Turkish Online Journal of Qualitative Inquiry (TOJQI)
Volume 12, Issue 8, July 2021: 3168-3184

Minimization Of Cost For Software Based Mutation Testing Using Surrogate Optimization Approach

ShobanaR[1], DrMaria priscillaG[2]

[1]Assistant Professor in Department of Computer Applications, Sri Ramakrishna College Of Arts and Science, (Autonomous), Coimbatore, Tamil Nadu, India. shobanarphd@gmail.com
[2]Professor in Department of Computer Science at Sri Ramakrishna College of Arts & Science, Coimbatore, Tamil Nadu, India. mariapriscilla@srcas.ac.in

ABSTRACT

Mutation testing is the efficient and costlier testing method for the fault identification process for the code. It is costlier one due to the different levels of operations to generate the test case scenario. But, an application should pass such type of test cases to perform in an effective manner. Several approaches were proposed to reduce the cost for the mutation testing. The approaches were genetic algorithm, multi-objective particle swarm optimization and it is based on minimizing the cost for generating the test cases. But, the minimum value cannot be determined effectively because it is based on the bounds given to the problem and its search space region. In this, the objective function for all the optimization approaches is same. Here, the cost function is based on the reachability, necessity and control parameters. The use of optimization approach is to determine the minimal value for the cost function with intensive searching which is similar to the mutation testing. The proposed multi-objective surrogate based optimization process to overcome the drawbacks of existing approaches. Because, the surrogate optimization main work is to determine the minimal value for the objective function. This property helps to reduce the cost effectively as compared to the previous approaches. Here, the cost function is based on the reachability, necessity and control parameters. The optimal value for this cost function will be determined through our proposed method and it will produce the best test case with minimal cost. The proposed method is implemented and tested using the emujava and it able to achieve high mutation score with lesser number of iterations. It also able to identify the doubtful mutants which helps to speed up the process and reduce the test cases.

Keywords: Mutation testing; necessary; reachable; control; cost; multi-objective;surrogate based optimization.

1. Preface

Software testing is an important process in the development of the software. Because, it check whether the design is meet with the requirements. It is an important action to fulfil the requirements of the user and produce a bug free application.

In software development, the testing process is performed from the starting of the application till its dispatch process. In order to ensure the details, each process in development phase is subjected to testing either in unit or in the bulk manner. The testing process will stop only if the requirements are meets, there is no bug while operating the applications, it meet up all the conditions while processing it. A tested application only provide a long term communication and attracts the user.

Testing main role is to validate the application that all the criteria or the conditions for operating the application is possible and it check the final requirements. Testing is of generally three types: white box, grey box and black box.

In white box, the tester has complete knowledge about the information and methodologies used in it. Because, it is an organized testing process and produce the less bug free application. The tester design the cases and information based on that. In black box, it is the vice versa of the white box. The tester need not require great knowledge about the application and simply check the test case scenario. The grey box which requires limited knowledge about the application to process it effectively.

Among the three testing methods, the white box testing is the high time consumption test due to the keen processing of the data. But, the end result of the testing is effective one as compared to the other testing methods. In this, Mutation testing is analysed to reduce the time for the processing is proposed.

Mutation testing is comes under the category of white box testing. Because, it is highly structure oriented based testing process. It involves more parameters for testing and time and cost consumption is high. Mutation testing is to reduce the code size and the vulnerability of the code if any changes occur to it.

Mutation testing is performed by changing the code. The code is changed by using any one of the method. First, the values used in the code can be changed. Second, the conditions used for the checking an operation is changed. Third, the parameters used are changed. In all the three methods, the code should pass the test for less bug free application by generating the different output as compared to the original code.

The mutation testing involves high cost because it creates the individual test case scenario for all the statements in the code. To automate this process, the Pit testing and Stryker tools were used. The evaluation is based on the mutation score. The number of killed mutants should be high for higher mutation score. Then, only the application is ready for the usage. The methods to use mutation test and its cost minimization is analysed in below papers.

Silva et al., (2017) studied about the role of optimization process in the mutation testing [1]. Mutation testing is highly cost oriented in terms of time and processing. Because, it creates a vast number of test cases for a problem. Hence, an automated process and optimal selection of test cases is important to reduce the cost of testing. To perform this process, the optimization algorithms like genetic, hill climbing and ant colony and other techniques were employed. Those algorithms will process based on the cost function. The cost function may be the mutant score or the killed mutants. The solution may give the optimal mutants or the operators for generating the mutants.

