults compared to existing approaches. It is also able to locate all faults that have been detected with fault localization score better or comparable to existing approaches.

Keywords. Testing; debugging; test case selection; optimization techniques; multi-criteria optimization.

1. Introduction

Software testing is a major phase in the development process of software. It is done with a purpose to find faults. Every time a modification is made in software, regression testing is carried on to test that changes have not led to any faults in the software. Executing all test cases, for regression testing is a time taking process. For reducing the time requisite for regression testing, test case selection, minimization or prioritization is done. If a fault is detected while regression testing, it needs to be debugged also. For debugging, first, it needs to be located in the code and then removed. Both these phases are necessary to perform because the delivery of correct version of software is dependent on them. It is suggested by Boehm [1], that from total development time, 50% should be given for testing and 25% for debugging.

As both phases take time and effort, optimization algorithms have been used to reduce the required effort by testers. Optimization techniques such as SPEA-2, MO-BAT, MO-PSO, NSGA-2, VEGA, PAES, TAEA, etc. have been used in research work [2–12] for effective optimization of test cases for fault detection. Various approaches [13–17] have been developed for minimizing and optimizing the effort required for fault localization.

The current scenario is that test cases are optimized for each phase but separately. This is because the subsets of test cases that are apt for detecting faults are not successful in locating faults effectively. Gonzalez *et al* [14] and Yu *et al* [18] have used test suite apt for regression testing on fault localization and found that test suite optimization has less effect on fault detection but a large consequence on fault localization. It leads to low fault localization capability of test suites. Thus, it becomes difficult to optimize test cases together for both phases. This increases the cost of whole software development. It has been suggested by Gupta *et al* [19] that testing may be done in a way that aids debugging. This would help in decreasing the time required for considered phase.

The solution to this scenario is that first, category for test suite optimization i.e., selection, minimization or prioritization for both phases should be decided. Then adequacy

*For correspondence

Published online: 06 June 2020

criteria used for each phase need to be identified according to the category of optimization. Adequacy criteria selection would be important here as test cases optimized for fault detection are not suitable for fault localization. Statement coverage, branch coverage, path coverage, execution time, mutation coverage are some of the adequacy criteria that have been used for fault detection. Mutation adequacy, statement Partitioning [15], mutant partitioning/D-criterion [20], diversity maximization speedup [21], DDU (Density-diversity-uniqueness) [17] are some of the adequacy criteria that have been used for fault localization.

For an integrated optimization approach, only a few fault localization adequacy criteria may be used because most of the adequacy criteria or fault localization techniques are proposed to handle fault localization after fault detection. Thus, they would initially need information about fault detecting test cases. Fault localization techniques may further be divided into two categories: Offline [14, 15, 17, 22] and Online [16, 21, 23, 24]. Offline techniques do not need pass or fail data of test cases whereas online techniques require this data. As integrated optimization has to be done before regression testing, only offline techniques may be helpful for this optimization. Thus, for an integrated solution, it is necessary to finalize the category of optimization and adequacy criteria to be used for each considered phase.

To tackle the considered problem, an integrated offline approach was proposed by Gupta *et al* [25]. They used statement, branch coverage and D-criterion for minimizing test cases. The proposed technique detected faults efficiently but it failed to locate non-mutable faults. Non-mutable faults are the faults in statements for which mutants cannot be created. If the fault is in non-mutable statement, then suspicion value of that non-mutable statement is determined as zero using mutant based fault localization technique. Therefore, suspicion value these statements may be derived from statement based fault localization approach. This was the major drawback of their work. Another limitation is that the test suite size criterion was not considered in their work. It is an important criterion for test suite reduction. If two test cases have the equal fault detecting and locating capabilities then it is better to use a smaller test suite. The performance of minimized test suite was compared with that of complete test suite. An approach is useful only when it is properly validated by comparison with other existing approaches. Execution time analysis is also important because if the approach takes large execution time then it will not help in saving time. They did not provide any time analysis which is necessary for selecting any approach.

To overcome these limitations, an approach for selecting test cases based on statement coverage, D-criterion, and test suite size has been proposed in the current paper. For locating both mutable and non-mutable faults effectively, a hybrid of statement and mutant based fault localization technique has been utilized. The mutable faults are those faults in statements for which mutants can be created and the suspicion value of those statements is determined by mutant based fault localization approaches while the non-mutable faults are those faults in statements for which mutants cannot be created, and the suspicion value of these statement are determined using statement based fault localization technique. After assigning suspicion value, all statements are inspected according to the descending order of suspicion value. Thus, both mutable and non-mutable faults are well located by the proposed approach. To amend these criteria together, SPEA-2 [26], NSGA-2 [27] and VEGA algorithms [28] have been used. A comparison of the proposed approach has been done with two base approaches. Base approach 1 uses statement coverage, test suite size and statement partitioning for selecting test cases whereas base approach 2 uses statement coverage, test suite size and K-criterion [20] for selecting test cases. Outcomes indicate that the presented approach performed better for both considered phases compared to the two base approaches. The presented approach has been analyzed on three applications of the Defects4j database [29] and assessed for fault detecting and locating abilities. Execution time analysis of all considered approaches has also been done.

Further, section 2 consists of the necessary background and similar work available in the literature. In section 3, proposed approach has been discussed. Details of the flow of experiment have been discussed in section 4. An example to demonstrate the working of considered approaches has been discussed in section 5. Results and answers to considered research questions are available in section 6. Sections 7 and 8 discuss threats to validity and future work. The conclusions are available in section 9.

2. Background

2.1 *Test suite optimization for fault detection*

With changes in software, both modified and non-modified parts of software need to be tested again. For testing the modified part of code while regression, test case selection is done. Executing all available test cases is not a feasible solution. Thus, optimization techniques have been used for selecting test cases that are suitable for testing the modified part of code.

A test case selection approach based on maximization of sum of coverage and maximum sum of coverage has been developed by Mirarab and Akhilaghi [2]. To select test cases, greedy approach and linear programming have been used. Results depict that test cases selected by the considered approach are able to detect most of the faults.

An approach called MOPSO-CDRLS has been proposed by Luciano *et al* [3]. The proposed approach has been compared with BMOPSO-CDR and NSGA-II algorithm.

Results depict that BMOPSO-CDRLS performs better compared to other algorithms.

Test case selection approach based on statement coverage and time of execution has been proposed by Lucia *et al* [4]. For optimization, an improved NSGA-2 algorithm has been utilized. Results on applications from SIR database depict that improved algorithm provided better solutions and had better convergence rate.

Mondal and Hemmati [5] have used NSGA-2 algorithm to select test cases on the basis of code coverage, diversity of test cases and execution time. Results on Apache Ant, JTopas and Space applications, depicts that proposed approach is 25% more efficient than the bi-objective approach.

Test case classification and selection based on statement, condition and branch coverage has been done by Kumar *et al* [6] using fuzzy entropy and ACO. First, they have filtered test cases using fuzzy entropy and backward search strategy. Then test cases are selected using ACO technique. Results improve as the stages progress.

2.2 *Fault localization techniques*

Fault localization techniques are used for finding the exact location of code that is the cause of the fault. In literature, various types of fault localization techniques are there but statement based and mutant based techniques are extensively utilized. In statement-based techniques, considered component for analysis is statements of code. Suspicion value is calculated on the basis of coverage data of statements and test cases that executed successfully or failed during implementation. Some of the statement based technique available in literature are Ochiai [30], Tarantula [31], Op2 [32], Dstar [33].

In mutant based techniques, the suspicion value of statements is calculated based on the suspicion value of mutants. Various techniques are available under this category such as MUSE [34], MUSEUM [35], Metallaxis [36], Debroy and Wong [37] and DAM-FL [38]. Papadakis and Traon [36] concluded that mutant based techniques are able to perform better than statement based techniques because, in mutant based techniques, a fault that has the probability of occurrence are induced before testing in the form of mutants. On the basis of kill data of mutants, suspicion value is calculated. In Metallaxis, traditional killed mutants have been used for calculation of suspicion value whereas in DAM-FL, distinguished mutants have been used for calculation of suspicion value. In terms of fault locating ability DAM-FL [38] was better than Metallaxis [36].

