JCSE International Journal of Computer Sciences and Engineering Open Access
Research Paper Vol.-7, Issue-6, June 2019 E-ISSN: 2347-2693

Generating Optimized Test Case from UML Diagram Using Meta-Heuristic Algorithm

Preeti[1*], Rohit Goyal[2]

[1,2] Department of Computer Science and Engineering, Himgiri Zee University, Dehradun, INDIA

[*]Corresponding Author: 10preetishivach@gmail.com

DOI: https://doi.org/10.26438/ijcse/v7i6.11771183 | Available online at: www.ijcseonline.org

Accepted: 24/Jun/2019, Published: 30/Jun/2019

***Abstract*—** Properly tested software is better in quality then the software tested using a poor approach or not tested. Increasing size and complexity of software makes manual testing process a time, cost and resource consuming task. Automating the testing process can improve software development process. The unified modeling language (UML) is the most widely used language to describe the analysis and designs of object-oriented software. Test cases can be derived from UML models more efficiently. In our work, we propose a novel approach for automatic test case generation from the combination of UML state chart, sequence and activity diagrams. In our approach, we first draw the UML state chart, sequence and activity diagrams. Then convert these diagrams to graphs and generate a combined graph. This graph is then used to generate test paths. We have integrated meta-heuristic algorithm i.e. Genetic Algorithm (GA) for this purpose and found fruitful results.

***Keywords*—**UMLDiagram, Sequence Diagram, Activity Diagram, Test case generation, Genetic Algorithm.

I. INTRODUCTION

Quality software can be developed when it is properly tested. Due to increase in the size and complexity of object-oriented software, manual testing has become time, resource and cost consuming. Properly designed test cases discover more errors and bugs present in the software. The test cases can be generated much early in the software development process, during the design phase. Software testing is a superiority phase of development of software which main aim is not to productivity only but also support to enhance the quality of software product from small scale to large scale. In fact we test the software until the product is valid and verifiable. As increasing the software complexity, the requirement of test coverage need for generated test case increases gradually **[1]**. In Software Development Life Cycle (SDLC), test process is the most important phase to check the Software System validation. It is mainly completed by running test and inspection of these processes. The whole Test process complete in three processes:

1. Generation of Test Cases.
2. Execution of Test cases.
3. Evaluation of Test cases.

The main aim of testing should be conveying advice to change and modified if necessity, by add on some value to entire test process. The main aim of design the test case is to rectify the different classes of error within less amount of time and effort. Software reliability and quality are mainly depends on the collection of data during testing. The UML is the most widely used language to describe the analysis and designs of object-oriented software. Test cases can be derived from UML models more efficiently. In our work, we propose a novel approach for automatic test case generation from the combination of UML sequence and activity diagrams. In our approach, we first draw the UML sequence and activity diagrams. Then convert these diagrams to graphs and generate a combined graph. This graph is then used to generate test paths. We have integrated meta-heuristic algorithm i.e. GA for this purpose and found fruitful results.

A. UML Diagram and Model-Based Testing

Unified Modeling Language was announced by the Object Management Group in1997. Object oriented prototypes are very rapidly used in industries and academics. UML is the most superior and controlling modeling language used in development of software. UML diagram are classified into twelve diagrams on the basis of two category level [2]:

A) Structural Level(class diagram, object diagram, package diagram, composite diagram, profile diagram)

B) Behavioral Level (use case diagram, state machine diagram, activity diagram, sequence diagram, communication diagram, interaction overview diagram, timing diagram).

Structural diagrammatic signifies the inter-relation between different component of system on different hierarchy of abstraction while the role of object including physical changes in object and shows the interaction within each-other comes in Behavioral diagram.

Model based testing (MBT) is also known as an object oriented testing techniques where generation of test cases is based on the models using UML diagram. MBT is becoming famous in both academias as well as in industry. Because of the increasing in complexity, many critical functions are performed and dependability requirement such as safety, reliability, availability, and security of the system are very crucial to the user of the system. On the basis of requirement specification the information are preserved by the model and it forms a basis of final implementation. Model determines logical paths, location of program boundaries, identify reachable problem. For the analysis of UML models it becomes a challenge because the information about the system is distributed across the different models views. For reducing the complexities and size of the problem, UML models are required for reducing that complexity.

