International Journal on Recent and Innovation Trends in Computing and Communication
ISSN: 2321-8169 Volume: 12 Issue: 2
***Article Received:** 25 November 2023 **Revised:** 12 December 2023 **Accepted:** 30 January 2024*

Automated Test Case Generation Model from UML Diagrams based on Monotonic Genetic Algorithm

Jyoti Gautam Tiwari
Department of Computer Science and Engineering, SGSITS, Indore, M.P., India
jyotitiwariscs@gmail.com

Ugrasen Suman
Professor, School of Computer Science& IT Devi Ahilya University,Indore, M.P., India
ugrasen123@yahoo.com

Abstract :The procedure of developing package includes software testing as an imperative phase. The three components of the testing process are test execution, test evaluation and test case generation. The creation of test cases remains at the heart of challenging automation. It decreases the amount of mistakes and flaws while saving time and effort. A new way to automate the testing process has been developed to reduce the high tot test evaluation al of software testing and to improve the dependability of the testing procedures. In this paper, a innovative technique for creating and refining test cases using UML Activity Chart diagrams is proposed. The Genetic Algorithm's crossover method was used to create the new test sequence, and the test sequences' effectiveness was assessed by Mutation Analysis. As a result, they are unable to effectively combat multilayer perceptrons when faced with incorrect properties. Monotonic genetic algorithm is a Concept that is easy to understand and Supports multi-objective. The radial basis function (RBF) neural network algorithm currently in use has challenges counting the amount of neurons in the hidden layer and has poor weight learning ability from the hidden layer to the output layer. RBF networks have the drawback of giving respectively attribute a comparable weight since all factors are taken into account equally while calculating distance unless the attribute weight parameters are included in the entire optimization procedure.

Keywords**: Mutation analysis, Genetic Algorithm, State Chart Diagram, Test Cases, Crossover, software Testing, Activity Diagram and activity design.**

1. INTRODUCTION

Software evaluation involves balancing performance cost, and period. Software testing accounts for over half of the whole charge and time of a typical software development project. Testing is more than just troubleshooting. Testing may be done for reliability estimation, superiority declaration, or validation and verification. The three primary stages of software testing remain, test execution, test evaluation and test case creation. It is simple to device the final deuce sections. However, the initial portion requires a certain amount of knowledge.

The Unified Modeling Language (UML) is "a standard programming language for establishing, illustrating, preserving, and modelling the various aspects that comprise software systems" and is also used in non-IT technologies and commercial applications. Applying excellent fundamentals of engineering to the development or modelling of massive, intricate structures has proven beneficial, and UML is a consequence of that [1]. With its emphasis on pictorial designations, the UML is an essential part of the software creation and the computer program development procedure. When project teams use the UML, they can validate the product's design and architecture, develop new ideas, communicate and collaborate more successfully.

A graphical language for imagining, defining, building, and authenticating the objects of software-intensive systems is called the Unified Modeling Language [2]. A standardized approach to writing a system's blueprint is provided by the UML. It covers both concrete and conceptual fundamentals, such as database schemas, courses printed in a specific software design language, and refillable package mechanisms. Conceptual elements include business processes and system functions.

The developmental process for creating software comprises, software testing [3] is a crucial task. A sizeable amount of the budgets of software companies are allocated to testing-related activities. Before being accepted, the customer will validate a thoroughly tested software system. A program is tested by running it through a series of test cases and comparing the outcome to what was anticipated [4]. Defect avoidance should be a primary focus of testing. Typically, software artifacts like design, implementation, or specifications are the source of test cases. A suitable model of the system can be created to gain an understanding of the implementation before testing the system.

Software testing remains an essential and fundamental step in the construction of software. The process involves running a program to identify any blunders in the cipher. Exercise or evaluation is the procedure of applying manual or automated methods to a system or system component to confirm that it meets the required specifications or to find discrepancies among the predictable and actual consequences. Demonstrating incorrectness remains the goal of testing, and when an error is found, the test is deemed successful [6]. A sizeable amount of the budgets of software companies are allocated to testing-related activities. Testing accounts for over half of the software development budget, according to studies [7].

