STM JOURNALS

Journal of Artificial Intelligence Research & Advances
ISSN: 2395-6720 (Online)
Volume 6, Issue 2
www.stmjournals.com

Modified Genetic Algorithm for Performing the Regression Testing

Lata Dubey*, Seema
Department of Computer Science Engineering, Chandigarh Group of Colleges, College of Engineering, Mohali, Punjab, India

Abstract

Software testing is an approach where deferent errors and bugs in the software are identified. To test software, we need the test cases. One of the mainly imperative activities in software maintenance is regression testing. The re-execution of all test cases throughout the regression testing is expensive and time consuming and even though several of the code based planned techniques by researchers address technical programs. In our research work we proposed a regression test case selection for optimizing the selected test case with genetic algorithm. We are executing genetic algorithm upon different crossover rates (CR) and analyzed the results on number of iterations. The test cases are automatically generated through path crawler tool. We have taken 100% path coverage of the given source code. The effectiveness of the approach was evaluated by calculating average percentage of Modified Genetic Algorithm (MGA) over Simple Genetic Algorithm (SGA). Proposed Approach (PA) provides significantly improved outcome in terms of average percentage.

Keywords: *Regression testing, test-cases, genetic algorithm*

**Author for Correspondence* E-mail : lata.dubey@cgc.edu.in

INTRODUCTION

Software testing is worked out to determine whether the outcomes are expected and the software system is free of lack. It involves implementation of a software component or system component. Software testing is a process applied on the software product or application for producing correct and reliable software for the customer or stakeholders. This phase is most important phase of Software Development Life Cycle (SDLC) [1]. Software testing also helps us to find out the faults, cracks or lost necessities in the actual requirements. Software testing is one of the processes for evaluating software item and a method used to find out the accuracy, totality and superiority of residential computer software. Software testing also helps to identify errors, intervals, or absenteeism compared to actual testing. Software testing can be done also manually or by using automatic tools [2]. While testing a software application, different types of software tests are used to achieve different purposes. But the regression testing is very important testing type of software testing. A regression test is consistent with each innovative release of the invention or web site, and allows repetitive assumptions. Such tests indicate that invention defects have been fixed for all new discharges and that no new superiority problems have been introduced in maintenance process. Although regression testing could be done manually and an automated test suite is often used to decrease the time and resources, mandatory to perform the necessary tests.

Regression testing is defined as a kind of software testing to verify any current program or code changes having not had adverse effects on existing features. There is nothing in the recurrence test, but there are full or limited selections of already test cases that are re-executed to cure the accessible tasks. These tests are completed to ensure that new code changes should not have any effect on the current work. This ensures that the old code still works after the new code changes are made [3]. It can be difficult to determine how important it is to re-test, especially near the

end of the development cycle. Whenever used to improve defect, testing of a set of issues that need to be run to verify the defect correction, is determined by the test team [4]. The intension of regression testing is to hold the new bugs which were by mistake made in the release. For small projects, we can examine the entire product with the help of testing cases. Whenever any modification or changes are introduced in the software, it is necessary to run a regression test without fail. Modifications and changes may also cause undesired errors that user should be crawling while roaming in the system. In addition, overall performance volatility can show that there must be many errors due to significant changes in the architecture [5]. Determining the frequency of regression testing after every modification or every build update or a bunch of bug fixes is always a challenge. Sometimes, the entire regression testing suite becomes difficult due to the lack of time and budget.

