Automatic Test Path Generation from Sequence Diagram Using Genetic Algorithm

Bahare Hoseini
Faculty of Electrical and Computer Engineering Department
Tarbiat Modares University
Tehran, Iran
B.hoseini@modares.ac.ir

Saeed Jalili
Faculty of Electrical and Computer Engineering Department
Tarbiat Modares University
Tehran, Iran
Sjalili@modares.ac.ir

***Abstract*— Software testing is an important and complicated phase of software development cycle. Software test process acquires test cases as input for the system under test to evaluate the behavior of the product. If test cases are prepared before coding, it will help the developers to control their code to conform to specification. White box testing requires a set of predefined test paths to generate test cases, therefore generating a set of reliable test paths is a critical task. The most common approach in white box testing is to generate test paths from source code while the generation process must be delayed until completion of source code. Using sequence diagram as an input artifact for generating test path is cost and time efficient due to the fact that test process starts before implementation phase. Furthermore, tester involvement in source code complexity is reduced to a minimum. Test paths are generated from the control flow graph, which is extracted from sequence diagrams. Among all graph based coverage criteria, Prime path coverage subsumes different graph based coverage criteria that lead us to complete path coverage. Also, prime path coverage concentrates on visiting all nodes and edges in the control flow graph rather than traversing all existing paths, which results in test effort reduction. Genetic algorithm is applied minimize the number of test cases required to reach the highest coverage.**

In this paper, we proposed a model to generate all prime paths automatically and extract minimum paths with shortest possible length, which covers all prime paths by means of genetic algorithm. The experimental results show the generated paths can easily turn into optimal test paths with the best prime path coverage having the least number of test paths.

Keywords*— genetic algorithm; test path generation; sequence diagram; prime path coverage

I. INTRODUCTION

Every software product needs to be tested to reassure it achieves all of its goals and detect any unexpected behavior. One of the most important features that should be tested is the accuracy of sequential steps, which are determined by analysis and design level, so it is essential to have test paths to map test cases on them. Every test path is a path p, possibly of length zero that starts at some node N_0 and ends at some node N_f [2]. Test paths are extracted from a graph that is constructed based on software specifications. Generating test paths from Unified Modeling Language (UML) diagrams help developers to reduce their efforts for learning new specifications so much because it is a mainstream technique in software development [1]. On the other hand, UML diagrams that are made in the design phase are useful sources for white box test case generation [3-7]. However source code is the most common artifact for test case generation in white box approach. To do so, all conditions and loops in the code are identified and then a directed graph is made to show entire possible paths of the code. Finally, a set of tests is generated that each one traverses a specified path. The purpose is to assure none of the paths in the code is uncovered. However, test paths will be dependent on source code. By applying a strategy to avoid conflicts with code complexity and details, test path generation will be exceeded up.

In order to have a test set with the least size and high-quality, an efficient method is needed. Also to quantify the amount of testing of the software, coverage-based testing can be used [8]. Among graph based coverage criteria, Prime Path Coverage (PPC) is the strongest one that can be practically applied in testing [9]. According to Fig. 1, all graph coverage criteria are subsumed into prime path coverage, so by satisfying it, it is possible to reach near complete graph coverage. One of important applications of genetic algorithm is to reduce the number of test cases required to reach the best coverage criteria. Genetic algorithms have been successfully used in many software testing activities such as test data generation, test case generation, test case selection and test case prioritization [10]. In recent years, test case generation has turned into a path search problem [11]. This paper presents a model to produce control flow graph corresponding to a sequence diagram. Then generates test paths from that graph. To reduce test effort genetic algorithm is used in our model to generate test path with the least length and is aimed to cover the entire prime paths.

The rest of the paper is organized as follows: Section 2 discusses and compares related works in test case generation, test path production and applying genetic algorithm in the test process. Section 3 introduces the main concepts and definitions. Section 4 presents our proposed model in test path generation from sequence diagram on the basis of genetic algorithm. Section 5 shows our experimental results through five tables. The last table represents final population which gets complete prime path coverage, and finally section 6 concludes this paper.