Evolutionary testing is used to search for optimal test parameter combination that satisfies a predefined test criterion. This criterion is represented by using a “cost function” that measures how well each of the automatically generated optimization parameters are satisfying the given test criterion [4]. It is a searching algorithm that is mainly based on the natural genetics. GA is based on Selection, Crossover and Mutation. The Fitness Function plays the main role in GA. The procedure of Selection is used for randomly selects the individual (Chromosome) from the initial population. The evaluation procedure evaluates the Fitness value of each individual and assigns a value to it. The Crossover performed by recombination of the two individual genes in Parents node. After Crossover the Mutation is performed by mutating the one bit in newly generated chromosome of Crossover. Mutation is applied on that chromosome that satisfies the mutation probability. In a basic form, Genetic Algorithm is mainly used for single objective and search optimization Algorithm.

Rest of the paper is organized as following. Section 2 presents the related work that has already done on generation of test cases from different UML diagrams using different testing techniques. Section 3 presents proposed work. Section 4 represents the case study of Railway Reservation System in which GA is implemented. Section 5 represents the conclusion and future work.

II. RELATED WORK

This section represents the various research papers that are surveyed and related to the generation of test cases techniques using various UML diagram.

Yoo-Min Choi, Dong-Jin Lim [1] proposed a method that holds the dependent transition pair for generation of transition path. The presented method describes that it protect the specific part of transition against randomly operation of genetic is performed by Grouping Genetic Algorithm (GGA). Experiment is performed on State chart diagram and an ATM of State Chart that holds all dependent transition and counter problem. The presented GGA defines that the generation of full-transition coverage FTP use state chart with all dependent transition pair with a 100% success rate.

Namita Khurana, R.S Chhillar [3] presented a technique for the generation of test cases using State Chart Diagram (SCD) and Sequence Diagram (SD). The presented techniques are used to convert the SCD to SCDG (State Chart Graph) and then SD to SDG (Sequence Graph). Finally, the integration of SCDG and SDG are completed to design a System Graph for testing. This System Testing Graph (SYTG) pre-stored the crucial information is required for the generation of test cases. Based on a Coverage criteria and a fault model, GA is applied on this SYTG to generate optimized test cases automatically. Hooda and Chhillar [5] applied GA and neural network for test case generation.

Abdelkamel Hettab et al. [6] proposed the techniques that are graph transformation and use the ATOM3 tool for the generation of test cases. Due to this purpose, they represent the two graph grammars. Firstly, intermediate diagram is extracted from the Activity Diagram that is known as EADG (Extended Activity Dependency Graph) where all the activities are captured that is important to generate test cases. At second stage as per the Hybrid coverage criteria the generation of test cases are done from the EADG model. Loop Structure and Concurrency problem is also solved in Activity Diagram.

Fernando Augusto et al.[7] proposed a method named EasyTest to generate the test cases using Activity diagram of UML ,the main focus is to integrate Modeling. And show the two different stages of test before and after coding stages. In first stage, approach helped TDD programmers to supply with the required information to generate error free code. In second stage, proposed approach helps in executing the automatic generation of test code on any of the Test Execution Tool. And for verification the scenario, EasyTest tool was used. EasyTest tool provides gain into cost, effort, and time

Meiliana et. al.[8] presented the Modified DFS algorithm to generate the test cases through Activity and Sequence

Diagram. To generate appropriate test cases, modified version of DFS algorithm is needed. Here, firstly the conversion of Activity diagram (AD) and Sequence Diagram (SD) are done with their respective graphs named as Activity Diagram Graph (ADG) and SDG then both of these Graphs are merged to generate SYTG. After the formation of SYTG, Modified DFS is applied over SYTG to generate the expected test cases. After the experimental result it is proved that Sequence diagram can produce better test case.

Rajiv Mall [9] proposed a method for the generation of test cases by integration of Use Case and Sequence diagram (SD). Both the diagrams are converted into Use Case Diagram Graph and SDG respectively. Thereafter these graphs are used to formulate the SYTG. But how this graph is integrated is not mentioned in paper and test case also not optimized.