Test cases have been generated from UML diagrams in a few previous studies. However different approaches taken and distinct cases presented by earlier researchers result in imprecise comparison and appraisal of this field. This study offers a single scenario case with multiple UML diagrams that can be utilized during the process of creating test cases. The state chart diagram and the activity diagram are two UML behaviours diagrams that will be used in the initial work. A use-case or business process's state flow of activities from the beginning to the end can be represented using an activity diagram, which can also be rummage-sale to model system logic. A state chart diagram, on the other hand, can provide more specific information about the relationships between processes, their sequence of interaction, and the lifespans of the objects about those messages. In this research, an additional graph is created by combining the activity and state diagrams, and it is also utilized to produce test cases.

The format of the article is as surveys. We go over the research that has been done on case study creation methods utilizing various UML diagrams within subsection 2. The translation of diagrams of sequences and activity chart representations for hospital administration to graphs is covered in section 3. The graphs are combined into a system testing graph, and test cases created with the help of the provided example are developed and optimized using a genetic algorithm. We finally wrap up the report in this section by outlining the conclusion and our plans for future research. The cited sources are provided in the final section.

2. BACKGROUND AND RELATED WORK

This section, we review several studies on test case-generating methods utilizing UML diagrams. The proposed technique creates test cases by combining a sequence diagram and a use case diagram [8]. They transform the sequence diagram into a sequence graph after first converting the use case diagram into a use case graph. Following that, a System Graph is created by integrating the two graphs. It never becomes clear how the two graphs are combined. Furthermore, the generated test cases lack optimization.

They offered a technique for producing cases for testing that makes use of the activity schematic and UML sequence chart [3]. Using this method, a sequence chart is first turned into a sequence chart, and the activity diagrams into an activity graph. A system graph is then created by combining the two graphs, the sequence graph and the activity diagram. The test cases are then created by traversing the System Diagram using the Depth First Search Method (DFS). This process has also been applied to the validation of ATM cards.
The test cases were created with the aid of diagrams of activities [10]. Using that method, a diagram of activities is used to to develop a flow of activities graph first. AFG is traversed using the depth-first traversal method. The generation of every activity path is then suggested by an algorithm. Lastly, using activity path coverage criteria, test cases are created.

The method for creating test scenarios with a combination UML diagram for systems that are object-oriented was presented [11]. This method creates a Sequence-Activity Graph, which is then crossed to produce test cases that decrease the explosion of test cases. However, the test cases aren't at their best.

A model-based evaluation methodology using scenarios for testing produced from UML Sequence diagrams has been developed by [12]. Test cases are developed for mobile phone applications when the order diagram is converted to labeled changeover arrangements. The test case runs flawlessly on a small-screen smartphone application. The number of test cases is larger than request features test coverage is unmoving an issue for larger applications with larger LTSs. Utilizing their method, it is also necessary to reduce test case redundancy.

In UML state diagrams, Samuel et al.'s automatic test case generation proposal is presented in [13]. It covers every occurrence connected to state diagrams. By checking the borders established by straightforward predicates, they have decreased the overall number of test cases. They have provided examples of their automated test cases for frozen treat vending machines. They were unable to use the combination of variable methods to arrive at a globally optimal solution. They recommended a genetic algorithm to accomplish the same thing.

The de-facto norm for modeling object-oriented applications is the Unified Modeling Language (UML). Diagrams are available in UML to depict both the dynamic and static behavior of a framework [14]. Activity, sequence, and state drawings are used to depict the dynamic behavior of the system, whereas class, element, and distribution drawings are applied to denote the system's static behavior. The operations can be carried out during the design phase itself thanks to the UML Activity Diagram, which displays the object's actions [15].

The proposed technique utilizes UML models, which are a valuable source of data for test case design [16]. The implementation of the operation during the design phase is described by the UML-based activity diagram model, which also perfectly supports the information or explanation of similar happenings and organization features complicated in various activities. This method effectively meets the maximum path coverage criteria by creating test cases by examining the corresponding sequence and class diagrams of respectively state. It also has the benefit of lowering test model criteria costs due to design reuse.

A System Testing Graph (SYTG) is a graph created by converting a Activity Diagram and an State Chart Diagram. The test case is generated and optimized using a genetic algorithm according to their criteria. However, this method uses UML diagrams for state charts and sequence diagrams, and it does not explicitly state how the results compare.

3. RESEARCH METHODOLOGY

In our future procedure, an organization below test is transformed into a graph known as the System Testing Graph (SYTG), which is created by fusing a sequence diagram and a state chart diagram. Primary, the state chart diagram is transformed into a graph, and the sequence diagram into a graph. To create the System Testing Graph, these two graphs are then combined. This graph contains all the data needed to generate test cases in advance.

