Journal of Information Technology Research
Volume 15 • Issue 1

Model-Based Test Sequence Generation and Prioritization Using Ant Colony Optimization

Gayatri Nayak, Siksha 'O' Anusandhan (Deemed), India*
https://orcid.org/0000-0001-5360-8149

Mitrabinda Ray, Siksha 'O' Anusandhan (Deemed), India

ABSTRACT

The paper presents an approach to generate and optimize test sequences from the input UML activity diagram. For this, an algorithm is proposed called Unified Modelling Language for Test Sequence Generation (UMLTSG) that uses a search-based algorithm, named Test Sequence Prioritization using Ant Colony Optimization (TSP ACO) to generate and optimize test sequences. The algorithms overcome the existing limitations of handling complex decision-making activity such as conditional activity, fork activity, and join the activity. The optimization process helps to reduce the number of processing nodes that leads to minimizing the time and cost. The proposed approach experiments on a well-known application Railway Ticket Reservation System (RTRS). APFD metric measures the effectiveness of our approach and found that the prioritized order of test sequences achieved 20% higher APFD score. Apart from this, the authors have also experimented on six real life case studies and obtained an average of 52.16% reduction in redundant test paths.

KEYWORDS

Activity Diagram, Ant Colony Optimization, Test Sequence Generation, Test Sequence Prioritization, Unified Modelling Language

1. INTRODUCTION

In conventional software projects, sixty percent of the exertion is used upon software testing to generate trustworthy software (Farooq et al., 2008). Thus, researchers should give more attention to adequate testing. Producing test cases are clumsy depending on the source code. Therefore, the generation of test sequences based on design models is one of the current research areas in software testing (Chen et al., 2007, Jena et al., 2014). The dynamic behaviour of software is represented by various UML interaction diagrams at design level. UML includes activity diagrams, use-case diagram, state chart diagrams, sequence diagrams, and collaboration diagrams, etc. The activity diagram is a sequence of activities performed in the software (Chandler et al., 2005). Literature study says that researcher have proposed various methods to generate test sequences from UML diagram such as activity diagram, sequence diagram (Bhattacharjee et al., 2018). The test sequence represents a precise order of test cases (Srivastava et al., 2010). A test sequence is also alternatively abbreviated as test path in this paper. It comprises different kinds of nodes and edges such as fork node, conditional node, join node,

DOI: 10.4018/JITR.299946 *Corresponding Author

etc. This work generates test sequences from the input activity diagram. The generation of test sequences for the activity diagram is a crucial task. So, the input diagram is converted to XMI code and then to a control flow graph (CFG) as an intermediate representation. It requires traversing all the nodes and edges of CFG to test all types of nodes and edges such that the maximum number of faults should get detected. We use Depth First Search (DFS), on a graph to generate test sequence. During the test process, each node performs some action, transition, which generates a test sequence. This is done at the design phase and implemented using the proposed algorithm named UML Test Sequence Generation (UMLTSG). This algorithm finds linearly independent paths (i.e. Test Sequences) to cover all nodes and edges of the obtained CFG. Test sequence prioritization is used for reordering of test sequences, to enhance test efficiency (Elbaum et al., 2004). Among various approaches, test sequences prioritization is one, which is used to order the test sequences according to the highest priority, based on some principles like execution time, coverage criteria, fault detection rate, etc. (Elbaum et al., 2004, Rothermel et al., 2001, Wong et al.,1997). Optimization of those generated test sequences is required for the effective testing of software (Sabharwal et al., 2010). Different optimization techniques such as GA (Kaur et al., 2011), ACO (Marijan et al., PSO (Lv et al.,2018) are used to optimize test sequences. We use ACO to find an optimal test path because of the following reasons:

- It is capable for finding better solutions.
- It avoids premature convergence.
- It requires less number of parameters to tune as compared to GA and PSO.

1.1 Motivation

There is a precise need to focus on generation and optimization of test sequence in design phase. Because, it gives an opportunity to the testers for early detection of bugs at design phase i.e. before coding phase. Generating test sequences at the design phase is a mechanism to improve the efficiency of testing software. The testing costs of software is reduced as we are able to find bugs at the design phase, that is without waiting up to coding phase. This work uses ACO algorithm for optimizing test sequence generation process. But, ACO technique sometimes suffers from falling into local optima and long search time. To remove these two limitations, we modify the objective function and restrict the number of feasible solutions to linear order (i.e. O(n)) using cyclomatic complexity metric. The second motivation is to find a better method to eliminate redundant test sequences. Usually, test sequences are generated by traversing a control flow graph, in which few nodes appear repeatedly in them, generating redundant paths. Therefore, these redundant paths should be removed.

1.2 Objective

UML activity diagram is the most important design artifacts among several UML models by considering various elements of activity diagram. We set the following objectives for this paper.

To generate test sequence, by including recent UML key features in the activity diagram such as fork node, join node, guarded condition and swim lane etc. To optimize the generated test paths by applying ACO. We named a path as critical path where critical path carries many conditions and more number of dependent activities.

1.3 Research Contribution

First contribution of this paper is converting the input activity diagram to CFG. Second contribution is to filter information regarding nodes and edges and draw a CFG using Graphviz tool. Third contribution is to define ACO algorithm that can find an optimal path from the obtained CFG. The optimal path refers to maximize the strength of the generated non-redundant test paths and limit the number of test paths as per cyclomatic complexity (CC) value. CC helps to cover all nodes and edges with minimum number of test paths. The strength of each test path is calculated by assigning different weight values to different types of nodes. The total strength of a path is accumulated by considering all the nodes present in a path. Thus, it generates test sequences optimally by reducing testing cost and removes node redundancy in the test node sequences.

The remaining part of the article is explained in the following sections. Section 2 presents basic concepts on UML Activity diagram and ACO, that are used in this work. Section 3 refers to the related work done on test sequence generation and test sequence optimization at the code level, and design level. Section 4 explains the proposed model and its components. Section 5, explain our method using a well-known case study, RTRS. Section 6 discusses some existing work. Section 7 highlights some threats to the proposed approach, and finally, Section 8 concludes the work by mentioning some future scope.

2. BASIC CONCEPTS

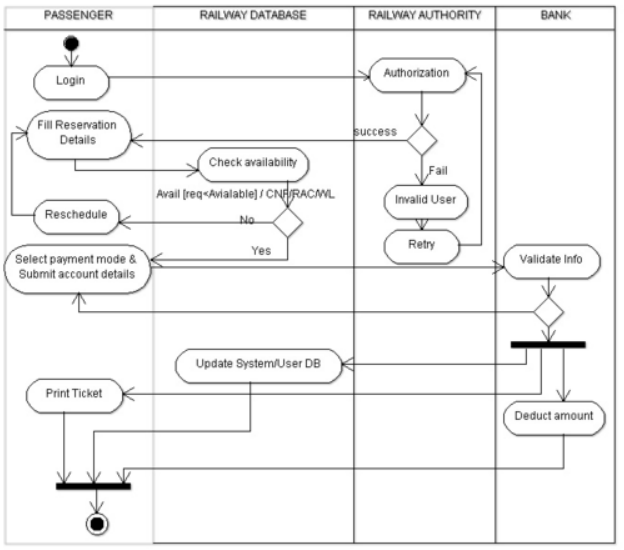
Here, we present the basic concepts which are required for our proposed approach. Then, we discuss about ACO algorithm that is used in our approach for test sequence optimization.