Vikas Panthi et. al. [10] presented a method to generate the test case using Activity Diagram. They derived a Flow graph from the Activity diagram. After generation of Flow graph, a traversal technique is used to traverse the CFG (control flow graph) by DFS (Depth First Search). All Activity Paths are generated through an algorithm. Lastly, the Activity path coverage generates the output test cases.

Saso Karakatic, Tina Schweighofer [11] presented a techniques named as Genetic Programming that is used for the generation of test cases from the Activity diagram, from which binary decision tree is constructed that is taken into use as the evolution process of Genetic Programming. This research paper focuses mainly on the different fitness function. This approach is simulated on the ATM which shows the potential and usefulness of proposed method.

Santosh Kumar Swain et. al. [12] presented a method to generate test cases through sequence diagram of UML models. A model is proposed in which a flow chart is created by sequence diagram of UML model; thereafter it is being transformed into flow graph (MCFG). By the help path coverage criterion of Sequence graph, Message Control Flow Graph is used to traverse and generate the test paths. So that, the generation of test cases are completed from MCFG. At last, GA is applied to generate the optimized test cases. They proposed a framework that covert a Sequence Diagram to Sequence Diagram Graph and apply a traversal technique DFS for the generation of test cases and ensure that visiting all nodes and remove redundant ones.

Arvinder Kaur, Vidhi Vig [13] presented a method by using the collaboration diagram for the automatic generation of test cases and validates the output by the help of mutation testing. From the inspection it is initiated found that generated automatic test cases ensured proper path coverage and avoid infinite loops. Rational Rose software is used for the creation of Collaboration diagram of the system and then it is converted into graph. For the traversing of tree the sequence numbers between each node in the graph must be captured. Mutation testing are used for the generation of valid, Invalid and terminated paths for different condition. Evolutionary algorithms have been applied in many research areas. Dubey et al. [14] applied GA for encryption and decryption purpose. Shahmohammadi and Mohammadi [15] applied evolutionary algorithm for key management in sensor networks.

III. PROPOSED APPROACH

In our proposed methodology a graph is obtained from the system under test called SYTG. Based on a Coverage criteria and a fault model, GA is applied on this SYTG for the generation of optimized test cases automatically. The proposed approach contains the three phases – (1) pre-processing phase, (2) Formation of SYTG and (3) Implementation of GA on SYTG.

1. **Pre-processing Phase –** This phase convert the UML diagrams to corresponding graphs. Example of Railway Reservation system has been taken.

(a) Converting Sequence Diagram (SD) to Sequence Diagram Graph (SDG)
The SDG is represented by the following -
SDG= {State, Edge, First, Last} where,
State represents the node of the graph.
Edge shows the transaction flow between the various nodes.
First represents the starting node of the graph.
Last represents the end node of the graph.

For the formulation of a SDG, a scheme is defined with quadruple values: <Id, Start state, message, Pass/Fail>.Id is used to represent a specific number for the identification of each message. Start state represent the initial point where the scenario initiates. A set of message shows the set of such events that occurs in operation scheme. Pass/Fail State represents the state that a system acquires on the accomplishment of condition. The final stage depends totally on the user. Fig. 1 presents the SD for Railway Reservation System and Fig. 2 is the SDG is the graph for system.
(b) Converting Activity Diagram (AD) to Activity Diagram Graph (ADG)
ADG= {Start, State, Edge, Last} where
Start represents the starting node of the ADG.
State represents the all nodes of the graph.
Edge represents the actual work flow behavior from one node to another.
Last represents final node of the graph.

Activity diagram represents the activities or chunk of processing which may or may not correspond to the method of classes. An Activity inherits all the interior action and multiple out degree transition of a state that automatically lead to the termination of the internal activity. A condition is used to identify the out degree transition if an activity has multiple out degree transition. AD supports the description of parallel activities and synchronization involved in different activities. Fig 3 (a) represents the AD for Railway Reservation system and Fig. 3 (b) represents the ADG for system.

