International Journal of Academic Research in Computer Engineering
Print ISSN:2476-7638 and Online ISSN: 2538-2411
Vol. 1, No. 2, Pages. 25-32, November 2016

International Journal of Academic Research in Computer Engineering
Volume 1, Number 2, Pages 25-32, November 2016
www.ijarce.org
IJARCE

Multi-Objective Test Case Prioritization Techniques: A Brief Review

Batool Abadi Khasragi[1*]

1. Department of Computer Engineering, Islamic Azad University Osku Branch, Osku, Iran.

Receive Date 2016.08.19; Accepted Date: 2016.10.25; Published Date: 2016.11.15

*Corresponding Author: B.A.Khasragi (batool.abadi@gmail.com)

Abstract
Software testing is considered one of the most important stages in the software development life cycle aiming to detect all software faults including different methods and approaches such as regression test method. When new alterations or features are added to software, the regression test examine the system to ensure that no software operation has been affected by new modification. Test case prioritization is an approach for organizing test cases with a means to minimize time consumption, cost, and efforts, and to provide faster fault detection. Multiple criteria-based prioritization techniques have the potential of the better improvement and enhancement of the effects of regression test than other techniques applying single criteria. In this paper, an overview of regression test and prioritization will be presented along with an introduction to multi-objective techniques used in prioritizing. Furthermore, a comparison will be made between single- and multiple-criteria-based prioritization methods.

Keywords: Test Case Prioritization, Test Suite, Software Testing.

1. Introduction

Software testing is considered one of the main and costliest stages in the software development life cycle. This procedure occurs in all stages of software development life cycle (SDLC) including the requirements stages through the management and maintenance stages of software. Software testing can be defined as "the process of executing (running) programs in order to find faults". This activity, which is considered a validation and verification procedure, is the process of evaluating software distinguishing given and expected input so that software features can be evaluated. Software testing is an important component in software quality assurance to the extent that many organizations spend 40% of their resources on testing procedures. A software changes frequently during development and maintenance. These changes can be caused through adding new features, correcting faults, and improving system performance. Changes made to software's code can have dire consequences on software components causing the system to fail. Therefore, it is necessary to ensure that adding new features or components to a software or making changes in the software toward correcting and ensuring software component stability does not result in any negative outcomes. Although it may seem that executing and testing all test components is a good way to ensure component compatibility, considering the overall time and costs of the problem, it is not practically possible, though theoretically it works quite well. In [1] stated that a test suite for software with 20000 lines of code requires 7 weeks to complete its execution. However, different techniques are used in software testing including regression test. The purpose of this test is to validate a modified software. This test ensures that modified components do not affect the quality of other software components. According to Rothermel et. al. [2] "test suites can reach large sizes so that executing each test case with each modification in the source code can result in major costs”. As a software develops, the need to perform regression tests increases, and new test cases are generated and added to the test suite. Re-running test cases in regression tests for modified components of a system requires high amounts of time and costs. This calls for different methods of regression tests. Test case prioritization is one of the operations of a regression test. This procedure is necessary for multiple test cases, even when lines of code are less than millions. Apart from the time required for running test cases, other operations also require resources. Such operations may include test environment preparation, test result documents, and evaluation of test procedures in order to improve software procedure. Due to this, researches have recommended the use of the following techniques for more efficient running of regression tests [3]:

1.1. Regression Test

Regression test is applied in maintenance stage to the modified components of a software. Regression test executes all former successful tests taken from the software so that it can ensure that the software has not failed randomly in previous runs. Therefore, in order to ensure that all previous capabilities are still compatible, previous tests are performed for newer versions before the new version of the software is published. This test can be performed for all or none of the previous levels. The test also focuses on post-modification re-testing. In general, the purpose of regression test is to identify side effects (unexpected behaviors) in a new version. Regression test is not a test phase; rather it is a testing technique that can be applied in different test phases. Test cases of a specific stage are sometimes in the form of a series of stages for testing the correct operation and features of an application. Rerunning all test cases in an application is impractical, time consuming with high costs and needs for resources. There are different techniques for reducing the complexity of regression testing. Regression tests are usually performed for system content, acceptance test, and when a program undergoing a testing procedure has a significant need for cooperating with other programs (such as integration tests). Regression testing is also used in black box testing techniques in order to test high-level requirements of the program being tested without considering the details regarding implementation. Rothermel et al. in [2] stated that there are three methods for reducing time and costs of regression tests including regression test selection, set minimization test, and test case prioritization techniques. These techniques gather information from the main program, the modified program, and the test suite. In test case prioritization, test cases are assembled according to specific criteria. The test cases with the highest priority are then executed to reach a certain operational goal (objective). In test suite reduction or minimization, redundant test cases are removed from the test suite in the course of time and are transformed into a smaller set of test cases. Test suite reduction techniques reduce the cost of regression testing through minimizing test suites while maintaining the coverage of the primary test suite according to certain coverage criteria. In regression test selection, a sub-suite of test cases is selected from the original and larger test suite. In other words, a subset of test cases that can identify faults in the altered system is selected.