2.1 Activity Diagram

In 1997, the UML diagram used to model a software system by Object Modulo Group (OMG) (Malcolm et al., 1997). The UML diagram is designed to support a regular approach to envision the system (Chandler et al., 2005). This diagram is used to define the software system's structures and performance. The different components of the system are defined by the structural diagram like a class diagram, a composite structure diagram, etc. (Farooq et al., 2008). This paper focuses only on an activity diagram with advanced elements like the fork, join, guard condition and its action, decision nodes, etc. The behavioral characteristics of a system are represented by the flow diagram known as the activity diagram. The activity diagram has some states to do some work. So, activity diagram represents sequence of activities. The arrangement of different activities of the system is also described using this diagram. Here different conditions of the work are represented, using decision and fork nodes, which stands for parallelism.

The edges are used to represent the changes from one activity to another. Every transaction is denoted by arrows. Every user reads various actions from the top most position by using the connection symbol arrows in an activity diagram. Figure 1 shows a sample activity diagram for the railway reservation system and we have tried to test normal activities as well as negative or exceptional activities that are captured for a particular scenario. Exceptional activities such as authentication failure,

Figure 1. Activity Diagram of Railway Ticket Reservation System



exceptions due to wrong input and stuck at loop. This diagram shows many advanced features such as swimlane to show parallel activities between the four objects, guard condition for check availability activity, fork to generate parallel activities and join to merge parallel links, etc.

2.2 Ant Colony Optimization

This optimization technique works on performing actual ants. Using the characteristics of ant, this ant colony optimization technique was used in the traveling salesman problem (Dorigo et al., 1991). Mainly the ant left the pheromone on the path so that another ant follows that path when they travel for food (Mann et al.,2015). Ants behavior is used to find different probabilistic solutions. It is also used for different combinatorial problems. This algorithm is applied to recognize the path in the activity diagram easily. In this approach, ACO is used to find the rest path on a weighted graph. The ants are used to form the solution by traveling on the weighted graph. The formulation of the solution is a stochastic procedure which is influenced by the pheromone model of ant (Bhattacherjee et al., 2018).

The set of components associated with this graph like nodes and edges can be changed using the behavior of an ant at execution time. Ants can trace all feasible paths using pheromone spread over the routes. But, the optimal path is decided based on the highest accumulated pheromone on a path (Bhattacherjee et al., 2018). Usually, an ant collects the following information from a path:

- Pheromone(p): The amount of pheromone leaves by an ant while traversing an edge.
- Heuristic(h): This helps in determining the next node to be visited by the ant.
- Visited status (v): It represents the state of a node, whether a node is visited or not.

3. LITERATURE FINDINGS

This section elaborates study on related existing work done using GA, PSO, and ACO for regression testing process. Sushant et al.,(2017) suggested a multi-objective approach to maximize fault finding rate in minimum time using ACO technique. The objective of their proposed approach was to increase fault detection rate with minimum regression cost. Their work helped to control the convergence, while achieving better path exploration. Their method gave special attention to select qualitative test cases such that, it could minimize regression test time. Yogesh Singh et al., (2010) proposed a method to prioritize test suite using the fault detection rate and execution time in a time reserved situation using ACO. Their approach was used to eliminate a set of recognized errors, where a test case was able to detect one or more faults. In their proposed ACO algorithm, the n number of the test cases were represented by the n number of ants. Yang et al., (2014) proposed an improved version of ACO algorithm for software test cases generation. Their approach was used to focus on three things, such as pheromone volatilization rate, local pheromone and the global path pheromone updating criteria for ant colony optimization (IGPACO). Their results had indicated that their method had effectively improved the search efficiency, restrained precocity, promoted case coverage, and reduced the number of iterations in regression testing.

Marijan et al., (2013) used ACO algorithm to prioritize test case. They considered three factors such as the number of faults detected, execution time, and fault severity of a software system for prioritization. These factors were used for the regression testing process. They had validated their proposed method using the APFD metric.

Mohapatra et al., (2016) suggested an ant colony optimization method to select the test sequences for prioritization process. The effectiveness of test cases was ordered and measured using APFD metric. Their proposed prioritization method had reduced time complexity and space complexity as compared to random ordering technique (Ahmed et al., 2013). All the above discussed prioritization approaches are based on source code which consider fault rate, execution time, etc. to generate and prioritize test sequences. Similarly, for design phase, UML models are used to generate test sequence and prioritize the test sequence using optimization algorithm as discussed below.

Suri et al., (2011) suggested a method to improve fault detection rate while prioritizing test cases. In their method, they considered each node of a UML activity diagram as a test case. Then, they had applied probabilistic method to find the next vertex in the activity diagram. They used a heuristic function to optimize time and fault detection rate. In their approach, the test cases were continuously generated until all faults are detected from the test suite. If the obtained solution found to be invalid, then the entire process was repeated. Their proposed approach had reduced the size of the test suites by 62.5%.

Chandler et al., (Chandler et al., 2005) suggested a method to produce test sequences using UML activity diagram. Their proposed method explained the procedure of retrieving the required meta-data from the XML file to generate test cases. They had used various types of software tools to extract XML code.

Farooq et al. (Farooq et al., 2008) had used a novel technique to generate test sequences automatically depending on coverage criterion like sequential and concurrent based on the execution of Colored Petri Nets. Their proposed technique was experimented on various applications such as service-oriented and concurrent applications.

Sangeeta et al., (Sangeeta et al., 2010) suggested a Genetic Algorithm (GA) based method to generate test sequences. They had assigned different weights values to different nodes of an activity diagram using FAN IN and FAN OUT technique. The strength of those test sequences were measured using the weights of the node and prioritize those test sequences.

Doerner et al., (2003) used the Markov Usage Model for selecting a set of test paths. They had calculated the probability of failure to know about the amount of improper selection of a test paths (Zhou et al., 2012). They had also analyzed the losses occurred at the failure time. Due to this reason, the testing cost for different test sequences were varied differently. This random variation of testing cost was motivated them to apply ACO for test sequences generation and then prioritization. But, in our approach, we generate random behavior based on the variance of weight at nodes of a graph. We propose a probabilistic method for defining the objective function such that ACO can be applied for test sequences generation and then prioritization.

