Performance Enhancement of Test Case Prioritization Using Hybrid Approach

Rajanroop Walia
D.A.V. Institute of Engineering and Technology
Jalandhar, India

Harpreet K. Bajaj
D.A.V. Institute of Engineering and Technology
Jalandhar, India

Abstract: Regression Testing is an indispensable part of software testing process. It validates all the modifications that have been introduced into the system throughout the development period. Although it is an expensive process in terms of time and cost, yet it cannot be avoided. Therefore, various techniques have been introduced in the past for reducing the expenses involved in this process. Test Case Prioritization is one such technique that schedules the execution order of test cases with an aim to improve the rate of fault detection. In this paper, a hybrid approach has been presented which is a combination of two approaches, Adaptive approach and Genetic algorithm. The approach works by firstly employing an adaptive approach to prioritize the test cases according to their statement coverage. Further, the leftover test cases are prioritized using Genetic Algorithm. Finally, the results of the proposed approach are compared with those of Genetic Algorithm based on two parameters: execution time and average percentage of statement coverage (APSC) values. The results confirm that the proposed approach performs better in terms of both parameters.

Keywords: regression testing, test case prioritization, genetic algorithms, adaptive approach.

1. INTRODUCTION

Regression testing aims to verify that the software still performs in the same manner as it did before it was changed [1]. However, regression testing can be expensive and time-consuming, especially when the test suite involved in testing the software is large. This limitation triggered the efforts to truncate these expenses and thus, led to the development of three main techniques namely, test case prioritization, test case selection and test suite minimization. Test case prioritization attempts to reorder the test cases so as to improve the rate of fault detection. Test case selection selects a subset of the original test suite for execution. Finally, test suite minimization shrinks the original test suite such that it still maintains the coverage. Among these techniques, test case prioritization is considered to be most efficient since it takes into account all the test cases contained in the test suite and identifies the best execution sequence that meets a certain testing criteria. This is not so in case of other two techniques as they do not cover all the test cases of a test suite and thus increase the risk of software containing undetected errors [2].

Various prioritization techniques have been proposed in the past including genetic algorithms, ant colony optimization, particle swarm optimization, history-based approach and adaptive approach. Among these techniques, Genetic algorithms are widely used in solving test case prioritization problems, by generating results using the techniques inspired by natural selection. But they consume too much time in doing so. This is so because they carry out test suite prioritization and execution as separate phases. On the other hand, an adaptive approach which is gaining popularity nowadays, saves time by carrying out prioritization and execution of test cases simultaneously. But it only schedules the order of those test cases which have achieved some amount of statement coverage on the previous program. This means all of the test cases are not prioritized by an adaptive approach, which further implies that full statement coverage has not yet been achieved. Therefore, a hybrid approach has been proposed in this paper, which is a combination of the above two approaches. It overcomes the limitations of both the approaches by achieving almost 100% statement coverage in minimum time.

This paper is organized as follows. Section 2 describes related work. Section 3 explains some existing test case prioritization approaches. Section 4 describes the proposed work. Section 5 explains how the experiment is carried out and presents the results. Section 6 concludes the paper.