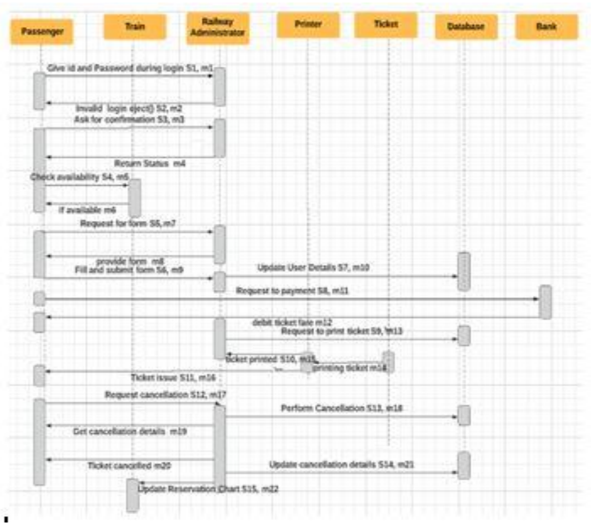
Passenger
Train
Railway Administrator
Printer
Ticket
Database
Bank
Give id and Password during login S1, m1
Invalid login eject() S2, m2
Ask for confirmation S3, m3
Return Status m4
Check availability S4, m5
If available m6
Request for form S5, m7
provide form m8
Fill and submit form S6, m9
Update User Details S7, m10
Request to payment S8, m11
debit ticket fare m12
Request to print ticket S9, m13
ticket printed S10, m15
printing ticket m14
Ticket issue S11, m16
Request cancellation S12, m17
Perform Cancellation S13, m18
Get cancellation details m19
Ticket cancelled m20
Update cancellation details S14, m21
Update Reservation Chart S15, m22

Figure 1: Sequence Diagram of Railway Reservation System

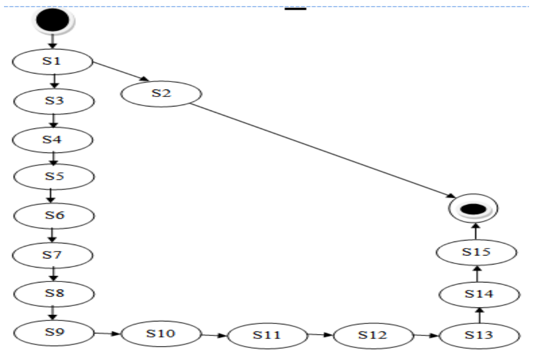

Figure 2: Sequence Diagram Graph of Railway Reservation System

2. **Formation of SYTG**

After the formation of three Diagram Graph: ADG and SDG. The next step is to integrate these UML diagram graph to form the SYTG which is used for generation of test cases. SYTG is defined as:

SYTG={S, T, F, L} where

S represents the all states presented in three graph SDG and ADG, i.e.

S= States(SDG) U States(ADG).

T represents the union of transaction flow from one state to another, i.e.

T= T(SDG) U T(ADG).

F represents the initial node of SYTG and L shows the end node of SYTG. Fig. 4 shows SYTG of Railway Reservation System.