Huang et al (2014) utilized Particle Swarm Optimization (PSO) calculation that proposed the gathering self-movement input (SAF) administrator and Gauss transformation (G) changing idleness weight to upgrade the presentation of molecule swarm advancement (PSO) [2]. Utilizing the upgraded calculation in programming experiment, the trials show that the presenting a solitary way wellness work structure and multi-way wellness calculation of equal reasoning gives better outcomes as than the cycle time in single way test contrasted with the standard PSO and is more compelling in the age of multi-way experiment.

Li et al (2015) suggested about reducing the expense of mutation testing, and presented an algorithm for mutation test generation, and then provided few reduction rules to minimize the set of test suite that is employed for killing mutants [3] depending on formal concept analysis. In the case of mutation testing, few representative errors are intentionally seeded into the SUT (System under Testing)to generate a set of faulty programs known as mutants, and all current test cases are executed on all the mutants. Designing efficient and helpful mutation operators are one among the key challenges of mutation testing. The results proved that this technique can produce a smaller size test suite compared to other techniques. Moreover, this technique can be of some assistance to mutation testing.

Gong et al (2015) presented a powerful change execution strategy and the transformation based flaw limitation plot with the method, alluded to as Faster-Mutation-based Fault Localization (MBFL) [4]. The dynamic change execution procedure includes two improvements, which are transformation execution enhancement and experiments execution advancement. These enhancements are centered on faster processing dubiousness estimations of articulations through the dynamic change of the request for execution of freaks and experiments.

Souza et al (2016) proposed a computerized test age methodology, utilizing slope moving, for incredible transformation [5]. It gradually focuses at executing the freaks emphatically, by focusing on the proliferation of freaks', i.e., the methods for slaughtering the mutants, which are murdered pitifully however not sufficient. Moreover, the exact outcomes worried about the expense and effectiveness of this methodology over a lot of 18Cprograms. This procedure achieved a higher changes center contrasted with irregular testing, by 19,02% on a normal, and the prior presented test age methods, which disregard the proliferation of freaks', by 7,2% on a normal. The outcomes likewise demonstrate the improvement of this strategy over the prior techniques.

Prajapati et al (2016) introduced an improved information stream based Quality Assurance (QA) model for Component-based Software Engineering (CBSE) by utilizing the Ant Colony Optimization (ACO) calculation [6] for upgrading the code given for programmed age and prioritization of ideal way in choice to choice Control Flow Graph (CFG) that, thusly, prompts an improved testing stage for QA model with limited unpredictability. At that point, the new ACO based method is likewise utilized for producing the test information to meet the made arrangement of ways. The outcomes show that better testing is refined by utilizing the new ACO put together procedure with respect to segment based programming. This procedure ensures total programming inclusion with least measure of repetition.

Ma et al (2016) presents a novel plan for the execution of lesser freaks while holding simply a similar degree of proficiency as produced by transformation testing utilizing a total arrangement of freaks [7]. This procedure plays out the dynamic grouping of articulation level pitifully murdered freaks, which are foreseen to produce a similar outcome under an experiment; only one freak from each bunch is totally executed under the experiment. This procedure was actualized and its exhibit demonstrated that it productively limited the cost of transformation testing with no misfortune in the adequacy.

Bashir et al (2017) showed an enhanced genetic algorithm, which can minimize the computational expense of mutation testing [8]. At first, it introduces a new state-based and control-specific fitness function, which makes efficient use of object-oriented program features for the evaluation of a test case. After this, it does the empirical evaluation of it employing this tool implemented, eMuJava, and then performs its comparison with standard fitness function. Results indicate that even though the new fitness function yields detailed information regarding the fitness of a test case but the standard genetic algorithm is quite not capable of making use of that with efficiency in order to correct the test cases. Therefore, new two-way crossover and adaptable mutation techniques, which intelligently utilize the fitness information for generating a fitter offspring is proposed. At last the enhanced genetic algorithm with random testing, standard genetic algorithm, and EvoSuite are compared. The experimental results illustrate that this new technique can detect the optimal test cases in lesser attempts (minimizes the computational expense). Also, it can find the software bugs from doubtfully equivalent mutants and these mutants are eventually killed (maximizes mutation score).

Devorey et al., (2016) utilized a new approach for the test case generation based on the application [9]. Here, the applications where the test cases generated are smart card system. The mutants are generated for using the card system based on its features. Due to the higher level of information is used for the test case generation. It reduces the number of test cases for the testing. It is possible by generating the conditions for the mutants using the higher order features from the program as compared enumerated mutation testing process. It able to analyse the system effectively with high mutation score and minimum time for processing it.

