Journal of Theoretical and Applied Information Technology
15th August 2023. Vol.101. No 15
© 2023 Little Lion Scientific

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

DESIGN OF MUTATION OPERATORS FOR TESTING USING PARALLEL GENETIC ALGORITHM FOR OPEN-SOURCE ENVIRONMENTS

SANDDEP KADAM[1], T. SRINIVASARAO[2]

[1]Department of Computer Engineering Gitam University,Visakhapattnaam, India

[2]Department of Computer Engineering Gitam University, Visakhapattnaam India

[1]sukadam.bscoer@gmail.com, [2] sthamada@gitam.edu

ABSTRACT

Specification-based testing approaches create test data without having any prior knowledge of the program's structure. However, the quality of this test data isn't always reliable enough to catch errors when non-functional modifications are made to the software. We offer a novel technique that combines formal requirements and the evolutionary algorithm to successfully produce test data. In this technique, Parallel Genetic Algorithm (PGA) rewrites formal requirements in order to create input values that kill as many mutants of the target programmed as feasible. To explain how the approach works, two famous instances are offered. The results suggest that the proposed technique may successfully produce test cases to eliminate programmed mutants, resulting in improved software maintenance.

Keywords: *Model-Based Testing; Fault Localization; Search-Based Algorithm; Automatic Test-Case Generation; Mutation-Based Testing;*

1. INTRODUCTION

Mutation testing, often known as programme mutation [39], is a methodology for creating test cases and assessing the effectiveness of current testing methods. Small changes are inserted into the original programme during mutation testing. Each altered version is referred to as a programme mutant, and test data is considered excellent if it kills programme mutants, that is, if it causes programme mutants to behave differently from the original programme. Both the programmes and the specifications are mutated in our method. The mutant's programme is used to assess the quality of altered specifications. We look for excellent modified specifications that can be used to create test data that can be utilized to find bugs.

Since characterizing circumstances describe how output variables relate to input variables, they are often used to check whether an execution of the programme is correct or not, rather than being used to directly generate input values. It's frequently impossible for a software to create input values that fulfil a defining condition without first knowing the output values. For example, if input variable x and destination variable y both match the defining condition ($x\ y > x + y$), we can't derive input x from ($x\ y > x + y$) since output y is unknown. As a consequence, ($x\ y > x + y$) is normally not used to assist in the generation of input values, but it may be used to examine the outcome of running the programme with input x. Our major focus is on devising a method for determining optimal correct output for the specification. These measurement results are then utilized to create modified specifications, which are merely input variable restrictions. The altered specs may then be utilized directly in testing phase to create input values. To do this, we use GA to find acceptable output values from the specifying condition. Furthermore, some modification is explored for reforming specifying criteria before using GA to generate modified requirements which are stronger in bug identification. In this modification, we make a little adjustment to the defining conditions in order to cause the produced test data to fulfil those reformed ones to cause as many undesirable programme behaviors as feasible. After implementing GA to the original specification, we create modified specifications in our technique. The modified specs may be acquired more accurately by following two rules:

1. Reforming the proposed techniques by introducing regression models into the defining environments so that test data that meets those reformed ones can cause the programme to behave

badly; 1. Reforming the proposed techniques by introducing regression models into the defining environments so that test data that meets those reformed ones can cause the programme to behave badly;

Our aim is to get a fresh version of the requirement from which to construct a test data set that will trigger as many errors in the software as feasible. The chromosomal forms for the redesigned requirement, as well as the crossover and mutation operators, will be defined next. Then we use GA to find relevant altered specs that may be used for bug identification.