Srivastava et al., (2010) defined ACO algorithm to generate test sequences automatically. They had claimed for getting a strong level of software coverage. They had generated the test sequences by traversing all the states obtained from the given UML state machine diagram. ACO algorithm was used to generate test sequences by covering maximum number of transitions.

Mann et al., (2015) defined an algorithm to generate optimal paths sequences, inspired by real Ant's behavior. The proposed algorithm was able to generate all DD paths from the CFG and prioritize them according to the path strength. The algorithm was set to stop only when all nodes were covered, thus making 100% path coverage.

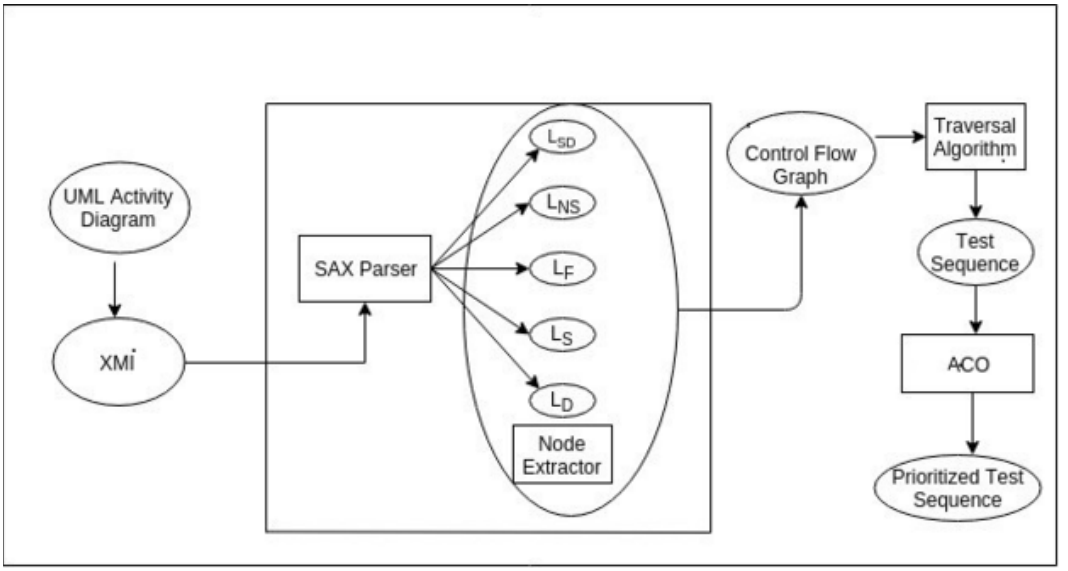
Li et al., (2004) proposed a state transition based technique to generate test sequences using UML State Machine diagram. They had suggested a procedure for converting UML diagram to generate test sequences using ACO. The generated test paths were feasible, adequate and non-redundant. Unlike their approach, we propose a probabilistic method with a different objective function for ACO algorithm.

Sharma et al., (2011) proposed an algorithm to generate test sequences from the state machine transitions of the given software. They had applied ACO algorithm in the control flow graph to generate test sequences. Their proposed method had tried to determine maximum coverage with less amount of redundancy. They had optimized time and coverage scores.

4. PROPOSED METHOD FOR TEST SEQUENCE GENERATION AND PRIORITIZATION

This section describes our whole approach to generate and prioritize test sequences using ACO. Figure 2 shows the block diagram of our approach. This approach suggests a new objective function for ACO technique that can achieve optimal test sequences by reducing number of redundant paths. Then, test sequences are prioritized by assigning some weight values to the decision nodes and the fork nodes. Chandler et al., (Chandler et al., 2005) had used a different method to generate test paths from UML activity diagram. We extend the same concept to generate a CFG from activity diagram to find test sequences.

Figure 2. Architecture of Proposed Model



4.1 Test Sequences Generation

First, we construct an activity diagram, as shown in Figure 1. Then, from this diagram XMI code is exported to a separate file. This XMI file is used to extract information regarding the flow of activities carried out in the activity diagram. Here, we have considered the Railway Ticket Reservation System (RTRS) for test sequence prioritization. For this system, an activity diagram is drawn using a commercial tool called RSA (IBM, 2018), which is shown in Figure 1. This tool helps to generate the XMI Metadata Interchange (XMI). Thus, it helps in the direct processing of UML models. Parsers are useful for processing the XMI data. Therefore, we use SAX parsers to parse the data from the XMI files generated from the UML diagram. In this work, an activity diagram is constructed for the given system called the Railway Ticket Reservation System (RTRS) using IBM RSA tool. Then, the XMI representation from the UML activity diagram is exported to a separate file. After that, the SAX parser is used to parse the XMI data (sax2, 2001). This parsing helps to extract node information from the XMI file in coded form. The node information is node id and node name. Using these nodes and edges, a control flow graph (CFG) is generated. DFS traversal algorithm is applied to this CFG for generating test sequences. These steps are written in Algorithm 1.

```
Algorithm 1: UML Test Sequence Generation (UMLTSG):
UML Test Sequence Generation(UMLTSG):
Input: XMI code of Activity Diagram
Output: Test Sequences
1.Construct the required activity diagram using RSA.
2.Generate an XMI file.
3. Use SAX parser to extract Nodes and Edges. This information has
stored in a linked list.
4.Group Source node, Destination nodes, and edges using different
linked lists.
5.Similarly Linked List also used for storing information about
decision nodes, fork nodes, and join nodes.
6.Create a CFG using these nodes and edges.
```

```
7. Apply DFS algorithm to generate test sequences.
8. Display test sequences.
```

Table 1. Node Type and corresponding weights

Serial No.	Node Type	Weight Assumed
1	Start Node	1
2	Merge Node	5
3	Decision Node	7
4	Fork Node	10
5	Guarded Decision	9
6	Loop	2
7	End	1

4.2 Prioritization

Test sequences prioritization method helps to decrease the testing cost and number of bugs as well. Therefore, testing has to be done using an effective method and choosing the appropriate test sequence is a challenging task in software testing.

4.2.1 Test Sequence Weight

An activity diagram represents a sequence of activities and their execution. These activities contain many conditional nodes and fork nodes that guide the execution flow. These nodes are called decisive nodes to generate a test path. Thus, these nodes should be given more importance and assigned with higher weight values. Therefore, we have assumed or assigned a weight value to each node such that the assigned weight can reflect their importance in making a test path. The assumption details about the types of nodes and their corresponding weight are shown in Table 1 after referring to some existing literature (Panthi, V. (2018)). This table indicates that we have assumed Start node and End node to carry equal contribution towards generating a test path. Decision node, merge node and Guard condition node are having moderate influence on test path weight. Similarly, Fork node has given the highest weight as it impacts much on generating test path. Because, Fork node is having more number of outgoing edges, representing high FAN OUT value. After weight assumption, the strength of a node is calculated using Eq. 1 with two parameters like FAN IN and FAN OUT.