Panichella et al., (2017) utilized the mutliobjective concept for generating the test cases for an application [10]. Here, the test cases are generated based on the concept of minimize the cost for coverage testing as fitness function Due to this, the number of test cases generated is high and effective one as compared to the traditional test case generation using multiple objective method. It able to increase the chances of killing of mutants and analyse all the possible conditions in the snippet for hacking.

The organization of the paper is arranged by explaining various techniques in mutation testing in the following section. It is followed by merits and demerits of the improved genetic algorithm and other evolutionary mutation testing in section 3. Section 4 elaborates the working surrogate based mutation testing. It is followed by implementation procedure and its discussion based on the results. Finally the paper is concluded by summary and its future extension method.

2. Literature Survey

Abuldjayan and Wedan (2018) utilized the higher order mutant concept to reduce the cost for the test case generation in the mutation testing [11]. Because, the higher order mutants are able to process the data effectively with the minimum number of test cases for a snippet. It results in the minimum computational time and cost for the mutation testing. To further reduce the computational complexity, the genetic algorithm were used for creating the optimal number of higher order mutant generation for a given snippet. It is tested on the sample code and evaluated using the mutation score. The mutation score is acted as the fitness function for the optimization process. this technique able to improve the result by 4% as compared to the other types of test case generation scenarios.

Zhang et al., (2018) utilized the prediction process for the test case generation in mutation testing [12]. Here, a classification model is built based on the test case and its corresponding nature that is it is killed or not. Based on this trained network, the future test cases for various applications can able to identify whether the corresponding mutant is able to kill or not. By this, the computational time for the test case generation and execution is reduced as compared to the traditional mutation testing. But, it able to predict accurately only for the limited number of mutants as compared to the other mutant testing process.

Chen and Zhang (2018) analysed the various approaches in minimize the test case generation using the regression test analysis [13]. Here, the survey on various approaches for the test case generation in the mutation testing process. it collects various test case scenarios using regression technique is collected. Then, those information were used as the format for the test case generation based on the regression test. This approach is able to reduce the computational time in the generation of test cases due to the forecasting and repeated analysis. But, the processing time for the test case will be high for getting high mutant score.

Ferrari et al., (2018) discussed about the various techniques utilized to reduce the mutation testing [14]. Here, the techniques which are available for reducing the computational time and cost for generating and processing the test cases using various approaches were discussed. It gives a clear idea about how the test case generation time can be reduced and what are the fitness functions that are required for reducing the test cases. It states that the test case can be generated based on the corresponding function and the fitness function is responsible for the test case analysis in mutation testing.

Khari et al., (2018) studied about the role of optimization algorithms like artificial bee colony and cuckoo search optimization in selecting the proper test cases for mutation testing is performed [15]. Here, the test cases are generated based on their property. Then, the optimal test from the suit is determined with the help of optimization algorithms. The five test suites for the case generation are boundary based, robustness based, worst case scenario, immediate worst case scenario and random test scenario. The optimization algorithms will work on the above test suites to generate the test cases for a problem automatically. The optimization helps to improve the performance and speed up the test case generation. It is highly preferred for the single level code based testing.

Carlvaho et al (2018) used the empirical study approach for analysing the pre-processor directives based mutation testing [16]. Generally, a part of the snippet only analysed but there is a chance of problem in the pre-processor directive access by creating a fault statements. This problem is overcome by using the empirical study analysis on the directives to create the test case generation and its effect on it. It is observed that this type of testing shows that some little modifications in defining the pre-processor directive will produce the same result as original. It leads to a problem in the higher stage analysis. Hence, a proper test case generation should be considered for the pre-processor drives and the smaller programs.

Ghiduk et al., (2018) proposed an approach in reducing the computational time for the higher order mutant generation [17]. Generally, the higher order mutants are framed with the help of combining the first order mutants. But, it results in high processing time and not able to detect the optimal higher order mutant. To overcome this problem, three techniques were used to reduce the higher order mutant generation time and to select the optimal mutant value. The three techniques were same category with repeated values, even-even or odd-odd, or the second category with repetition. These techniques were able to reduce the higher order mutants as compared to the combinational approach. But, the technique has to be properly selected.

Sanchez et al., (2018) utilized the evolutionary concept in the mutation testing process [18]. Usually, the mutation testing is performed by evaluating the test cases for the program by changing the function names or the operators in it. But, it leads to highly expensive one and also it causes a great drawback in the processing time. It overcome by using the searching mechanism to identify the proper mutants for the testing. It is performed with the help of evolutionary algorithms to determine the optimal mutant cases for the testing and its performance is evaluated.