978-1-4799-5359-2/14/$31.00 ©2014 IEEE

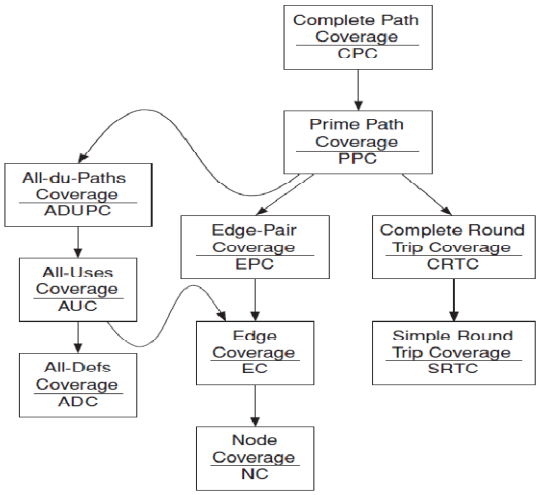

Fig. 1. Subsumption relations among graph coverage criteria

II. Related Work

There are two approaches in software testing: (a) Black box testing, which tests the results of the given inputs without knowing the internal structure of the software, and (b) White box testing that is aimed to test the internal structure of the software under consideration completely. Both approaches are employed in an automatic test case generation [12]. Meta-heuristic techniques are used both to get a complete coverage over test results and to have an appropriate size of the test set [13]. These techniques include Ant Colony Optimization algorithm [14-17], Genetic Algorithm [1], and other search-based test data generation methods [18]. But a few researches are focused on test path generation. An extended firefly algorithm is applied to produce critical test paths based on a state transition diagram and control flow graph to achieve high path coverage [19]. As a strategy, an intermediate representation of the sequence diagram is built that is called Sequence Dependency graph (SDG). This algorithm complies with UML 2.0. Then the message sequences are incorporated into the SDG. Test sequences are generated from SDG by a traversal algorithm [4]. Recently, genetic algorithm has been used in test path generation in white box testing approach [10]. It focuses on generating a set of independent basis paths from a directed graph. Every basis path should satisfy three conditions as below:

- *Every Path Should be an Independent Path.*
- *All Edges in a Control Flow Graph Should be Covered by All Paths in The Basis Set.*
- *Every Path Not Contained in This Basis Set of Paths Can be Constructed by Linear Operations Among Paths in This Set.*

To cover these three conditions, genetic algorithm is used. Every chromosome is mapped into a path in the control flow graph. The fitness value of a chromosome relies on the number of the adjacent edges in the directed graph of the software under test. It has redefined genetic algorithm concepts which lead to considering with variable length chromosomes that is applied to the program.

III. Basic Concepts

A. Genetic Algorithm

Genetic Algorithm (GA) has combined optimization techniques with search-based methods. It starts with analyzing initial population and optimizes every generated population by defining an appropriate fitness function and finally, GA offers possible solutions to a problem by representing chromosomes in the population because the last population contains best solution. The pseudo code of a basic GA is as follows [20]:

```
Initialize (population)
Evaluate (population)
While (stopping condition, not satisfied)
{
    Selection (population)
    Crossover (population)
    Mutate (population)
    Evaluate (population)
}
```

Evaluation method calculates the fitness value of each chromosome. According to an expression, each chromosome in a population gets a fitness value.

Selection method determines which chromosomes in current population are selected as the parents of offspring in the next population on the basis of their fitness value.

Crossover and mutation are used to generate offspring from parents to fill nest population. One-point and two-point crossover are more common techniques among others. Both of them require two parent chromosomes to generate at least two novel chromosomes as their offspring. In mutation process, the value of a gene in a chromosome changes to build new chromosome.

B. Prime Path Coverage

Path is considered as a valid sequence of nodes in a graph. A path from node *i* to node *j* in a graph is simple if no node appears more than once on the path, with the exception that the first and last nodes may be identical. A path from node *i* to node *j* is called prime if it is simple and it does not appear as a proper sub-path of any other simple path. If a set of paths contains all prime paths, it will reach complete prime path coverage [2].

For example, consider the graph in Fig 2. The valid node sequences {1, 2, 4, 2, 3} and {2, 3} cannot be prime paths because node {2} is repeated in the middle of the first sequence and the second sequence is the sub path of {1, 2, 3}. On the other hand {2, 4, 2} and {1, 2, 3} are both prime paths.