Other problems include infinite loop of the event, which are not recognized and are unable to create test data for the complex programs; these technologies have been made inappropriate to create test data. Therefore, this necessitates preparing test data using the search-based technology. The use of genetic algorithms in software testing is a rising field of research that brings the cross fertilization of thoughts in two domains. Genetic algorithm can be used to generate the test cases, to ensure the cases of mass test that are not superfluous. It maximizes test coverage for test-born cases [6]. To satisfy the effectiveness of the examination cases and to use the appropriate suit of software testing, it is necessary for test data, quantification, measurement and ideal modeling. Testing is used to test and test data for the number, complexity, quality and test cases prepared for making more reliable and quantitative. A genetic algorithm can be used to detect the solution in a very short time. Although this may not be the perfect solution, but it can find the closest solution to problem? The fundamental idea of genetic algorithms is to randomly generate the initial population, in which there are different solutions for a problem called chromosomes, and then develops this population after several iterations of the generations. Every chromosome is evaluated during each generation, using the some fitness values. Selection, crossover and mutation are the three fundamental genetic operations employed in genetic algorithms. The chosen solutions are modified through these operations and the most appropriate offspring is selected to be passed on to succeeding generations [7].

Here, we are presenting an evolutionary approach that will be selecting the test cases from the source code by using the Path Crawler tool. The Path Crawler tool will help to generate the automatic test paths from the given source code. Since a large number of test cases arise, we have refined those test cases according to their weightage. The fitness value is computed by Crossover Rate (CR) and Mutation Rate (MR). This approach will reduce the time and provide the best fitness values of the test cases used in testing the customized program.

RELATED WORK

The following is the summary of the review performed by various research fellows.

Sánchez *et al*. [8] derived to the potential large number of products, the Software Product Line (SPL) test is challenging. For reducing this problem, many contributions have been proposed, while still good coverage has been proposed for deducting the number of software products. However, the products that have been tested have not been given much attention. Test career priority technology reconstitutes test cases to gather a certain performance goal.

Kaur *et al*. [9] proposed a new genetic algorithm method for the proposed regression testing suite, which prioritizes test cases based on full code exposure. Genetic algorithm will be automating the test case priority procedure. Result codes represent the efficiency of the algorithms that are represented with the average percentage of the metrics (APCC).

Musa *et al*. [10] proposed regression test case of selection method for the object-oriented

softwares on the basis of source code dependency graph model analysis and optimized the selection test case by using genetic algorithms. The main objective of this method is to select the revised test cases and sort them according to their fitness values, by the prior history of fault severity.

He *et al*. [11] introduced, a mathematical model which has been explained for the problem of lack of this test-suit and it has been changed to a linear integer documentation. After this, the paper examines the use of an evolutionary approach to the problem of reducing this test-suit, which is called genetic algorithm. The study shows that the algorithms can greatly decrease the size and the price of test-suit and the test-execution price is one of the most significant features that should be taken into the consideration for diminution in the test-suite.

Musa *et al*. [12] presented an evolutionary regression test case predilection for the object-oriented softwares based on the dependent graph model examination of the affected programs using genetic algorithms. This approach is based on optimization of the test case selected from Test Suite T. Its main purpose is to identify test cases for the purpose of identifying dependence such as reliance, control dependence and shelter relation, on inheritance and polymorphism, changes in body of a method, using effective statements and using them on GA on their fitness basis.

You *et al*. [6] defined the problem of formally reducing the time-tested retrograde test. They also provided a novel genetic algorithm for the problem reduction in timed retrograde testing. They determined the demonstration and fitness functions of genetic algorithm, and also introduced the basic collection of genetic algorithms, crossover and interaction operators.

Baradhi *et al*. [7] compared five algorithms for regression testing, including slicing, incremental, firewall, genetic, and masking algorithms. This contrast is the basis of ten quantitative and qualitative parameters like, Performance Time, Number of Selected Ratings, Purity, Completion, User Parameter, and Operation of Global Variables, types of Tests, Type of Test, Stage of Testing and Types of strategy for new results.

Nachiyappan *et al*. [8] examined the use of an evolutionary approach for the reduction of test-suite, which is called genetic algorithm. The proposed model creates an initial population based on the test history, uses the coverage to check the fitness value and gives time to run the test case, and then the selection of the next generation uses genetic operations and only the lower suit Fit allows testing.