2. LITERATURE REVIEW

According to [1], The degree to which presently utilized mutation testing procedures in DL is by the conventional understanding of mutation testing might be questioned. We notice that the creation of machine learning models parallels the test-driven development (TDD) method, in which a training algorithm ('programmer') develops a model (software) that fits data points (test data) to labels (implicit assertions) up to a particular threshold. However, when using this TDD paradigm to analyze suggested mutation testing methodologies for ML systems, production and test code difference is fuzzy. They might question the realism of mutation operators. The competent programmer and coupling effect hypotheses are discussed as basic ideas underpinning conventional mutation testing. These theories do not easily convert to ML system development, as we shall demonstrate, and more intentional and explicit scoping and concept mapping will be required to draw similarities fully. According to [2] seeking to develop a complete technique that can (1) discover define-usage issues and (2) produce test data for UML state machines automatically (3): modify the states and flows to get an efficient mutation score by varying the amount of complexity. (4): offer comprehensive coverage of the best def-use route (5): This rule applies to all UML diagrams. This work-in-progress is the first step in that direction, and we've verified our approach with a working implementation that they can use with state diagrams.

According to [3] a method for detecting redundancy in mutations by using dynamic subsumption relations between mutants. The subsumption relations among modifications of an expression or statement, referred to as "mutation target," are the topic of this paper. We create subsumption relations for hundreds of mutation targets in which the MUJAVA tool may apply mutations by concentrating on targets and depending on automated test generating tools. Subsequently applied these relationships in MUJAVA-M, a programme that creates a smaller collection of mutants for each target, eliminating duplicate mutants. According to [4] The two particular div measures (based on accuracy and Matthew's correlation coefficient, respectively) are compared to artefact-based diversity (a-div) to prioritize the test suites of six distinct open-source projects. In all of the projects we looked at, our b-div measures outperformed a-div and random selection. According to [5] a new collection of Java Scripts mutation operators that address aspects not addressed by current Java Script mutation operators. Our suggested operators created a wide range of flaws that do not overlap with current operators. We conduct tests on various case studies, and the findings show that the suggested mutation operators do not seed duplicate flaws.

According to [6] PRIMA is a revolutionary test input prioritizing strategy for DNNs that uses intelligent mutation analysis to identify more bug-revealing test inputs sooner for a short duration, allowing DNN testing to be more efficient. PRIMA is built on the following important insight: a test input that can kill a lot of mutated models and provide different prediction outcomes with a lot of mutated inputs is more likely to expose DNN problems. Hence they should prioritize it higher. PRIMA incorporates learning-to-rank (a type of supervised machine learning for solving ranking problems) to intelligently combine these mutation results for effective test input prioritization after obtaining several mutation results from a series of our designed model and input mutation rules for each test input. We performed a thorough investigation on 36 popular issues, considering their variety across five dimensions (i.e., different domains of test inputs, various DNN tasks, other network structures, different types of test inputs, and different training scenarios). According to [7] a scalable approach to mutation testing depending on the relevant main ideas: (1) mutation testing is done incrementally, mutating the changed code during code review rather than the entire code base; (2) mutants are filtered, removing mutants that are likely to be irrelevant to developers and limiting the number of mutants per line and per code review process; (3) mutants have been selected based on the historical performance of mutation operators, further eliminating irrelevancy; and (4) mutants are selected based on the historical performance of mutation operators, The suggested technique is experimentally validated in this study by examining its performance in a code-review-based context, which over 24,000 engineers utilized on over 1,000 projects. The findings reveal that the suggested method generates orders of magnitude fewer mutants and that context-based mutant filtering and selection improve mutant quality and actionability. According to [8] It increases the performance of spectrum-based fault localization; mutation testing should be used to identify successful test suites (SBFL). In our tests, we employ two famous SBFL methodologies, Ochiai and Jaccard, to demonstrate the influence of test suite efficacy on SBFL performance. To assess and choose test cases, we utilized the free PIT as a source mutation testing tool, and to write and execute the test suites, we used the unit testing framework JUnit. According to our experimental data, the suggested technique may greatly enhance the suspiciousness rating of incorrect claims.