2.3 *Test suite optimization for fault localization*

For locating faults accurately whole test case data is necessary. But using data of all test cases would require more effort in terms of time and resources. To overcome this issue various techniques have been proposed for fault localization where less effort is required by testers. Baudry *et al* [13] have proposed a criterion called "test for diagnosis" which suggests that small dynamic basic blocks should be covered instead of a large one. Results depicted that with smaller basic blocks coverage better fault localization results were available compared to that of code coverage based technique. RAPTOR technique based on the idea of decreasing ambiguity groups has been proposed by Gonzalez *et al* [14] for test prioritization while fault localization. It is based on the principle that there should be a balance in coverage of all test cases. It is an offline technique so it does not require the final results of test cases. Partitioning is among the widely used techniques in fault localization. Hao *et al* [15] have minimized test cases based on statement partitioning and they achieved better results in comparison to tests reduced based on statement coverage. An online fault localization technique based on coverage and path vector data is developed by Dandan *et al* [16]. It located faults effectively even with a test suite of smaller size. A metric based on density-diversity and uniqueness principles is developed by Perez *et al* [17] and they have called it DDU approach. It is able to locate faults with better accuracy compared to the approach based on branch coverage.

2.4 *Similar work*

A unified test suite minimization approaches have been presented by Vidacs *et al* [39] for dealing with both considered phases. For detecting faults, they have minimized test suite based on additional code coverage and for locating faults they have minimized test suite based on statement partitioning. After that, they have merged the two test suites to have a final test suite suitable for both the phases. For optimization, greedy technique has been used by them. Results on project space, GCC and Webkit depict that with partitioning the fault localization results have improved.

Statement coverage and statement partitioning have been used by Geng *et al* [40] also for selecting test cases for both phases. Instead of merging separate test suites suitable according to each criterion, they have optimized both criteria together using NSGA-2 algorithm and got one test suite that helped in both considered phases.

Approach named MOTSD has been developed by Correia *et al* [41]. For optimization, history coverage and DDU metric adequacy criteria have been utilized. Results on project Out_Systems depict that MOTSD had better fault detecting and locating capability in comparison to coverage based reduced test suite. But compared to whole test suite, test suite selected by MOTSD had very few failing test cases. This was one of drawbacks of MOTSD.

An integrated approach for test case prioritization has been proposed by Yoo *et al* [42]. First, they have prioritized tests for the detection of faults. After getting one fault detecting test, test cases are ordered such that it helps in improved fault localization. Thus, very few works are available in literature where united effort is made for both considered phases.

2.5 *Motivation for selecting base approaches for comparison*

For comparison of approaches, it is necessary that all should be of the same type. As discussed in section 1 that there are two types of fault localization techniques: online and offline. It would be unjustified to compare an offline technique with online technique as in online technique information of failing and passing test cases is available beforehand. Keeping this perspective for comparison, it was found that from similar work available in subsection 2.4, only two approaches proposed by Vidacs *et al* [39] and Geng *et al* [40] were applicable for comparison. Geng *et al* [40] were motivated by the work of Vidacs *et al* [39] in proposing an integrated approach. They have considered the same criteria as in the work of Vidacs *et al* [39] and one more additional criterion i.e., test suite size. Another modification is that instead of uniting separate test suite for fault detection and localization they have used the NSGA-2 algorithm for selecting test cases. This way Geng *et al* [40] have improved the approach proposed by Vidacs *et al* [39]. Thus, the approach proposed by Geng *et al* [40] is considered as base approach 1. In base approach 1, adequacy criteria used for optimization are the same as that used by Geng *et al* [40] i.e., statement coverage, statement partitioning, and test suite size. Ochiai formula has been used for suspicion value calculation in current work as well as in work by Geng *et al* [40]. Thus due to this reason, it helps in comparison of the proposed approach and base approach 1.

The approach by Correia *et al* [41] is not used for comparison because it is an online approach. It uses previous coverage history and DDU metric that requires prior information of failing test cases whereas the proposed approach in current work is offline technique. Thus, it is unjustified to compare these two approaches. Similarly, approach by Yoo *et al* [42] is an online approach as it first needs to find out one fault detecting test case. So, it has also not been considered for comparison.

Base approach 2 is also proposed in the current paper for comparison with the main proposed approach. It uses statement coverage, test suite size, and traditional mutation adequacy criteria. The traditional mutation adequacy criterion (K-criterion) has been used for fault localization in Metallaxis [36]. For suspicion value calculation they have used the Ochiai formula as used in the current work. With K-criterion, statement coverage and test suite size have been used so that it may be justified that in the proposed approach results are not due to statement coverage but due to D-criterion. So, to balance out and use a hybrid of SBFL and MBFL technique, base approach 2 consists of statement coverage, test suite size and K-criterion for selecting test cases.

2.6 *Partitioning*

In partitioning, the considered element of code is divided into equivalent classes. Elements that are a member of distinct class/partition are said to be distinguished from each other. For locating faults mainly two elements are considered: statements and mutants.

2.6.1 *Statement partitioning:*
In statement partitioning, test coverage data of statements is used for dividing them into partitions. Statements covered by the same test cases belong to the same partitions. Thus, statements of different partitions are covered by different test cases and they are considered as distinguished from each other. Definition of statement partitioning explained in strategy 1 by Hao *et al* [15] has been considered as the base for study. According to them two statements from a set of statements S = $\{s_1, s_2 \ldots s_n\}$ are indistinguishable from each other if each test in test suite T, either executes or not execute both the statements at all. Thus, even if one test case from the test suite executes differently on statements then they are distinguished from each other. Thus, statement partitioning by test suite T may be defined as follows in Eq. (1).

$$\mathit{Statement\ Partitioning}\ (T) = \frac{\mathit{No.\ of\ unique\ partitions\ possible\ by\ T}}{\mathit{Total\ No.\ of\ unique\ partitions}} \quad (1)$$

2.6.2 *Mutant partitioning:*
The idea of mutant partitioning is similar to statement partitioning but the difference is that considered component is mutants. It has been developed by Shin *et al* [20] and they call it "D-criterion". In their work mutant are distinguished/partitioned from each other based on their kill details. Mutants that are killed by same test cases will be in one partition. Thus, mutants that belong to different partitions are distinguished from each other. Partitioned mutants are helpful because they are capable of covering the diverse behavior of a program and redundant mutants

Table 1. Coverage information of killed mutants.

Tests	A	B	C	D
1	1	1	1	1
2	0	1	0	1
3	0	0	1	1

are removed. To explain the concept, coverage details of mutants are available in table 1. Table 2 consists of mutant partitions, K-score and D-score by the considered test suite. P_o is the original program. K-score defines the number of mutants killed by the test suite, T. It is computed using the formula in Eq. (2). D-score defines the number of mutants diversified by test suite. It has been discussed in detail in section 3. It may be observed from table 2 that as the number of test cases increases, D-score also improves whereas by all test suites K-score is 1. It may happen that by K-criterion adequate test suite {1}, a fault is left undetected. Thus, it is better to use D-criterion because if a fault is detected then only it would be located.

$$K\!-\!score(T) = \frac{No.\ of\ mutants\ killed\ by\ (T)}{Total\ No.\ of\ killed\ mutants\ by\ whole\ test\ sulte} \tag{2}$$

3. Proposed integrated approach for multi-objective test case selection

An integrated test case selection approach has been developed for both considered phases. Test cases have been selected on the basis of maximization of statement coverage and D-criterion and minimization of test suite size. A mixture of statement and mutant based adequacy criteria has been considered because by this variety of elements will be covered and it may strengthen the selected test suite. Statement coverage has been used by many researchers for selecting test cases that assist in both fault detection and localization. D-criterion has been the choice of Shin *et al* [20] for fault detection and Gupta *et al* [38] for fault localization. Thus, these criteria are suitable for both phases.