Raju *et al*. [13] developed and authenticated the need on the basis of system level trial case priority plan for improving the quality of customer-treated software by using genetic algorithm (GA) to reveal more serious faults in the first stage. For this, they proposed a set of the priority factors for designing introduced system. In proposed technique, see these factors as a priority factor (PF).

Mansour et al. [10] presented to determine the minimum number of expected test cases in the maintenance phase to modify the modified software. They proposed two natural optimization algorithms, namely genetic algorithm and annealing algorithm, to solve the problem.

Rothermel *et al*. [11] used test performance information, and several techniques to prioritize test cases for testing, which included strategies for test cases on the basis of total coverage of their code. Those techniques have not been included before their code coverage components, and those techniques which include the code components covered by them, and give credit to the order of test cases based on the estimated capacity of crystals.

Zhai *et al*. [12] provided a suite of matrix and started introducing them to input-guided techniques and point-of-interest (POE) conscious test case-oriented techniques. It is different that the expected output of test cases in place of location information whether the location is in place or not. This state reports case study on LBS-enabled service.

According to Sampath *et al*. [13], regression testing is a major component of most major test systems, but only the web services have been started to be implemented. Many different methods have been studied to make most of the value of accumulated test suites like Mitigation, Selection and Priority.

Musa *et al*. [14] provided an evolutionary algorithm case preference for object-oriented software based on extended system dependence graphic affected programs by using genetic algorithms. This approach is based on the optimization of the selected test case which is related to source code dependency analysis. They proposed their approach which is based on optimization of the test case selected from the dependency analysis of source code.

Krishnamoorthi *et al*. [15] proposed an examination case-priority technique using genetic algorithms (GA). The proposed strategy gives preference after the original test suite so that the new suit can be executed within the time-delayed execution environment and compared to the rates of priority test trials; it would be a better rate for randomly detecting the mistake. This experiment for prioritizing test cases analyzes genetic algorithm in relation to effectiveness and time zone by using structural-based criteria. An average percentage defect (APFD) metric is used for establishing the effectiveness of the case sequence of new test.

Huang *et al*. [16] developed a method that automatically improves the GUI testing suite, which is possible to create new test cases. We are using a genetic algorithm to develop new test cases, which increase the coverage of our test suits while avoiding the inhale series. They experimented with this algorithm on the set of synthetic programs with dissimilar types of constraints for diverse length test sequences. The results show that they can generate new test cases to cover up the most potential coverage and the genetic algorithm leaves behind all the random algorithms annoying to achieve the similar goals in almost all cases.

RESEARCH METHODOLOGY

During the software life cycle, most test cases should be written by adequately testing software. As the software is changed, maintenance activity called regression testing is performed. This is undertaken to make sure the validity of the modified software. Regression testing evaluates whether a development change in a portion of the software affects the functionality of the application. To test the modified software, the new test cases to test the change of went need to be added to the test suite, which would increase the size of the test suite, the cost and time constraints [17, 19].

In most cases, as the product goes into adding new features, the regression test suite becomes heavier due to the trials adding to the period of time; it will create new problems to complete all the tests long in the entire test set. Therefore, to improve the efficiency of software testing, improving regression testing will help in reducing the cost of software.

