A Hybrid Approach for Test Case Prioritization and Optimization using Meta-Heuristics Techniques

[1]Pavi Saraswat and [2]Abhishek Singhal
Department of CSE, ASET
Amity University Uttar Pradesh
Noida, India
[1]pavisaraswat@gmail.com, [2]asinghal1@amity.edu

***Abstract*—Software testing is a very crucial and important phase for (SDLC) software development life cycle. Software is being tested on its effectiveness for generating good quality software. Regression testing is done by considering the constraints of resources and in this phase optimization of test suite is very important and crucial. This paper mainly aims to make use of hybrid approach of meta-heuristics, It comprises of two algorithms first is genetic algorithm and second is particle swarm optimization. In addition to algorithm the comparison of proposed algorithm hybrid GA_PSO with other optimization algorithms are been done. To validate the research Average Percentage Fault Detection (APFD) metric is used for comparison and fitness evaluation of the proposed algorithm.**

Keywords—Software testing; regression testing; test case prioritization; test case optimization; genetic algorithm; partucle swarm optimization.

I. INTRODUCTION

In the life cycle of software development, software testing is a crucial phase and it consumes substantial amount of work. It is a process of exercising software with the intent of revealing defect and evaluating quality [5]. A good test case is the one which has high probability of detecting yet undetected errors and test results should be inspected meticulously.

Regression testing is one of the software testing type in which the previously developed and tested software is verified that it works properly after modification or changes in the previous code [9]. These changes can be software version updates, configuration and many more. It is performed under greater time constraint as less time is provided for this in schedule[15]. Previously generated test cases, test suites and test plans can be used again in this and it tests only the changed or modified part not the whole software code.

Test suite prioritization is a technique in which multiple test suites are present in test suite pool from which the best test suite is chosen and it is done on the basis of some criteria like average percentage fault detection[13], code coverage[11], statement coverage and many more. Optimization of test suites can be done from many ways like genetic algorithm [15], ant colony optimization [10], particle swarm optimization [12], bee colony optimization [11] and many other heuristic approaches. In this initially a population of multiple combinations of test suite is taken from which the best one is selected to run in the process of testing.

In software testing initial phase is of creating test cases which validates all possible conditions but there are no such specific tools which can exhaustively work in every condition of testing the software[14]. Automatic test case generation was introduced because of the fact that human errors can create a huge loss in testing phase and can even create inappropriate test cases which can even increase the overall cost of the test cases. There are four basic methodologies to create test cases[16] which are as follows:

a) Artificial Intelligence Based Test Case Generators:

Here there are multiple combination (sets) of test cases which are liable to uncover the faults and these faults are potent which have potential to get detected and are not defined properly[17] . Here there are no such boundaries defined for the faults so it becomes difficult to generate or design the test cases that can detect the error. In some cases when there is a halt in executing the test cases or a local optimization problem is achieved and there are no such algorithms which can solve the problem. So to solve those problems we go for heuristic approaches for generating the test cases. Here Case Based Reasoning (CBR) is used in accordance with backtracking method.

b) Use Case Based Test Case Generators:

In today's scenario almost 50% of total production cost is used for the testing. So to save cost and time use case diagrams are used as a source for the test case generations and it is pictorial representation that is based on the Unified Modelling Language (UML).

c) Use Case Diagrams are helpful because of the following reasons:

- It helps customer in knowing what to expect?
- It helps developer in knowing what to code?
- It helps technical writer in knowing what to document?
- It helps tester in knowing what to test?

Test cases can be designed or generated from a use case scenario by initially generating a full set of scenarios[18] . Then from all the scenarios generate test cases and conditions to execute them. Lastly for all the test cases data values should be provided or identified for testing.

978-1-4673-6984-8/16/$31.00 © 2016 IEEE

d) Test Case Generation By Specification Based Software Testing:

Here test cases are designed on the basis of functional and the requirement specifications. But while transforming the functional specification in test cases there are few issues like it should be user friendly, easily convertible into machine language and small test data inputs are acquired due to this[19] . In order to achieve these conditions a tool called CAT (Computer Assisted Testing) is used. The CAT tool uses Test Script Graph (TSG) and Test Script Language (TSL) to convert the functional specifications into the test cases.

e) GNU/ Linux as a Testing Tool:

All these operating system with .NIX act as testing tool in their own and in them many functionality [20] and tools are available in them to test the software.