1.2. Test Case Prioritization

The issue of test case prioritization was first presented in 1997 by Wong et al. in [4]. This technique assigns a priority to each test case. The priorities are assigned according to certain criteria, and test cases with the highest priority are executed first. The main purpose of test case prioritization is to speed up fault detection rate in regression testing. The technique also tries to meet operational objectives such as fault detection rate, program code coverage rate, and amount of increased reliability of the system. This technique also has the advantage of not removing test cases from the test suite. The solution space for the prioritization problem is extremely large. For example, a suite consisting of 20 test cases includes 20! = 2,432,902,008,176,640,000 different arrangements for test cases. No algorithm can provide an optimal solution for such a large solution space; however, it is possible to develop very useful algorithms such as a genetic algorithm. Many authors have shown methods for improving regression test prioritization techniques [2]. Rothermel et al. in [5] have defined the prioritization problem as Equation (1).

$$(\forall T'')(T'' \in PT)(T'' \neq T')[f(T'') \geq f(T'')] \quad (1)$$

A test suite T along with a test set PT containing permutations of the test suite and a function f relating PT to real numbers.

Test case prioritization is performed to execute test cases in an ordered and regular fashion so as to save costs and time.

While decreasing test suite and regression test selection causes a reduction in the number of test cases, test case prioritization has no effect on the number of test cases. When test cases are decreased using test suite reduction and regression test selection, a few faults may occur due to loss of test cases. Therefore, it seems that test case prioritization is a more reliable and efficient approach compared to other methods.

The overall structure of the paper organized as follows: In the Section 2, test case prioritization techniques are explained. At the Section 3, comparison of single-criteria and multi-criteria will be described and finally in the Section 4, conclusion will be explained.

2. Test Case Prioritization Techniques

Please Test case prioritization techniques using two features [6]; test cost and fault severity. The cost of a test case can be computed according to test execution, launching, and test validation. The fault severity of each of the two methods regarding time required for locating faults and correcting them and impact of failures due to faults can be measured through this approach. The performance of the proposed method is less than that of prioritization approaches based on genetic algorithm.

A value-driven approach for system-level test case prioritization was proposed by Hema Srikanth et al. in [7]. This approach is also known as Prioritization of Requirements for Test (PORT). The PORT algorithm uses four factors for prioritizing test cases including requirements, execution complexity, fault rate for requirements, and client priority. This approach uses the value factor and weight factor for computing prioritization factor value (PFV) for all requirements. The fault detection rate is improved in this approach.

Walcott et al. [8] proposed a time-aware based approach for prioritizing test cases using two criteria of execution time and code coverage (block coverage and method). A genetic algorithm is used in this approach for prioritizing regression test suites according to these two criteria. Test case tuples are first selected according to their execution time.

These tuples are then used as the primary population for the genetic algorithm. Fitness of these tuples are evaluated according to code coverage. In experiments regarding this approach, the GradeBook and Jdepend applications were used both of which show faults. After these experiments, it was observed that time-aware prioritization used in this approach performed better than other prioritization techniques.

Researchers in [9] proposed a hybrid approach based on a particle swarm optimization algorithm in order to prioritize test cases in embedded real-time systems. The proposed method is based on three criteria including function, statement, and branch. The proposed algorithm can perform the best search for test cases in order to prioritize modified software components. It can also find test cases with the highest coverage. Experiments have been performed on 20 test cases from the Junit test suite.

Test case prioritization using multi-objective fitness function was proposed by Amr Abdel Fatah Ahmed et al. in [10]. This approach uses various control-flow coverage criteria for prioritization. The disadvantage of the proposed method is that it does not consider whether it affects regression test performance in computing execution cost and time for test cases.