Zhu et al., (2018) surveyed the role of compression techniques in one of the white box testing mechanism named mutation [19]. In this, the compression techniques were used to increase the testing process. first, the mutants and the cases were clustered using overlapping and formal concept analysis. Then, the mutation is weighted based on its reachable and necessary state. Using this, the stronger and weaker mutants are generated. Then, the weaker mutants are compressed and only the stronger mutants were used for the analysis. It able to speed up the process but, sometimes the important mutant may be destroyed due to the suppressing mechanism.

Hooda and Chillar (2018) used the optimization and machine learning approach to generate the test cases in mutation testing [20]. Because, in normal mutation testing the redundancy of test case is higher is due to the doubtful nodes which has no effect in the output in both the original and mutant program. This problem is overcome by generating the test cases using genetic algorithm. The genetic algorithm used the cross over operator to generate different possiblities of the test cases. Then, the optimal test cases are selected based on the feedback from the output using the artificial neural network. It able to reduce the redundancies but there is a chance in the redundancy due to the mutation operator. NEH-Heurestic model is reviewed in [22].

3. Existing method

Mutation testing is performed to determine the possible vulnerabilities in the program. A successful test is achieved only if it able to kill all the mutants in the program. Otherwise the testing process will be performed till it kill all the mutants. Due to this, it require high processing time and also requires high cost for the testing process. To overcome this problem, the evolutionary based testing is introduced to reduce the time for the test cases generation in the mutation testing. Several techniques like genetic algorithm, random testing and improved genetic algorithm were implemented. Those methods were utilized by only one cost function which is based on the sufficient cost. This problem is overcome by using the three types of cost function which includes all the possibilities of the test case generation in the mutation testing. This fitness function is used in the improvised version of the genetic algorithm. Here, the two way cross operator is implemented to reduce the cost for the fitness function evaluation. Those functions were able to reduce the time for the test case generation as compared to the traditional evolutionary testing mechanism. This method is tested on the java snippets for the combined programs. It has the following merits:

- It reduces all types of costs for the mutant generation.
- It requires the minimum number of iterations for processing.
- It reduces the computational overhead of the testing.

Despite of its major advantages, it requires some modifications to improve the performance better and also reduce the time for the convergence rate. Hence, in this, the surrogate based optimization is used. This optimization generally finds the minimum value for the function which helps to achieve the convergence rate faster and also to reduce the test case generations. It is explained elaborately in the following sections.

4. Proposed method

In this, the evolutionary based mutation testing is improved by finding the minimum test case generation for mutation testing using surrogate based optimization. The aim of the proposed surrogate based optimization in mutation testing is as follows:

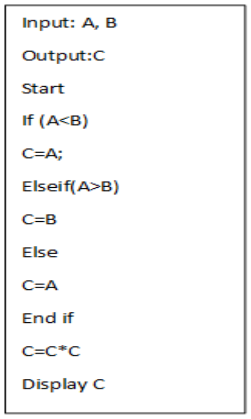
- Reduce the test case generation.
- Achieve best solution with lesser convergence rate.
- Reduce the computational cost and overhead problem.

The above goals can be achieved by finding the minimum value for the objective function. Here, the objective function is created based on the three important points in the mutation testing. Those points were as follows:

- Reachable state
- Necessary state
- Control state.

The detail explanation for the above states can be explained with the help of the following psuedocode. The psuedocode is to determine the square of a larger number between the two numbers is shown in the below figure 1.

Figure 1. Psuedocode for square of a larger number



```
Input: A, B
Output:C
Start
If (A<B)
C=A;
Elseif(A>B)
C=B
Else
C=A
End if
C=C*C
Display C
```

The above figure 1 utilize all the three states for the mutant generations. The role of each state in mutant generation is explained below:

4.1. Reachable state:

The reachable state is used for the evaluations of the conditions of the program. It is mainly used to evaluate the, if else conditions and other conditions related blocks in the program. In this, the reachable state is evaluated by selecting the, if else conditions of A and B in the figure 1. This objective function is further divided into two categories to evaluate the functions better as compared to simple reachable state functions.

The categories of the reachable state is as follows:

- Nature of state (N_s)
- Coverage of state (C_s)

Based on this, the reachable state is given using the following equation 1.

$R_s = \{ N_s, C_s\}$	(1)

1.1.1 4.1.1. Nature of state:

The nature of state is to determine the output of the, if condition in the loop. If the condition is satisfied, then the state nature will be 1 otherwise 0. It is calculated using the distance between the states as branch distance (B_{ds}). It is given in equation 2.