This paper is basically divided in 5 Sections; Section 2 describes literature review and background overview of test suite reduction using Hybrid GA_PSO. Section 3 describes proposed work and algorithm of hybrid GA_PSO. Section 4 describes results in which studies are done and also the comparison of the results of algorithms is shown and section 5 includes the conclusion and future scope for work of the work study done.

II. Literature Review

This section basically surveys the previous techniques and algorithms used for the prioritization of the test cases. Test case prioritization is a process of finding a order of test cases which is set on the basis of some criteria and after running gives the optimized output. There are two most important things to focus on while considering test case prioritization, those are fault detection [13] and fault impact. Fault detection rate should be maximized after running the prioritized test case because that is the aim. Fault impact should also be maximized as this is the main criteria to decide optimized test cases and test suites. There are many algorithms used for test case prioritization some of which are genetic algorithm, glowworm optimization, particle swarm optimization, ant colony optimization, artificial bee colony optimization and many more. [1] Defines a hybrid genetic algorithm technique which works on multiple criteria as its fitness function like code coverage and APFD, but this hybrid algorithm prioritizes the test cases using complete total code coverage of test case. Automation of test cases is done by this hybrid genetic algorithm. And in this paper comparison of the proposed technique is done with other test case prioritization techniques. The technique proposed in [2] is a test case prioritization which is cost cognizant and which is meant to remove the limitations of other prioritization techniques defined in that paper. The main motive of this technique is basically to find a test case from the existing test suite and this technique has greatest efficacy in terms of cost cognizant that too in the foregoing regression testing. The technique defined in [3] is using 100 test cases and 1000 faults and these faults are detected by improved genetic algorithm and after prioritization of test cases using improved genetic algorithm most number of faults are detected in less number of test case execution.

The technique proposed in [4] is multi objective particle swarm optimization (MOPO) which optimizes two factors firstly fault coverage and secondly execution time. Here the priority that is which test case is preferable over others for test cases is calculated by the ratio of two factors; fault coverage and the execution time, higher is the ratio maximum is the priority of respective test case. An experimental analysis is performed on the maximum number of faults and time taken for execution is less. Comparison of this technique is done with other test case prioritization techniques. [5] Proposes technique for prioritization of test cases which is based on potential code coverage and testing time. Evolutionary search algorithm reorders test suite using genetic algorithm in any order so that maximum code is covered by test suite. In [6] a technique of framework is proposed that will prioritize test case on black box testing on the new software product by using the ant colony optimization and the execution history of test cases on similar types of product. A simulation of this algorithm is shown on two products for the practicality of technique and effectiveness of the approach. In [7] a component based software test case prioritization framework is proposed which uncovers the extreme bugs in early stages using genetic algorithm on java platform. For this framework, a set of prioritized keys are also proposed like size, scope of the code, bugs inclination, and information stream. The integrity of keys is measured using key assessment metric.

In [8] technique proposed prioritizes the test cases on the basis of total code coverage using modified genetic algorithm as it uses some concepts of bee colony optimization to modify the genetic algorithm. As for the complex problems, the complex algorithm works better like bee colony than simple genetic algorithm. Here the performance is evaluated on the basis of average percentage condition coverage (APCC) and comparison of proposed technique is done with other test case prioritization techniques. [9] Proposes a technique which prioritizes test cases using ant colony optimization. This technique considers three factors for prioritizing fault severity, number of faults detected and execution time. These three factors are used by ant colony optimization to detect impactful faults at the earlier stage of the testing. The effectiveness is calculated by using average percentage fault detection (APFD) metric.

III. Proposed Hybrid Prioritization Technique

The proposed technique hybrid GA_PSO consists of basic two parts first part of the algorithm is genetic algorithm (GA) [15] code and after running the genetic algorithm code on initial population we have optimized set of population after some amount of iterations. Then the optimized population of the genetic algorithm is given to the particle swarm optimization (PSO) code so it has optimized population of genetic algorithm as its initial population. Now in the second part the particle swarm optimization code is been executed then in the output we have better results which are optimized according to the criteria Average Percentage Fault Detection (APFD). The output is more optimized after execution of second part. And here 'N' in the detailed flow chart is the number of generation for the algorithm to run. And it can be specified according to the need.