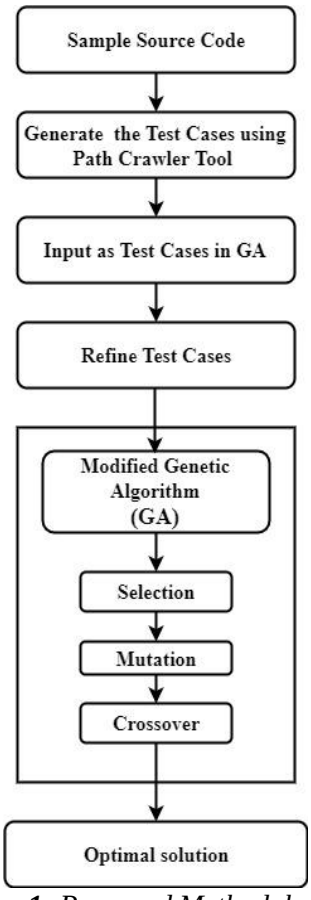

Fig. 1: *Proposed Methodology.*

In the proposed methodology we have proposed a regression test case selection method for analyses and optimized the selected test case with the genetic algorithm (Figure 1). In this we have generated test cases by using the Path Crawler tool. Since a large number of test cases arise, we have refined those test cases according to their weightage. Then the genetic algorithm is applied on the generated chromosomes and fitness of each chromosome is calculated [21-23].

Algorithm of Modified Genetic Algorithm
We are using modified genetic algorithms for global search technology. In the later stages

Step 1: Set the number of chromosomes, mutation rates and crossover rate values.
Step 2: Select the parents according to the fitness value. Select (two chromosomes).
Step 3: Apply crossover operator on selected chromosomes.
Step 4: Apply mutation operator to get more variation in the offspring.
Step 5: Evaluate the new generated children.
Step 6: Compare new children with the parents.

If (fp<fc)
Generate new population with children.
Else
Do not change the population.
Step 7: Check the optimization solution
If (solution! = possible)
Step 8: Go to STEP 5
Else
END.

IMPLEMENTATION

Selection Operator
In the selection operator, we are selecting two chromosomes according to our accumulated value for the performance of crossover. Fitter individuals are more likely to be selected for the next generation.

Crossover Operator
In genetic algorithms, crossover is a genetic operator that changes the programming of chromosomes or chromosomes from one generation to another. In this crossover we have used single point crossover of the parent and then connected the parent to the contemporary point to make the child. In the one point crossover, random crossover points have been selected and the tail of its two parents is swapped to obtain new off springs.

Mutation Operator
Here, we described the most used mutation operator, which is inversion mutation operator. In the inversion mutation operator, we are selecting the subset of genes, and reversed the overall string in the subgroup.

RESULTS AND DISCUSSION

Increasing the rate of mutation increases the diversity of population and the average distance between individuals. Selecting the mutation rate is an important thing because the population of chromosomes to determine the mutation reduced rate is similar to each other, thus growth can slow down or even stop. Due to the increase in the rate of the desired mutation, the population will be scattered and thus the guidelines will be lost for best solution. That is the reason we are choosing mutation rate 0.3 on crossover rate 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.6, 0.7, 0.8, 0.9 and 1.

Table 1: *Comparing Modified Genetic Algorithm on Simple Genetic Algorithm.*

	Modified Genetic algorithm		Simple Genetic algorithm	
Crossover Rate	Average Best fitness	Average Best Time	Average Best fitness	Average Best Time
0.1	491.772	2.678	507.538	3.558
0.2	512.418	2.714	503.178	2.684
0.3	545.564	2.38	483.192	3.848
0.4	530.408	2.508	474.748	3.752
0.5	524.278	2.664	506.11	2.918
0.6	529.942	3.304	551.27	3.578
0.7	501.07	3.138	520.712	3.264
0.8	520.956	3.144	436.512	3.606
0.9	538.98	3.192	547.322	4.69
1	529.284	3.326	487.992	3.818

In the Figures 2 and 3 given below we are comparing the graph of time between Modified Genetic Algorithm (MGA) and Simple Genetic Algorithm (SGA) on the basis of above Table 1.

In the above graph, y axis shows the time of both Modified Genetic Algorithm (MGA) and

Simple Genetic Algorithm (SGA), whereas x axis shows the crossover point. This graph shows the average of time on the different Crossover Rate (CR) on same Mutation Rate (MR) 0.3 (Figure 2). For each crossover rate (0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.6, 0.7, 0.8, 0.9 and 1), the average best time obtained from MGA is less than SGA considering the mutation rate as 0.3 [24].