$N_s = \{ B_{ds}\}$	(2)

4.1.2. Coverage of state:

Then, after the state nature analysis, the distance totally covered by the parameter is considered by using the distance between the choices level and the distance between the variables in the choices.

$C_s = \{ Choice_s, B\_V_{ds}\}$	(3)

The term $Choice_s$ is the difference between the level of the states in the, if condition. The term $B_V_{ds}\}$is the distance between the parameters used in the choices.

By substituting the equations 3 and 2 in 1, the equation 1 becomes as

$R_s = \{ B_{ds}, Choice_s, B\_V_{ds}\}$	(4)

The equation 4 shows that the reachable state evaluates all the possible fault conditions and generate the test cases based on it. It helps to evaluate the system more effectively.

4.2. Necessary state:

In this, the operators used in the program is analysed. It is performed in the two modes. It is given in the following equation 5.

$$NE_s = \{ O_s, F_s \} \quad (5)$$

The two modes are as follows:

- Operator mode
- Function mode

4.2.1. Operator mode:

In this, the operators used in the program is changed to generate the test cases. It helps to evaluate the relation between the operators and the output of the program. It is given using the following equation 5

$$O_s = \{ arithmetic\ and\ logical\ operators\ (ALO)\ modified \} \quad (6)$$

4.2.2. Function mode:

In this, the parameters used in the calling of a function is changed to generate the test cases. It helps to analyse the role of parameters in a calling function. It is given in equation 7.

$$F_s = \{ parameters\ in\ calling\ function\ (CF)\ is\ modified \} \quad (7)$$

Substitute the equations 6 and 7 in equation 5, it becomes,

$$NE_s = \{ ALO_s, CF_s \} \quad (8)$$

4.3. Condition state:

In this state, the parameters which has no role in the program is determined. Even though the parameters is mentioned in the reachable mode, it has no effect on the results. Because, the test cases output for those parameters will remain same as the original code. This type of nodes are called as the doubtful nodes or suspicious nodes. It is given in the following equation 9.

$$C_s = \{ NP \} \tag{9}$$

The term NP indicates that it has no effect in the test case output. Due to this, the kill of the mutant will be reduced and the processing time will be high to kill all the mutants.

The combined objective function for the surrogate optimization is given in equation 10

$$objective\ function = \{ R_s, NE_s, C_s \} \tag{10}$$

4.4. Implementation in EmuJava using Surrogate optimization:

The steps to be used in the mutation testing using emuJava is given below:

- Initialization
- Test case generation
- Objective function evaluation
- Optimization algorithm
- Adjustable mutation test cases

1.1.2 4.4.1. Initialization:

In this, the parameters for the optimization and the tool for the mutation testing is mentioned. Here, the search agents are considered as 50 for the mutant generations.

1.1.3 4.4.2. Test case generation

In this, the test cases for each mutants is generated by changing the parameters in the program or function. In program, the variables and operators are changed which is called as selective mutation test case generation.

If the variables and operators are changed in the function, then it is called as the object oriented programming test case generation.

1.1.4 4.4.3. Objective function evaluation

The test cases generated for each mutant is evaluated using the equation 10 to determine the mutation score. The optimization algorithm is used to maximize the output with minimum test cases. The equation 10 is replaced by corresponding values and it becomes,

$$objective\ function = \{ B_{ds}, Choice_s, B\_V_{ds}, ALO_s, CF_s, C_s \} \tag{11}$$

The equation 11 shows that the proposed method is able to generate and evaluate all types of test case scenario.

4.4.4. Surrogate optimization

Surrogate optimization is mainly used for finding the minimum value for an objective function. Its main advantage is does not require any pre-process step in the data to smooth its values. The other advantage of this method is able to provide the best solution in lesser time interval.

It find out the minimum test case scenarios of mutation testing by evaluating the objective function. The evaluation of the objective function is performed with the help of following two process.

- Explore the search space area.
- Boost the process to find the minimum test case with minimum iterations.

The process in the surrogate based optimization is shown in the figure 2.