Here, FAN IN is a number that represents incoming edges to a node. Similarly, FAN OUT is used to represent several outgoing edges of a node. So, a test sequences having many decision nodes should be considered as more critical test sequence for testing purposes. Then, total weight value of each test sequence is computed for ranking test sequences. This leads us to prioritize the test sequences.

$$\text{Weight}(\text{Node}_i) = \text{FAN IN}(i) \times \text{FAN OUT}(i) \quad (1)$$

4.2.2 Ant Colony Optimization

It is difficult to find a testing criteria, which is applied to cover all kinds of faults. So, we have to follow a probability-based method for test sequence generation. The number of test sequences are determined using the cyclomatic complexity mentioned in Eq. 2, where `e' and `n' denotes number of edges and nodes in the given graph. An optimal path is selected from these generated test sequences.

Cyclomatic Complexity (CC) = e-n + 2 (2)

```
Algorithm 2: Test Sequence Prioritization using ACO (TSP_ACO):
Input: Control Flow Graph
Output: Optimal Test Sequence
1. Initialize each edge pheromone value p(ij)= 1.
2. Initialize each edge heuristic value h(ij)= 2.
3. Initialize each node Visited status = 0.
4. // Visited status = 0 means not visited
5. Assign initial weight of each node.
6. Initialize current node = start
7. CC = e - n + 2 // Cyclomatic Complexity(CC) in Eq. 2
8. Set i= current_node index
9. Visited status(i) = 1
10. if (i = Decision_node)
```

11. $Prob(ij) = \frac{\left[p(ij)*h(ij)\right]^{-1}}{\left[\left[p(ij)*h(ij)\right]^{-1}\right]*\left[\left[p(ik)*h(ik)\right]^{-1}\right]}$

```
12.  // prob. value for left node
```

13: $Prob(ik) = \frac{\left[p(ik)*h(ik)\right]^{-1}}{\left[\left[p(ij)*h(ij)\right]^{-1}\right]*\left[\left[p(ik)*h(ik)\right]^{-1}\right]}$

```
14: // prob. value for right node
15: if (Prob(ij)>= Prob(ik))
16: select edge (ik)
17: otherwise select edge (ij)
18: p(ij) =[p(ij)+h(ij)]^-1
//Update Pheromone
19: h(ij) = 2*h(ij) // Update Heuristic
20: Weight = Weight +Weight(selected_node) //Use Eq. 1
21: Strength[CC] = Weight
22: current_node = selected_node
23: EndIf
24: if current node! = \END node”
25: Print the path and Strength[CC]
26: Otherwise go to Step 8
27: CC = CC -1
28: if CC>0
29: Find a node j having node visit status(v)=0, set
30: current node= node j and go to Step 8
31. EndIf
32: Display Optimal Test Sequence
33: End
```

4.2.3 Prioritization Technique

There are many prioritization techniques available, out of which one criterion may be the length of the test sequence (path). Based on this criterion, the highest priority is assigned to the longest path. The existing methods can be enhanced by considering potentially contributing nodes of each path.

This approach does some enhancement in adding some special nodes with a weight assignment policy for better results. Here, special nodes like fork node, join node, the node with a guard condition is added in this approach to strengthen the prioritization process using ant colony optimization. The obtained paths are ranked based on their weight determined for prioritization.

The step wise execution of the optimal test sequence generation method is explained in the Algorithm 2. It takes CFG as input and gives an optimal test sequence as output. Here, steps 1 to 5 are used to initialize the solution space. Step 6 is used to find CC of the inputted CFG. Steps 9 to 17 show how do the ants take decision at a conditional node. At a decision node, the probability of movement towards the left node and right node are calculated for the current node. Based on the selected edge, the weight is updated, and the next node becomes the current node. This process is repeated using Steps 18 to 21. Steps 22 to 24 say about the stopping criteria that until all the nodes and components of the graph are visited, the algorithm does not stop.