2. RELATED WORK

An in-depth analysis of regression testing was presented in order to remove the constraints associated with it. In [1] Y. Li gave a detailed description of regression testing including its definition and types. Apart from this, they also compared the *retest all* and *selective* regression testing strategies and concluded that there is tradeoff between the both. However, it was explained in [2] that as the size of test suite increases, *retest all* strategy becomes infeasible because of time and cost constraints. Thus, it revealed an increasing trend towards the different techniques to remove these constraints namely, test case prioritization, test suite minimization and test case selection. However, test case prioritization gained much popularity which is evident from the vast amount of work that has been done in this field. Y.C. Huang in [3] proposed a cost-cognizant prioritization technique that ordered test cases according to their history information by using genetic algorithm. The technique prioritized test cases on the basis of their test costs and fault severities, without analyzing the source code. Its efficiency was evaluated using a UNIX utility program and the results confirmed the usefulness of the proposed technique. In [4], a technique for identifying the test path that must be tested first in case of static testing was proposed. Test paths or scenarios were derived from source code. In order to find the path to be tested first, the approach made use of Information Flow model and Genetic Algorithm.Y. Huang in [5], proposed a method of cost-cognizant test case prioritization which was based on the use of historical records. The historical records were gathered from the latest regression testing and then a genetic algorithm was proposed to determine the most effective order. Evaluation results proved that the proposed method improved the fault detection effectiveness. In [6], the necessity of Component-Based Software testing prioritization framework was developed and proved, which uncovered more extreme bugs at an early stage and enhanced software product deliverable quality utilizing Genetic Algorithm with java decoding technique. For this, they proposed a set of prioritization keys. An algorithm to prioritize test cases based on total coverage using a modified genetic algorithm was proposed in [7]. The performance of the proposed algorithm was compared with five other approaches and the results indicated that the proposed algorithm was better than other approaches. However, the same could not be guaranteed for bigger test suites. In [8], Y. Singh presented a regression test prioritization technique based on Ant Colony Optimization to reorder test suites in time constrained environment. On the other hand, a modified version of Ant Colony Optimization for test case prioritization was also presented in [9]. The performance in both the cases was evaluated using the Average Percentage of Faults detected (APFD) metric and the results proved the effectiveness of these techniques. Tyagi in [10] proposed a 3-step approach to perform regression testing using Multi Objective Particle Swarm Optimization. The proposed MOPSO outperformed other approaches like No Ordering, Reverse Ordering and Random Ordering as it achieved maximum fault coverage and maximum value of APFD in minimum execution time. In [11], history-based approach for prioritizing the test cases was extended to modified lines. The modified lines were prioritized first and then subsequently followed by the test cases. The results showed that the proposed approach was able to detect faults faster and with less effort as compared to previous approach. Dan Hao in [12] presented an adaptive TCP approach, which worked by determining the test case execution order simultaneously during the execution of test cases on the modified program. The results showed the proposed adaptive approach to be significantly better than the total test case prioritization approach and comparable to additional statement-coverage based test case prioritization approach. In [13], L. Mei proposed Preemptive Regression Testing (PRT), a novel strategy that rescheduled test cases based on the changes of the service under test detected in the course of each actual regression test session. Three particular PRT strategies, integrated with existing test case prioritization techniques were proposed to generate new techniques. The experimental result confirmed that one of the PRT-enriched techniques was able to test workflow-based web service. A novel family of input-based local-beam-search adaptive-randomized techniques was proposed in [14]. The results showed that these techniques achieved either higher or same mean APFD values as the existing code-coverage-based greedy or search-based prioritization techniques. A. Schwartza in [15] empirically studied the existing strategies and developed two additional Adaptive Test Prioritization (ATP) strategies using fuzzy analytical hierarchy process (AHP) and the weighted sum model (WSM). The empirical studies provided in this research showed that utilizing these strategies can improve the cost-effectiveness of regression testing.

3. EXISTING TEST CASE PRIORITIZATION APPROACHES

3.1 Genetic Algorithm

Genetic algorithm is an evolutionary method that generates solutions to optimization problems using the techniques which are based on the principles of natural selection. It works by repeatedly evolving a population of individuals represented as chromosomes, towards a better solution. During each step, it chooses individuals from the current population based on their fitness values, which are calculated in accordance to some objective function in the problem being solved. Once the best fit chromosomes are selected, they are then modified by applying the following genetic operators in order to produce the next generation:

a) Crossover: Crossover operator is used to vary the chromosomes from one generation to next in such a manner that the new chromosome formed after applying crossover is better than original chromosomes. In other words, it mimics the process of biological evolution by taking more than one chromosomes as parents and then producing a child chromosome from them. In case of one-point crossover, a random crossover point is selected in both the parent chromosomes and then their tails are swapped to get new off-springs as shown below:

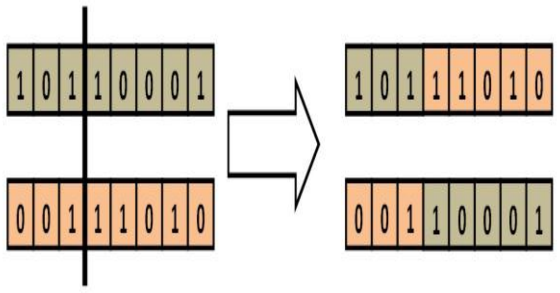

Fig 1: Crossover operation

b) Mutation: Mutation operator is applied to inject diversity in the population of chromosomes by altering one or more gene values in a chromosome. It can lead to a solution which is entirely different from the previous solution. In case of bit-flip mutation, one or more random bits are selected and flipped as shown below:

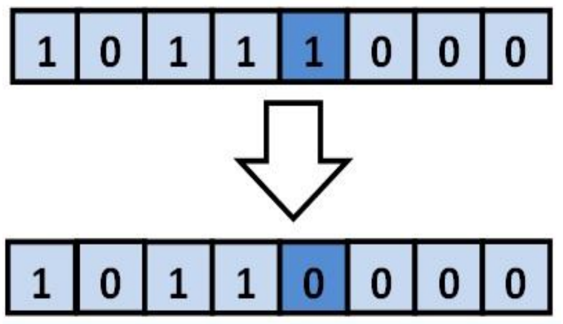

Fig. 2: Mutation operation

3.2 Adaptive Approach

An adaptive approach for solving prioritization problems has gained much popularity in the recent years. Unlike existing test case prioritization approaches that prioritize the test cases before running them on the modified program, an adaptive approach works by prioritizing the test cases simultaneously during their execution. It does so by calculating the initial fault detection capability (denoted by *Priority (t)*) of each test case according to its statement coverage on the previous program and selects a test case t_s with the largest *Priority*. This *Priority (t)* is given by the following equation:

$$Priority(t) = \sum_{S\ is\ executed\ by\ t} Potential(s) \qquad (1)$$

where *Potential (s)* denotes how likely a statement s contains faults that have not been discovered by the existing test suite where *Potential (s)* of any statement lies in the interval [0,1]. Initially, all statements have *Potential* 1. The adaptive approach then runs the test case with the largest priority and then modifies the *Potential* of each statement *s* according to whether its output is passed or failed. In other words, it modifies the *Potential* on the basis of the following equation:

$$Potential(s) = \begin{cases} Potential'(s), & s\ is\ executed\ by\ t' \\ Potential'(s) * q, & s\ is\ executed\ by\ t' \wedge t'\ is\ passed \\ Potential'(s) * p, & s\ is\ executed\ by\ t' \wedge t'\ is\ failed \end{cases} \quad (2)$$

where *Potential'(s)* represents the probability of any statement comprising new faults before running any test case *t'*. *p* and *q* are two non-negative constants, whose value lies between 0 and1. This process is repeated until all the test cases are prioritized and executed.

4. PROPOSED WORK

Genetic Algorithms provide excellent solutions to prioritization problems but take significant amount of time to do so. This is so because firstly they schedule the order of test cases and then execute them. On the other hand, adaptive approach prioritizes the test cases on the basis of their output information. In other words, test case prioritization and execution take place simultaneously in case of adaptive approach. Since both processes occur concurrently, time expenses are reduced to a great extent. But adaptive approach prioritizes only those test cases that achieve some amount of statement coverage. The test cases which are unable to cover any statements are left non-prioritized which implies that statement coverage has not been done perfectly. Therefore, in order to prioritize all the test cases, a hybrid approach has been designed. In this approach, the test cases that cover the code statements are prioritized first using an adaptive approach. The leftover test cases that do not cover any statement are prioritized using Genetic Algorithm by applying four operations: parent selection, crossover, mutation and duplicate elimination. The benefit of this approach is that besides saving time, it achieves almost 100% statement coverage.

The step-by-step working of hybrid approach is shown below in the flowchart given in Figure 3:

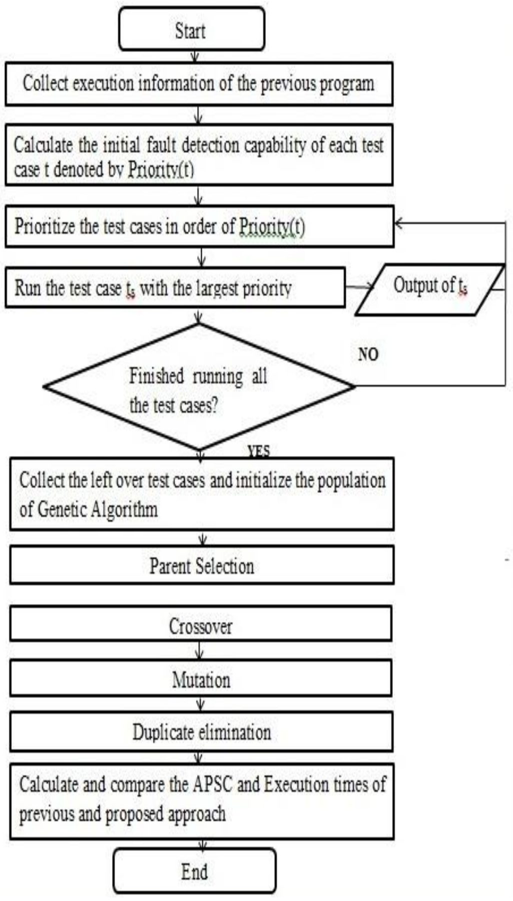
Start
Collect execution information of the previous program
Calculate the initial fault detection capability of each test case t denoted by Priority(t)
Prioritize the test cases in order of Priority(t)
Run the test case ts with the largest priority
Output of ts
NO
Finished running all the test cases?
YES
Collect the left over test cases and initialize the population of Genetic Algorithm
Parent Selection
Crossover
Mutation
Duplicate elimination
Calculate and compare the APSC and Execution times of previous and proposed approach
End