Constructed on a failure model and a coverage criterion, the genetic algorithm (GA) is functional to this network to automatically produce and enhance the test cases. We go over the approach we recommend in the section that follows.

3.1 Conversion of SCD into ACDG

First, we define the state chart figure and activity chart figure graph in this segment. After that, we demonstrate how to change SCD into ACDG. An activity chart figure depicts the active movement of a switch within a system beginning one state to another.

ACDG Definition: The description of the Activity Chart Graph is

ACDG ={S, T, GC, Si, Sf}

S=States in state chart figures stand for a collection of individuals' worth groupings where an item responds to events in the same way. T= the move beginning one state to extra is represented by a transition. A guard state must be satisfied to allow the change to that it fits to occur: Guard circumstances can be utilized to show that a given event can result in a variety of transitions, depending on the condition. Si stands for the source of all objects and is their initial condition. Because there are no items in this state yet, it is not a typical state. Sf = Final state denotes the object's final state of existence. Because the things in a final state do not exist, it is not a real state.

We will now talk about converting SCD to ACDG. The Activity chart design allows for the mapping of every state as a node. The continuous dependence of one on a different is represented by each transition from one state to the next.

SDG Definition: The definition of a state diagram graph is

The State is traditional of all nodes that represent the numerous stages of a situation, and SDG stands for **{State, Edge, First, Last}.** Edges indicate the exchange of information between several states. The beginning node, which represents the preliminary state, comes first. The last node represents the state in its ultimate form.

3.2 Combination of SDG and ACDG into SYTG

The following step after creating SDG and ACDG is to combine both of the graphs into one graph named a System Testing Graph (SYTG). We now outline SYTG.

SYTG definitions Condition = State (SDG) U Activity (ACDG) is the collection of entire the states of the order figure and state chart figure, and the SYTG is distinct as SYTG={S, T, F, L}.

T is the combination of the transitions from the Activity Chart Graph and State Graph, and its formula is T = T (SDG) U T (ACDG).

The initial node in the ACDG is F, and the last collection of nodes in the SYTG is L.

The algorithm to produce SYTG from SDG and ACDG is now presented.

Algorithm 1: GENERATE-SYTG

SYTG was created by combining the activity diagram graph and state chart diagram. This gave the test cases additional feature and covered the over-all framework of a system that users would use after start to finish, but they also included redundant information. It shows the Algorithm 1.

```
Input: Activity Chart Graph (ACDG) and State Diagram        Graph (SDG)
Output: System Testing Graph (SYTG)
    1.  P = Recognize overall the paths of (ACDG).
    2.  For every path pi є P do
    3.  SCj = SCi   //Begin by the SCi the initial node
    4.   T←ϕ
    5.   For evry state SCj of path pi do
    6.   If ci є SCi //present state consuming numerous resulting states
    7.   α = SCi-← SDG //Edge beginning the preceding node to the sequence Graph
    8.  β = SDG (Final)    SCi+1 // edge beginning the preceding node of SDG to the
    Following node of ACDG. Edge from SDG's failed concluding node to node SCi+1, here V=0
otherwise, edge beginning the SDG's effective terminal node to node SCi+1, here V=1.
    9.  End If
    10. T←T U T1
    11. If ci є\ SCi
    12. Y=SCi←SCi+1   // the current node of the identical State Chart Graph has an edge leading to
the next node.
    13. End If
    14. End For
    15. End
```

After that, we need to create test cases once we have gathered all the data in the SYTG Graph. Test cases must be optimized using any evolutionary technique once they have been generated. One evolutionary technique that we have used to create and optimize test cases on the SYTG (System Testing Graph) is called G.A. The technique we propose.