According to [9] a novel method for extracting characteristics from mutant programmes based on mutant death criteria, such as reachability, need, and sufficiency, as well as the mutant significance and test suite metrics. A deep learning Keras model is presented to forecast dead and living mutants from each programme. First, the features are retrieved using the Eclipse JDT library and programme dependency analysis. Second, preparation methods like Principal Component Analysis and Synthetic Minority Oversampling are utilized to minimize data dimensionality and address the unbalanced class issue, respectively. Finally, fine-tune parameters such as dropout and dense layers, activation function, error, and loss rate are used to improve the deep learning model. The suggested study analyses five open-source applications from the GitHub repository containing thousands of classes and LOCs. According to [10] a representation technique for static code characteristics that combines graph and vector-based representations. Our findings, based on 50 changes in 21 Coreutils applications, show that our approach has a significant prediction capacity, with AUC values of 0.80 (ROC) and 0.50 (PR-Curve) and precision and recall values of 0.63 and 0.32, respectively. These predictions are significantly better than random guesses, with AUCs of 0.20 (PR-Curve), precision and recall of 0.21 and 0.21, respectively. They lead to strong relevant tests that kill 45% more relevant mutants than randomly sampled ones (either from those residing on the changed component(s) or from the changed lines). According to [11] a mutation-based framework for testing compliance between virtual/silicon device implementations and their requirements effectively and efficiently. Based on our established mutation operators, device specifications may be automatically instrumented with mild mutant-killing restrictions to represent likely erroneous device behaviours. Our method uses a suitable symbolic execution technique to quickly automate test case development and compliance testing for virtual/silicon devices to exclude all possible mutations. Our approach correctly measures if the designs have been appropriately vetted and reports conflicts between device specifications and implementations by symbolically executing the instrumented specifications using virtual/silicon device traces generated through cooperative execution.

According to [12] To test network protocols, they considered fault-based testing methodologies. The criteria for fault-based testing are derived from network protocol standards. This project's major purpose is to see whether fault-based testing approaches can detect flaws or defects that traditional network protocol testing techniques can't. Conformance testing is one of the major functional testing domains where fault-based methods might be useful. They may see whether the network protocol is strong enough to verify test cases that follow protocol specifications and invalidate test cases that don't. Several investigations have shown that fault-based testing may confirm conformity with less effort than other testing methods. The network simulator uses the test scenarios that have been created as input. The quality of test scenarios is assessed from three angles: code coverage, (ii) mutation score, and (iii) testing effort. In NS2, we built the testing framework. According to [13] a Systematic Literature Review on AS and CAS testing, one of the purposes of which was to characterize fault types for ASs and CASs. We examined 11 key research that addressed fault types to achieve this aim. We also provided code samples to show how different faults might arise. Finally, we looked at the flaws addressed in the other seven studies on fault-based testing for ASs and CASs. Results: We give a list of particular fault types and fault type categories (6 in whole) for AS and CASs, as well as a discussion of the fault types' link to existing fault-based testing methodologies. Conclusion: When compared to the state-of-the-art, our findings are novel since we presented the first classification of fault types for ASs and Cass.

According to [14] a mutation-analysis-based benchmarking system that they may use to assess the recall of clone detection methods for various sorts of clones and certain types of clone alterations, all without the need for human labour. The system uses a clone synthesis editing taxonomy to generate thousands of fake clones, inject them into codebases, and analyses the subject clone detection techniques using a mutation analysis method. According to the framework's characteristics, custom clone pairs might also be utilized in the framework for assessing the subject tools. There allows for the evaluation of specialized tools in specialized situations, such as detecting sophisticated Type-4 clones or real-world clones, without writing complex mutation operators for them. The performance of 10 recent clone detection methods is evaluated over two clone granularities (function and block) and three programming languages (Java, C, and C#). According to [15] MeMu is an innovative methodology for decreasing the execution time of mutants by memoizing the system's most costly processes. When repeated inputs are discovered, memorization is an optimization approach that permits expensive processes to be skipped. There may be a user menu in combination with other methods of acceleration.