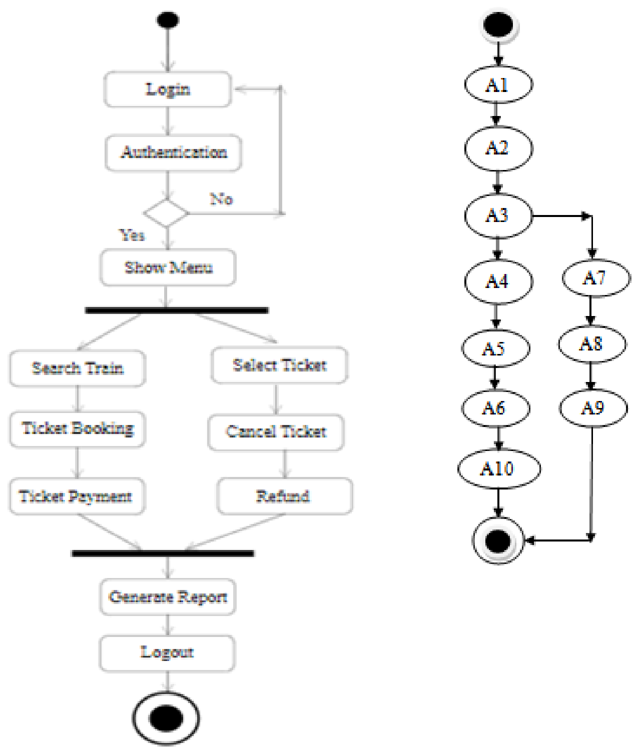

Figure 3(a) AD Railway Reservation system (b) ADG for Railway Reservation system

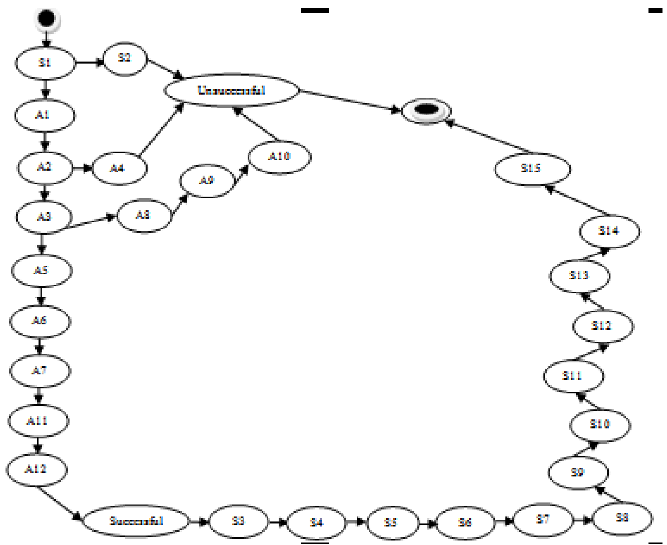

Figure 4: SYTG by integration of SDG, and ADG

Weights are assigned to all edges from Source to final node. Edge is used for showing the control flow in between two nodes. Edges are also called as transition flow. One node is dependent to another by using this edge. Dependent node will not perform the action till the node does. The integrated SYTG represents the weighted directed graph. Assigning the weight to each edge from source to terminal node because it helps to calculate the fitness value. On the basis of these weights of each path of SYTG, fitness function will be calculated. The following path shows the connection of all paths with edge from source to destination and weight also assigned to each edge.

Path1:S1=>A1=>A2=>A3=>A5=>A6=>A7=>A11=>A12=> Successful=>S3=>S4=>S5=>S6=>S7=> S8=>S9=>S10=>S11 =>S12=>S13=>S14=>S15=>Final Node.

Weight:(1=>2=>3=>4=>5=>6=>7=>8=>9=>10=>11=>12=> 13=>14=>15=>16=>17=>18=>19=>20=>21=>22=>23)

Path2: S1=>A1=>A2=>A3=>A8=>A9=>A10=>Unsuccessful=>Final Node.

Weight: (1=>2=>3=>4=>24=>25=>26=>81)

Path3: S1=>A1=>A2=>A4=>Unsuccessful=>Final Node.

Weight: (1=>2=>3=>27=>81)

Path4:S1=>S1=>Unsuccessful=>Final Node.

Weight: (1=>28=>81)

To generate the SYTG, the initial node starts with ADG starting node. And then check that if multiple states derived from the current state of ADG then add the node of SDG. Last node of ADG of system is added with SDG's starting node. Last node of SDG S12 is passed through the Successful node because path1 shows the successful steps towards the Railway Reservation system. After that, the rest of the nodes of ADG are added and then the ADG is added to the end of SDG and finally ended with Final Node. On the other hand, the invalid steps pass through the Fail node of the SYTG. The invalid steps are generated from the ADG and SDG are passed through Fail invalid node.

3. Implementation of GA to generate optimized test cases

After the collection of required information from SYTG, the next step is to generate the test cases and that test cases can be optimized using evolutionary Algorithm. GA is the most basic and popular evolutionary algorithms that implemented on the SYTG for the generation of optimized test case. Our proposed algorithm is:

Input: SYTG

Output: Optimized Test case

1. Firstly identify the paths P={p1, p2, p3.....} present in the SYTG from the source node to target node.

2. Assigning the weights to the entire individual node related to SYTG that shows the similar actual weight of the child node as well as Parent node. If in any condition, a child has multi-parents in that case the weight of that child node is calculated as the sum of the weight of parent's node. The Path's weight is allocated from the left to right in the SYTG.