It is considered that mutation testing is a stronger testing technique compared to coverage based [43] because with mutation early intuition may be made of the faults that may occur. For fault localization, mutant based techniques perform better compared to statement based techniques [36] because simulated seeded faults may act like actual faults. Due to equivalent behavior with seeded faults, real faults may get detected and localized.

D-criterion is mutation testing based criteria that uses the concept of partitioning to distinguish mutants. It is already discussed that statement partitioning helps in locating faults. Thus, to analyze how D-criterion will work for both considered phases together, it has been utilized in an integrated approach. The proposed integrated approach has been compared with two base approaches discussed in section 1. For selecting test cases, SPEA-2, NSGA-2 and VEGA algorithms have been used. Further, considered objectives have been discussed in detail.

Table 2. Mutant Partitions, K-score and D-score.

Test suite	Partitioned set	K-score	D-score
{1}	{p0}, {A, B, C, D}	5/5	2/5
{1, 2}	{p0}, {A, C}, {B, D}	5/5	3/5
{1, 2, 3}	{p0}, {A}, {B}, {C}, {D}	5/5	5/5

- D-Score

D-criterion is used to partition mutants. For best detection and localization results, maximum partitioning needs to be done. D-score is calculated by dividing "the number of unique d-vectors to the maximum number of unique d-vectors". Unique d-vectors represent the partitions of mutants. So, the main aim is to maximize D-score and get a value of 1. For calculation of D-score, the formula in Eq. (3) is used. Here T represents the considered test suite and M represents the set of mutants and p_o represents original program.

$$D-score(T,M,p_o) = \frac{|\{d(TS,p_o,m)|m \in M'\}|}{|M'|} where\, M' = M \cup \{p_o\} \tag{3}$$

- Test suite size

Aim in current work is to select test cases such that both D-score and statement coverage is maximized. If two test suites have the same value for considered objectives, then it is obvious that smaller ones should be selected. Thus, one of the objectives is also to have minimum possible test cases in the selected test suite.

- Statement coverage

Statement coverage maximization helps in better fault detection and localization. Metric S_{cover}, Eq. (4), measures the percentage of statement covered by considered test suite.

$$S_{cover} = \frac{\text{Statements covered by test cases}}{\text{Total number of statements}} \tag{4}$$

After selecting test cases, the final test suite should be analyzed for fault detecting and locating capabilities. For fault detection, the widely used metric is the percentage of fault detected (FD Metric) [2, 4–9]. It is defined in Eq. (5). It is desired to have a high value of the percentage of fault detected. For all considered approaches i.e. proposed and base approaches, FD Metric is calculated.

$$FD\ Metric = \frac{\text{Total fault detected}}{\text{Total faults}} \times 100 \tag{5}$$

For evaluating fault localization effectiveness, first suspicion value of the considered element of code needs to be calculated. In the proposed approach, as the considered elements are statements and mutants, hybrid of statement and mutant based technique "MCBFL-hybrid-failover" developed by Pearson *et al* [44] has been utilized for locating faults. It suggests that suspicion value to all lines should be assigned first using MBFL technique. As non-mutable lines will be assigned zero value, so after the MBFL technique, SBFL technique should be used to assign value for these lines. Thus, instead of having zero suspicion value, the non-mutable line would have some suspicion value and will be placed accordingly.

In base approach 1, the considered element for the study is just statements of code thus SBFL technique is used for locating faults. For base approach 2, considered elements are statements and mutants. Thus, "MCBFL-hybrid-failover" is used in base approach 2 also. For all considered approaches, suspicion value is calculated using the formula in Eq. (6). Here, e represents the element of code for which suspicion has to be calculated. As discussed earlier e in this study are either statements or mutants. For statements, coverage details are used and for mutants kill details are used. Failed (e) denotes the number of test cases that have failed and covered/killed the considered element, e. Passed (e) represents test cases that have covered/killed considered element e but passed while execution on the non-mutated program. Total failed represents all test cases that have failed while execution on the non-mutated program.

$$\begin{aligned}&\textit{Suspicion value}(e)\\&=\frac{\text{Failed (e)}}{\sqrt{\text{total failed}\times(\text{Failed(e)}+\text{Passed(e)})}}\end{aligned}\tag{6}$$

After calculation of suspicion value, lines are arranged according to their suspicion value. Line having the highest suspicion value is given rank 1 and would be the first line to be examined in code. After arranging statements, score (S) [45] is calculated using the formula in Eq. (7). A large value of score (S) is desirable as it tells the percentage of statements that need not be examined for debugging fault.

$$S=\frac{\text{Total number of statements}-\text{rank of faulty line}}{\text{Total number of statements}}\times 100\tag{7}$$

4. Experimental description

To validate the efficacy of the proposed integrated approach, it is necessary to answer the following research questions:

- **RQ 1:** Which approach performs best for detecting and locating faults among all considered approaches?
- **RQ 2:** Is the acquired result because of the large size of test suite?
- **RQ 3:** What is the average execution time required by each considered approach?

To answer the above questions, the experimental set-up has been designed as illustrated in figure 1. We use the applications of Defescts4j database [29]. For gathering the statement coverage data and mutant related details COBERTURA [46] and MAJOR [47] tools have been utilized, respectively. For statement partitioning, statement coverage data is used. For mutant partitioning, D-criterion is used. All these details are used by considered integrated approaches. For selecting test cases by considered approaches three algorithms have been used. For implementing these algorithms MOEA framework [48] has been used. After selecting test cases, they are evaluated for both fault detecting and locating capabilities.

4.1 *Subject faults*

The experiment has been conducted on three applications: chart, time and lang of Defects4j database. As the approach is proposed for selecting test cases that may be used before regression testing only modified and loaded classes of applications have been utilized. Changed classes are those that had faults and test cases that helped in detecting faults are triggering tests. Class loaded while the implementation of fault detecting tests is loaded class. Each application had different versions. In total 118 versions are there. From these versions, analysis of three versions, two from time and one from chart was not available. Thus, they have not been included in the study. Thus finally 115 versions were left for the study. Details about considered subjects in study are provided in table 3.

4.2 *Test suites*

For conducting the study, relevant test cases associated with each application version have been utilized. Relevant test cases are test cases that load at least one of the changed or loaded classes. Only relevant test cases have been considered because test case selection is performed. Test case selection in regression testing is done only for the changed classes. Thus, for this reason, relevant test cases are contemplated as whole test suite.

4.3 *Parameter setting of algorithms*

While optimization it is necessary that parameters discussed in table 4 should be set properly otherwise it would be difficult to obtain an optimal solution. The value of each parameter has been provided in table 4.

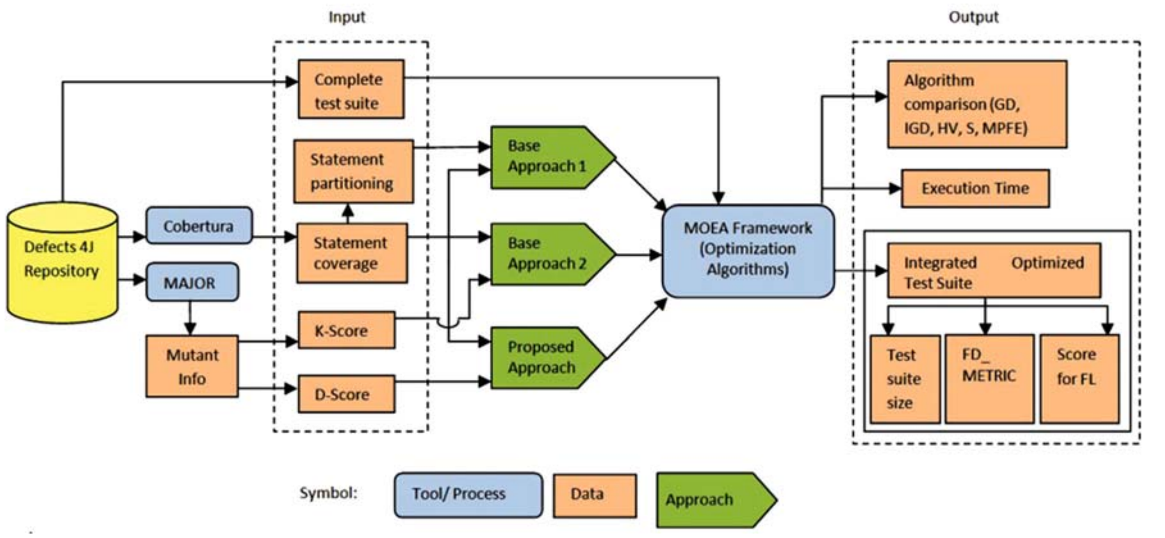

Figure 1. Experimental process for implementing all the considered approaches.