Fig. 3: Flowchart of the proposed technique

Finally, the efficiency of the proposed approach is evaluated by comparing its results with those of Genetic Algorithm on the basis of two parameters: APSC and Execution time.

5. EXPERIMENTAL EVALUATION

In order to prove the effectiveness of the proposed technique, 100 test cases along with their statement coverage have been collected from Apache Open Source by interfacing it in Eclipse and then testing it with Junit test toolkit. This dataset has been used for implementation of the proposed approach. For the purpose of comparison, Genetic Algorithm has also been implemented on the same dataset. Post implementation, the performance of both the approaches have been calculated according to two parameters: Execution Time and Average Percentage of Statement Coverage (APSC) values. APSC is defined as the degree to which a prioritized test suite covers the statements. It is calculated as shown below:

$$APSC = 1 - \frac{TS_1 + TS_2 + \cdots + TS_m}{nm} + \frac{1}{2n} \quad (3)$$

where

TSi denotes the id of first test case that first covers the statement i in the execution sequence.

M denotes the number of statements.

N denotes the number of test cases.

The first set of each of these values for both the techniques has been acquired by altering the number of test cases in the dataset, as given by Table 1. This is done by creating four different subsets of the original dataset, containing 25, 50, 75 and 100 test cases respectively. Figures 4 and 5 show the bar graphs for APSC and Execution Time respectively. It is clearly visible that the proposed hybrid approach maximizes the statement coverage up to 5 percent and minimizes the execution time to a great extent.

Table 1: Comparison based on the number of Test Cases

No. of Test Cases	APSC values (in %)		Execution Time Values (in ms)	
	APSC (GA)	APSC (HY)	Time (GA)	Time (HY)
25	98.7	99.9	51394	14950
50	97.35	99.79	114518	61974
75	97.23	99.61	187018	115882
100	95.91	99.57	331653	292261

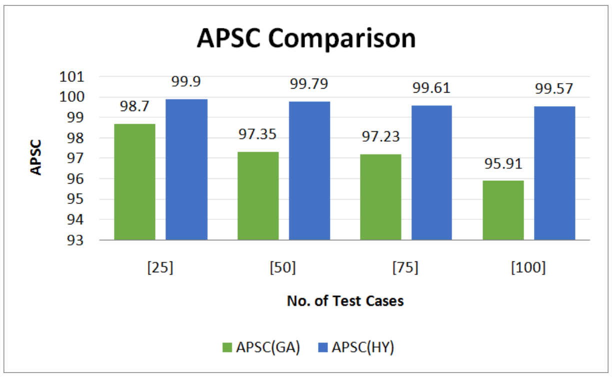

Fig. 4: Graph showing APSC values of Genetic Algorithm and Hybrid Approach corresponding to the number of test cases.

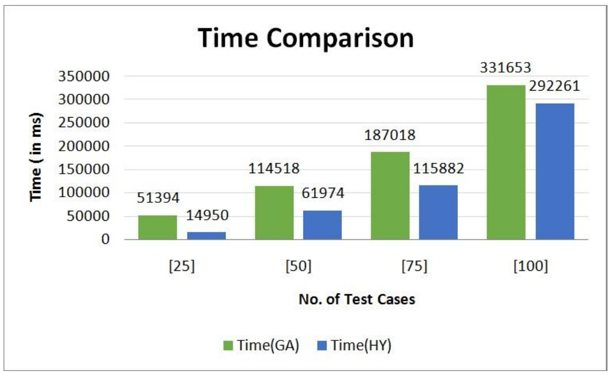

Fig. 5: Graph showing Execution Time values of Genetic Algorithm and Hybrid Approach corresponding to the number of test cases.