3. Now calculate the each path's cost(X) that is the sum of the weight of that path originates from Start node and terminates on target node.

4. Apply GA to the SYTG.

5. Calculate the Fitness Function value

a. The cost of each path is calculated.

*b. Fitness function is applied for each path in the SYTG as F(X)=X*X.*

c. Now the probability of each individual path is calculated as p(i)=F(X)/∑F(X).

6. Now the next step is to choose the best individual from large initial population. For this we start to perform some genetic operation for the generation of solution.

a. To choose the best individual from the initial population, probability ranges is divided into bins and the size of bins is based on the relative fitness of the solution.

b. Now generate the random values that check the bin where that value will fall into then select the individuals for the next generation.

7. Again perform the crossover to the pair of chromosomes. Firstly, mating the two individual pairs by applying single point crossover from the 4th bit from right. And for next two pair single point crossover from 3rd bit from right.

8. Apply the mutation operation by mutating the every fourth bit in case of when random number is less than 0.2.

9. The whole process will be repeated till the fitness function's value is maximized or minimized the number of generation is reached or all paths have been covered.

10. Finally, best path is generated or we can say that test cases are optimized.

11. End.

IV. Case Study

The case study is done on the example of Railway Reservation System. The mathematical representation of the table is performed by applying the some organic evolution such as selection, recombination, and mutation. The sequentially assigned value is used to calculate the Weight of path. Each individual path of SYTG has weight that is used to calculate the Fitness value. Table 1-17 shows step-wise results of applying GA to the system (CP is used for cumulative probability). The number of possible paths obtained from SYTG is:-

Path1:S1=>A1=>A2=>A3=>A5=>A6=>A7=>A11=>A12=> Successful=>S3=>S4=>S5=>S6=>S7=> S8=>S9=>S10=>S11 =>S12=>S13=>S14=>S15=>Final Node. Cost=276.

Path2: S1=>A1=>A2=>A3=>A8=>A9=>A10=>Unsuccessful=>Final Node. Cost=166

Path3: S1=>A1=>A2=>A4=>Unsuccessful=>Final Node, cost=114

Path4: S1=>S2=>Unsuccessful=>Final Node. Cost=110.

Table 1: Fitness of Initial Population

Path No.	Cost(X)	X*X	Chromosome	Probability	CP	Associated Bin
1	276	76176	100010100	0.5913	0.5913	0.0-0.6
2	166	27556	010100110	0.2138	0.8051	0.6-0.8

3	114	12996	001110 010	0.1008	0.9060	0.8-0.9
4	110	12100	001101 110	0.0939	1.0	0.9-1.0

Table 2: Selection of New Generation

Random No.	Fall into bin	Selection	Crossover	Mutation
0.3834	1	100010100	100010100	100010100
0.7862	1	100010100	100010100	100010100
0.8926	2	010100110	010100010	010100010
0.0837	3	001110010	001110110	001010110

Table 3: Fitness of new Generation

Path No	Cost(X)	X*X	Chromosome	Probability	CP	Associated bin
1	276	76176	100010100	0.4095	0.4095	0.0-0.4
2	276	76176	100010100	0.4095	0.8191	0.4-0.9
3	162	26244	010100010	0.1411	0.9602	0.9-1.0
4	86	7396	001010110	0.0397	1.0	1.0-1.0

Table 4: Selection of New Generation

Random No	Fall into bin	Selection	Crossover	Mutation
0.9253	1	100010100	100010100	100010100
0.8980	2	100010100	100010100	100010100
0.6321	2	100010100	100010010	100010100
0.1966	3	010100010	010100100	010000100

Table 5: Fitness of New Generation

Path No	X	X*X	Chromosome	Probability	CP	Associated bin
1	276	76176	100010100	0.3111	0.3111	0.0-0.4
2	276	76176	100010100	0.3111	0.6222	0.4-0.7
3	274	75076	100010010	0.3066	0.9288	0.9-1.0
4	132	17424	010000100	0.0711	0.9999	1.0-1.0