Figure 3. Generated XMI code from developed activity diagram

```
<packagedElement xmi:type="uml:Activity" xmi:id="_yky-otUaEeO18ZGvLO-emQ" name="Activity1">
  <node xmi:type="uml:InitialNode" xmi:id="_yky-o3UaEeO18ZGvLO-emQ" name="start" outgoing="_ykzlwdUaEeO18ZGvLO-emQ"/>
  <node xmi:type="uml:OpaqueAction" xmi:id="_yky-pNUaEeO18ZGvLO-emQ" name="visit the website" outgoing="_ykzlaNUaEeO18ZGvLO-emQ" incom
  <node xmi:type="uml:DecisionNode" xmi:id="_yky-pdUaEeO18ZGvLO-emQ" name="browse or search item" outgoing="_ykzlu9UaEeO18ZGvLO-emQ _y
  <node xmi:type="uml:OpaqueAction" xmi:id="_yky-ptUaEeO18ZGvLO-emQ" name="browse item" outgoing="_ykzlytUaEeO18ZGvLO-emQ" incoming="_
  <node xmi:type="uml:OpaqueAction" xmi:id="_yky-p9UaEeO18ZGvLO-emQ" name="search item" outgoing="_ykzlx9UaEeO18ZGvLO-emQ" incoming="_
  <node xmi:type="uml:DecisionNode" xmi:id="_yky-qNUaEeO18ZGvLO-emQ" name="found desired item or not" outgoing="_ykzl09UaEeO18ZGvLO-em
  <node xmi:type="uml:MergeNode" xmi:id="_yky-qdUaEeO18ZGvLO-emQ" name="find item" outgoing="_ykzlzdUaEeO18ZGvLO-emQ" incoming="_ykzlx
  <node xmi:type="uml:OpaqueAction" xmi:id="_yky-qtUaEeO18ZGvLO-emQ" name="decide an item to buy" outgoing="_ykzl0NUaEeO18ZGvLO-emQ" i
  <node xmi:type="uml:OpaqueAction" xmi:id="_yky-q9UaEeO18ZGvLO-emQ" name="view item" outgoing="_ykzl1tUaEeO18ZGvLO-emQ" incoming="_yk
  <node xmi:type="uml:OpaqueAction" xmi:id="_yky-rNUaEeO18ZGvLO-emQ" name="add to cart" outgoing="_ykzl2dUaEeO18ZGvLO-emQ" incoming="_
  <node xmi:type="uml:DecisionNode" xmi:id="_yky-rdUaEeO18ZGvLO-emQ" name="proceed to payment" outgoing="_ykzl4tUaEeO18ZGvLO-emQ _ykzl
  <node xmi:type="uml:OpaqueAction" xmi:id="_yky-rtUaEeO18ZGvLO-emQ" name="via internet banking" outgoing="_ykzl3NUaEeO18ZGvLO-emQ" in
  <node xmi:type="uml:OpaqueAction" xmi:id="_yky-r9UaEeO18ZGvLO-emQ" name="enter your banking credentials" outgoing="_ykzl39UaEeO18ZGv
  <node xmi:type="uml:OpaqueAction" xmi:id="_yky-sNUaEeO18ZGvLO-emQ" name="enter your address" outgoing="_ykzl8dUaEeO18ZGvLO-emQ" inco
  <node xmi:type="uml:ActivityFinalNode" xmi:id="_yky-sdUaEeO18ZGvLO-emQ" name="end" incoming="_ykzl6NUaEeO18ZGvLO-emQ"/>
  <node xmi:type="uml:OpaqueAction" xmi:id="_ykzlsNUaEeO18ZGvLO-emQ" name="exit the website" outgoing="_ykzl6NUaEeO18ZGvLO-emQ" incomi
  <node xmi:type="uml:OpaqueAction" xmi:id="_ykzlsdUaEeO18ZGvLO-emQ" name="Cancelling payment" outgoing="_ykzl69UaEeO18ZGvLO-emQ" inco
  <node xmi:type="uml:OpaqueAction" xmi:id="_ykzlstUaEeO18ZGvLO-emQ" name="Try another time" outgoing="_ykzl7tUaEeO18ZGvLO-emQ" incomi
  <node xmi:type="uml:OpaqueAction" xmi:id="_ykzls9UaEeO18ZGvLO-emQ" name="Checkout" outgoing="_ykzl99UaEeO18ZGvLO-emQ" incoming="_ykz
  <node xmi:type="uml:OpaqueAction" xmi:id="_ykzltNUaEeO18ZGvLO-emQ" name="Display confirmation page" outgoing="_ykzl-tUaEeO18ZGvLO-em
  <node xmi:type="uml:OpaqueAction" xmi:id="_ykzltdUaEeO18ZGvLO-emQ" name="Print or save page" outgoing="_ykzl_dUaEeO18ZGvLO-emQ" inco
  <node xmi:type="uml:OpaqueAction" xmi:id="_ykzlttUaEeO18ZGvLO-emQ" name="Display thank you message" outgoing="_ykzmANUaEeO18ZGvLO-em
  <node xmi:type="uml:DecisionNode" xmi:id="_ykzlt9UaEeO18ZGvLO-emQ" name="Write feedback for website" outgoing="_ykzmA9UaEeO18ZGvLO-e
  <node xmi:type="uml:OpaqueAction" xmi:id="_ykzluNUaEeO18ZGvLO-emQ" name="Write feedback" outgoing="_ykzmBtUaEeO18ZGvLO-emQ" incoming
  <node xmi:type="uml:OpaqueAction" xmi:id="_ykzludUaEeO18ZGvLO-emQ" name="Submit feedback" outgoing="_ykzmDNUaEeO18ZGvLO-emQ" incomin
  <node xmi:type="uml:OpaqueAction" xmi:id="_ykzlutUaEeO18ZGvLO-emQ" name="display visit again message" outgoing="_ykzmD9UaEeO18ZGvLO-
  <edge xmi:type="uml:ControlFlow" xmi:id="_ykzlu9UaEeO18ZGvLO-emQ" name="browse" source="_yky-pdUaEeO18ZGvLO-emQ" target="_yky-ptUaEe
    <guard xmi:type="uml:OpaqueExpression" xmi:id="_ykzlvNUaEeO18ZGvLO-emQ">
      <body></body>
```

5. IMPLEMENTATION

The proposed approach is implemented on a controlled environment using Java language. The proposed algorithms help to generate test sequences and also to prioritize them.

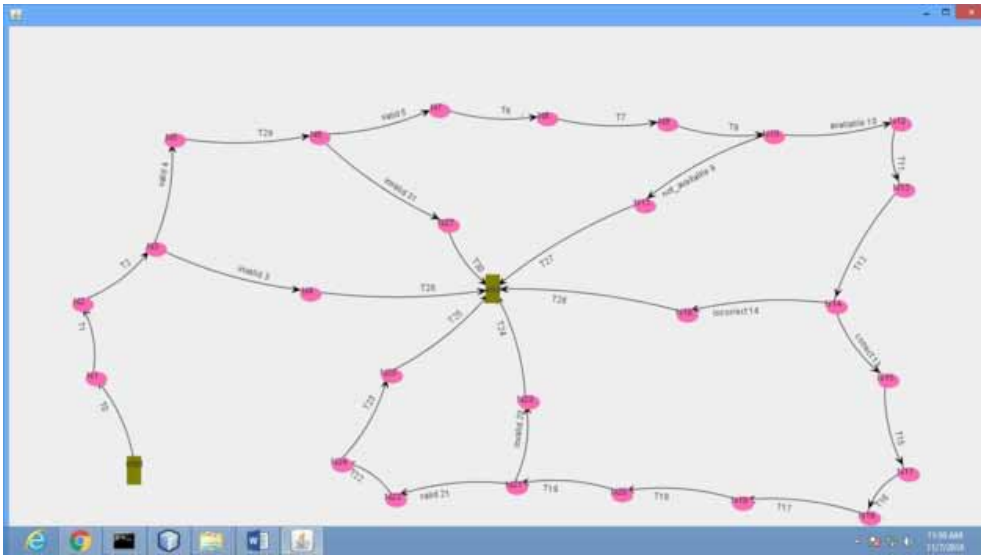
5.1 Setup

This work is implemented with a system having configuration like Windows 10, 64-bit operating system. Apart from this configuration, we have used the following tools to implement the proposed approach.

1. RSA
2. Netbeans8.0
3. Graphviz

Here, a design model is tested by generating test sequences. RSA (Rational Software Architect) is a commercial tool that is used to draw UML diagrams. Thus, we use this to draw the activity diagram of RTRS for a particular scenario. Graphviz is a free tool used for the pictorial representation of the control flow graph. Netbeans8.0 is used to implement ACO algorithm for test sequence optimization and prioritization.

Figure 4. Generated graph through Graphviz



5.2 Materials & Methods

This section describes materials in terms of tools used and methods to demonstrate flow of execution of our proposed approach. Using RSA tool, an activity diagram is constructed for Railway Ticket Reservation System, as shown in Figure 1. The reason behind choosing RSA over other tools is that, it is a commercial tool that supports UML2.x advanced features. Then, XMI code is generated for the constructed activity diagram, which is shown in Figure 3. This figure shows that some node and its random id are generated. Then, a SAX parser is used to parse this obtained XMI code. During this parsing process, information about a possible number of nodes, their ids and edges are extracted and then stored using different linked lists. Then, node types are categorized and stored. Nodes are accessed using Node name and Node id, as shown in Figure 3. Table 2 shows the list of nodes (written as Nd), their symbolic names and their adjacent nodes, etc. The symbolic names Nd_0; Nd_1; Nd_2; … Nd_{27} etc. are named by us, where start node is named as Nd_0 and end node is named as Nd_{26}.