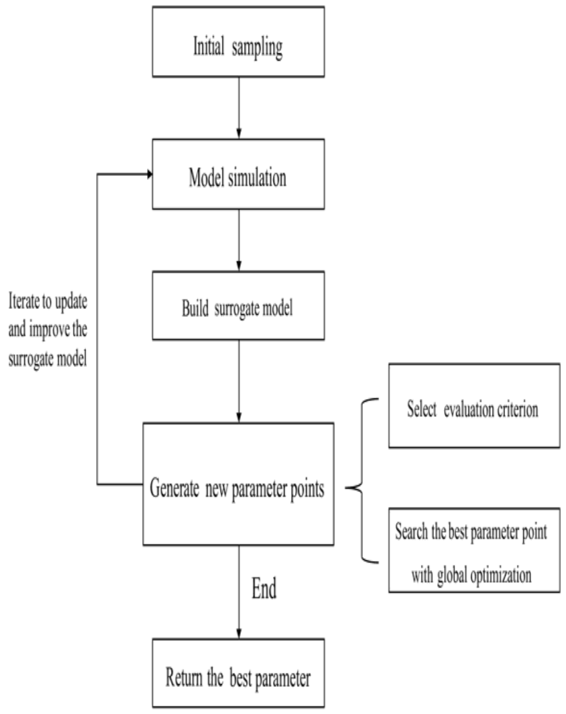

Initial sampling
Model simulation
Build surrogate model
Iterate to update and improve the surrogate model
Generate new parameter points
Select evaluation criterion
Search the best parameter point with global optimization
End
Return the best parameter

Figure 2 Surrogate optimization approach

Step 1: first the Sample points are taken from the data which are responsible for the output.

Step 2: then, using those sample points, the objective function in equation 11 is evaluated.

Step 3: then, the model is designed using the sample points and radial basis as its functions.

Step 4: again, the test cases are generated based on the model

Step 5: if (objective function is minimum)

Step 6: the optimal test case scenario is obtained

Step 7: otherwise

Step 8: repeat the step 3.

Step 9. The process is continues till it reaches the minimum objective function value.

Step 10: stop.

1.1.5 4.4.5. Adjustable mutation test cases

In this, the control state able to determine the doubtful nodes. It helps to remove those mutants and then process again for the testing. It helps to reduce the computational time and cost of mutation testing. The implementation and its results are discussed in the forthcoming sections.

5. Implementation and discussion

In this, the optimized mutation testing using surrogate based optimization is proposed to reduce the computational time and cost for the test case generation in the mutation testing. This whole process is realized using the emuJava application and by implementation of the proposed approach it is updated as emuJava version 3. It requires necessary Netbeans software and the other jar files for the implementation process and results.

The proposed approach is tested on the following snippets:

- Autodoor
- Hashtable
- Stack
- Cgpa calculator
- Calculator
- Triangle
- Binary search tree.

The proposed approach is tested on the above snippets to perform mutation testing using emuJava. In the emuJava tool, the test is processed for the 10 iterations to kill all the mutants and calculate the mutation score. The performance of the optimized mutation testing is evaluated based on the following two metrics

- Mutation score
- Detection of false statements in the snippet.

5.1. Mutation score:

Mutation score is used as a measure to analyse the algorithm based on the number of kills of the test cases. Generally, this value should be higher for an algorithm to pass the mutation testing. It is calculated using the following formula.

$$mutation\ score = number\ of\ mutants\ killed \text{ / total number of mutants} \quad (11)$$

Based on this, the proposed approach surrogate based optimized mutation testing (SBO_MT) is evaluated and compared with the existing techniques like Random technique (RT) and improved genetic algorithm (IGT) and it is shown in the below table.

Table 1. Comparison of mutation scores of proposed vs. Existing techniques

technique	RT		IGT		SBO_MT	
snippet	Iterations	Mutation score (%)	Iterations	Mutation score (%)	Iterations	Mutation score (%)
Autodoor	310	7750	140	3500	135	7751
Hash table	1052	15375	289	7225	280	8500
stack	91	26300	208	5200	200	1000
Cgpa calculator	380	2325	74	1850	65	2000
calculator	130.93	3250	157	3925	150	4000
triangle		10800	373	9325	370	9400
Binary search tree	263	6575	255	6375	350	6450

From the table 1, it is observed that the proposed surrogate based mutation testing is able to improve the mutation score with minimum number of iterations to kill the maximum number of mutants. Due to this, the number of test cases generated will also be reduced. The pictorial format of the table 1 is shown in the below figure,

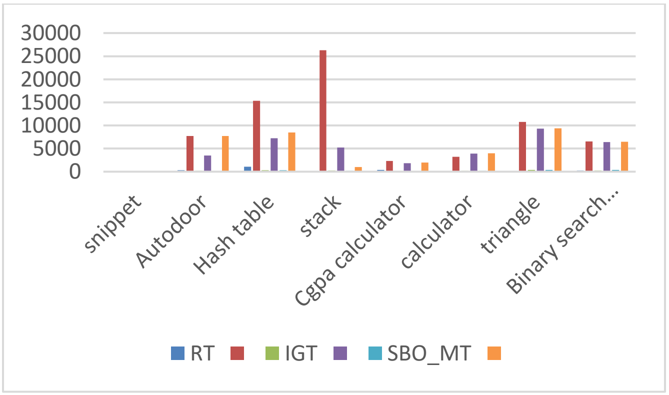


Figure 3. Comparison of mutation score versus iteration

From the figure 3, it is observed that the proposed techniques require only minimum number of iterations to kill more number of mutants as compared to the existing technique. Due to this, the computational time for creating the test cases and process for killing the mutants is reduced.