According to [16] a unique fault localization method based on the integration of spectrum and mutation. The methodology increases the accuracy of the spectrum-based fault localization method while also reducing the time consumption overhead of the mutation-based fault localization method. In contrast to the spectrum-based fault localization technique, the experimental findings suggest that the strategy enhances fault localization accuracy. According to [17] They can improve RISC-V compliance testing using a mutation-based technique by delivering more detailed data. As a result, we define RISC-V-specific mutation classes to evaluate the CT's quality and give a symbolic execution mechanism to construct additional test cases that kill undiscovered mutations. The proved the usefulness of a mutation-based technique to increase RISC-V compliance testing. Based on our mutation classes, we detected many severe holes in the Compliance Test-suite (CT) and created additional tests to reinforce the CT by filling these gaps. Our method has also proven successful in locating flaws in RISC-V simulators. Finally, we had a lengthy conversation in which we sketched out several intriguing future research topics. According to [18] The use of third-party transformations to assess the efficacy of ATL mutation operators provided in the literature and additional operators built based on actual data of real-world developer mistakes. Similarly, we evaluate the effectiveness of widely used test model generating strategies. If a test suite fails to identify an inserted problem, we create test models to detect it. We provide a framework that automates this procedure for ATL as a technical contribution.

According to [19] Aspect preserving mutation is a novel approach that stochastically maintains the desired features, termed aspects, that we wish to be retained throughout mutations. In our fully-fledged JavaScript fuzzer, DIE, we show aspect preservation using two mutation techniques, namely, structure and type preservation. Compared to state-of-the-art JavaScript fuzzes, DIE's aspect-preserving mutation successfully detects new problems and provides legitimate test cases. Core Java, Java Script, and V8 all have 48 high-impact issues, according to DIE. According to [20] Used a random walk mutation-based differential evolution (DE) with an estimate of distribution algorithm (EDA) to solve this issue. The following are the main characteristics: I random walk mutation preserves population diversity, guiding individuals to different promising regions; ii) probability selection is used to provide suitable parent individuals for evolution, and iii) EDA is used to accelerate convergence and obtain the roots. They chose a test set of 30 NESs with various characteristics to assess the performance of our technique.

According to [21] The TransRepair is a completely automated method for checking and fixing machine translation system consistency. TransRepair detects inconsistency problems by combining mutation and metamorphic testing (without access to human oracles). To resolve the discrepancies, it uses probability-reference or cross-reference to post-process the translations in a grey-box or black-box way. TransRepair is the first method for automatically testing and improving the consistency of context-similar translations. TransRepair uses a context-similar mutation to create slightly changed (mutated) words that there may use to evaluate machine translation systems. Translation and comparison of the original and altered texts are used for testing. TransRepair calculates the similarity of the translation subsequences to determine consistency. TransRepair considers it a possible problem when context-similar mutations cause above-threshold disruption in the translation of the fundamental component. According to [22] It is a better optimization framework that combines the advantages of many methods, including a multi-operator differential evolution technique and evolutionary approach to covariance matrix co. Reinforcement learning is utilized in the former to automatically find the optimum differential evolution operator. Three benchmark sets of bound-constrained optimization problems (73 problems) with 10, 30, and 50 dimensions are solved to assess the proposed framework's performance. Furthermore, it put the suggested technique to the test by solving 100-dimensional optimization problems from the CEC2014 and CEC2017 benchmark problems. A data set from a real-world application has also been translated. The best variation is compared against a variety of state-of-the-art algorithms in a series of tests to examine the impact of various proposed framework components.

According to [23] an elitist Genetic Algorithm (GA) with a better fitness function that exposes the most errors while reducing the cost of testing by producing fewer complicated and asymmetric test cases. It employs a selective mutation technique to generate low-cost artificial defects with fewer redundant and comparable mutations. For evolution, the traces of test execution and mutant identification natural reproduction operator selection determine whether to diversify or strengthen the prior population of test instances. The size of the test suite is further reduced by iteratively eliminating redundant test cases. This research compares the effectiveness of the suggested technique to Initial Random testing and a commonly used evolutionary framework in academia, namely Evosuite, using 14 Java applications of notable sizes. Our method is proven to be more stable in practice, with a considerable increase in the test case efficiency of the optimized test suite. According to [24] a new algorithm for creating tests SGO-MT uses the social group optimization algorithm (SGO) to find as many flaws in software as possible. SGO is based on learning the characteristics of a group of persons. It comprises two phases: acquiring (learning from society) and developing (learning from the instructor), aiming to improve each individual's fitness. Another impacts each test case in learning from culture, while test data are developed concerning the fittest test case in the latter situation. SGO-MT stops working when it fulfils its goal to discover as many fake problems as feasible. According to [25] MUTAPI is the first method for detecting API abuse trends using mutation analysis. We initially constructed eight effective mutation operators inspired by the prevalent features of API misuses to simulate API misuses based on proper usages successfully. MUTAPI creates mutants by running these mutation operators on a series of client projects, then collecting mutant-killing tests and stack traces. Misuse patterns are detected in the deceased mutants, which are then prioritized depending on their potential of producing API misuses based on the data obtained.