The modified genetic algorithm (MGA) generates the high fitness values against simple genetic algorithm (SGA). This graph shows the high efficiency of the MGA by optimizing the best fitness with less time over SGA (Figure 3). So MGA gives the better results by performing best fitness in less time. Whereas SGA takes more time to perform and provide less fitness values.

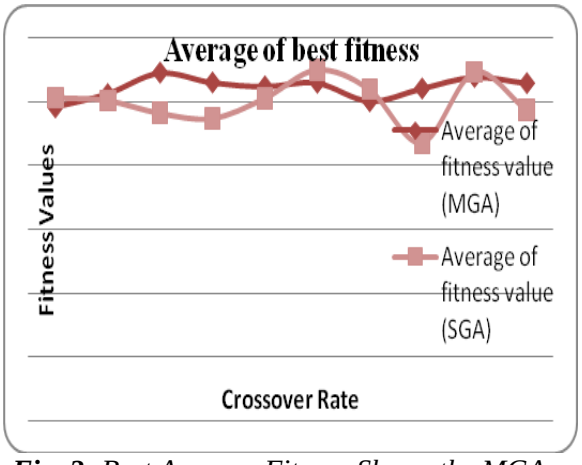

Fig. 3: *Best Average Fitness Shows the MGA and SGA.*

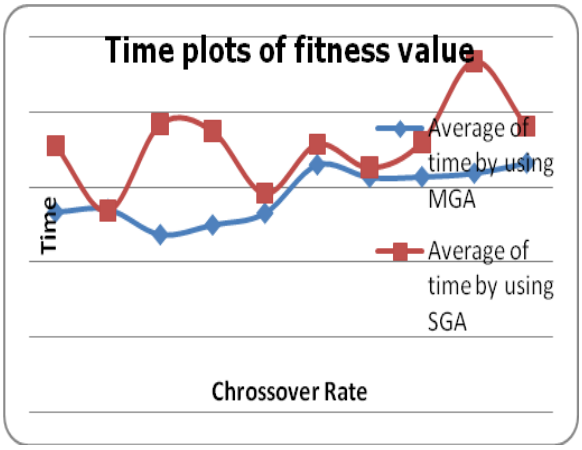

Fig. 2: *Best Average Times Comparing between Two Mutation Operators.*

CONCLUSION AND FUTURE SCOPE

The major contribution of this work is given below:

- The key contribution of the technique is to generate test cases of the sample source code for performing on further methodologies.
- The founding technique of modified genetic algorithm. This proposed technique solves the problem of time of regression testing, from which an optimal solution is obtained.

The proposed technique has focused on to reduce the time and the cost of the regression testing, but there are following points that can be explored further. The proposed technique optimizes the solution on the parameter of dissimilar crossover rates (0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.6, 0.7, 0.8, 0.9 and 1) on same mutation rate 0.3. The proposed technique can extend on another mutation rate. The proposed method can extend to generate test cases for complex coverage measures.

REFERENCES

1. Khan Rijwan, Mohd Amjad. Automatic Test Case Generation for Unit Software Testing Using Genetic Algorithm and Mutation Analysis. *Electrical Computer and Electronics (UPCON), 2015 IEEE UP Section Conference on.* IEEE; 2015.
2. Planning Strategic. The Economic Impacts of Inadequate Infrastructure for Software Testing. 2002.
3. Harrell Frank. *Regression Modeling Strategies: With Applications to Linear Models, Logistic and Ordinal Regression, and Survival Analysis.* Springer, 2015.
4. Spichkova Maria, *et al*. Human Factors in Software Reliability Engineering. *arXiv preprint arXiv:1503.03584*. 2015.
5. Darlington Richard B, Hayes Andrew F. *Regression Analysis and Linear Models: Concepts, Applications, and Implementation.* Guilford Publications; 2016.
6. Suman, Seema. A Genetic Algorithm for Regression Test Sequence Optimization. *International Journal of Advanced Research in Computer and Communication Engineering (IJARCCE).* 2012; 1(7): 478–481p.