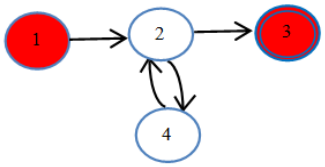

Fig. 2. Sample directed graph

IV. Test Path Generation by GA

For projects in large scale, sequence diagrams are large and complex and their corresponding graphs include too many

edges and nodes. Also, most of them have many loops, so extracting all possible paths and assigning them to test sequences needs too much time and effort. By defining prime path coverage it is not necessary to test all paths in sequence diagram. By applying genetic algorithm, we have presented a model that has three steps. 1) It makes the control flow graph from a given sequence diagram, 2) It extracts prime paths from the control flow graph, and 3) It Generates an optimal set of test paths that its size is smaller than the number of prime paths, with a high prime path coverage property. Our proposed model is shown in fig. 3:

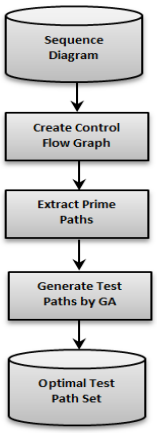

Fig. 3. Our proposed model

A. *Creation of control flow graph from sequence diagram*

To create a control flow graph from a given sequence diagram, first a set of operational scenarios must be identified. An operation scenario is a sequence of methods that are called. It starts from an actor invocation, i.e., an initial state in first lifeline and continues until getting to a final state, where it terminates. Each node shows a method name, caller and callee object [20]. The representation of an Operation Scenario (OS) is as follows:

OS_i = *{(method1(), caller 1, callee 1), ...}*

The control flow graph from sequence user authentication sequence diagram of Fig. 4, is shown in Fig. 5 and its nodes created based on a sequence diagram of Fig. 4 are:

Node 1: GetCardInfo (), a, b
Node 2: SetConFig(), b, c
Node 3: SaveLoginAttempt(), c, c
Node 4: RequestPassword(), c, d
Node 5: CheckCharacterValidity (), d, e
Node 6: EnableTransaction (), c, f

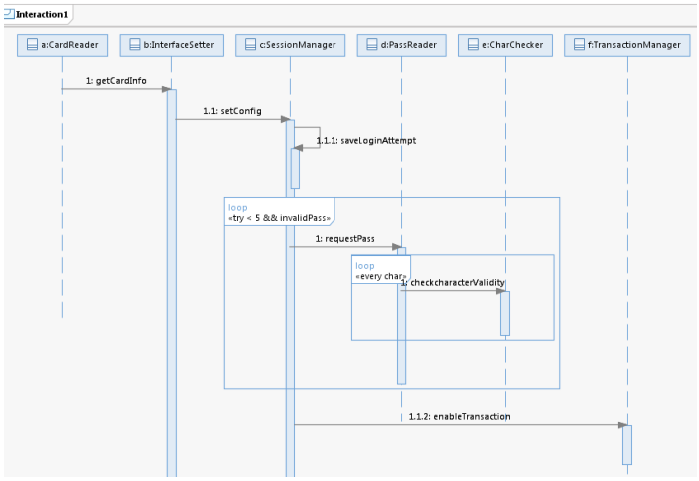

Fig. 4. Sequence diagram of user authentication in an ATM system

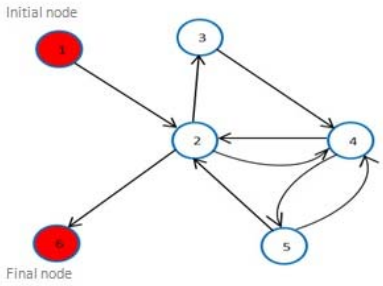

Fig. 5. Control flow graph based on the corresponding sequence diagram

According to UML 2.X, sequence diagram supports nested sequence, condition and loop, therefore, result graph contains loops and multiple branches per node.

B. *Prime Path Extraction*

The second step in our proposed model finds all prime paths of the included graph (Fig. 5). Our algorithm to find all prime paths is as follows:

1. import adjacent matrix of graph
2. Calculate all existing paths
3. N = 1 //initial path size
4. nodeCount = number of graph nodes
5. primePathList = empty
6. pathList1, pathList2 = empty //both lists will contain all available paths with length of N and N+1 sequentially
7. **while**(N<nodeCount){
8. import all paths with length N to pathList1
9. import all paths with length N+1 to pathlist2
10. **if** ((any member in pathList1 does not exists in pathList2) && (mentioned member follows both conditions of a prime path))
11. add the path to primePathList
12. N++}