Table 6: Selection of New Generation

Random_No	Fall into bin	Selection	Crossover	Mutation
0.1906	1	100010100	101000101	100110100
0.2814	1	100010100	101000101	100010100
0.1221	1	100010100	101000101	100110100
0.5591	2	100010100		100010100

Table 7: Fitness for new generation

Path_No	X	X*X	Chromosome	Probability	CP	Associated bin
1	308	94864	100110100	0.2773	0.2773	0.0-0.3
2	276	76176	100010100	0.2226	0.4999	0.3-0.5
3	308	94864	100110100	0.2773	0.7772	0.5-0.8
4	276	76176	100010100	0.2226	1.0	0.8-1.0

Table 8: Selection of New Generation

Random_No	Fall into bin	Selection	Crossover	Mutation
0.6858	1	100110100	100110100	100110100
0.4951	1	100110100	100110100	100110100
0.1451	2	100110100	100110100	100010100
0.1803	3	100010100	100010100	100110100

Table 9: Fitness for New Generation

Path_No	X	X*X	Chromosome	Probability	CP	Associated bin
1	308	94864	100110100	0.2629	0.2629	0.0-0.3
2	308	94864	100110100	0.2629	0.5258	0.3-0.6
3	276	76176	100010100	0.2111	0.7369	0.6-0.8
4	308	94864	100110100	0.2629	0.9998	0.8-1.0

Table 10: Selection for New Generation

Random_No	Fall into bin	Selection	Crossover	Mutation
0.1103	1	100110100	100110100	101000101
0.3448	1	100110100	100110100	101000101
0.2310	2	100110100	100110100	101000101
0.5617	2	100110100	100110100	100110100

Table 11: Fitness for New Generation

Path_No	X	X*X	Chromosome	Probability	CP	Associated bin
1	276	76176	100010100	0.2111	0.2111	0.0-0.3
2	308	94864	100110100	0.2629	0.4740	0.3-0.5
3	308	94864	100110100	0.2629	0.7369	0.5-0.8
4	308	94864	100110100	0.2629	0.9998	0.8-1.0

Table 12: Selection for New Generation

Random_No	Fall into bin	Selection	Crossover	Mutation
0.7943	1	100110100	100110100	100110100
0.8714	3	100110100	100110100	100110100
0.6778	3	100110100	100110100	100110100
0.1480	4	100010100	100010100	100110100

Table 13: Fitness for New Generation

Path_No	X	X*X	Chromosome	Probability	CP	Associated bin
1	308	94864	100110100	0.25	0.25	0.0-0.3
2	308	94864	100110100	0.25	0.50	0.3-0.5
3	308	94864	100110100	0.25	0.75	0.5-0.8
4	308	94864	100110100	0.25	1.0	0.8-1.0

Table 14: Selection for New Generation

Random_No	Fall into bin	Selection	Crossover	Mutation
0.0911	1	100110100	100110100	100010100
0.4540	1	100110100	100110100	100110100
0.4062	2	100110100	100110100	100110100
0.0835	2	100110100	100110100	100010100

Table 15: Fitness for New Generation

Path_No	X	X*X	Chromosome	Probability	CP	Associated bin
1	276	76176	100010100	0.2226	0.2226	0.0-0.3
2	308	94864	100110100	0.2773	0.5	0.3-0.6
3	308	94864	100110100	0.2773	0.7773	0.6-0.8
4	276	76176	100010100	0.2226	1.0	0.8-1.0

Table 16: Selection for New Generation

Random_No	Fall into bin	Selection	Crossover	Mutation

0.5173	2	100110100	100110100	101000101
0.3497	3	100010100	100010100	101000101
0.7820	3	100010100	100010100	101000101
0.7794	3	100010100	100010100	101100001

Table 17: Fitness for New Generation

Path_No	X	X*X	Chromosome	Probability	CP	Bin
1	276	76176	100010100	0.25	0.25	0.0-0.3
2	276	76176	100010100	0.25	0.50	0.3-0.5
3	276	76176	100010100	0.25	0.75	0.5-0.8
4	276	76176	100010100	0.25	1.0	0.8-1.0

V. CONCLUSION AND FUTURE SCOPE

The purpose of this designed method is to use combination of two UML diagrams to generate test cases and optimization of test cases using GA. The designed approach is beneficial to generate the maximal number of test cases: numeral of test cases generated may not be valid. Integration of two UML diagram describes that it cover maximum number of test cases or all possibilities. GA is mainly used for optimization purpose that leads to optimal result. In near future, we will try to analyze other Evolutionary Algorihms.