Table 3. Subject details.

Subjects	Size (LOC) (modified + loaded classes)	Years of project	No. of test cases	Size of selected test suite by proposed approach
Chart	[285–25,164]	[2007–2015]	[5–559]	[5–291]
Time	[175–7558]	[2010–2013]	[422–4102]	[243–2001]
Lang	[26–2817]	[2006–2013]	[10–836]	[6–456]

Table 4. Algorithm parameter settings.

Parameter	Setting
Chromosome size	Total number of tests
Population size	Larger than the length of chromosome
No. of generation	5000
Crossover function	Single point crossover
Mutation function	Bit flip mutation
Stopping criteria	No. of evaluations or threshold value (α) of objectives. α value of statement coverage = statement coverage by whole test suite and α for d-score=1.

4.4 *Metrics for performance comparison of algorithms*

To evaluate which algorithm is most suited for solving the considered problem, the following metrics have been considered:

- **Hyper-volume (HV)** [49]: For a set of objective function vectors Y = {F(x_1)... F (x_n)}, hyper-volume is the volume of the objective space dominated by Y. A higher value of hyper-volume is desired.
- **Generational distance (GD)** [50]: It tells the distance between the obtained Pareto front and true Pareto front. Its lower value is desired.
- **Inverted generational distance (IGD)** [50]: It measures both convergence and diversity of solutions. It is complementary to the GD metric and measures distance from true Pareto front to unknown Pareto front in solutions. Its lower value is desired.
- **Maximum Pareto front error (MPFE)** [51]: It measures the largest distance between any individual in the approximation front and the corresponding closest vector in the true Pareto front. Its lower value is desired.
- **Spacing (S)** [52]: It measures the relative distance between consecutive solutions of a non-dominated set of solutions. The algorithm which finds non-dominated solutions having smaller spacing is better.

For computing these metrics, a class called analyzer available in the MOEA framework has been used. If algorithms performance for the considered metric is different from other algorithms then analyzer tells by itself. For computing GD, IGD and MPFE a true Pareto front is required.

The solution of multi-objective optimization (MOO) has a strong association with the nature of the problem and way of formulation of the problem along with a wide choice of parameters and operating circumstances. In the search space driven approach, the MOO problem has a set of solutions and these set of optimal solutions also represents trade-off among the considered different objectives. The set of Pareto optimal solutions is called a Pareto Optimal Front (PFO) and it is a set of non-dominated solutions. These solutions are not superior in all objectives. These solutions are better in one objective and not worse in other objectives. Therefore, the foremost aim of MOO algorithms is to explore the search space for the identification of a precise approximation of the exact or true Pareto optimal solutions. True Pareto front is the Pareto front that is able to exactly solve a problem and composed of optimal solutions that converged with uniform coverage of all objectives [53]. It may be constructed using an exhaustive search i.e., by applying an enumerated approach that checks all possible solutions. Since it is not always feasible, in most cases, the true Pareto front is unknown, thus approximated Pareto fronts are built and used. A detailed explanation of the true Pareto front can be found in the work by Coello *et al* [50].

The problem considered in current work is new and complex thus a true Pareto front is not available for it. So, a Reference Pareto Frontier has been created from the best-known approximation set that consists of all Pareto optimal solutions produced by the optimization algorithms. As the three considered algorithms are stochastic, 20 seeds are used to run each algorithm.

5. Example

In this section a working example for application Chart, version 16 has been evaluated with all considered approaches. Results show that when lines suspicion value was calculated using the proposed approach, better fault detection and localization results were obtained compared to considered base approaches. Table 5 consists of a line's suspicion value when the test suite is selected using base approach 1. Similarly, tables 6 and 7 represent the same for base approach 2 and the proposed approach. As the number of lines in code are 100. So, only the first 10 suspicious lines have been included in tables. The fault was in line number 208. So, it can be seen in tables 5, 6 and 7 that the rank of faulty line was 2, 3, and 1 by base approach 1, 2 and proposed approach respectively. Score by base approach 1, base approach 2 and proposed approach was 98%, 97%, and 99%. These results show that the proposed approach is best among all considered approaches.

Table 5. Rank and suspicion value of line when base approach 1 is used.

Line no.	Line suspiciousness	Rank
208	0.66	2
156	0.59	5
150	0.59	5
163	0.59	5
207	0.66	2
342	0.31	7
338	0.31	7
335	0.21	10
573	0.21	10
789	0.21	10

6. Results

6.1 *Evaluation of all approaches for fault detecting and locating ability*

In order to compare the proposed approach with base approach 1 and base approach 2, FD metric and Score metric (S) discussed in section 3 have been used. For each application version, these metrics were calculated based on the selected test suite by all three approaches. The proposed approach was able to detect 96.50% of faults, base approach 1 was able to detect 73.04% of faults and base approach 2 was able to detect 88.69% of faults. For all considered applications, number of detected faults by each considered approach is provided in figure 2.

All considered approaches have been evaluated for fault locating capability on the basis of Score metric (S). To validate the effectiveness of all approaches, the score value calculated by each considered approach has been compared with that of the whole test suite. Score achieved by a whole test suite is the best achievable score. Thus, the difference in score value of considered approaches and the whole test suite is computed. A larger difference in score values means the considered approach is less effective. Figures 3, 4 and 5 depict the difference in score value by all considered approach for chart, time and lang applications, respectively. In these plots, it may be clearly seen that the proposed approach had the least difference value. The maximum and average difference value of all considered approaches for each application is available in table 8. A large value of difference is mostly for those cases, where the fault was not detected. The width of each plot represents the number of application-versions that had the corresponding value of difference on the Y-axis. Even if a large value of difference is there by the proposed approach it may be observed that the width of plot is very small. So this clearly depicts that large difference in score value by proposed approach is for few projects only whereas by base approach 1 and 2 width of plots is larger. Average value of

Table 6. Rank and suspicion value of line when base approach 2 is used.

Line no.	Mutant (suspicion value)	Line suspiciousness	Rank
208	70-(0.44)71-(0.79)72-(0.77)	0.79	3
156	17-(0.59)18-(0.59)20-(0.59)22-(0.59)	0.59	5
150	3-(0.59)4-(0.59)6-(0.59)7-(0.59)9-(0.59)11-(0.59)	0.59	5
163	24-(0.79)26-(0.0)27-(0.64)28-(0.64)29-(0.79)30-(0.0)31-(0.79)	0.79	3
207	67-(0.0)68-(0.79)69-(0.63)	0.79	3
342	159-(0.44)161-(0.44)162-(0.44)164-(0.44)166-(0.44)	0.44	8
338	152-(0.44)153-(0.44)155-(0.44)157-(0.44)	0.44	8
335	148-(0.44)150-(0.44)	0.44	8
573	398-(0.31)399-(0.31)	0.31	10
789	510-(0.31)512-(0.31)	0.31	10

Table 7. Rank and suspicion value of line when proposed approach is used.