Fig. 6. Prime Path Extraction Algorithm

In every iteration that refers to line 7 of Fig. 6, two valid path list (pathList1,pathList2) are used that are called pathList1 and pathList2. They contain all valid graph paths with length N and N+1, consecutively. There are finite prime paths in a graph and maximum size of a prime path is equal or smaller than the number of graph nodes, so parameter N is

definitely smaller than the number of nodes in the graph. As a result, our proposed algorithm never faces to an endless loop. According to the graph shown in Fig. 5, there is no path with length of 1 which is not subpath of any path with 3 nodes, so there is not any prime path with 2 nodes. Executing the proposed algorithm results in 21 Prime Paths (PP) that are represented in Table I:

TABLE I. ALL PRIME PATHS CORRESPONDING TO CONTROL FLOW GRAPH

Prime Paths		
3 nodes	***4 nodes***	***5 nodes***
{1,2,6}	{1,2,4,5}	{1,2,3,4,5}
{2,4,2}	{2,3,4,2}	{2,3,4,5,2}
{4,2,4}	{2,4,5,2}	{3,4,5,2,3}
{5,4,5}	{3,4,2,3}	{3,4,5,2,6}
{4,5,4}	{3,4,2,6}	{4,5,2,3,4}
	{4,2,3,4}	{5,2,3,4,5}
	{4,5,2,4}	
	{5,2,4,5}	
	{5,4,2,3}	
	{5,4,2,6}	

C. *Test Path Generation Using GA*

The mid step of the proposed model generates all test paths to reach complete prime path coverage with the possible smallest size that contain the maximum number of prime paths. To satisfy this purpose, GA is applied. Each chromosome represents a path in the graph and is supposed to be a test path. Each gene has a value between zero and the maximum node number. As a matter of fact, zero doesn't refer to any node. When a gene is set to zero, it means that the gene is not counted in node sequence. Every zero value in chromosome decreases one unit from test path size. This technique helps up to have test paths with variable size, so it is possible to consider test path length as a parameter in the fitness function. Chromosome max length depends on the longest path in a graph that transits any possible loop once. This statement prevents test paths from having extra nodes. Population size has a direct correlation with the number of prime paths in the graph. A set of chromosomes in the last population is considered as a solution to reach complete prime path coverage. The algorithm continues until getting a specified percentage of the coverage. If it does not reach the desired coverage, it will terminate after 100th generation. Due to our experimental results on different sequence diagrams, the proposed algorithm reaches the complete coverage after at most one hundred generations.

To build a chromosome in the initial population, one of the nodes is selected randomly. Then a valid path with random length is found in the graph that is started with the selected node.

Evaluating test paths depends on their fitness value. Test paths with higher fitness values are selected for the next generation. The fitness value of a chromosome is calculated based on the number of prime paths it contains and its corresponding path length. Suggested algorithm to calculate fitness value for each test path is as follows:

```
1. Fitness = -length(testPath)
   //the shorter test path, the better
2. for each primePath in list of all Primepaths{
3.     if testPath contains primePath
4.         Fitness += 10 + length(primePath)
               //10 points are assigned to fitness per prime path contained
5. }
               //looking for prime paths that are subsets of the test path
```

Fig. 7. An algorithm for fitness value calculation

Two different crossover techniques (one-point and two-point) are used. According to experiments, one point crossover results in a better generation. Mutation operation is applied on chromosomes after crossover operation. Every time mutation is performed, the value of a random gene is replaced with another node in the graph. In the proposed model a set of chromosomes leads to a complete coverage because each chromosomes just a path.

V. EXPERIMENTAL RESULTS

As it is presented in Tables II, III, IV, V and VI, after executing our proposed model, all 21 prime paths in the control flow graph in Fig. 2 were covered within reached 100% prime path coverage in 40th generation from a control flow graph in Fig. 2. The generated test paths are:

TABLE II. PATHS OF THE INITIAL POPULATION, ACCOMPANIED BY PRIME PATH

Generation#1		
	Path	Fitness
#1	{4, 5, 2, 4, 5, 2, 0, 0, 0, 0, 0}	36.0
#2	{1, 2, 3, 4, 5, 2, 3, 4, 5, 0, 0}	66.0
#3	{6, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}	-1.0
#4	{1, 2, 4, 5, 2, 3, 4, 5, 0, 0 , 0}	50.0
#5	{2, 3, 4, 5, 4, 5, 2, 3, 0, 0, 0}	18.0
#6	{1, 2, 6, 0, 0, 0, 0, 0, 0, 0, 0}	7
Total Coverage	23.08%	

Total coverage is obtained by dividing the number of covered prime path in the population, on all existing prime paths.