Graph traversal algorithm i.e. DFS is applied to the obtained CFG to generate all possible paths, and the resultant graph is shown in Figure 4. This figure indicates that there are 28 nodes and 32 edges obtained, where each node is represented with its alphanumeric node id and edges with transition id or transition name. Here, Start and End node appear in rectangular box and other intermediate nodes are shown in oval shapes. Usually, DFS is preferred over BFS technique for graph traversal to have depth wise movement such that we can start from Start node up to End node in a single traversal. The list of generated paths and their node names are shown in Table 3. This traversal technique can generate all possible paths, but there may be a repetition of sub-paths. This may cause an increase in cost for traversal. Thus, to reduce the cost of testing, we have considered CC to optimize the number of paths. This path includes at least a new node in the visited path. Then, Cyclomatic Complexity value of this obtained graph is computed and found to be 6 by using Eq. 2. All these obtained six test paths are listed in Table 3 where we can find a new node is being included in the newly generated path than old one. Thus, these test paths are called linearly independent to each other. During prioritization, the net weight of each path is calculated. Based on this final weight of each path, some rank is being assigned to each path. We may assume that rank1 as the highest priority. These results for the ranking process are listed in Table 4. From the obtained result, it is found that test

Table 2. Generated Node (Nd) Information

Serial No	Node Name	Node Symbolic Name	Adjacency List
1	Start	Nd_0	Nd_1
2	passenger.visit the website	Nd_1	Nd_2
3	passenger.Enter user name and Password	Nd_2	Nd_3
4	Authenticate user	Nd_3	Nd_4 Nd_6
5	Railway Authority.user name or pasword is incorrect	Nd_4	End
6	passenger.Enter the source and destination station code	Nd_5	Nd_6
7	verify source and destination station code	Nd_6	Nd_7 Nd_{27}
8	passenger.Enter the date of journey	Nd_7	Nd_8
9	Railway Authority.display list Of train	Nd_8	Nd_9
10	passenger.Select the train	Nd_9	Nd_{10}
11	check seat availability	Nd_{10}	$Nd_{11}Nd_{10}$
12	Railway Authority.display seat Not available	Nd_{11}	End
13	Railway Authority.provide information detail	Nd_{12}	Nd_{13}
14	passenger.submit reservation form	Nd_{13}	Nd_{14}
15	verify form detail	Nd_{14}	Nd_{15} Nd_{16}
16	Railway Authority.process reservation detail	Nd_{15}	Nd_{17}
17	Railway Authority.display information is incorrect	Nd_{16}	End
18	passenger.select payment mode	Nd_{17}	Nd_{18}
19	Railway Authority.processing payment detail	Nd_{18}	Nd_{19}
20	passenger.select online bank	Nd_{19}	Nd_{20}
21	passenger.pay amount	Nd_{20}	Nd_{22} Nd_{23}
22	verify customer account	Nd_{21}	Nd_{22} Nd_{23}
23	Bank.customer does not exist	Nd_{22}	End
24	Bank.debit amount	Nd_{23}	Nd_{24}
25	Railway Authority.print ticket	Nd_{24}	Nd_{25}
26	Railway Authority.update registration form	Nd_{25}	End
27	Railway Authority.source or password is incorrect	Nd_{27}	End
28	End	Nd_{26}	Destination

Table 3. Generated Test Sequence

Test Sequence Number	Generated Test Sequence
Test_Sequence1	Nd_0-> Nd_1->Nd_2->Nd_3->Nd_4->Nd_{26}
Test_Sequence2	Nd_0-> Nd_1->Nd_2->Nd_3->Nd_5->Nd_6->Nd_{27} ->Nd_{26}
Test_Sequence3	Nd_0-> Nd_1->Nd_2->Nd_3->Nd_5->Nd_6->Nd_7 ->Nd_8->Nd_9->Nd_{10}->Nd_{11}->Nd_{26}
Test_Sequence4	Nd_0-> Nd_1->Nd_2->Nd_3->Nd_5->Nd_6->Nd_7 ->Nd_8->Nd_9->Nd_{10}->Nd_{12}->Nd_{13}->Nd_{14}->Nd_{16}->Nd_{26}
Test_Sequence5	Nd_0-> Nd_1->Nd_2->Nd_3->Nd_5->Nd_6->Nd_7 ->Nd_8->Nd_9->Nd_{10}->Nd_{12}->Nd_{13}->Nd_{14}->Nd_{15}->Nd_{17}->Nd_{18}->Nd_{19}->Nd_{20}->Nd_{21}->Nd_{23}->Nd_{24}->Nd_{25}->Nd_{26}
Test_Sequence6	Nd_0-> Nd_1->Nd_2->Nd_3->Nd_5->Nd_6->Nd_7 ->Nd_8->Nd_9->Nd_{10}->Nd_{12}->Nd_{13}->Nd_{14}->Nd_{15}->Nd_{17}->Nd_{18}->Nd_{19}->Nd_{20}->Nd_{21}->Nd_{22}->Nd_{26}

Table 4. Results obtained from TSP_ACO Algorithm

Test Sequence	Total Strength	Number of Nodes	Decision Nodes	Rank
TS 5	56.10	16	4	1
TS 6	53.92	19	4	2
TS 4	38.31	15	3	3
TS 3	29.23	10	2	4
TS 2	18.81	8	2	5
TS 1	11.75	7	2	6

sequence number 5 has the highest priority or rank 1, because it contains the maximum number of decision nodes. Its total strength computed and found to be 56.10. It contains a total of 16 nodes, in which 4 nodes are decision nodes. On the other hand, test sequence number 3, test sequence number 2 and test sequence number 1 all got same number of decision nodes. Test sequence1 got the least strength value and assigned lowest rank that is 6. Hence, ACO is proven to be an effective approach to optimize test sequence generation.

6. RESULTS DISCUSSION

It is a must to assess effectiveness of the proposed approach using some metric. The proposed technique is evaluated on a sample program using Average Percentage of Fault Detected (APFD) metric. Further, our technique is also validated upon implementing on another six case study programs.