3.3 proposed methodology

3.3 Proposed Methodology

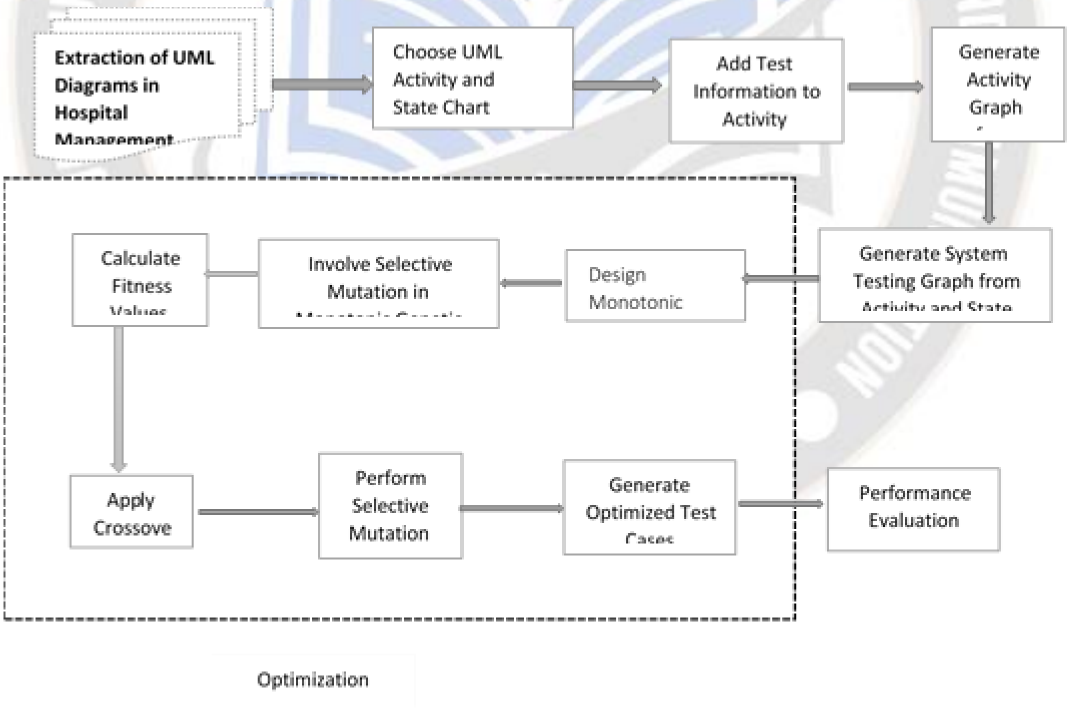

Extraction of UML Diagrams in Hospital Management
Choose UML Activity and State Chart
Add Test Information to Activity
Generate Activity Graph
Generate System Testing Graph from Activity and State
Design Monotonic
Involve Selective Mutation in
Calculate Fitness Values
Apply Crossove
Perform Selective Mutation
Generate Optimized Test Cases
Performance Evaluation
Optimization

Fig 1. Over all work Flow diagram

Fig.1 shows the overall work Flow diagram the first step in the process is to identify and choose the UML diagrams that will be used as the groundwork for creating test cases, specifically the UML Activity and State Chart diagrams. Data from UML Activity and State Graphs should be used. Specific UML Activity and State Chart figures are picked for further processing from the selected UML diagrams. The process of creating a system testing graph using the chosen UML diagrams. Add test data to the activity diagram. Relevant test-related information is incorporated into the

UML Activity Figure. The improved Activity Figure is used to provide an activity graph that shows the progression of decisions and actions. State Chart Figure to State Chart Graph Conversion. The State Chart Figure produces a State Chart graph that displays the states and transitions. From an activity and state chart graph, create a framework testing graph. A thorough System Testing Graph that captures system behaviour is produced by fusing the Activity Graph and State Chart Graph. The process of creating test cases is optimized using monotonic genetic algorithms.

There are stages in the procedure designed for monotonic improvement.

i. Fitness value

ii. Best individuals

iii. Crossover

iv. Perform selective Mutation

v. Determine the best solution

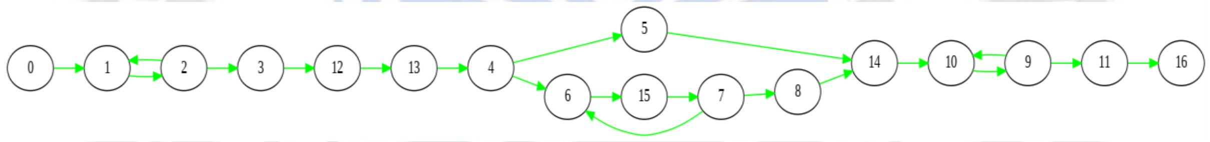
Create enhanced test cases through applying the specified Monotonic Genetic Procedure to the System Testing Graph, the procedure generates optimal test cases. The performance of the created optimized test cases is assessed using a variety of test coverage criteria.