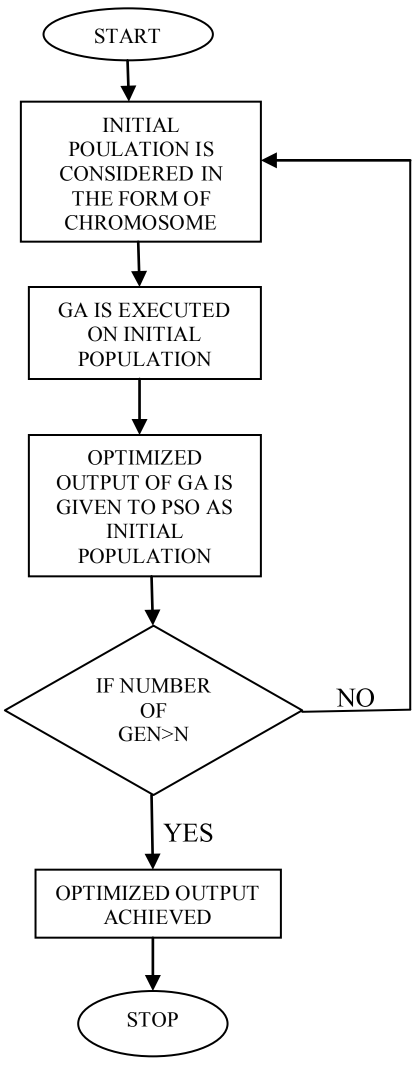
START
INITIAL POULATION IS CONSIDERED IN THE FORM OF CHROMOSOME
GA IS EXECUTED ON INITIAL POPULATION
OPTIMIZED OUTPUT OF GA IS GIVEN TO PSO AS INITIAL POPULATION
IF NUMBER OF GEN>N
NO
YES
OPTIMIZED OUTPUT ACHIEVED
STOP

Fig. 1. Proposed hybrid technique of GA and PSO

The genetic algorithm is defined in detailed flow chart below. In genetic algorithm, the initial population is set of chromosomes of size 20 which means a test suite contains 20 test cases. Initial population of test cases is generated randomly by the random number generator. Initial population is being encoded in the form of permutation encoding because we have 20 unique number of test cases. Then selection is done which is random selection from the initial population of test cases for the other operators to perform on. The fitness function on the basis of which better offspring are selected is (APFD) Average Percentage Fault Detection, which is also the criterion to choose the best test suite among the population. APFD is the measure of the fault detection per percentage rate of the test suite execution. The calculation of APFD is done by taking weighted average of the faults detected percentage during the test suite execution. It is ranged between 0 and 100. The formula of APFD which is the fitness function in the genetic algorithm code is given in equation 1:

$$APFD = 1-\{(Tf1+Tf2+Tf3+\ldots\ldots+Tfm)/mn\} + (1/2n) \quad (1)$$

Here in the above defined equation 'n' value is the total number of test cases of fault matrix; 'm' is the total number of faults of fault matrix and (Tf1, Tf2, Tf3……Tfm) values are the position of first test that expose the faults in the fault matrix. And 'N' is the number of generations.