N.Prakash and T.R.Rangaswamy in [11] proposed a modular-based multiple test case prioritization technique. The program is divided into multiple modules, and the number of test cases are prioritized according to each module. In the second stage, the prioritized test sets for a single module are combined and prioritized for the whole program. Each module is created by a number of test cases and test case fault coverage. The performance of the proposed method is better than the greedy algorithm and redundant greedy algorithm methods. However, it is less than the genetic algorithm approach. This prioritization approach considers only the fault coverage criteria and neglects other indexes such as code coverage, cost, time, etc.

Mahfuzul Islam et al. in [12] proposed a multi-objective test case prioritization approach based on latent semantic indexing. Latent semantic indexing is the method of information retrieval (IR). This proposed technique counts code coverage, software requirements, and executional costs for test cases. IR-based traceability recovery approach is applied to bind software artifacts (such as requirement specification) to the code. The test case arrangement is defined using multi-purpose optimization and performed using NSGA-II (Non-dominated Sorting Genetic Algorithm II) algorithm. This approach has been evaluated using two small java software applications.

Sudhir and Srinivas in [13] worked on an evolutionary search algorithm for test case prioritization. The proposed algorithm operates based on test time and code coverage information in order to arrange test suites using a genetic algorithm. The test cases of the proposed technique are generated using a genetic algorithm and consumes less time compared to test cases generated using random and optimal prioritization techniques. This technique can detect the maximum number of faults in a limited executional time. The fitness function for this algorithm is complicated.

Wang and Zeng in [14] presented a multi-criteria dynamic test case prioritization approach. Test case prioritization values are computed independently according to five criteria: coverage, potential probability of fault exposure, requirements, historical information, and execution time. The total weight of optimal results is also measured in the proposed method. The test set computed based on value is then arranged by total weight. This approach has been investigated in small test case samples. The proposed method is quite complex consuming a large amount of time for test case prioritization.

Manika and Malhotra in [15] presented a novel method for test case prioritization using MOPSO (Multi-Objective Particle Swarm Optimization). The proposed method uses maximum fault coverage and minimum execution time for test case prioritization. This algorithm uses a three-stage approach for test case prioritization. The first stage consists of removing abundant test cases. The second stage applies the multi-objective particle swarm optimization algorithm for selecting test cases from a test set according to both fault coverage and execution time objective functions. The third is when test case prioritization takes place and test cases from the second stage are prioritized. The multi-objective particle swarm optimization approach performs better than other approaches such as the non-sorting, reverse sorting, and random sorting techniques.

A multi-objective test case prioritization technique using genetic algorithm was proposed by Mitrabinda Ray et al. in [16]. The proposed approach prioritizes components based on their effect on reliability of the system being tested and then applies a test case selection method for selecting a constant number of test cases from storage. The test case selection approach is in fact a multi-purpose optimization problem used for minimizing deviance maximizing test suite qualification for selecting test cases. The performance of this approach was estimated using simulated experiments.

N. Prakash and K. Gomathi [17] proposed a test case prioritization approach using more than one criterion such as code coverage, branch coverage, function coverage, path coverage, and fault coverage. Coverage information is collected and analyzed both manually and automatically. According to coverage information, multiple coverage criteria are used for test case prioritization. Experimental results of this approach have been reviewed by three standard programs and were compared with other existing prioritization approaches. Comparison results show that the proposed method improves regression test performance.

Manika and Malhotra in [18] presented a regression test case prioritization method based on three factors: Rate of Fault Detection (RFT), Percentage of Fault Detection (PFD), and Risk Detection Ability (RDA). RFT is defined as the average number of defects found per minute by each test case. PFD is the percentage of faults detected by each test case over all faults. RDA is defined as the ability to detect severe faults per unit time. For each test case, these three factors have all been computed. Test cases are then sorted in descending order according to computed values. This multi-objective approach performs better than other approaches including non-sorting, reverse-sorting, and random sorting techniques.

Saini and Tyagi in [19] proposed a multi-objective test case prioritization algorithm (MTCPA) which is based on two objective functions. The objective functions include statement coverage and test case execution time using genetic algorithm. The proposed method has been compared with different prioritization techniques in order to find the optimal solution. Experimental results have shown that the proposed algorithm returns a test case suite with maximum fault coverage and minimum execution time with maximum APFD criteria as the solution.

The proposed multi-objective test case prioritization techniques are shown in Table 1.