3.3.1 Hospital Management System

Before drawing UML diagrams for the system, the first step is to collect as much hospital management information as you can. You need to complete every field to provide them with a high-quality system. Following that, the UML diagrams will be built using the data that was gathered. The class diagram for the hospital management system shows how the information or data the system will manage is structured. To denote these proofs or bits of information, classes will be applied. Depending on the methodologies it employs, each class will have its groupings. The categories that must be made in a hospital include patients, users, doctors, nurses, admissions, and transactions.

3.3.2 **Model Diagram for Activity Diagram**

In this step we have converted Activity chart diagram to Activity chart Graph the possible paths from the .xml files based on the kind of the state like State, Edge, First, Last, and condition of the state (like yes or no)



Possible paths from the Activity Diagram:

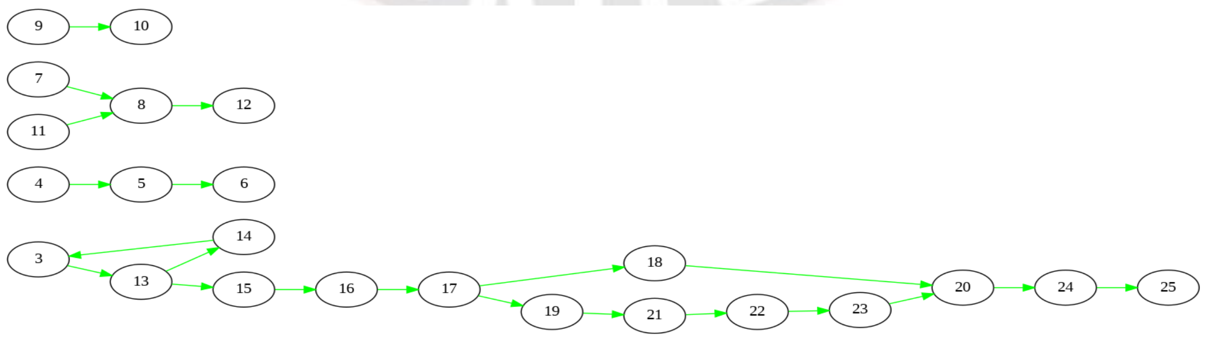
Path1 ➔0🡒1🡒
2🡒3🡒12🡒13🡒4🡒5🡒14🡒10🡒9🡒11🡒16

Path2
➔0🡒1🡒2🡒3🡒12🡒13🡒4🡒6🡒15🡒7🡒8🡒14🡒10🡒9🡒
11🡒16

Path3
➔0🡒1🡒2🡒3🡒12🡒13🡒4🡒6🡒7🡒8🡒14🡒10🡒9🡒11🡒
16

3.3.3 Model diagram for State Diagram:

In this step we have converted a state chart figure to State chart Graph the possible paths from the .xml files based on the kind of the state like State, Edge, First, Last and condition of the state (like yes or no).



Possible paths from the State Diagram:

Path4 ➔ 3🡒13🡒15🡒16🡒17🡒18🡒20🡒24🡒25

Path5 ➔
3🡒13🡒15🡒16🡒17🡒19🡒21🡒22🡒23🡒20🡒24🡒25

Path6 ➔ 3→14→13→15→16→17→18→20→24→25

Path7 ➔
3→14→13→15→16→17→19→21→22→23→20→24→25

Path Finalization
From the System testing Graph, we collected the possible paths.

All Possible Paths are,

- Path1 ➔0→1→2→3→12→13→4→5→14→10→9→11→16
- Path2 ➔0→1→2→3→12→13→4→6→15→7→8→14→10→9→11→16
- Path3 ➔0→1→2→3→12→13→4→6→7→8→14→10→9→11→16
- Path4 ➔ 3→13→15→16→17→18→20→24→25
- Path5 ➔ 3→13→15→16→17→19→21→22→23→20→24→25
- Path6 ➔ 3→14→13→15→16→17→18→20→24→25
- Path7 ➔ 3→14→13→15→16→17→19→21→22→23→20→24→25

3.4 Proposed Monotonic Genetic Algorithm for path selection

The steps in the algorithm we recommend are as follows:

1. Create a state flow diagram for a particular project or program.

2. Examine and compile every route that could lead from the starting point to the desired state.

3. Repetition of Steps 4 and 5 will prevent the most distinct path from happening.

4. Pick two potential directions. P1 and P2 {p11, p12, p13... p1n} and {p21, p22, p23... p2n}

5. Conduct crossover on P1 and P2 to enable the creation of a original sequence. The sequence P3= {p31, p32, p33...p3n} will be chosen based on the frequency of a state at file location i.