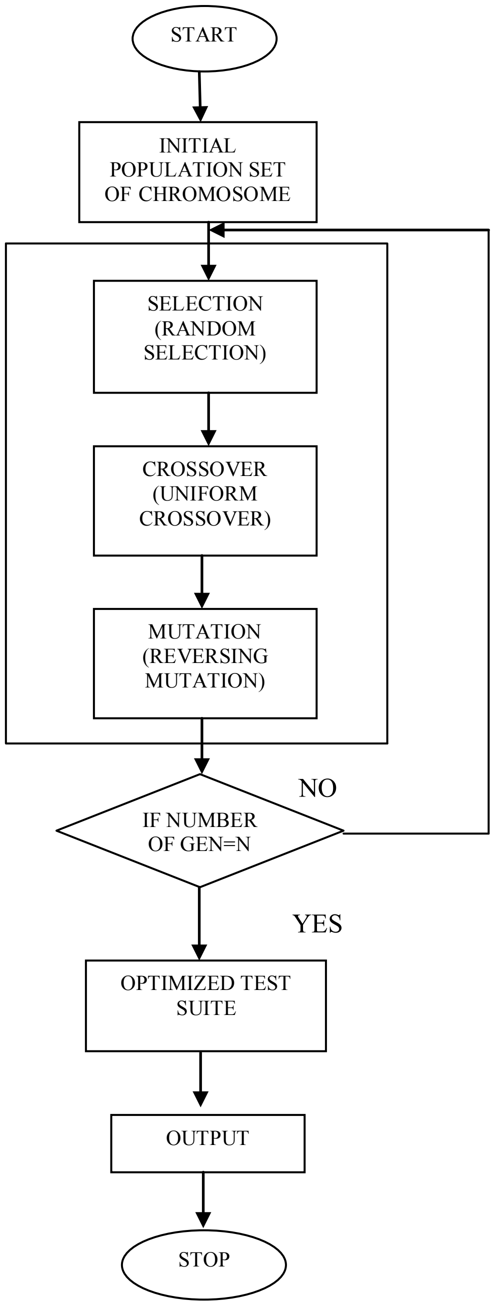
START
INITIAL POPULATION SET OF CHROMOSOME
SELECTION (RANDOM SELECTION)
CROSSOVER (UNIFORM CROSSOVER)
MUTATION (REVERSING MUTATION)
IF NUMBER OF GEN=N
NO
YES
OPTIMIZED TEST SUITE
OUTPUT
STOP

Fig. 2. Detailed work flow of genetic algorithm

The detailed work flow of PSO is shown below. Initially particle swarm optimization (PSO) is provided with random particles which are results or solutions and then updating of the generations done after that optimal solution can be achieved. In all the iterations, each particle of algorithm is updated by two factors (values) which are personal best (pbest) and global best (gbest). Personal best (pbest) is the fitness (solution) which is obtained so far and fitness is then stored for future purpose. Global best (gbest) is then tracked by optimizer of particle swarm optimization and it is the finest value achieved by any particle till that iteration. The main motive of this phase is to bring other solution near to the gbest which is best solution among all. After having the two of the best values, then velocity is calculated and positions are updated accordingly. Equation (2) [3] and equation (3) [3] defines velocity and updating of positions.

$$v[] = v[] + c1 * rand() * (pbest[] - present[]) + c2 * rand() * (gbest[] - present[]) \quad (2)$$

$$present[] = present[] + v[] \quad (3)$$

Here v[] is velocity of particle, present[] is the particle solution(fitness) of current population, rand() is number taken randomly between 0 and 1, c1 and c2 are the learning factors which are usually taken equal to 2.

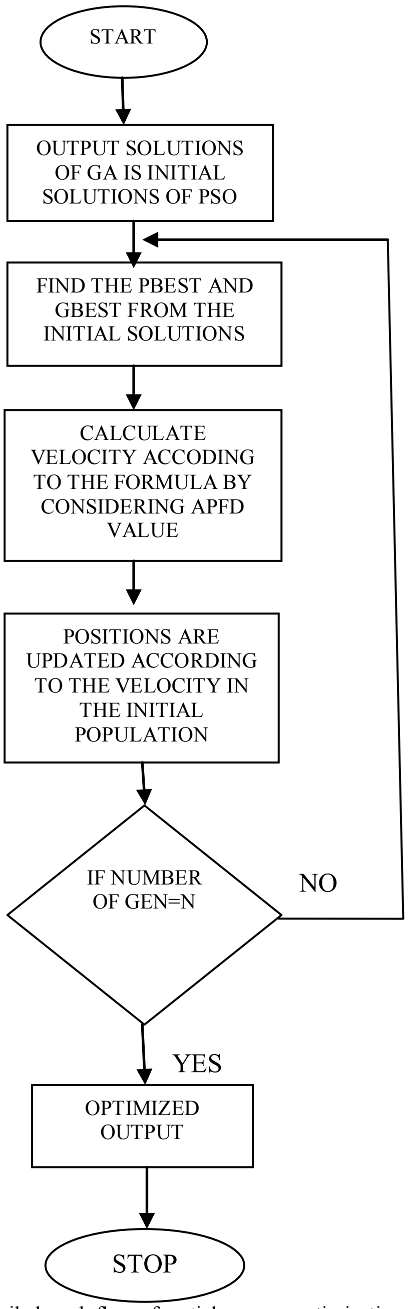
START
OUTPUT SOLUTIONS OF GA IS INITIAL SOLUTIONS OF PSO
FIND THE PBEST AND GBEST FROM THE INITIAL SOLUTIONS
CALCULATE VELOCITY ACCODING TO THE FORMULA BY CONSIDERING APFD VALUE
POSITIONS ARE UPDATED ACCORDING TO THE VELOCITY IN THE INITIAL POPULATION
IF NUMBER OF GEN=N
NO
YES
OPTIMIZED OUTPUT
STOP