Table 1. Multi-Objective Test Case Prioritization Techniques

Approaches	Metrics	Objectives	Authors
Greedy	APFDc	Statement coverage, Test cost and Fault severity	S. Elbaum, et.al.[6]
Value-driven approach	TSFD, ASFD	Requirements, execution Complexity, Fault rate for requirements, Client priority	Hema Srikanth, et.al.[7]
Time-aware based approach, Genetic Algorithm	APFD	Execution time, Code coverage (Block coverage and Method)	K.R Walcott, et.al. [8]
PSO	APFD	Statement coverage, Branch coverage, Function coverage	Khin Haymar Saw Hla, et.al. [9]
Genetic Algorithm	APFD	Control-flow coverage, Statement coverage, Fault severity	Amr Abdel Fatah Ahmed et.al.[10]
Modular based	APFD	Fault coverage	N. Prakash, T.R. Rangaswam[11]
Genetic Algorithm	APFD	Code coverage, software Requirements, , Executional costs for test cases	M.M.Islam et.al. [12]
Genetic Algorithm	APFD	Code coverage, Execution time	Sudhir Kumar, et.al. [13]
Optimized results	APBC	Coverage, potential, Probability of fault exposure, Requirements, Historical information, Execution time	Xiaolin Wang, Hongwei Zeng[14]
Multi objective PSO	APFD	Maximum fault coverage, Minimum execution time	Manika Tyagi and Sona Malhotra[15]
Genetic Algorithm	No metric	Reliability of the system being tested, Minimizing deviance maximizing test suite qualification for selecting test cases	Mitrabinda Ray, et.al. [16]
Optimized results	No metric	Code coverage, branch coverage, Function coverage, Path coverage, Fault coverage	N.Prakash and K.Gomathi [17]
Optimized results	APFD	Rate of Fault Detection (RFT), Percentage of Fault Detection (PFD), Risk Detection Ability (RDA)	Manika Tyagi, Sona Malhotra [18]
Genetic Algorithm	APFD	Statement coverage, Execution time	Anita Saini and Sanjay Tyagi[19]

3. Comparison of Single-Criteria and Multi-Criteria

Many prioritization techniques focus on one objective such as coverage or fault detection rate for prioritization. Recently, researchers have created hybrid criteria (also multi criteria, multi-objective techniques, and breaking ties). Most of these researches believe that fault detection is a complicated procedure, and using a unit criteria can significantly limit the ability of regression test in detecting defects.

The main idea of hybrid criteria is that they are comprised of multiple criteria and use the advantages of individual criteria in making decisions about selecting the next test case in a prioritization problem. A hybrid criterion may be comprised of multiple single criteria (namely primary criteria, secondary criteria, etc.) and are prioritized using the primary criteria. The secondary criterion is only used when test cases in the primary criterion are tied (for example they estimate the primary criterion).

Another reason for using hybrid criterion is that single criterion can transform into unit criterion. For example, let us assume that test cases are prioritized using total coverage for statements and branches (the most significant test cases covering the most statements and branches). Therefore, a hybrid method is needed to apply all criteria simultaneously. Multiple criteria have the potential to improve the effect of regression testing than techniques using single criterion.

The following example is used to define the concepts of single and multiple criteria prioritization. The test set is comprised of five test cases, each of which is a series of criteria such as events, statements, branches, executional time, and relative faults. Table 2 shows each of these test cases. 0 indicates criteria coverage and 1 indicates no coverage of criteria.

Single criteria prioritization can now be defined considering test cases shown in the table above. Let us assume that we want to prioritize five test cases using a redundant statement coverage prioritization approach [2]. This approach is as so: test cases with the highest statement coverage are selected first. The statements covered by test cases are then identified. Coverage information in all remaining test cases is configured so that statements not yet covered can be identified, and the process is repeated until all statements are covered by at least one test case.

When multiple test cases cover a common statement, one test case is randomly selected for that statement. According to this definition, the T_1 test case in Table (2) is selected first. This test case covers four statements. This test case is placed within the prioritization test suite. After this test case, the test case with a higher coverage degree and covering statements not yet covered is selected. According to these conditions, test cases T_2 and T_4 are selected in the next iteration. The reason why T_3 was not selected is that it covers statements previously covered by test case T_1. Test cases T_2 and T_4 both have the same conditions; therefore, one of them is selected randomly (T_4 in this case). This test case is placed in the prioritization test suite. With this test case, all statements are covered, and therefore, the algorithm terminates. The prioritization test suite includes {T_1, T_4}, and remaining test cases are added to the suite in an ordered fashion. The final suite for prioritization is { T_1, T_4, T_2, T_3, T_5}.