6. Carry out the alteration on P3 to carry out the dead state trimming.

7. Exit

3.5 Mutation analysis

The Mutation is the name given to a binary modification procedure. When a single genotype is exposed to it, a changed mutant is produced as the offspring or child. The Mutation is typically thought to result in an objective, random change. Theoretically, the mutation has a purpose. It can ensure that the area is interconnected. The result of mutation is that all conceivable chromosomes are accessible. The examination is limited to alleles that exist in the original sample even with crossover and even inversion. This can be overcome using the mutation operator by simply randomly choosing and altering a somewhat bit position in a string. This is helpful because if novel alleles do not develop in the first generation, crossover and inversion may not be capable of processing them.

Suppose we've already obtained an original string via crossover: [A=> B=> C=> E=> H =>A.]

Accept the alteration amount is 0.001, which is typically a low number. After that, we choose a random number between 0 and 1 for the A. The initial A must mutate if the amount is smaller than the mutation rate (0.001). For every test case, we produce and obtain a number. The procedure is recurrent using similar steps for the test cases following processing A and B in the same manner.

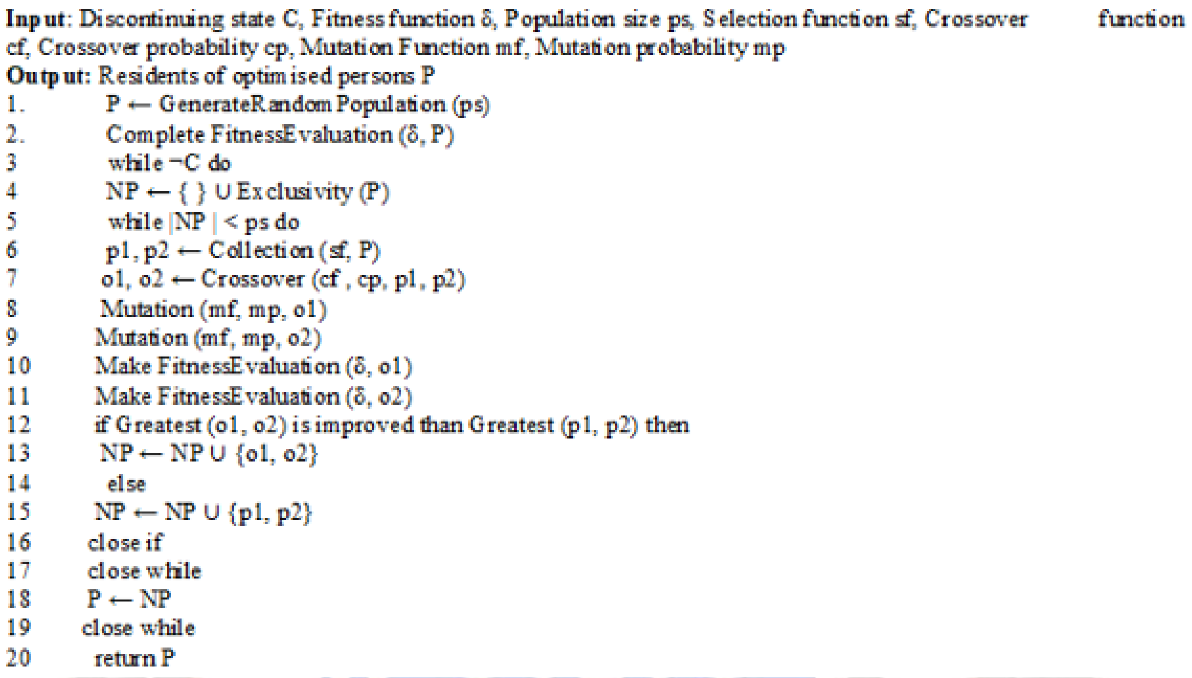
In our example, the new chromosome will look like the one below if only the first test changes and the respite of the tests remains the same. [B=> C=> E=> H =>A]

A single number at random is chosen for respective order in the parental residents, and this sequence has a 1% probability of evolving through mutation. If the above sequence is chosen for mutation, the operation sequence procedure is reversed and a copy of the sequence is created. To ensure that a workable schedule is always produced by the mutation, only activities from separate jobs will be reversed. According to the results of the investigation, reciprocal exchange (RX) and order crossover (OX) are effective together and are hence in use. We obtain the test case's final sequence following a random crossover.