Fig. 3. Detailed work flow of particle swarm optimization

IV. Results

The proposed algorithm hybrid GA_PSO is been validated for its effectiveness using average percentage fault detection metric. It is being implemented in java. For this case study test cases of an e-learning website is been taken into consideration. From those test cases, multiple fault matrix is being created which have test suite number and the number of faults detected by the respective test suite. The proposed algorithm performs test suite prioritization and the output given by the algorithm is the optimized test suite. For this experimental study two fault matrices tables are taken into account and then the proposed hybrid algorithm runs for specific fault matrix table and provide the fitness in terms of APFD percentage and the optimized test suite.

TABLE I. Test Case Fault- Matrix For First Study

	T 1	T 2	T 3	T 4	T 5	T 6	T 7	T8	T 9	T10	T11	T12	T13	T14
F1						X	X	X	X					
F2											X	X	X	
F3									X	X	X			
F4					X							X		
F5												X	X	
F6											X	X		
F7										X		X	X	X
F8					X			X						
F9	X					X	X	X	X					
F1 0									X					

Table 1 defines the fault matrix for first study which is having 15 test cases in a test suite. The x represent that fault is detected by the particular test case. After applying the multiple set of test cases are generated in the population and best test suite is selected from that on the basis of the APFD value of that test suite. Table 2 describes different types of test cases order in different prioritization techniques, the order provided by the hybrid GA_PSO is the output of the proposed algorithm for APFD table defines for first study. These different techniques order will in comparing the result of the proposed algorithm with others. It is then followed by a table which defines APFD percentage for all techniques in table 2.

TABLE II. Order Of Test Cases For Various Test Case Prioritization Approaches For First Study

ORIGINAL ORDER	RANDOM ORDER	REVERSE ORDER	OPTIMAL ORDER	HYBRID GA_PSO
T1	T6	T15	T6	T9
T2	T9	T14	T11	T11
T3	T1	T13	T5	T5
T4	T12	T12	T10	T1
T5	T15	T11	T12	T10
T6	T3	T10	T1	T12
T7	T8	T9	T9	T15
T8	T5	T8	T8	T6
T9	T2	T7	T13	T13
T10	T13	T6	T15	T4
T11	T4	T5	T14	T8
T12	T7	T4	T3	T7
T13	T11	T3	T7	T3
T14	T14	T2	T2	T2
T15	T10	T1	T4	T14

Table 3 is showing the comparison of all the techniques considered in table 2 on the basis of average percentage fault detection (APFD) percentage. From the values achieved in table 3 it is clear that non-prioritized test case technique always provides non-optimal results. Random order can be near to optimal sometimes but can never be the best. Optimal order and Hybrid GA_PSO mostly provides near optimal solutions and which are mostly preferred. The proposed algorithm is giving solution better than optimal order in first study. And it is detecting more faults at earlier stages and is a better algorithm. As genetic algorithm is always a simple preferred approach and hybrid GA_ PSO performs well for fault detection.

TABLE III. COMPARISION BETWEEN HYBRIDGA_PSO AND OTHER PRIORITIZATION TECHNIQUES FOR FIRST STUDY

TEST CASE PRIORITIZATION TECHNIQUES	APFD (100%)
ORIGINAL ORDER/NON PRIORITIZED TEST CASE ORDER	65.68%
REVERSE ORDER	45.65%
RANDOM ORDER	39.73%
OPTIMUM ORDER	87.55%
HYBRID GA_PSO	89.78%

TABLE IV. TEST CASE FAULT- MATRIX FOR SECOND STUDY

	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11	T12	T13	T14	T15
F1			X			X	X					X	X		
F2				X				X	X	X					
F3							X		X						
F4								X	X	X					
F5												X	X	X	
F6				X						X	X	X			
F7										X	X		X		X
F8								X		X		X	X	X	
F9							X			X	X	X			
F10									X	X			X		X

Table 4 defines the fault matrix values for second study having 15 test cases in a test suite and it is provided to hybrid GA_PSO and the output is given in the form of best test suite among randomly generated population in the form of test suite.