The same example will now be explained for multi-objective prioritization. The algorithm is the same algorithm as before and statement, and event coverage are selected as the primary and secondary criteria, respectively. After executing the algorithm, the T_1 test case is selected first based on primary criteria. In the next iteration, the T_2 and T_4 test cases are selected. According to the primary criterion, a tie occurs between these two test cases and the secondary criterion is applied. According to the secondary criterion, test case T_2 is selected since it covers event e_1 whereas the T_4 test case does not cover e1. The prioritization test suite now contains {T_1, T_2}.

Table 2. Test Cases With and Without Relative Criteria Coverage

Test Cases	Events			Statements					Branches		Executional Time	Faults				
	e_1	e_2	e_3	s_1	s_2	s_3	s_4	s_5	b_1	b_2	Exec (sec.)	f_1	f_2	f_3	f_4	f_5
T_1	0	1	1	1	0	1	1	1	0	0	0.5s	0	0	1	0	1
T_2	1	0	1	0	1	0	1	0	0	0	0.4s	0	1	0	1	0
T_3	1	1	0	1	0	1	0	1	1	1	0.8s	1	0	1	0	0
T_4	0	0	1	0	1	0	1	0	1	0	0.3s	0	1	0	0	0
T_5	1	1	0	1	0	0	0	0	0	1	0.9s	0	0	0	0	0

In the next iteration, no test case is selected according to the primary criterion since all statements have been covered and the algorithm is called using the secondary criterion and once again no test case is selected since all events are covered by the T_1 and T_2 test cases. Remaining test cases are added to the suite in an ordered fashion, and the final test suite for prioritization includes {T_1, T_2, T_3, T_4, T_5}.

As can be seen in the two examples above, the hybrid approach achieves a desirable coverage in program code. In the first example, the algorithm randomly selected a test case when two test cases were tied and therefore, the test case covering more statements than the test case selected may be removed, and the desirable code coverage may not be achieved and fault detection may not be improved as much.

In order to increase the effect of fault detection, another example will be examined according to figure 1. This example contains a program with 14 lines of code and 5 test cases. According to the table in Figure 1, the branches (if and else) of a program that are covered by test cases have also been shown.

```
1: Read (a,b,c,d); B1:If(a>0)
2: X=0;
3: else
4: X=5;
5: end if
6: if  (b>0)
7: end if
      B3: if(c>0)
      B4:  if(d>0)
8: output(x);
9: else
10: output(10);
11: end if
12: else
13: output(1/(y-6));
14:  end if
```

Test Case:
T_1: (a=1,b=1,c=-1,d=0)
T_2: (a= -1,b= -1,c=-1,d= -1)
T_3: (a= -1,b=1,c=-1,d=0)
T_4: (a= -1,b=1,c=-1,d=0)
T_5: (a= -1,b= -1,c=-1,d=0)

Figure 1. Sample Program with Relative Test Cases

Table 3. Test Cases That Relative with Sample Program

Test Case	B^T_1	B^F_1	B^T_2	B^F_2	B^T_3	B^F_3	B^T_4	B^F_4
T_1	x		x			x		
T_2		x		x	x			x
T_3		x	x			x		
T_4		x	x		x		x	
T_5		x		x	x		x	

The objective is to create a minimized test suite using the HGS test suite minimization algorithm [20] using only one criterion and the data shown in table 3. Initially, since both B_1^T and B_4^F branches are only covered by the T_1 and T_2 test cases, these two test cases are added to the minimum suite. Then, all branches covered by these two test cases are marked. Test case T_3 is redundant since it covers branches previously covered by the T_1 and T_2 test cases. Therefore, it is removed. In the next step, the B_4^T branch remains uncovered. The remaining T_5 and T_4 test cases cover this branch. The T_4 test case is selected randomly. The minimum test suite includes {T_1, T_2, T_4}, and all branches are covered by this suite. If we look closely at Figure 1, it can be seen that test case T_3 that detects the divide by zero fault (line 13) is not included in the minimum suite. Therefore, the effect of fault detection will decrease in the minimum test suite. This problem can be overcome by expanding this algorithm using a different criterion. It is clearly seen that hybrid algorithms perform better than single criterion algorithms and can detect more faults.