According to [26] is prompted by the fact that the success of created network protocols is highly dependent on the beginning circumstances and assumptions of the testing scenarios in many situations in the literature. Network services are deployed in complex contexts. The testing and simulation results might change from one environment to the next and even from time to time within the same environment. Our objective is to offer mutation-based integration testing for network protocols used as Built-in Tests (BiT). According to [27] a mutation-oriented framework for property-based testing that is entirely automated. Our programme improves the efficiency of the testing loop by using unique algorithms and detecting complicated flaws in seconds. We test MUTAGEN by producing random WebAssembly applications to detect faults in a broken validator. According to [28] a method for detecting redundancy in mutations by using dynamic subsumption relations between mutants. The subsumption relations among modifications of an expression or statement, referred to as "mutation target," are the topic of this paper. We create subsumption relations for hundreds of mutation targets in which the MUJAVA tool may apply mutations by concentrating on targets and depending on automated test generating tools. These relationships are subsequently used in MUJAVA-M, a programme that creates a smaller collection of mutants for each target while eliminating repetitive mutants. According to [29] From two angles, the employment of intelligent technology increases the efficacy and efficiency of mutation testing. A machine learning approach called fuzzy clustering is used to classify mutants into distinct groups. Then, when the issue of test data production is seen as an optimization problem, a multi-population genetic algorithm with individual sharing is used to create test data for killing mutants in separate clusters simultaneously. The execute the suggested strategies, a complete framework called FUZGENMUT is developed. According to [30] MutShrink is a mutation-based test data selection approach for SQL regression testing (Mutation-based Test Database Shrinking Method). The objective is to reduce testing costs while maintaining the same efficacy as the original database. Tests using a benchmark that included some difficult SQLs and a huge database are conducted.

3. PROPOSED SYSTEM IMPLEMENTATION

The proposed system described the generation of mutation testing using a parallel genetic algorithm in a real-time source code environment. In the first phase, we generate some mutant code from the original programs using third-party API or tools. The test suite has been generated by using J-unit testing for all directories and major classes. The major objective behind the generation of mutant classes is to validate the test case with the original and mutant classes. In the analysis phase, we evaluate both mutant and original classes using a parallel genetic algorithm. The fitness score is important to decide whether the test case is killed or not. In the below section, we describe in detail the execution process of the proposed architecture with below figure 1.

The proposed approach begins by modifying both the source as well as mutant programmes, allowing each programme to be regarded as a collection of discrete modules. The approach does not go through the complete programme at once. It goes through each mutated unit and tries to kill them one by one. If the data states of the mutated object and the original programme differ, it re-tests with fresh test data created using genetic algorithms. It provides a fitness function using the parameter of mutant declaration expressing the value of the mutant programme, interpretation value of the original system, test data, and mutation score in order to generate fresh test data. We address the cost of connectivity and required and sufficient conditions to produce fitness function.