Line no.	Mutant (suspiciousness)	Line suspiciousness	Rank
208	70-(0.28)71-(0.73)72-(0.81)	0.81	1
163	24-(0.73)26-(0.0)27-(0.61)28-(0.61)29-(0.73)30-(0.0)31-(0.73)	0.73	3
207	67-(0.0)68-(0.73)69-(0.57)	0.73	3
150	3-(0.61)4-(0.61)6-(0.61)7-(0.61)9-(0.61)11-(0.61)	0.61	4
156	17-(0.57)18-(0.57)20-(0.57)22-(0.57)	0.57	6
573	398-(0.47)399-(0.47)	0.47	6
338	152-(0.40)153-(0.40)155-(0.40)157-(0.40)	0.4	9
335	148-(0.40)150-(0.40)	0.4	9
342	159-(0.40)161-(0.40)162-(0.40)164-(0.40)166-(0.40)	0.4	9
296	128-(0.28)129-(0.28)	0.28	10

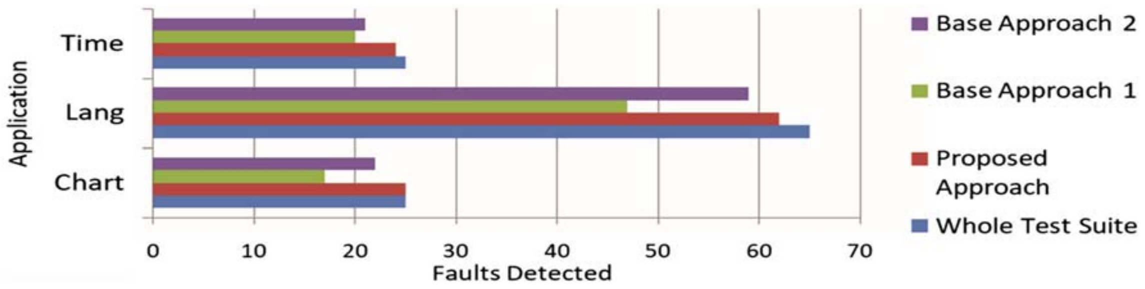

Figure 2. No. of detected faults for each application by each approach.

difference by proposed approach is also least among all considered approaches. So, this clearly indicates that the proposed technique was best among all considered techniques for locating faults.

Proposed approach in current paper performed better in terms of fault localization of non-mutable faults compared to approach by Gupta *et al* [25]. It their work non mutable statements got suspicion value of zero whereas in present work, non mutable statements are assigned suspicion value according to "MCBFL-hybrid-failover" technique. In a few versions where the fault was in non-mutable lines, base approach 1 performed better compared to the proposed approach and base approach 2. As base approach 1 is based only on statements and suspicion value of lines is directly calculated instead of using mutants, it performed better for those non-mutable faults. Otherwise, for most of the versions, the proposed approach performed better than base approach 1 and base approach 2.

The proposed approach had better fault localization scores in most of the applications because test suites

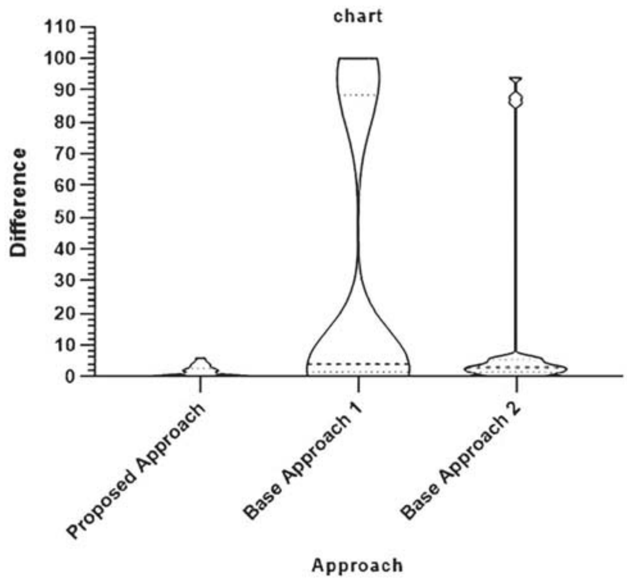

Figure 3. Difference in Score values for chart application.

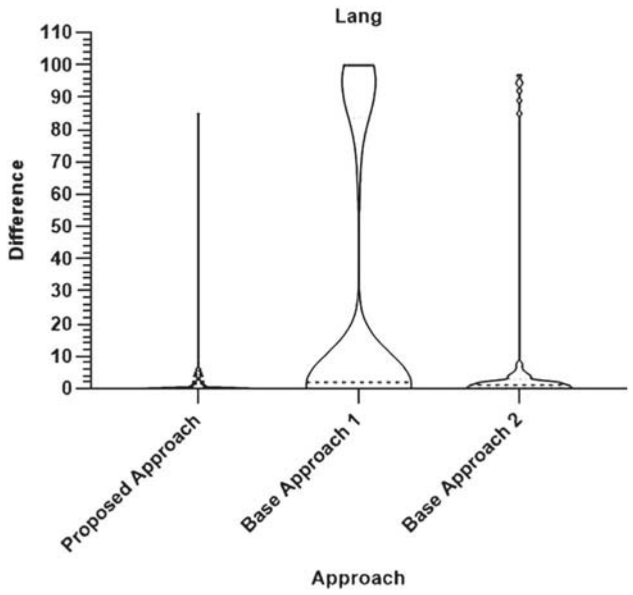

Figure 5. Difference in Score values for lang application.

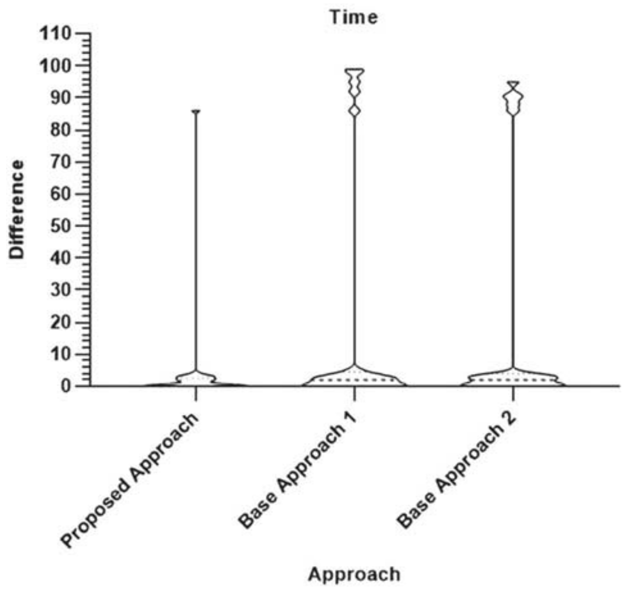

Figure 4. Difference in Score values for time application.

selected by it had most of the fault triggering test cases. Figures 6, 7, and 8 depict the number of fault triggering test cases by all considered approaches for applications chart, time and lang, respectively. These figures clearly show that the proposed approach had the largest number of fault triggering test cases among all considered approaches.

Thus, it may be concluded that combination of statement coverage and distinguished mutants is capable of selecting most fault triggering test cases as well as is able to maintain diversity among test cases. Due to these reasons, it is capable of detecting and locating faults very well.

6.2 *Analysis of size of selected test suite by each considered approach*

The proposed technique has the largest number of selected test suite among all considered approaches. But still, the difference between test suite sizes among all approaches is not very large. The reduction percentage in test suite size from that of the whole test suite has been evaluated for each considered approach. Figure 9 represents the percentage of reduction in all approaches for chart application. Figures 10 and 11 represent the same for time and lang applications. Reduction percentage that maximum versions of considered applications have by all considered approaches is presented in table 9. There is not much difference in the reduction percentage of all considered approaches. Thus, it may be concluded that good results are not because of the size of test suite but because of the adequacy criteria on basis of which test cases have been selected. The average percentage of reduction for all applications by the proposed approach, base approach 1 and base approach 2 is 54%, 59%, and 56% respectively. 54% reduction is a fair reduction percentage thus it may be concluded that results are not because of the large size of the test suite rather than it is due to used adequacy criteria.

6.3 *Execution time of approaches*

It is important to compare approaches on the basis of execution time. Table 10 depicts the average time for each considered approach. Execution time is dependent on the number of test cases and the numbers of components under study i.e., statements and mutants. As the number of test cases and components under study increases, execution

Table 8. Maximum and average difference value of all considered approaches.