4. Conclusion

Regression testing is applied to modified components of software. Regression testing reruns all previous tests successfully executed by the software in order to ensure that the software has not randomly failed in previous operations. Different techniques are used for reducing regression test complexity amongst which is the test case prioritization technique. Test case prioritization is an approach for regulating test cases in order to minimize time consumption, costs, and efforts, and to increase fault detection rate. The regression test and different activities in a test were initially explained in this article. Different test case prioritization techniques were then reviewed. Finally, it was observed that multiple criteria-based prioritization techniques have the potential to improve the effect of regression testing techniques than single criterion-based approaches.

References

[1] S. Elbaum, P. Kallakuri, A.G. Malishevsky, G. Rothermel, S. Kanduri, Understanding the Effects of Changes on the Cost-Effectiveness of Regression Testing Techniques, Journal of Software Testing, Verification, and Reliability, pp. 65-83, 2003.

[2] G. Rothermel, R.J. Untch, C. Chu, Prioritizing Test Cases for Regression Testing, IEEE Transactions on Software Engineering, pp. 929-948, 2001.

[3] S. Yoo, M. Harman, Regression Testing Minimization, Selection and Prioritization: A Survey, Software Testing, Verification and Reliability, Vol. 22, No. 2, pp. 67-120, 2010.

[4] W.E. Wong, J.P. Horgan, S. Londonm, H. Agrawal, A Study of Effective Regression Testing in Practice, 8th International Symposium on Software Reliability Engineering, pp.230–238, 1997.

[5] G. Rothermel and S. Elbaum, Putting Your Best Tests Forward, IEEE Software, Report, pp.22-25, 2003.

[6] S. Elbaum, A. Malishevsky G. Rothermel, Incorporating Varying Test Costs and Fault Severities into Test Case Prioritization, 23rd International Conference Software Engineering, pp. 329-338, 2001.

[7] H. Srikanth, L. Williams, J. Osborne, System Test Case Prioritization of New and Regression Test Cases, 4th International Symposium on Empirical Software Engineering, pp.62-71, 2005.

[8] K.R. Walcott, M.L Soffa, G.M. Kapfhammer, R. Roos, Time-Aware Test Suite Prioritization, Software Testing and Analysis, pp.1-12, 2006.

[9] K.H.S, Hla, Y. Choi, J.S. Parl, Applying Particle Swarm Optimization to Prioritizing Test Cases for Embedded Real Time Software Retesting, Eighth IEEE international conference Computer and Information Technology Workshops, pp.527-532, 2008.

[10] A.A.F. Ahmed, M. Shaheen, E. Kosba, Software Testing Suite Prioritization Using Multicriteria Fitness Function, IEEE ICCTA, pp.160-166, 2012.

[11] N. Prakash, T.R. Rangaswamy, Modular Based Multiple Test Case Prioritization, IEEE International Conference on Computational Intelligence and Computing Research, pp.1-7, 2012.

[12] M.M. Islam, A. Marchetto, A., Susi,G. Scanniello, A Multi Objective Technique to Prioritize Test Cases Based on Latent Semantic Indexing, 16th European Conference Software Maintenance and Reengineering, pp.21-30, 2012.

[13] S.K. Mohapatra, S., Prasad, Evolutionary Search Algorithm for Test Case Prioritization, IEEE ICMIRA, pp.115-119, 2013.

[14] X. Wang, H., Zeng, Dynamic test case prioritization based on Multi-Objective, 15th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD), pp.1-6, 2014.

[15] M. Tyagi, S., Malhotra, Test Case Prioritization using Multi Objective Particle Swarm Optimizer, IEEE International Conference on Signal Propagation and Computer Technology (ICSPCT), pp.390-395, 2014.

[16] M. Ray, D.P. Mohapatra, Multi-objective test prioritization via a genetic algorithm, Innovations System Software Engineering Springer, pp.261-270, 2014.

[17] N. Prakash, K., Gomathi, Improving Test Efficiency through Multiple Criteria Coverage Based Test Case Prioritization, International Journal of Scientific & Engineering Research, Vol. 5, No. 4,pp.420-424, 2014.

[18] M. Tyagi, S. Malhotra, An Approach for Test Case Prioritization Based on Three Factors, MECS I.J. Information Technology and Computer Science, pp.79-86, 2015.

[19] A.Saini, S. Tyagi, MTCPA: Multi-Objective Test Case Prioritization Algorithm Using Genetic Algorithm, International Journal of Advanced Research in Computer Science and Software Engineering, Vol. 10, No.4, pp. 261–270, 2015.

[20] Harrold, M.J., Gupta, R., Soffa, M.L., A Methodology for Controlling the Size of a Test Suite, ACM Transactions on Software Engineering and Methodology, pp.270-285, 1993.