REFERENCES

[1] Yoo-Min Choi and Dong-Jin Lim. Automatic feasible transition path generation from UML state chart diagrams using grouping genetic algorithms. 94:38–58, 2018.

[2] Rathee N. & Chhillar," A Survey on Test Case Generation Techniques Using UML Diagrams", *Journal of Software*, vol. 12, 8 August 2017.

[3] Khurana N., R.S Chillar,"Test Case Generation and Optimization using UML Models and Genetic Algorithm", *3rd International Conference on Recent Trends in Computing 2015*, Sciencedirect, PP.996-1004.

[4] Akshat Sharma, Rishon Patani, Ashish Aggarwal,"Software Testing Using Genetic Algorithms", International Journal of Computer Science & Engineering Survey vol.7,No.2, April 2016, PP-21-33.

[5] Itti Hooda, R.S Chillar,"Test Case Optimization and Redundancy Using GA and Neural Networks", International Journal of Electrical and Computer Engineering vol.8, No.6, December 2018, PP-5449-5456.

[6] Abdelkamel Hettab, Elhillali Kerkouche, Allaoua Chaoui,"A Graph Transformation Approach for Automatic Test Cases Generation from UML activity Diagram",*C3S2E 2015*,ACM,2015.

[7] Fernando AugustoDiniz, Glaucia Braga e Silva,"EastTest: An approach for Automaric Test Cases Generation from UML Activity diagram.", *Springer,2018*

[8] Meiliana, Irwandhi Septian, Ricky Setiawan Alianto, Daniel, Ford Lumban Gaol,"Automated Test Case Genartaion from UML Activity Diagram and Sequence Diagram using Depth First Search Algorithm", *2nd International Conference on Computer Science and Computational Intelligence 2017,ICCSCI*,ScienceDirect,october2017,PP-629-637.

[9] Monalisa Sharma, Debashish Kundu, Rajib Mall,"Automatic Test Case Generation from UML Sequence Diagrams" the proceeding of IEEE Conference on Software Maintainance,2007,IEEE. PP-996-1004.

[10] Ranjita Kumari Swain, Vikas Panthi, Prafulla Kumar Beher,"Generation of test cases using Activity Diagram" ,ISSN(PRINT):2231-5292, Vol.3,Issue-2,2013

[11] Saso Karakatic, Tina Schweighofer,"A Novel Approach to Generating Test Cases with Genetic Programming",Springer,2015,PP-260-271.

[12] Ajay Kumar Jena, Santosh Kumar Swain, Durga Prasad Mohapatra," Test Case Creation from UML Sequence Diagram: A Soft Computing Approach", Springer, Proceedings of ICCD 2014, Volume 1

[13] Arvinder Kaur, Vidhi Vig," Automatic test case generation through collaboration diagram: a case study", Springer, 2017.

[14] S. Dubey, R. Jhaggar, R. Verma, D. Gaur, "Encryption and Decryption of Data by Genetic Algorithm", International Journal of Scientific Research in Computer Science and Engineering, Vol. 5 No. 3, pp. 42-46, June 2017.

[15] G.R. Shahmohammadi and Kh.Mohammadi, "Key Management in Hierarchical Sensor Networks Using Improved Evolutionary Algorithm", International Journal of Scientific Research in Network Security and Communication, Vol. 4, No. 2, pp. 5-14, April 2016.

Authors Profile

Preeti is pursuning her masters from Himgiri Zee University, Dehradun. Her main research work focuses on Requirement Engineering, Model-Based Testing. She has 3 years of teaching experience.

Rohit Goyal is an Assistant Professor in Himgiri Zee University, Dehradun. He is currently pursuing Ph.D. He has published many research papers reputed international journals His main research work focuses on IOT, Software Testing.