Application	Proposed approach (%)		Base approach 1 (%)		Base approach 2 (%)	
	Maximum	Average	Maximum	Average	Maximum	Average
Chart	6	1	100	29	96	13
Time	86	4	100	20	96	19
Lang	85	2	100	25	97	10

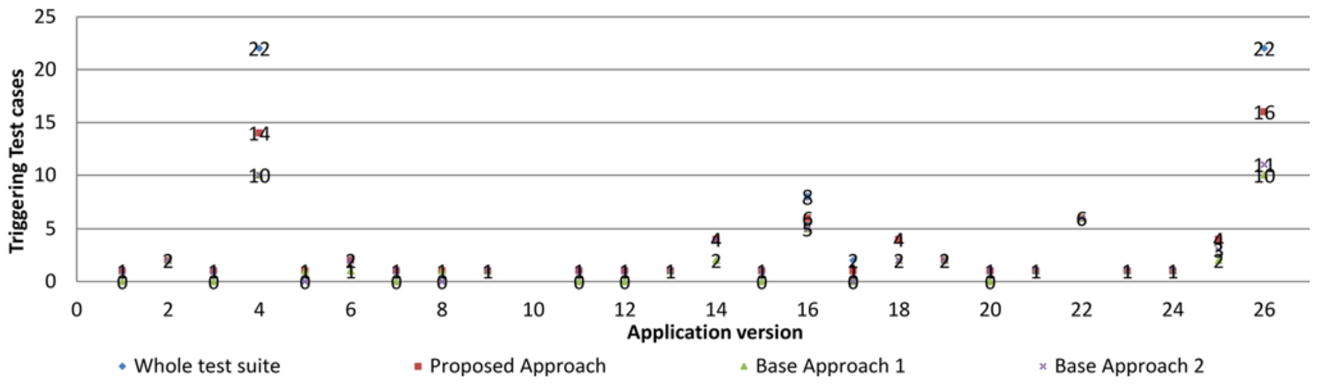

Figure 6. Number of triggering test cases by each considered approach for chart application.

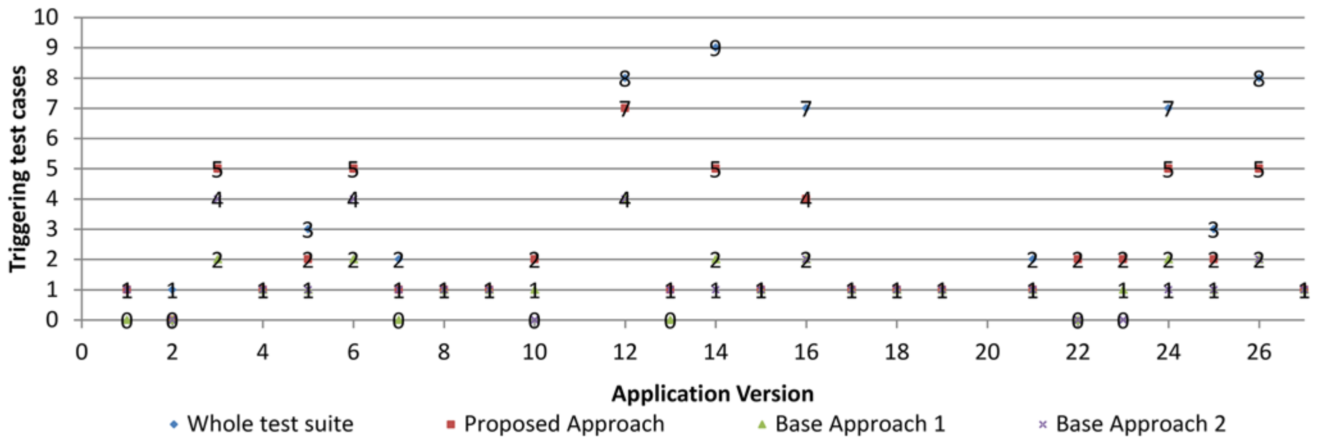

Figure 7. Number of triggering test cases by each considered approach for time application.

time also increases. Thus, different application-versions will have a wide range of execution time. For this reason, the average value has been calculated. Base approach 1 requires the least execution time because in this case, the considered component for the study is statements only. In base approach 2 and the proposed approach as the considered components for study are both statements and mutants, they take more time compared to base approach 1. As the proposed approach needs to distinguish mutants also, it requires little more time compared to base approach 1. But the extra time utilized by the proposed approach is worth for the acquired fault detection and localization results.

6.4 *Performance comparison of algorithms*

Results after the execution of analyzer for all considered algorithm is available in table 11. For each algorithm minimum, median, average and maximum value for all considered metrics has been provided. From these values, it may be concluded that in terms of HV and S all considered

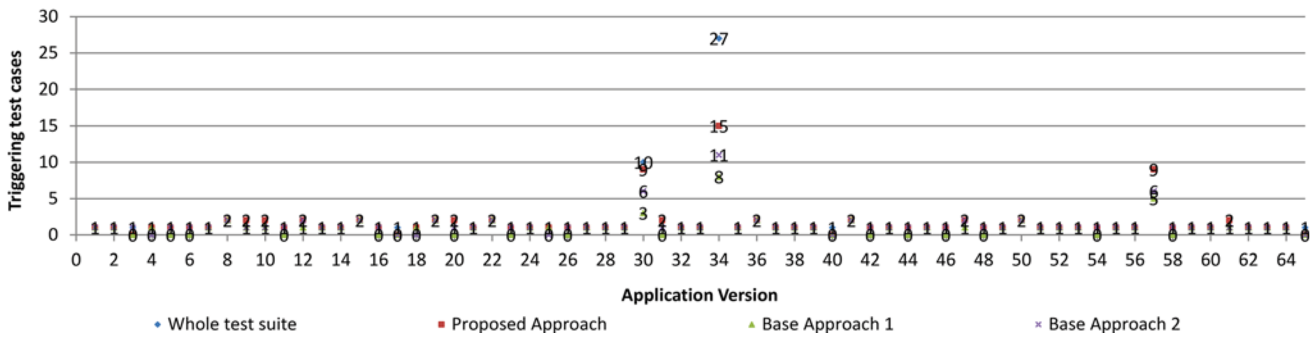

Figure 8. Number of triggering test cases by each considered approach for lang application.

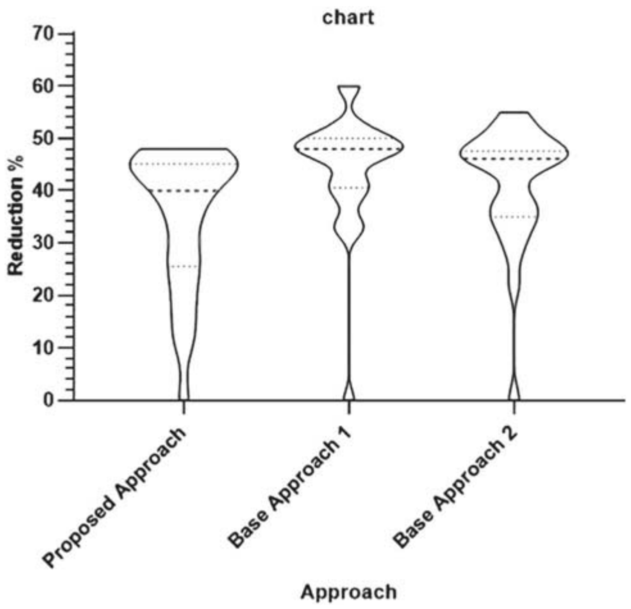

Figure 9. Test suite reduction% for chart application.