The next set of APSC and execution time values of both approaches has been obtained by taking into account, the number of generations. Figures 6 and 7 show the bar graphs for APSC and Execution Time respectively. Figures 8 and 9 show the line graphs for the same. From both the graphs, it can be observed that the proposed hybrid approach outperforms the genetic algorithm by 5 percent in terms of APSC values. As far as execution time is concerned, a significant difference can be observed in that also.

Table 2: Comparison based on the number of Generations

No. of Generations	APSC values (in %)		Execution Time values (in ms)	
	APSC (GA)	APSC (HY)	Time (GA)	Time (HY)
[2]	97.53	99.74	462020	212564
[3]	97.4	99.6	533850	327632
[4]	96.67	99.59	538903	338389
[5]	94.55	99.5	660924	491002

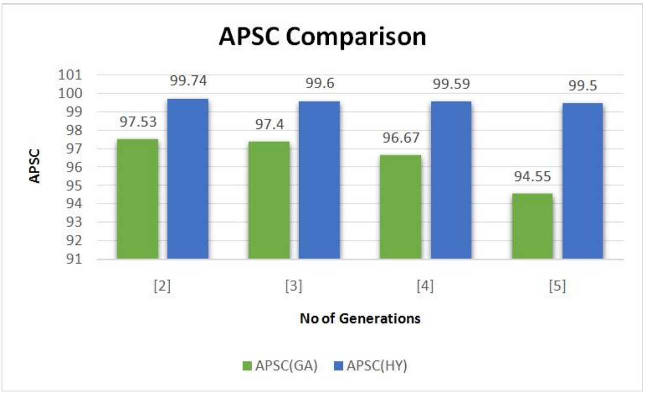

Fig. 6: Graph showing APSC values of Genetic Algorithm and Hybrid approach corresponding to number of Generations.

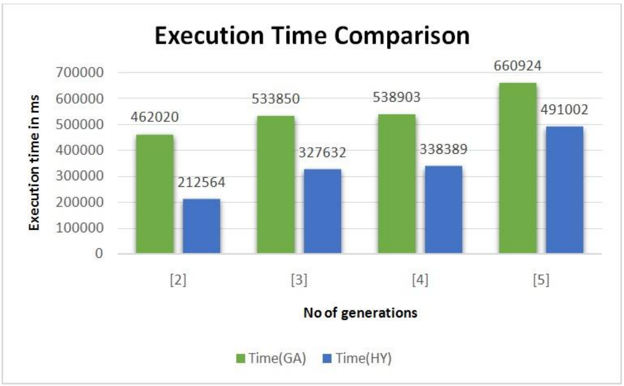

Fig. 7: Graph showing Execution Time values of Genetic Algorithm and Hybrid approach corresponding to number of Generations.

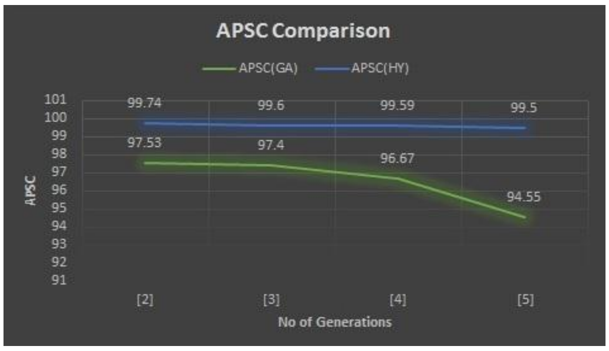

Fig. 8: Graph showing APSC values of Genetic Algorithm and Hybrid approach corresponding to number of Generations.

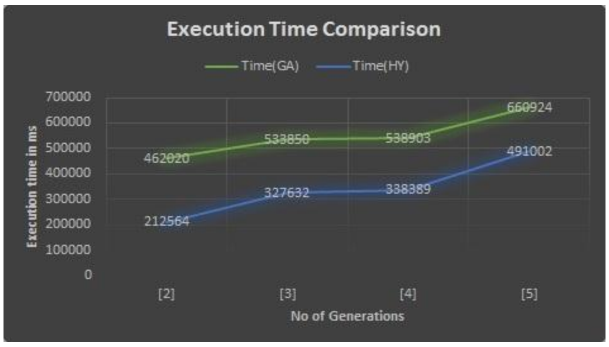

Fig. 9: Graph showing Execution Time values of Genetic Algorithm and Hybrid approach corresponding to number of Generations.