7. Said Gamal Abd El-Nasser A, Mahmoud Abeer M, El-Horbaty El-Sayed M. A Comparative Study of Meta-Heuristic Algorithms for Solving Quadratic Assignment Problem. *arXiv preprint arXiv:1407.4863*. 2014.
8. Sánchez Ana B, Sergio Segura, Antonio Ruiz-Cortés. A Comparison of Test Case Prioritization Criteria for Software Product Lines. *Software Testing, Verification and Validation (ICST), 2014 IEEE 7th International Conference on*. IEEE; 2014.
9. Kaur Arvinder, Shubhra Goyal. A Genetic Algorithm for Regression Test Case Prioritization Using Code Coverage. *International Journal on Computer Science and Engineering (IJCSE)*. 2011; 3(5): 1839–1847p.
10. Musa Samaila, *et al*. Software Regression Test Case Prioritization for Object-Oriented Programs Using Genetic Algorithm with Reduced-Fitness Severity. *Indian J Sci Technol*. 2015; 8(30): 1–9p.
11. He Zhen-Feng, Bin-Kui Sheng, Cheng-Qing Ye. A Genetic Algorithm for Test-Suite Reduction. *2005 IEEE Int Conf Syst Man Cybern*. IEEE. 2005; Vol. 1.
12. Musa Samaila *et al*. A Regression Test Case Selection and Prioritization for Object-Oriented Programs Using Dependency Graph and Genetic Algorithm. *Research Inventy: International Journal of Engineering and Science*. 2014; 4(7): 54–64p.
13. Raju S, Uma GV. Factors Oriented Test Case Prioritization Technique in Regression Testing Using Genetic Algorithm. *Eur J Sci Res*. 2012; 74(3): 389–402p.
14. Jorgensen Paul C. *Software Testing: A Craftsman's Approach*. CRC Press; 2016.
15. Digitalvidyacom. (2017). Digital Vidya. Retrieved 2 July, 2019, from http://istqbexamcertification.com/what-is-software-testing-life-cycle-stlc/
16. Huang C-Y, Lin C-T. Software Reliability Analysis by Considering Fault Dependency and Debugging Time Lag. *IEEE Trans Reliab*. 2006; 55(3): 436–450p.
17. Pressman Roger S. Software-Engineering. 7th Edn. McGraw-Hill Publishing Company; 2014.
18. Young Michal. *Software Testing and Analysis: Process, Principles, and Techniques.* John Wiley & Sons; 2008.
19. Pohl Klaus, Günter Böckle, van Der Linden Frank J. *Software Product Line Engineering: Foundations, Principles and Techniques.* Springer Science & Business Media; 2005.
20. Tzimiropoulos Georgios. Project-Out Cascaded Regression with an Application to Face Alignment. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition.* 2015.
21. Schroeder Larry D, Sjoquist David L, Stephan Paula E. *Understanding Regression Analysis: An Introductory Guide*. Vol. 57. Sage Publications; 2016.
22. Myers Raymond H, Montgomery Douglas C, Anderson-Cook Christine M. *Response Surface Methodology: Process and Product Optimization Using Designed Experiments.* John Wiley & Sons; 2016.
23. Fox John. *Applied Regression Analysis and Generalized Linear Models.* Sage Publications; 2015.
24. Montgomery Douglas C, Peck Elizabeth A, Geoffrey Vining G. *Introduction to Linear Regression Analysis.* John Wiley & Sons; 2015.
25. Dhiman Gaurav, Vijay Kumar. Spotted Hyena Optimizer: A Novel Bio-Inspired Based Metaheuristic Technique for Engineering Applications. *Adv Eng Softw*. 2017; 114: 48–70p.

Cite this Article
Lata Dubey, Seema. Modified Genetic Algorithm for Performing the Regression Testing. *Journal of Artificial Intelligence Research & Advances*. 2019; 6(2): 69–75p.