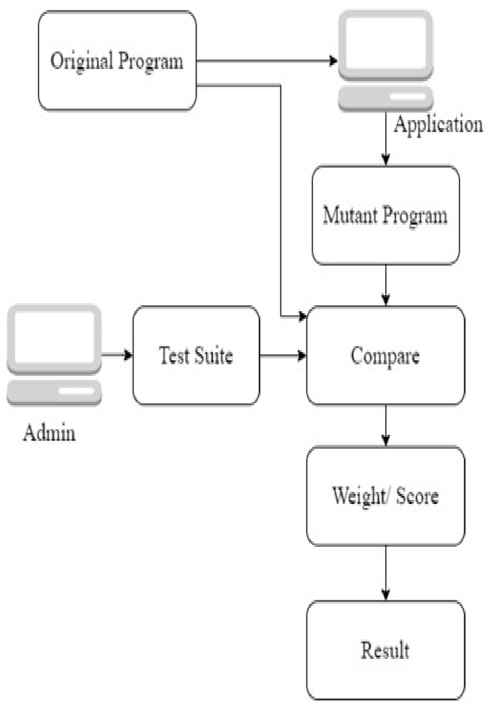

Figure 1 : Proposed System Architecture

The reachability cost comprises two parts: I the cost of a route difference and ii) the expense of failing a branch predicate. The required environment is a condition that must be met on the modified item or on any expression that contains the object for the mutant to be destroyed. For example, if the integer variable x is transformed to abs(x) in a programme, then x must be negative for the mutant to be eliminated. Even though there is a state variance and mutant programmes at the recombination statement, a mutant may sometimes survive since the difference is not conveyed to the output. We follow the execution of a working example and keep track of the original and mutant program's data states. The source and mutant programmes are both instrumented in this technique so that each unit's input and output behavior can be tracked. To compare and trace the output of each unit, we utilize a checker module. If the mutant unit survives, the checker logs a 1, and if the mutant unit dies, it logs a 0. We may find the malfunctioning unit by looking at the location where one occurs in the actual output sequence of the complete programme.

4. ALGORITHM DESIGN

In the implementation of the proposed system, we utilized a parallel genetic algorithm after the generation of mutant programs. The extracted features from both classes are the initial population, which is fed to 1st step of the genetic algorithm. After the generation of a random population, the single-point crossover has been utilized and generate new chromosomes from to Parent chromosomes. In the mutation phase, we have randomly changed the specific value of specific genes of the chromosome. The fitness function or generate the similarity score after applying the test Suite on both classes. According to generated fitness score finally, we decide, the test case is killed or not. In the below section, we determine two sections of proposed parallel genetic algorithm.

1: Parallel Genetic Algorithm

Input: random population from training dataset C[c[i]….c[n]]

Output: A rule set generation as intrusion pool for IDS

```
Step 1: Read each instance form pop
        $c = \sum_{k=0}^{n} (pop[k])$
If(k%2==0) then cros[i]
Else cros[i]+1
NewCh[]=crossover(cros[i],cros[i+1])
Step 2:
        $Nc = \sum_{j=0}^{n} (Newch[j])$
AftMut[] = Mutate(Nc)
Step 3: Calculate fitness
        $f(x) = \sum_{i=0}^{n} AftMut[i] \sum_{j=0}^{n} (Train[j])$
Step 4:  add fitness if list
                F [i] =f(x)/ sum f(x) ….. (4)
Step 5:   Apply selection operator on F[n]
          Check if C(x) met
          Sort best F[n]
```

Algorithm 2 : Sub Function for calculate the weight for each QoS parameter Calculate_Weight ()

Input : Statement_policy[], Attribute_val, Reward_Count, Penalty_Count

Output : Weight for current attribute W.

```
Step 1: while (event==true)
validate incomming attribute values using desired policy
   If(Attribute_val .equals(Statement_policy[]))
      Reward_Count = Reward_Count +1
      Total_Events = +1
Step 2: Else Penalty _Count = Penalty _Count +1
   Total_Events = +1
Update in log table penalty values for reward and penalty
End while
Step 3: Calculate the penalty weight
 W= Reward_Count / Total_Events
Return W
```

5. RESULTS AND DISCUSSION

The Implementation of the proposed system has been done with an open-source Java environment. The initial programs have been chosen from the Core Java environment which contains some statistical statements and object-oriented evaluations. The below Table 1 depicts the parameter initialization for proposed GA to evolution the test suite

Operators of GA	**Input Values**	**Accuracy**
Population	100	NA
Crossover Rate	0.40 -0.70	NA
Crossover Type	Single point	NA
Mutation Rate	0.30-0.60	NA
Fitness Function	F= F(x)/SumF(x)	NA
Selection Criteria	20%-80%	87.50%