5.2. Detection of false statements in the snippet:

There is a chances of errors in the snippet which results in the non-stop processing of the data. Due to this, it repeats the loop unconditionally which results in unconditional time to kill the test cases. Hence, the test case should be able to determine such conditions effectively. Based on this, the identification of false condition based on the three states of the proposed surrogate based mutation testing and improved genetic algorithm testing is shown in the table.

Table 2. Comparison of mutation score for false statement identification

Snippet	IGT (%)	SBO_MT (%)
Autodoor	82	90
Hash table	74	80
stack	85	89
Cgpa calculator	76	82
calculator	86	91
triangle	50	52
Binary search tree	74	78

Table 2 is shown in the pictorial format in the below figure 4.

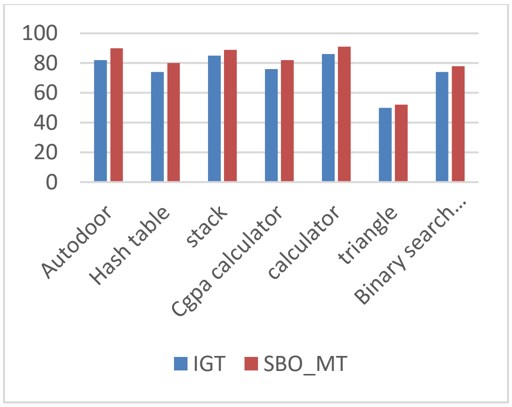


Figure 4. False condition identification

From the table 2 and figure 4, it is observed that the proposed Surrogate based mutation testing is able to identify the false conditions effectively as compared to the improved genetic algorithm.

From both the evaluations, the proposed approach surrogate based mutation testing able to improve the number of mutants killed with minimum number of iterations and it also able to determine the unconditional errors in the snippet effectively. Due to this, the proposed surrogate based optimization able to reduce the computational time, cost and overhead of the test case generation in the mutation testing.

6. Conclusion

Mutation testing is performed to determine the possible vulnerabilities in the program. A successful test is achieved only if it able to kill all the mutants in the program. Otherwise the testing process will be performed till it kill all the mutants. Due to this, it require high processing time and also requires high cost for the testing process. Several techniques like genetic algorithm, random testing and improved genetic algorithm were implemented. Those methods were utilized by only one cost function which is based on the sufficient cost. This problem is overcome by using the three types of cost function which includes all the possibilities of the test case generation in the mutation testing. Those approaches were suffer from the higher time for achieving the convergence for the complex problem and it results in the high computational time. This problem is overcome by the surrogate based optimization technique to determine the minimum number of the test case generation and lesser convergence rate for determine it. Hence, the proposed method surrogate based optimized mutation testing able to reduce the both computational overhead and the minimum iterations with high mutation score for all the simple and complex programs. Due to the high mutation score in the complex problems, the surrogate based mutation testing is best as compared to the improved genetic algorithm based mutation testing.

7. Future works

In future, the proposed optimization technique will be replaced by human or biological or swarm based optimization techniques to evaluate the performance of mutation testing.

References

1. Silva, R. A., de Souza, S. D. R. S., & de Souza, P. S. L. (2017). A systematic review on search based mutation testing. Information and Software Technology, 81, 19-35.
2. Zhang, Wei, et al. "A Method of Software Reliability Test Based on Relative Reliability in Object-Oriented Programming." Information Processing, 2009. APCIP 2009. Asia-Pacific Conference on. Vol. 1. IEEE, 2009.
3. Huang, Ming, Chunlei Zhang, and Xu Liang. "Software test cases generation based on improved particle swarm optimization." Information Technology and Electronic Commerce (ICITEC), 2014 2nd International Conference on. IEEE, 2014.
4. Li, Liping, and HonghaoGao. "Test suite reduction for mutation testing based on formal concept analysis." Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD), 2015 16th IEEE/ACIS International Conference on. IEEE, 2015.
5. Gong, Pei, Ruilian Zhao, and Zheng Li. "Faster mutation-based fault localization with a novel mutation execution strategy."Software Testing, Verification and Validation Workshops (ICSTW), 2015 IEEE Eighth International Conference on. IEEE, 2015.
6. Prajapati, Neha, and Naveen Kumar. "Data flow based quality testing approach using ACO for component based software development." Computing, Communication and Automation (ICCCA), 2016 International Conference on. IEEE, 2016.