6.1 APFD Score Analysis

APFD metric quantities the fault detection rate in the proposed approach (Doerner et al.,2003). APFD is calculated by using Eq.3 and its value lies between 0 to 100%.

$$\text{APFD} = 1 - \left\{\frac{Tf1 + Tf2 + \ldots. + Tfm}{mn}\right\} + \left\{\frac{1}{2n}\right\} \tag{3}$$

where n, m and Tfi represents the number of test sequences, number of faults and position of first test sequence that detects the fault. Let us consider the set of possible five faults as listed in Table 5. These faults are defined after referring to some existing literature (Chen, M. et al.(2008), Panthi et al. (2018)). Table 6 is used to calculate APFD score. This table shows the position of the first test case that detects the fault. Then, the APFD score is calculated and compared, which is shown in Figure 5.

With referring to Table 5 and Table 6, APFD value for non-prioritized test sequences is calculated as follows:

APFD = 1- ((4+3+4+3+2)/5*5)] + [1/(2*5)]=0.46 i.e. APFD%=46.0%.

Similarly, APFD score of the prioritized order (i.e. T5, T6, T4, T3, T2, T1) is calculated as follows:

So, APFD = 1- (3+1+2+2+3/5*5) + (1/(2*5)]=0.66.i.e. APFD%=66.0%.

From Figure 5, it is inferred that the APFD value of the prioritized order is more than the non-prioritized order of the test sequences. Thus, the proposed method validates the effectiveness of our approach in prioritizing test sequences.

6.2 Case Study Analysis

This section tries to compare the obtained results of different case studies like Hotel Management System (HMS), Online Shopping Management System(OSMS), Personal Loan Management System (PLMS), Online Card Management System (OCMS), vehicle Rental Process System (VRPS) and Social Group Site Management System (SGSMS).

Let us extend our discussion of theses case studies in detail on the context of the proposed approach. OSMS system help to manage activity of shopping mall like inventory management, vendor management and goods management. The constructed activity diagram of OSMS system has a total of

Table 5. Fault Description

Fault Item	Description
F1	Incorrect Start Activity
F2	Incorrect End Activity
F3	Missing activity
F4	Stuck at dead activity
F5	Stuck at loop

Table 6. Fault Matrix before Prioritization

Faults\Test Cases	T1	T2	T3	T4	T5
F1				F	F
F2			F	F	
F3				F	F
F4			F		F
F5		F		F	

Figure 5. APFD score comparison

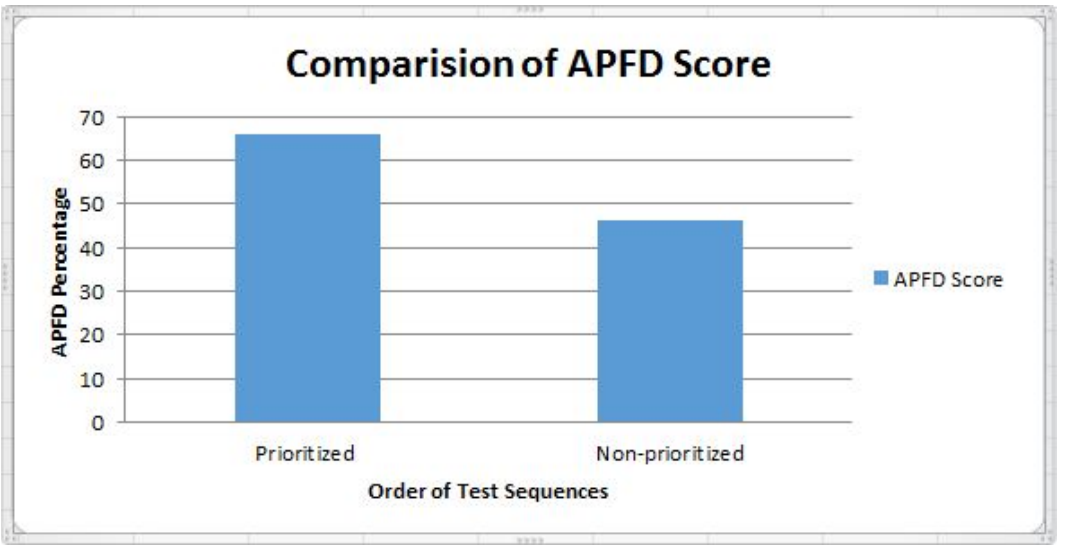


Table 7. Comparison of Case Studies

Sl. No.	Case Study	Total Path	Optimal Path	Reduction Rate(%)
1	OSMS	72	46	36.11
2	HMS	62	35	43.54
3	SGSMS	46	22	52.17
4	PLMS	24	12	50.00
5	OCMS	31	11	64.51
6	VRPS	12	4	66.66

98 nodes that are used to generate a total of 72 number of test paths. Then, the proposed optimization algorithm is used to optimize the number of test path, which results to 46 number of test paths. This evaluation process and the obtained results validates our proposed approach using ACO. Similarly, the rest of the case studies are executed on the same discussed context and the obtained results are shown in Table 7. PLMS (Personal loan management system) helps different banks and financial institution to manage their lending issues. This system has many features such as eKYC Processing, Borrower Profile management, Co-Borrower Loan Reschedule Functionality, NPA Management. Collateral Management etc. For such features, on the context of a particular scenario, the constructed Activity diagram. This diagram has 34 number of nodes that are executed and we found 12 number of optimal test paths from total of 24 number of test paths. Like this, we have experimented on 6 different case study based on the real life software products and the experimental evaluation has achieved an average of 52.16% reduction of test paths. Table 7 shows a comparison of different case study based on total number of test paths and optimized test paths obtained using our proposed approach.

7. DISCUSSION

The proposed work can be compared and analyzed with the existing ACO technique or other optimization technique such as Firefly, PSO (Particle Swarm Optimization) etc. Sharma et al. (2011) had used ACO to generate test sequences. But, their approach failed to get non- redundant paths. They had not discussed the regression testing cost. Also, they did not emphasize on decision nodes, fork node and join node etc. But, our proposed approach overcomes all these limitations.

Bhattacherjee et al., (2018) tested the input programs using an ACO algorithm to generate test sequences. But, their result had many redundant test sequences. It was found that many number of transitions were done to cover all possible paths. Srivastava et al., (2009) had used Firefly algorithm to optimize number of test sequences. They took UML state transition diagram to find test paths. Also, both these approaches did not use complex nodes such as decision, guard condition nodes, fork and join nodes.

Ansari et al. (2016) proposed an optimal way to reduce regression cost, time and uncover maximum faults for test prioritization using ACO. They have used fault matrix and pheromone matrix to optimize multiple objectives. But, they have not discussed about redundant paths, which are generated during testing. This limitation is overcome in our approach by considering cyclomatic complexity of the program under test.

Marijan et al. (2013) used ACO technique to prioritize test cases based on three parameters that are fault, its severity level and time. They have assumed ten number of faults and their severity level like minor, normal and major. They have used APFD metric to measure the performance of their approach.

Biswas et al., (2015) proposed ACO based algorithm to generate an optimal path. This approach was used to generate test sequence within the data domain. Their approach guaranteed for full coverage with less redundant nodes. They had used cyclomatic complexity of the module under test for redundancy removal.