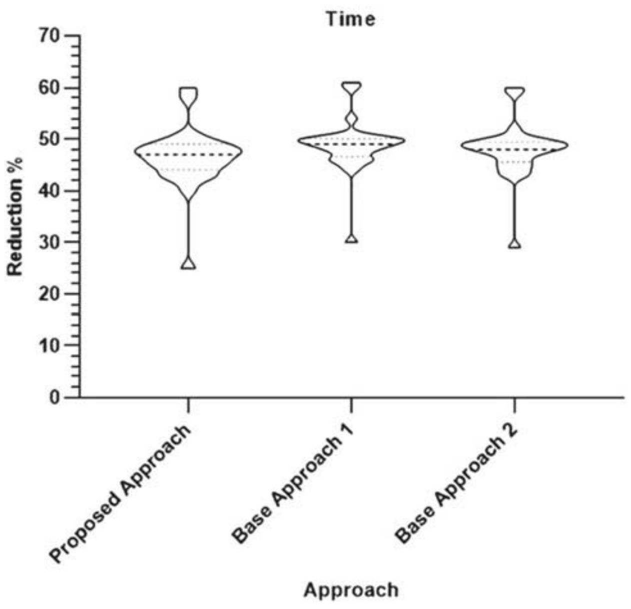

Figure 10. Test suite reduction% for time application.

algorithms gave similar results. In terms of GD, IGD, and MPFE, VEGA algorithm performed the worst and NSGA-II and SPEA-II algorithms performed quite equally. NSGA-II was better than SPEA-II for MPFE whereas SPEA-II was slightly better in terms of GD and IGD compared to the NSGA-II algorithm. Thus, from all these results it may be concluded that SPEA-II and NSGA-II are quite suitable for solving test case optimization problem considered in this paper.

7. Threats to validity

Many factors affect the validity of our conducted experiment. The most important factor is that the study has been conducted on applications of Defects4j database. Another benchmark repository such as SIR repository may be used to validate the considered approaches in current work. Mutation testing is involved in the conducted experiment and we have used widely accepted mutation operators. Results may vary with different mutation operators. Test cases have been used from developer written test suite. From these test cases, all mutants are not killed. This affects the K-score and D-score value. Thus, if the test suite is improved such that it kills non-killed mutants then results may further improve.

8. Future work

In this paper, fault localization score is dependent on the ochiai formula and "MCBFL-hybrid-failover" technique that are utilized for fault localization. Other formulas such as tarantula, OP-2, DSTAR etc are also available in the literature. Techniques: "MCBFL-hybrid-avg" and "MCBFL-hybrid-max" are proposed by Pearson *et al* [44].

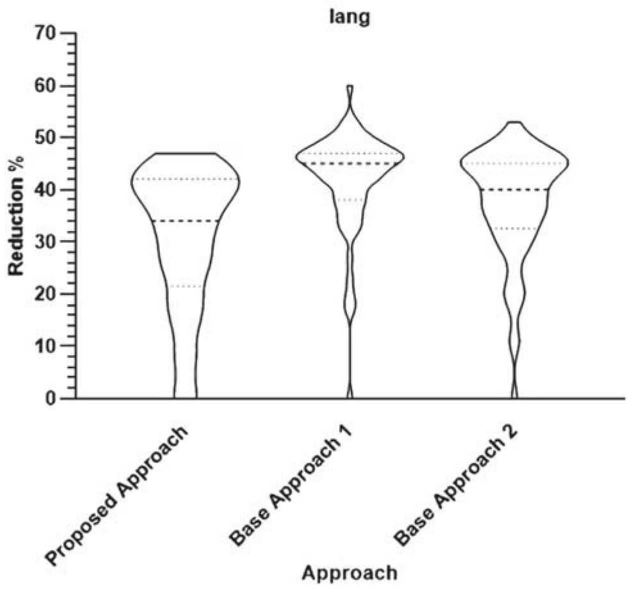

Figure 11. Test suite reduction% for lang application.

Table 9. Reduction percentage.

Application	Proposed approach (%)	Base approach 1 (%)	Base approach 2 (%)
Chart	45	48	47
Time	46	50	49
Lang	42	47	46

Table 10. Time analysis.

Approach	Time (ms)
Proposed approach	2,913,672
Base approach 1	2,292,042
Base approach 2	2,776,901

These techniques may vary the results. So, future work would be to analyze any change in fault localization score after using different formulas and techniques.

9. Conclusion

Integrated test case selection for fault detection and localization is rarely done and these two phases are considered different. In the current paper, approach has been developed to unite selection of test cases such that they are suitable for both considered phases. The proposed approach has been compared with two base approaches and results show that the proposed approach is capable of detecting 23.46% and 7.81% more faults compared to base approach 1 and base approach 2. In terms of locating faults, proposed approach was best among all approaches and had score value comparable to that of the whole test suite.

Table 11. Performance comparison of considered algorithms.

Metric	NSGA-II	SPEA-II	VEGA
(HV)			
Min	0.1740	0.1004	0.0019
Median	0.3159	0.1520	0.1034
Average	0.2945	0.2001	0.0984
Max	0.4354	0.4100	0.2463
(GD)			
Min	0.2590	0.1689	2.2608
Median	0.7620	0.6945	2.9481
Average	0.8734	0.7859	3.0019
Max	1.6879	1.6002	3.9537
(IGD)			
Min	0.4617	0.4013	1.5464
Median	0.9205	0.8776	4.0170
Average	1.5412	1.3796	2.9651
Max	2.7859	2.1051	5.7717
(MPFE)			
Min	0.0258	1.2671	3.6786
Median	1.8582	4.7731	6.4193
Average	2.1670	3.3073	5.2150
Max	4.8043	5.8841	7.8086
(S)			
Min	0.0059	0.0089	0.0100
Median	0.0255	0.0276	0.0522
Average	0.0279	0.0310	0.2799
Max	0.889	0.954	0.7350

List of symbols

D-criterion	Diversity aware mutation adequacy criteria
NSGA-II	Non dominated sorting genetic algorithm
MOO	Multi-objective optimization
HV	Hyper volume
GD	Generational distance
IGD	Inverse generational distance
MPFE	Maximum pareto front error
T	Test Suite
M	Mutant set
S	Score

References

[1] Boehm B W 1984 Software engineering economics. *IEEE Transactions on Software Engineering* 1: 4–21

[2] Mirarab S, Akhlaghi S and Tahvildari L 2011 Size-constrained regression test case selection using multi-criteria optimization. *IEEE Transactions on Software Engineering* 38: 936–956

[3] De Souza LS, Prudêncio RB and Barros FDA 2014 A hybrid binary multi-objective particle swarm optimization with local search for test case selection. In: *IEEE 2014 Brazilian Conference on Intelligent Systems*, pp. 414–419

[4] De Lucia A, Di Penta M, Oliveto R and Panichella A 2012 On the role of diversity measures for multi-objective test case selection. In: *2012 7th International Workshop on Automation of Software Test (AST)*, pp. 145–151

[5] Mondal D, Hemmati H and Durocher S 2015 Exploring test suite diversification and code coverage in multi-objective test case selection. In: *2015 IEEE 8th International Conference on Software Testing, Verification and Validation (ICST),* pp. 1–10

[6] Kumar M, Sharma A and Kumar R 2015 An empirical evaluation of a three-tier conduit framework for multifaceted test case classification and selection using fuzzy-ant colony optimization approach. *Software: Practice and Experience* 45: 949–971

[7] Panichella A, Oliveto R, Di Penta M and De Lucia A, 2014 Improving multi-objective test case selection by injecting diversity in genetic algorithms. *IEEE Transactions on Software Engineering* 41: 358–383

[8] Turner A J, White D R and Drake J H 2016 Multi-objective regression test suite minimization for mockito. In: *International Symposium on Search Based Software Engineering Springer, Cham*, pp. 244–249

[9] Pradhan D, Wang S, Ali S and Yue T 2016 Search-based cost-effective test case selection within a time budget: An empirical study. In: *Proceedings of the Genetic and Evolutionary Computation Conference 2016*, pp. 1085–1092

[10] Yoo S and Harman M 2010 Using hybrid algorithm for Pareto efficient multi-objective test suite minimization. *Journal of Systems and Software 83*: 689–701