7. Ma, Yu-Seung, and Sang-Woon Kim. "Mutation testing cost reduction by clustering overlapped mutants." Journal of Systems and Software 115 (2016): 18-30.
8. Bashir, Muhammad Bilal, and AamerNadeem. "Improved Genetic Algorithm to Reduce Mutation Testing Cost." IEEE Access 5 (2017): 3657-3674.
9. Devroey, X., Perrouin, G., Papadakis, M., Legay, A., Schobbens, P. Y., & Heymans, P. (2016, May). Featured model-based mutation analysis. In Proceedings of the 38th International Conference on Software Engineering (pp. 655-666).’]
10. Panichella, A., Kifetew, F. M., &Tonella, P. (2017). Automated test case generation as a many-objective optimisation problem with dynamic selection of the targets. IEEE Transactions on Software Engineering, 44(2), 122-158.
11. Abuljadayel, A., &Wedyan, F. (2018). An approach for the generation of higher order mutants using genetic algorithms. International Journal of Intelligent Systems and Applications, 10(1), 34.
12. Zhang, J., Zhang, L., Harman, M., Hao, D., Jia, Y., & Zhang, L. (2018). Predictive mutation testing. IEEE Transactions on Software Engineering, 45(9), 898-918.
13. Chen, L., & Zhang, L. (2018, April). Speeding up mutation testing via regression test selection: An extensive study. In 2018 IEEE 11th International Conference on Software Testing, Verification and Validation (ICST) (pp. 58-69). IEEE.
14. Ferrari, F. C., Pizzoleto, A. V., & Offutt, J. (2018, April). A systematic review of cost reduction techniques for mutation testing: preliminary results. In 2018 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW) (pp. 1-10). IEEE.
15. Khari, M., Kumar, P., Burgos, D., & Crespo, R. G. (2018). Optimized test suites for automated testing using different optimization techniques. Soft Computing, 22(24), 8341-8352.
16. Carvalho, L., Guimarães, M. A., Ribeiro, M., Fernandes, L., Al-Hajjaji, M., Gheyi, R., &Thüm, T. (2018, February). Equivalent mutants in configurable systems: An empirical study. In Proceedings of the 12th International Workshop on Variability Modelling of Software-Intensive Systems (pp. 11-18).
17. Ghiduk, A. S., Girgis, M. R., &Shehata, M. H. (2018). Reducing the Cost of Higher-Order Mutation Testing. Arabian Journal for Science and Engineering, 43(12), 7473-7486.
18. Sánchez, A. B., Delgado-Pérez, P., Medina-Bulo, I., & Segura, S. (2018, July). Search-based mutation testing to improve performance tests. In Proceedings of the Genetic and Evolutionary Computation Conference Companion (pp. 316-317).
19. Zhu, Q., Panichella, A., &Zaidman, A. (2018, April). An investigation of compression techniques to speed up mutation testing. In 2018 IEEE 11th International Conference on Software Testing, Verification and Validation (ICST) (pp. 274-284). IEEE.
20. Hooda, I., &Chhillar, R. S. (2018). Test Case Optimization and Redundancy Reduction Using GA and Neural Networks. International Journal of Electrical and Computer Engineering, 8(6), 5449.
21. Han, Z. H., & Zhang, K. S. (2012). Surrogate-based optimization. Real-world applications of genetic algorithms, 343.
22. G. Jeeva Rathanam, A. Rajaram, “Improved NEH-Heuristic Job Scheduling for An Optimal System Using Meta-Heuristic GA–INSMG”, International Journal of u- and e- Service, Science and Technology, Vol.9, No. 7 (2016), pp.213-226
23. Premanand, R.P., Rajaram, A. Enhanced data accuracy based PATH discovery using backing route selection algorithm in MANET. Peer-to-Peer Netw. Appl. 13, 2089–2098 (2020). https://doi.org/10.1007/s12083-019-00824-1
24. Rajaram.A., Dr.S.Palaniswami . Malicious Node Detection System for Mobile Ad hoc Networks. (IJCSIT) International Journal of Computer Science and Information Technologies, Vol. 1 (2) , 2010, 77-85
25. Dr.S.Palaniswami, Ayyasamy Rajaram. An Enhanced Distributed Certificate Authority Scheme for Authentication in Mobile Ad hoc Networks. The International Arab Journal of Information Technology (IAJIT).vol.9 (3),291-298.