Table 5 provides the order of test cases in a test suite in different test case prioritization techniques. The Hybrid GA_PSO defines order which is output of the proposed hybrid algorithm. Table 6 provides the comparative study of all the techniques named in table 5 on the basis of APFD percentage value.

In table 6 the values of APFD percentage for the algorithms like reverse order and non-prioritized order is not optimal as maximum faults are not being detected at early stages. But the value of optimum order is near to Hybrid GA_PSO order, which means that the proposed technique mostly provides the optimum results. The consistency in the result of hybrid GA_PSO is maintained from first study to the second study. As in the first study proposed technique hybrid GA_PSO gives the result 89.78% and now in second study it is 90.85% which shows the clear consistency of the technique. And again, optimal and hybrid GA_PSO provides almost similar values in second study and other techniques are not able to provide the most optimal solution.

TABLE V. ORDER OF TEST CASES FOR VARIOUS TEST CASE PRIORITIZATION APPROACHES FOR SECOND STUDY

ORIGINAL ORDER	RANDOM ORDER	REVERSE ORDER	OPTIMAL ORDER	HYBRID GA_PSO
T1	T8	T15	T4	T9
T2	T1	T14	T8	T12
T3	T11	T13	T3	T7
T4	T2	T12	T11	T8
T5	T15	T11	T12	T3
T6	T9	T10	T9	T10
T7	T3	T9	T6	T4
T8	T13	T8	T13	T11
T9	T4	T7	T7	T13
T10	T14	T6	T15	T14
T11	T5	T5	T2	T2
T12	T6	T4	T14	T1
T13	T7	T3	T1	T6
T14	T10	T2	T10	T15
T15	T12	T1	T5	T5

TABLE VI. COMPARISION BETWEEN HYBRID GA_PSO AND OTHER PRIORITIZATION TECHNIQUES FOR SECOND STUDY

TEST CASE PRIORITIZATION TECHNIQUES	APFD (100%)
ORIGINAL ORDER/NON PRIORITIZED TEST CASE ORDER	50.45%
REVERSE ORDER	46.78%
RANDOM ORDER	61.25%
OPTIMUM ORDER	89.86%
HYBRID GA_PSO	90.85%

V. CONCLUSION AND FUTURE SCOPE

In this research a hybrid GA_PSO algorithm is proposed which works on the concept of both genetic algorithm and particle swarm optimization. It is performed in two phases initially genetic algorithm [8] generates initial population randomly and then genetic operators are applied on the initial population and the fitness function used is APFD which measures fault detected per test suite. As mutation is used in genetic algorithm so it prevents the algorithm from getting trapped in local optimal solution and provides the optimal population after generations. Now the second phase is the particle swarm optimization in which the initial population is the output of genetic algorithm code. In PSO there is no mutation and crossover. Here velocity is calculated to update the population and only global best value is shared with others not the whole population. As in PSO gbest is the best test suite among whole population, so the velocity of test suites after every iteration tries to move the other test suites in the population towards the gbest. And at the end of both the phases most optimal test suite is achieved. The technique is described in details with all the steps and detailed flow diagram.

For future work this algorithm can be extended by taking multiple criteria for fitness function calculation which helps in deciding the best test suite. Here APFD is used in future APFD, code coverage and execution time can be used for fitness function calculation which will give more effective solution as it will be based on more than one criteria. In genetic algorithm phase, genetic operators can be changes for better results but here in these studies the set of operators gives optimal result. Different set of operators can be used for different experiments and framework.

REFERENCES

[1] Ahmed, A. Abdelbaky, Mahboob Shaheen, and Essam Kosba. "Software Testing Suite Prioritization using Multi-Criteria Fitness Function." 22nd International Conference on Computer Theory and Applications (ICCTA). IEEE, pp. 160-166, (2012).

[2] Huang, Yu-Chi, Chin-Yu Huang, Jun-Ru Chang, and Tsan-Yuan Chen. "Design and Analysis of Cost-Cognizant Test Case Prioritization using Genetic Algorithm with Test History." 34th Annual on Computer Software and Applications Conference (COMPSAC). IEEE, pp. 413-418. (2010).