TABLE III. PATHS OF 10TH POPULATION, ACCOMPANIED BY PRIME PATH COVERAGE

Generation#10		
Path		Fitness
#1	{1, 2, 3, 4, 5, 2, 3, 4, 2, 3, 4}	91.0
#2	{2, 4, 5, 2, 3, 4, 5, 2, 6, 0, 0}	65.0
#3	{2, 3, 4, 5, 4, 5, 0, 0, 0, 0, 0}	20.0
#4	{3, 4, 5, 2, 3, 4, 5, 2, 6, 0, 0}	66.0
#5	{3, 4, 5, 2, 3, 4, 5, 2, 6, 0, 0}	66.0
#6	{1, 2, 6, 0, 0, 0, 0, 0, 0, 0, 0}	7
Total Coverage	57.14%	

TABLE IV. PATHS OF 20TH POPULATION, ACCOMPANIED BY PRIME PATH COVERAGE

Generation#20		
Path		Fitness
#1	{1, 2, 3, 4, 5, 2, 3, 4, 2, 3, 4}	91.0
#2	{1, 2, 4, 5, 2, 4, 5, 4, 2, 4, 2}	84.0
#3	{4, 4, 5, 4, 5, 2, 3, 4, 5, 2, 6}	75.0
#4	{1, 2, 4, 5, 4, 2, 3, 4, 2, 6, 0}	72.0
#5	{4, 5, 2, 4, 5, 2, 3, 4, 2, 6, 0}	75.0
#6	{1, 2, 6, 0, 0, 0, 0, 0, 0, 0, 0}	7
Total Coverage	90.47%	

TABLE V. PATHS OF 30TH POPULATION, ACCOMPANIED BY PRIME PATH COVERAGE

Generation#30		
Path		Fitness
#1	{3, 4, 5, 2, 3, 4, 5, 2, 4, 5, 2}	91.0
#2	{2, 4, 2, 4, 5, 4, 2, 3, 4, 2, 6}	84.0
#3	{5, 4, 5, 3, 4, 2, 3, 4, 5, 2, 6}	60.0
#4	{1, 2, 4, 5, 2, 3, 4, 5, 4, 2, 6}	74.0
#5	{5, 2, 3, 4, 5, 2, 3, 4, 1, 2, 6}	62.0
#6	{1, 2, 6, 0, 0, 0, 0, 0, 0, 0, 0}	7
Total Coverage	95.23%	

As it is shown in Table VI, some chromosomes (paths) cannot be considered as test path, so by adding an initial node or final node to such paths out test path set is complete. The test paths obtained from last population are presented in Table VII.

TABLE VI. PATHS OF THE LAST POPULATION, ACCOMPANIED BY PRIME PATH COVERAGE

Generation#40		
Path		Fitness
#1	{1, 2, 3, 4, 5, 2, 3, 4, 2, 3, 4}	91.0
#2	{1, 2, 4, 5, 2, 4, 5, 4, 2, 4, 2}	84.0
#3	{2, 4, 5, 4, 5, 2, 3, 4, 5, 2, 6}	75.0
#4	{1, 2, 6, 4, 5, 4, 2, 3, 4, 2, 6}	71.0
#5	{1, 2, 4, 5, 2, 4, 5, 4, 2, 6, 0}	72.0
#6	{1, 2, 6, 0, 0, 0, 0, 0, 0, 0, 0}	7
Total Coverage	100%	

TABLE VII. TEST PATHS OBTAINED FROM THE LAST POPULATION

Test Paths	
#1	{1, 2, 3, 4, 5, 2, 3, 4, 2, 3, 4, 2, 6}
#2	{1, 2, 4, 5, 2, 4, 5, 4, 2, 4, 2, 6}
#3	{1, 2, 4, 5, 4, 5, 2, 3, 4, 5, 2, 6}
#4	{1, 2, 6, 4, 5, 4, 2, 3, 4, 2, 6}
#5	{1, 2, 4, 5, 2, 4, 5, 4, 2, 6}
#6	{1, 2, 6, 0, 0, 0, 0, 0, 0, 0, 0}
Total Coverage	100%

VI. CONCLUSION

In this paper, we have proposed a model to generate test paths from UML sequence diagram using genetic algorithm. Our model achieves prime path coverage, which is the strongest graph based coverage criteria. Using genetic algorithm has an impressive effect on reduction of test cases required for software testing because it results in minimum test paths. Due to our experiments union of test paths in some cases can get to even 100 percent coverage and length of each path is as short as possible. Unlike other attempts in white box testing, our model creates test paths from sequence diagram rather than source code to prevent facing code details and speed up test path generation process.