6. CONCLUSION

In this paper, two test case prioritization approaches, adaptive approach and genetic algorithms, have been combined to form a hybrid approach. Unlike other prioritization approaches, adaptive approach carries out prioritization and execution of test cases simultaneously. Firstly, it selects a test case according to its initial fault detection capability (priority) in the previous program. Then it executes that test case and records its output. Based on the output of first test case and the execution history of next unselected test case, it prioritizes that test case. This process continues till all the test cases which cover code statements are prioritized and executed. Further, the test cases that are unable to cover any statements are taken by Genetic algorithm and prioritized using four operations, parent selection, crossover, mutation and duplicate elimination. The performance of the hybrid approach is further compared with that of Genetic Algorithm. The experimental results show that the proposed approach outperformed the latter in terms of execution time and APSC values.

7. ACKNOWLEDGMENTS

I express my sincere gratitude and thanks to Ms. Harpreet K. Bajaj, Head of Department, Computer Science and Engineering at DAVIET Jalandhar. Under her valuable guidance and continuous support, I have been able to complete this paper.

8. REFERENCES

[1] Li Y, Wahl N J. (1999). An Overview of Regression Testing. ACM SIGSOFT Software Engineering Notes, 25(1), 69-73.

[2] Yoo S, Harman M. (2012). Regression testing minimization, selection and prioritization: a survey. Software Testing, Verification and Reliability, 22(2), 67-120.

[3] Huang Y C, Huang C Y, Chang J R. (2010). Design and Analysis of Cost-Cognizant Test Case Prioritization Using Genetic Algorithm with Test History. In: Proceedings of 34th IEEE Annual Computer Software and Applications Conference, 413-418.

[4] Sabharwal S, Sibal R, Sharma C. (2011). A Genetic Algorithm based Approach for Prioritization of Test Case Scenarios in static testing. In: Proceedings of International Conference on Computers and Communication Technology (ICCCT), 304-309.

[5] Huang CY, Peng KL, Huang YC. (2012). A history-based cost-cognizant test case prioritization technique in regression testing. Journal of Systems and Software, 85(3), 626-637.

[6] Mahajan S, Joshi S D, Khanaa V. (2015). Component-Based Software System Test Case Prioritization with Genetic Algorithm Decoding Technique Using Java Platform. In: Proceedings of IEEE International Conference on Computing Communication Control and Automation, (ICCUBEA), 847-851.

[7] Ramingwong L, Konsaard P. (2015). Total Coverage Based Regression Test Case Prioritization using Genetic Algorithm. In: Proceedings of 12th IEEE International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology (ECTI-CON), 1-6.

[8] Singh Y, Kaur A, Suri B. (2010). Test Case Prioritization using Ant Colony Optimization. ACM SIGSOFT Software Engineering Notes, 35(4), 1-7.

[9] Solanki K, Singh Y, Dalal S. (2015). Test Case Prioritization: An Approach Based on Modified Ant Colony Optimization (m-ACO). In: Proceedings of IEEE International Conference on Computer, Communication and Control (IC4), 1-6.

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

[11] Gupta A, Mishra N, Tripathi A, Vardhan M, Kushwaha DS. (2015). An Improved History- Based Test Prioritization Technique Using Code Coverage. Advanced Computer and Communication Engineering Technology, 315, 437-448.

[12] Hao D, Zhao X, Zhang L. (2013). Adaptive Test-Case Prioritization Guided by Output Inspection. In: Proceedings of 37th IEEE Annual Computer Software and Applications Conference (COMPSAC), 169-179.

[13] Mei L, Chan W K, Tse T H, Jiang B. (2015) Preemptive Regression Testing of Workflow-based Web Services. IEEE Trans. On Services Computing, 8 (5): 740-754.

[14] Jiang B, Chan W K. (2015). Input based adaptive randomized test case prioritization: A local beam search approach. Journal of Systems and Software, 105, 91-106.

[15] Schwartza A, Do H. (2016). Cost-effective regression testing through Adaptive Test Prioritization strategies. Journal of Systems and Software, 115, 61-81.