[11] Marchetto A, Islam M M, Asghar W, Susi A and Scanniello G 2015 A multi-objective technique to prioritize test cases. *IEEE Transactions on Software Engineering* 42: 918–940

[12] Agrawal A P and Kaur A 2018 A comprehensive comparison of ant colony and hybrid particle swarm optimization algorithms through test case selection. In: *Data engineering and intelligent computing Springer, Singapore,* pp. 397–405

[13] Baudry B, Fleurey F and Le Traon Y 2006 Improving test suites for efficient fault localization. In: *Proceedings of the 28th international conference on Software engineering*, pp. 82–91

[14] Gonzalez-Sanchez A, Abreu R, Gross H G and van Gemund AJ 2011 Prioritizing tests for fault localization through ambiguity group reduction. In: *2011 26th IEEE/ACM International Conference on Automated Software Engineering (ASE 2011),* pp. 83–92

[15] Hao D, Xie T, Zhang L, Wang X, Sun J and Mei H 2010 Test input reduction for result inspection to facilitate fault localization. *Automated Software Engineering* 17: 5–34

[16] Dandan G, Tiantian W, Xiaohong S and Peijun M 2013 A test-suite reduction approach to improving fault-localization effectiveness. *Computer Languages, Systems & Structures* 39: 95–108

[17] Perez A, Abreu R and van Deursen A 2017 A test-suite diagnosability metric for spectrum-based fault localization approaches. In: *2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE)*, pp. 654–664

[18] Yu Y, Jones J and Harrold M J 2008 An empirical study of the effects of test-suite reduction on fault localization. In: *2008 ACM/IEEE 30th International Conference on Software Engineering*, pp. 201–210

[19] Gupta N, Sharma A and Pachariya M K 2019 An insight into test case optimization: ideas and trends with future perspectives. *IEEE Access* 7: 22310–22327

[20] Shin D, Yoo S and Bae D H 2017 A theoretical and empirical study of diversity-aware mutation adequacy criterion. *IEEE Transactions on Software Engineering* 44: 914–931

[21] Gong L, Lo D, Jiang L and Zhang H 2012 Diversity maximization speedup for fault localization. In: *Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering*, pp. 30–39

[22] Zhang X Y, Towey D, Chen T Y, Zheng Z and Cai K Y 2015 Using partition information to prioritize test cases for fault localization. In: *2015 IEEE 39th Annual Computer Software and Applications Conference*, pp. 121–126

[23] Xuan J and Monperrus M 2014 Test case purification for improving fault localization. In: *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering,* pp. 52–63

[24] Sahoo S K, Criswell J, Geigle C and Adve V 2013 Using likely invariants for automated software fault localization. In: *Proceedings of the eighteenth international conference on Architectural support for programming languages and operating systems,* pp. 139–152

[25] Gupta N, Sharma A and Pachariya M K 2020 Multi-objective test suite optimization for detection and localization of software faults. *Journal of King Saud University-Computer and Information Sciences*

[26] Zitzler E, Laumanns M and Thiele L 2001 SPEA2: Improving the strength Pareto evolutionary algorithm. *TIK-report, 103*

[27] Deb K, Pratap A, Agarwal S and Meyarivan TAMT 2002 A fast and elitist multi-objective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation* 6: 182–197

[28] Schaffer J D 1985 Multiple objective optimization with vector evaluated genetic algorithms. In: *Proceedings of the first international conference on genetic algorithms and their applications,* pp. 23–34

[29] Just R, Jalali D and Ernst M D 2014 Defects4J: A database of existing faults to enable controlled testing studies for Java programs. In: *Proceedings of the 2014 International Symposium on Software Testing and Analysis*, pp. 437–440

[30] Abreu R, Zoeteweij P, Golsteijn R and Van Gemund A J 2009 A practical evaluation of spectrum-based fault localization. *Journal of Systems and Software* 82: 1780–1792

[31] Jones J 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

[32] Naish L, Lee H J and Ramamohana rao K 2011 A model for spectra-based software diagnosis. *ACM Transactions on Software Engineering and Methodology (TOSEM)* 20: 1–32

[33] Wong W E, Debroy V, Gao R and Li Y 2013 The DStar method for effective software fault localization. *IEEE Transactions on Reliability* 63: 290–308

[34] Parejo J A, Sánchez A B, Segura S, Ruiz-Cortés A, Lopez-Herrejon R E and Egyed A 2016 Multi-objective test case prioritization in highly configurable systems: A case study. *Journal of Systems and Software* 122: 287–310

[35] Bian Y, Li Z, Zhao R and Gong D 2017 Epistasis based aco for regression test case prioritization. *IEEE Transactions on Emerging Topics in Computational Intelligence* 1: 213–223

[36] Papadakis M and Le Traon Y 2015 Metallaxis-FL: mutation-based fault localization. *Software Testing, Verification and Reliability* 25: 605–628

[37] Debroy V and Wong WE 2010 Using mutation to automatically suggest fixes for faulty programs. In: *2010 Third International Conference on Software Testing, Verification and Validation*, pp. 65–74

[38] Gupta N, Sharma A and Pachariya M K 2019 A novel approach for mutant diversity-based fault localization: DAM-FL. *International Journal of Computers and Applications* 45:1–10

[39] Vidács L, Beszédes Á, Tengeri D, Siket I and Gyimóthy T 2014 Test suite reduction for fault detection and localization: A combined approach. In: *2014 Software Evolution Week-IEEE Conference on Software Maintenance, Reengineering, and Reverse Engineering (CSMR-WCRE)*, pp. 204–213

[40] Geng J, Li Z, Zhao R and Guo J 2016 Search based test suite minimization for fault detection and localization: A co-driven method. In: *International Symposium on Search Based Software Engineering Springer, Cham,* pp. 34–48

[41] Correia D, Abreu R, Santos P and Nadkarni J 2019 MOTSD: a multi-objective test selection tool using test suite diagnosability. In: *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pp. 1070–1074

[42] Yoo S, Harman M and Clark D 2013 Fault localization prioritization: Comparing information-theoretic and coverage-based approaches. *ACM Transactions on Software Engineering and Methodology (TOSEM)* 22: 1–29

[43] Zhu H 1996 A formal analysis of the subsume relation between software test adequacy criteria. *IEEE Transactions on Software Engineering* 22: 248–255

[44] Pearson S, Campos J, Just R, Fraser G, Abreu R, Ernst M D, Pang D and Keller B 2017 Evaluating and improving fault localization. In: *2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE),* pp. 609–620

[45] Abreu R, Zoeteweij P and Van Gemund A J 2007 on the accuracy of spectrum-based fault localization. In: *Testing: Academic and Industrial Conference Practice and Research Techniques-MUTATION (TAICPART-MUTATION 2007),* pp. 89–98

[46] Cobertura, http://cobertura.sourceforge.net/

[47] Just R 2014 The Major mutation framework: Efficient and scalable mutation analysis for Java. In: *Proceedings of the 2014 International Symposium on Software Testing and Analysis*, pp. 433–436

[48] MOEA framework, http://moeaframework.org/

[49] Deb K 2001 *Multi-objective optimization using evolutionary algorithm*. New York, US: John Wiley & Sons, 16

[50] Coello C A C, Lamont G B and Van Veldhuizen D A 2007 *Evolutionary algorithms for solving multi-objective problems*. New York: Springer, 5: pp. 79–104

[51] Van Veldhuizen D A 1999 Multi-objective evolutionary algorithms: classifications, analyses, and new innovations (No. AFIT/DS/ENG/99-01). *AIR FORCE INST OF TECH WRIGHT-PATTERSONAFB OH SCHOOL OF ENGINEERING.*

[52] Schott J R 1995 Fault tolerant design using single and multi-criteria genetic algorithm optimization (No. AFIT/CI/CIA-95-039). *Air force inst of tech Wright-Patterson OH.*

[53] Mirjalili S 2016 Dragonfly algorithm: a new meta-heuristic optimization technique for solving single-objective, discrete, and multi-objective problems. *Neural Computing and Applications* 27: 1053–1073