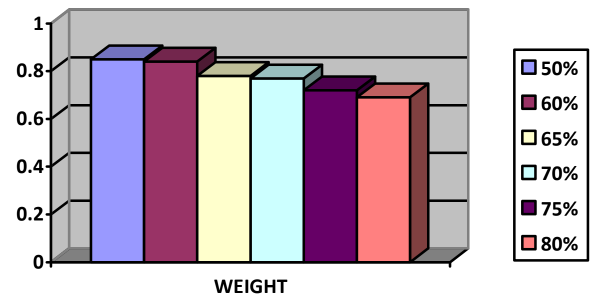

Figure 2 : Similarity Weight Generation From Fitness Function After Various Percentage Of GA Selection Criteria

The above figure 2 demonstrates similarity score generated by various selection criteria of GA selection. The parallel mutant has generated in first phase and evaluate the multiple mutants with original class with proposed GA. In a result the 50% route let selection criteria gives better similarity score by using PGA.1.

6. CONCLUSION

Test suite creation has been a prominent issue in automated testing development to provide efficient unit tests. This research revealed that the PGA's testing process showed more defects and ran more operations in the test class than other techniques used to produce real automated tests for Software components and sophisticated applications. These experimental findings were achieved by setting all algorithms' constants to the same value. Furthermore, all algorithms were run on the same tool and machine to eliminate bias in the findings. This shows that the PGA is not just a better GA for routing in a communication network, but it could also produce whole test suites in product testing. The same variable adjustment may restrict the algorithm's performance. Similarly, each method has optimum parameter values depending on the issue situation. This investigation's findings are just an initial effectiveness test towards creating a full test suite. As a result, proper values for PGA should be supplied to build test cases that are much more productive at discovering flaws and evaluating the source code. Moreover, various coverage criteria choices and connections with software system utilization yielded mixed results. Future research into entire test suite production using PGA must concentrate on boosting performance to discover more defects and reach more assertions in the test class by using procedures to choose the best chromosomal for the next iteration or integrate with other approaches.

REFERENCES:

[1] Panichella, Annibale, and Cynthia CS Liem. "What Are We Really Testing in Mutation Testing for Machine Learning? A Critical Reflection." 2021 IEEE/ACM 43rd International Conference on Software Engineering: New Ideas and Emerging Results (ICSE-NIER). IEEE, 2021.

[2] Mehboob, Fozia, Abdul Rauf, and Raza Ur Rehman Qazi. "Evaluating the Optimized Mutation Analysis Approach in Context of Model-Based Testing." 2020 International Conference on Emerging Trends in Smart Technologies (ICETST). IEEE, 2020.

[3] Guimarães, Marcio Augusto, et al. "Optimizing mutation testing by discovering dynamic mutant subsumption relations." 2020 IEEE 13th International Conference on Software Testing, Validation and Verification (ICST). IEEE, 2020.

[4] de Oliveira Neto, Francisco Gomes, Felix Dobslaw, and Robert Feldt. "Using mutation testing to measure behavioural test diversity." 2020 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW). IEEE, 2020.

[5] Muzamal, Muneeb, and Aamer Nadeem. "Improving test adequacy assessment by novel JavaScript mutation operators." 2019 16th International Bhurban Conference on Applied Sciences and Technology (IBCAST). IEEE, 2019.

[6] Wang, Zan, et al. "Prioritizing Test Inputs for Deep Neural Networks via Mutation Analysis." 2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE). IEEE, 2021.

[7] Petrovic, Goran, et al. "Practical Mutation Testing at Scale: A view from Google." IEEE Transactions on Software Engineering (2021).

[8] Saxena, Amol, Roheet Bhatnagar, and Devesh Kumar Srivastava. "Improving Effectiveness of Spectrum-based Software Fault Localization using Mutation Testing." 2021 2nd International Conference for Emerging Technology (INCET). IEEE, 2021.

[9] Naeem, Muhammad Rashid, et al. "Scalable mutation testing using predictive analysis of deep learning model." IEEE Access 7 (2019): 158264-158283.

[10] Ma, Wei, et al. "MuDelta: Delta-Oriented Mutation Testing at Commit Time." 2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE). IEEE, 2021.