In our instance, the last test procedure is [A=> B=> C=> E=> H]

Algorithm 2: Monotonic Genetic Algorithm

Optimised persons (p) have a more complex data structure or it can be a binary string. One can generate the initial pool of optimised persons manually or at random. The fitness function evaluates an optimised person's suitability for achieving a given goal. The optimised persons that take part in the evolutionary stage of the genetic algorithm comprised of the crossover and mutation operators are determined by the selection function. By exchanging genes from two p, the crossover operator produces two new optimised persons. A optimised person’s gene is altered by the mutation operator, adding a new optimised person.



```
Input: Discontinuing state C, Fitness function δ, Population size ps, Selection function sf, Crossover        function
cf, Crossover probability cp, Mutation Function mf, Mutation probability mp
Output: Residents of optimised persons P
1.        P ← GenerateRandomPopulation (ps)
2.        Complete FitnessEvaluation (δ, P)
3         while ¬C do
4         NP ← { } ∪ Exclusivity (P)
5         while |NP | < ps do
6         p1, p2 ← Collection (sf, P)
7         o1, o2 ← Crossover (cf , cp, p1, p2)
8         Mutation (mf, mp, o1)
9        Mutation (mf, mp, o2)
10       Make FitnessEvaluation (δ, o1)
11       Make FitnessEvaluation (δ, o2)
12       if Greatest (o1, o2) is improved than Greatest (p1, p2) then
13        NP ← NP ∪ {o1, o2}
14         else
15       NP ← NP ∪ {p1, p2}
16      close if
17      close while
18      P ← NP
19     close while
20       return P
```

4. RESULTS AND DISCUSSIONS

Test path with maximum fitness value will be the optimum path. From this research, ['path3', 'path2', 'path7', 'path4', 'path1'] shows higher fitness value (0.86) compared with others (0.737,0.775,0.73). Thus, we could conclude that ['path3', 'path2', 'path7', 'path4', 'path1'] are the optimum paths that has maximum coverage.

Optimized Paths are:

Path3
➔0→1→2→3→12→13→4→6→7→8→14→10→9→11→16

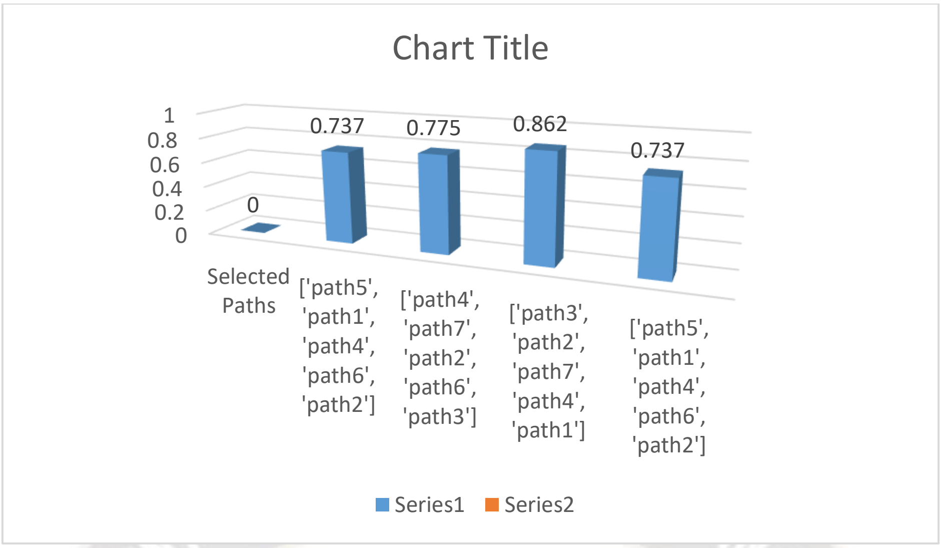
Path2
➔0→1→2→3→12→13→4→6→15→7→8→14→10→9→11→16