[3] Cingiz, M. Ozgur, Sefik Temei, and Oya Kahpsiz. "Test Case Prioritization with Improved Genetic Algorithm." 22nd International conference on Signal Processing and Communications Applications Conference (SIU). IEEE, pp. 1223-1226, (2014).

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

[5] Mohapatra, Sudhir Kumar, and Santasriya Prasad. "Evolutionary Search Algorithms for Test Case Prioritization." 2013 International Conference on Machine Intelligence and Research Advancement (ICMIRA). IEEE, pp. 115-119, (2013).

[6] Noguchi, Tadahiro, Hironori Washizaki, Yoshiaki Fukazawa, Atsutoshi Sato, and Kenichiro Ota. "History-Based Test Case Prioritization for Black Box Testing using Ant Colony Optimization." 8th International Conference on Software Testing, Verification and Validation (ICST). IEEE, pp. 1-2, (2015).

[7] Mahajan, Surendra, Shashank D. Joshi, and V. Khanaa. "Component-Based Software System Test Case Prioritization with Genetic Algorithm Decoding Technique using Java Platform." International Conference on Computing Communication Control and Automation (ICCUBEA). IEEE, pp. 847-851, (2015).

[8] Konsaard, Patipat, and Lachana Ramingwong. "Total Coverage based Regression Test Case Prioritization using Genetic Algorithm." 12th International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology (ECTI-CON). IEEE, pp. 1-6, (2015).

[9] Gao, Dongdong, Xiangying Guo, and Lei Zhao. "Test Case Prioritization for Regression Testing based on Ant Colony Optimization." 6th IEEE International Conference on Software Engineering and Service Science (ICSESS). IEEE, pp. 275-279, (2015).

[10] Solanki, Kamna, Yudhvir Singh, and Sandeep Dalal. "Test case Prioritization: An Approach based on Modified Ant Colony Optimization (m-ACO)." International Conference on Computer, Communication and Control (IC4). IEEE, pp. 1-6, (2015).

[11] Konsaard, Patipat, and Lachana Ramingwong. "Using Artificial Bee Colony for Code Coverage Based Test Suite Prioritization." 2015 2nd International Conference on Information Science and Security (ICISS). IEEE, pp. 1-4, (2015).

[12] Mohemmed, Ammar W., Nirod Chandra Sahoo, and Tan Kim Geok. "Solving Shortest Path Problem using Particle Swarm Optimization." Applied Soft Computing 8, no. 4, pp. 1643-1653, (2008).

[13] Ahmed, Bestoun S. "Test Case Minimization Approach using Fault Detection and Combinatorial Optimization Techniques for Configuration-Aware Structural Testing." Engineering Science and Technology, an International Journal(2015).

[14] Diaz-Gomez, P.A. and Hougen, D.F. “Three Interconnected Parameters for Genetic Algorithms.” In Proceedings of the 11th Annual conference on Genetic and Evolutionary Computation. ACM, pp. 763-770, (2009).

[15] Raju, S., and G. V. Uma. "Factors Oriented Test Case Prioritization Technique in Regression Testing using Genetic Algorithm." European Journal of Scientific Research 74, no. 3, pp. 389-402, (2012)

[16] Hooda, Itti, and Rajender Chhillar. "A Review: Study of Test Case Generation Techniques." International Journal of Computer Applications 107, no. 16 , (2014).

[17] Howe, Adele E., Anneliese Von Mayrhauser, and Richard T. Mraz. "Test Case Generation as an AI Planning Problem." Automated Software Engineering, no. 1, pp. 77-106, (1997).

[18] Chen, Lizhe, and Qiang Li. "Automated Test Case Generation from Use Case: A Model Based Approach." 3rd IEEE International Conference on Computer Science and Information Technology (ICCSIT).vol. 1,IEEE, pp. 372-377, (2010).

[19] Brucker, Achim D., Matthias P. Krieger, Delphine Longuet, and Burkhart Wolff. "A Specification-Based Test Case Generation Method for UML/OCL." In Models in Software Engineering, pp. 334-348. Springer Berlin Heidelberg, (2010).

[20] Yuan, Xun, and Atif M. Memon. "Generating Event Sequence-Based Test Cases using GUI Runtime State Feedback." , IEEE Transactions on Software Engineering, 36, no. 1,IEEE, pp. 81-95, (2010).