REFERENCES

[1] C. Doungsa-ard, K. Dahal, A. Hossain, and T. Suwannasart, "GA-based automatic test data generation for UML state diagrams with parallel paths," In Advanced Design and Manufacture to Gain a Competitive Edge, pp. 147-156, Springer London, 2008.

[2] P. Ammann and J. Offutt, "introduction to software testing," Cambridge University Press, 2008.

[3] S. Monalisa, D. Kundu, and R. Mall, "Automatic test case generation from UML sequence diagram," Advanced Computing and Communications, International Conference on IEEE, pp. 60-67, 2007.

[4] P. Samuel and A. Teresa Joseph, "Test sequence generation from UML sequence diagrams," In Software Engineering, Artificial Intelligence,

Networking, and Parallel/Distributed Computing, 2008. SNPD'08. Ninth ACIS International Conference on, pp. 879-887. IEEE, 2008.

[5] S.Kumar Swain, D. Prasad Mohapatra, and R. Mall. "Test case generation based on use case and sequence diagram," International Journal of Software Engineering, IJSE, pp. 21-52, 2010.

[6] M. Khandai, "A novel approach of test case generation for concurrent systems usin1g UML Sequence Diagram," Electronics Computer Technology (ICECT), 2011 3rd International Conference, vol.1, pp.157 – 161, 2011.

[7] V. Panthi and D. Prasad Mohapatra, "Automatic Test Case Generation Using Sequence Diagram," In Proceedings of International Conference on Advances in Computing, pp. 277-284, Springer India, 2012.

[8] P. Ranjan Srivatsava, B. Mallikarjun, and Xin-She Yang, "Optimal test sequence generation using firefly algorithm," Swarm and Evolutionary Computation 8 , pp. 44-53, 2013.

[9] A. H. Bushmais, "Graph based unit testing," The University of Texas, Austin, 2011.

[10] A. S. Ghiduk, "Automatic generation of basis test paths using variable length genetic algorithm," Information Processing Letters ,pp. 304-316, 2014.

[11] Y. Yao, "New test case generation method based on genetic algorithm," Computer & Digital Engineering 231(1), pp. 18–21, 2009.

[12] A. Saswat, et al, "An orchestrated survey of methodologies for automated software test case generation," Journal of Systems and Software 86.8, 1978-2001, pp. 2013.

[13] P. McMinn, "Search-based software test data generation: a survey," Software testing, Verification and reliability 14.2, pp. 105-156, 2004.

[14] T. Stutzle, M. Dorigo, "The ant colony optimization metaheuristic: algorithms, applications, and advances," International Series in Operations Research and Management Science, vol. 57, Springer, New York, pp. 250-285, 2003.

[15] L. Hjuaizhong and C. Peng Lam, "An ant colony optimization approach to test sequence generation for state based software testing," In: Proceedings of the Fifth International Conference (IEEE) on Quality Software, QSIC, pp. 255264, 2005.

[16] P. Ranjan Srivastava, K. Bby, and G. Raghurama, "An approach of optimal path generation using ant colony optimization," in: Proceedings of the TENCON – IEEE Region 10 Conference, Singapore, pp. 1-6, 2009.

[17] P. Ranjan Srivastava and K. Baby, "Automated software testing using metahurestic technique based on an ant colony optimization," in Proceedings of the International Symposium on Electronic System Design, ISED, Bhuba- neswar, pp. 235–240, 2010.

[18] Y. Chen and Y.Zhong, "Automatic path-oriented test data generation using a multi-population genetic algorithm," in Proceedings of the Fourth International Conference (IEEE) on Natural Computation, ICNC, pp. 566–570, 2008.

[19] P. Ranjan Srivastava, V. Ramachandran, M. KumarGourab Talukder, V. Tiwari, and P. Sharma, "Generation of test data using meta heuristics approach", in Proceedings of the TENCON 2008 - IEEE Region 10 Conference, India, pp. 1–6, 2008.

[20] S. Sabharwal, R. Sibal, and C. Sharm, "Applying Genetic Algorithm for Prioritization of Test Case Scenarios Derived from UML Diagrams," International Journal of Computer Science Issues (IJCSI) 8.3, 2011.