[11] Gu, Haifeng, et al. "Specification-Driven Conformance Checking for Virtual/Silicon Devices Using Mutation Testing." IEEE Transactions on Computers 70.3 (2020): 400-413.

[12] Zarrad, Anis, Izzat Alsmadi, and Abdulrahmane Yassine. "Mutation Testing Framework for Ad-hoc Networks Protocols." 2020 IEEE Wireless Communications and Networking Conference (WCNC). IEEE, 2020.

Journal of Theoretical and Applied Information Technology
15th August 2023. Vol.101. No 15
© 2023 Little Lion Scientific

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

[13] Siqueira, Bento R., et al. "Fault sTypes of Adaptive and Context-Aware Systems and Their Relationship with Fault-based Testing Approaches." 2020 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW). IEEE, 2020.

[14] Svajlenko, Jeffrey, and Chanchal Roy. "The mutation and injection framework: Evaluating clone detection tools with mutation analysis." IEEE Transactions on Software Engineering (2019).

[15] Ghanbari, Ali, and Andrian Marcus. "Toward Speeding up Mutation Analysis by Memoizing Expensive Methods." 2021 IEEE/ACM 43rd International Conference on Software Engineering: New Ideas and Emerging Results (ICSE-NIER). IEEE, 2021.

[16] Jia, Minghua, et al. "SMFL integrating spectrum and mutation for fault localization." 2019 6th International Conference on Dependable Systems and Their Applications (DSA). IEEE, 2020.

[17] Herdt, Vladimir, et al. "Mutation-based compliance testing for RISC-V." 2021 26th Asia and South Pacific Design Automation Conference (ASP-DAC). IEEE, 2021.

[18] Guerra, Esther, Jesús Sánchez Cuadrado, and Juan de Lara. "Towards effective mutation testing for ATL." 2019 ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems (MODELS). IEEE, 2019.

[19] Park, Soyeon, et al. "Fuzzing javascript engines with aspect-preserving mutation." 2020 IEEE Symposium on Security and Privacy (SP). IEEE, 2020.

[20] Liao, Zuowen, et al. "Random Walk Mutation-based DE with EDA for Nonlinear Equations Systems." 2019 IEEE Congress on Evolutionary Computation (CEC). IEEE, 2019.

[21] Sun, Zeyu, et al. "Automatic testing and improvement of machine translation." Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering. 2020.

[22] Sallam, Karam M., et al. "Evolutionary framework with reinforcement learning-based mutation adaptation." IEEE Access 8 (2020): 194045-194071.

[23] Rani, Shweta, Bharti Suri, and Rinkaj Goyal. "On the effectiveness of using elitist genetic algorithm in mutation testing." Symmetry 11.9 (2019): 1145.

[24] Rani, Shweta, and Bharti Suri. "Adopting social group optimization algorithm using mutation testing for test suite generation: SGO-MT." International Conference on Computational Science and Its Applications. Springer, Cham, 2019.

[25] Wen, Ming, et al. "Exposing library API misuses via mutation analysis." 2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE). IEEE, 2019.

[26] Alsmadi, Izzat, Anis Zarrad, and Abdulrahmane Yassine. "Mutation Testing to Validate Networks Protocols." 2020 IEEE International Systems Conference (SysCon). IEEE, 2020.

[27] Mista, Agustín. "MUTAGEN: Faster Mutation-Based Random Testing." 2021 IEEE/ACM 43rd International Conference on Software Engineering: Companion Proceedings (ICSE-Companion). IEEE, 2021.

[28] Guimarães, Marcio Augusto, et al. "Optimizing mutation testing by discovering dynamic mutant subsumption relations." 2020 IEEE 13th International Conference on Software Testing, Validation and Verification (ICST). IEEE, 2020.

[29] Dang, Xiangying, et al. "Enhancement of Mutation Testing via Fuzzy Clustering and Multi-population Genetic Algorithm." IEEE Transactions on Software Engineering (2021).

[30] Toledo, Ludmila I., Celso G. Camilo, and Cássio Leonardo Rodrigues. "MutShrink: a Mutation-based Test Database Shrinking Method." 2020 IEEE International Conference on Systems, Man, and Cybernetics (SMC). IEEE, 2020.