Path7 ➔
3→14→13→15→16→17→19→21→22→23→20→24→25

Path4 ➔ 3→13→15→16→17→18→20→24→25

Path1 ➔0→1→
2→3→12→13→4→5→14→10→9→11→16

Selected Paths	**Fitness Values**
['path5', 'path1', 'path4', 'path6', 'path2']	0.737
['path4', 'path7', 'path2', 'path6', 'path3']	0.775
['path3', 'path2', 'path7', 'path4', 'path1']	0.862
['path5', 'path1', 'path4', 'path6', 'path2']	0.737



5. CONCLUSION AND FUTURE WORK

The proposed approach involves creating test cases using activity charts and state chart diagrams, and then optimizing those test cases using a GA. The advantage of utilizing this technique is that the determined amount of test cases is produced; beyond this point, no other legitimate test cases can be produced. This strategy makes use of an activity chart diagram; on an Activity chart, one Activity can lead to numerous Activities that will produce the best results. Most interactions between the items during the procedure are covered by a sequence diagram. Therefore, if two UML diagrams are combined, the determined number of test cases, or all possible scenarios, will be covered. pre-post condition issues We created UML diagrams using Rational Rose software. Additionally, we have enhanced test cases using a genetic algorithm, which yields the best results. Furthermore, since this remains an automated method, we can automate the entire process in subsequent work.

REFERENCE

[1] Jones, B. F., Sthamer, H. H., & Eyres, D. E. “Automatic structural testing using genetic algorithms”. *Software engineering journal*, *11*(5), 299-306.

[2] Linzhang, W., Jiesong, Y., Xiaofeng, Y., Jun, H., Xuandong, L., & Guoliang, Z. “Generating test cases from UML activity diagram based on gray-box method”. In 11th Asia-Pacific software engineering conference (pp. 284-291). IEEE.

[3] Engels, G., Förster, A., Heckel, R., & Thöne, S., Process modeling using UML. “Process-Aware Information Systems: Bridging People and Software through Process Technology”, 83-117.

[4] Dennis, A., Wixom, B., & Tegarden, D., “Systems analysis and design: An object-oriented approach with UM”, John wiley & sons.

[5] Fowler, M. “UML distilled: a brief guide to the standard object modeling language”, Addison-Wesley Professional.

[6] Akhil Reddy, B. “Designing Microservices with Use Cases and UML” (Doctoral dissertation, University of Dayton).

[7] Booch, G. The unified modeling language user guide. Pearson Education India. Mateen, A., Nazir, M., & Awan, S. A. “Optimization of test case generation using genetic algorithm (GA)”, *arXiv preprint arXiv:1612.08813*.

[8] Sarma, M., Kundu, D., & Mall, R. “Automatic test case generation From UML sequence diagram”, In 15th International Conference on Advanced Computing and Communications (ADCOM 2007) (pp. 60-67). IEEE.

[9] Tripathy, A., & Mitra, A. “Test case generation using activity diagram and sequence diagram”, In Proceedings of International Conference on Advances in Computing (pp. 121-129). Springer India.

[10] Swain, R. K., Panthi, V., & Behera, P. K. “Generation of test cases using activity diagram”, International journal of computer science and informatics, 3(2), 1-10.

[11] Dalai, S., Acharya, A. A., & Mohapatra, D. P. “Test case generation for concurrent object-oriented systems using combinational UML models”, International Journal of Advanced Computer Science and Applications, 3(5).

[12] Cartaxo, E. G., Neto, F. G., & Machado, P. D. “Test case generation by means of UML sequence diagrams and labeled transition systems”, IEEE International Conference on Systems, Man and Cybernetics (pp. 1292-1297). IEEE.

[13] Samuel, P., Mall, R., & Bothra, A. K. "Automatic test case generation using unified modeling language (UML) state diagrams", *IET software*, *2*(2), 79-93.
[14] Sapna, P. G., & Mohanty, H. "Automated scenario generation based on uml activity diagrams", In 2008 International Conference on Information Technology (pp. 209-214). IEEE.
[15] Thanakorncharuwit, W., Kamonsantiroj, S., & Pipanmaekaporn, L. "Generating test cases from UML activity diagram based on business flow constraints", In Proceedings of the Fifth International Conference on Network, Communication and Computing (pp. 155-160).
[16] Sabharwal, S., Sibal, R., & Sharma, C. "Prioritization of test case scenarios derived from activity diagram using genetic algorithm", In International Conference on Computer and Communication Technology (ICCCT) (pp. 481-485). IEEE.