Suri et al., (2011) analyzed the regression test prioritization technique in time constraint environment using sample programs. Their experimental results claimed several facts such as i) significant reduction in test

suite size. ii) reduction in the execution time iii) achieved very high correctness for most of the test programs iv) discovered more number of faults earlier. But, in our proposed work, we consider test path for prioritization with covering these constraints during testing.

Srivastava et al., (2009) used ACO technique for optimal path generation. Their method focused on generation of test paths, equal to Cyclomatic Complexity (CC) of the input program. They guaranteed complete path coverage. Their proposed algorithm is able to select a path that covers maximum possible nodes in a single visit. Thus, this removed redundancy and achieved better path coverage. Our proposed method does same task along with an extra work for prioritizes the generated test paths.

8. THREATS TO VALIDITY

To ensure and maintain the quality of testing the validity to threads need to be discussed. This section highlights some threats to validate such as construct validity, internal validity and external validity for the proposed approach. Here, threats are analyzed and narrowed down by using the validity category.

Our work has some internal threats to be taken care during execution. Internal threats are related to the assumptions made about the validity of the parameters. Such threats can be narrow down by considering some simple points during workout. One such parameter is "node weight" consideration based on its importance in making a test path (Panthi, V. (2018). During implementation, the weight values of the nodes are static and based on user assumption.

The external validity targets to perform generality analysis and draw inferences on meeting requirements. This includes reasonable resources utilization. The resources used in the system are Core i3 processor and Windows 10 with a 64-bit operating system. We have used the RSA7.0 version to draw the activity diagram and Netbeans8.0 for implementing the proposed model. The effect of this threat, such as project selection for our experiments and selecting activity diagram on a particular scenario (Touseef, M., (2015), Rhmann, W., (2015)). We have considered an activity diagram for a railway reservation system for a particular scenario not for complete system. So, for a different scenario, the number of activities, number of objects, number of guards conditions may change, which may change the whole result.

The construct validity are the threats related to parameters used towards measuring accuracy and performance of the model. We have tuned the input parameters of ACO algorithms to get the best performance depending on the execution platform. The measurement metric such as APFD is already been widely accepted and used in many literatures (Panthi, V. (2018), Srivastava, P. R. (2010)). One more threat is "Pheromone Evaporation" rate which is considered as 10%. But, it differs in case of real life ants. Here, it is considered after referring to existing literature (Singh, Y. (2010), Dorigo, M. (2006)). Hence, we also use the same value to maintain the performance level.

9. CONCLUSION AND FUTURE SCOPE

This approach finds optimal test sequence using ACO algorithm. This algorithm saves cost by covering minimum number of non-redundant test sequences and thus performs better. These test sequences are optimized in terms of net strength of each path. This is validated from the obtained result and shown in Table 4. From the results Table 4, it is observed that test sequences having higher priorities, holds more decision nodes. It also removes the redundant test sequences to get an optimal test sequence.

In the future, readers may try to use different UML diagrams as input to generate and prioritize test sequences. Another future scope is to use other meta-heuristic algorithms for better prioritization. Another future work, researchers may propose a new intermediate model to make the process faster.

FUNDING AGENCY

The publisher has waived the Open Access Processing fee for this article.

REFERENCES

Ahmed, S. U., Sahare, S. A., & Ahmed, A. (2013). Automatic test case generation using collaboration UML diagrams. *World Journal of Science and Technology, 2*.

Ansari, A., Khan, A., Khan, A., & Mukadam, K. (2016). Optimized regression test using test case prioritization. *Procedia Computer Science*, *79*, 152–160. doi:10.1016/j.procs.2016.03.020

Bhattacharjee, G., & Dash, S. (2018). Test path prioritization from uml activity diagram using a hybridized approach. *International Journal of Knowledge-Based Organizations*, *8*(1), 83–96. doi:10.4018/IJKBO.2018010106

Biswas, S., Kaiser, M. S., & Mamun, S. A. (2015, May). Applying ant colony optimization in software testing to generate prioritized optimal path and test data. In *2015 International Conference on Electrical Engineering and Information Communication Technology (ICEEICT)* (pp. 1-6). IEEE. doi:10.1109/ICEEICT.2015.7307500

Chandler, R., Lam, C. P., & Li, H. (2005, December). AD2US: An automated approach to generating usage scenarios from UML activity diagrams. In *12th Asia-Pacific Software Engineering Conference (APSEC'05)* (pp. 8-pp). IEEE. doi:10.1109/APSEC.2005.25

Chen, M., Mishra, P., & Kalita, D. (2008, May). Coverage-driven automatic test generation for UML activity diagrams. In *Proceedings of the 18th ACM Great Lakes symposium on VLSI* (pp. 139-142). doi:10.1145/1366110.1366145

Chen, Y., Probert, R. L., & Ural, H. (2007, July). Model-based regression test suite generation using dependence analysis. In *Proceedings of the 3rd international workshop on Advances in model-based testing* (pp. 54-62). doi:10.1145/1291535.1291541

Doerner, K., & Gutjahr, W. J. (2003, July). Extracting test sequences from a Markov software usage model by ACO. In *Genetic and Evolutionary Computation Conference* (pp. 2465-2476). Springer. doi:10.1007/3-540-45110-2_150

Dorigo, M., Maniezzo, V., & Colorni, A. (1991). *Positive feedback as a search strategy*. Academic Press.

Dorigo, M., & Socha, K. (2006). An introduction to ant colony optimization. Handbook of Metaheuristics, 26.

Elbaum, S., Rothermel, G., Kanduri, S., & Malishevsky, A. G. (2004). Selecting a cost-effective test case prioritization technique. *Software Quality Journal*, *12*(3), 185–210. doi:10.1023/B:SQJO.0000034708.84524.22

Farooq, U., Lam, C. P., & Li, H. (2008, March). Towards automated test sequence generation. In *19th Australian Conference on Software Engineering (aswec 2008)* (pp. 441-450). IEEE. doi:10.1109/ASWEC.2008.4483233

IBM. (2018). https://www.ibm.com/developerworks/rational/

Jena, A. K., Swain, S. K., & Mohapatra, D. P. (2014, February). A novel approach for test case generation from UML activity diagram. In *2014 International Conference on Issues and Challenges in Intelligent Computing Techniques (ICICT)* (pp. 621-629). IEEE. doi:10.1109/ICICICT.2014.6781352

Kaur, A., & Goyal, S. (2011). A genetic algorithm for regression test case prioritization using code coverage. *International Journal on Computer Science and Engineering*, *3*(5), 1839–1847.

Kumar, M. S., & Srinivas, P. (2016). An ant colony algorithm to prioritize the regression test cases of object-oriented programs. *Indian Journal of Science and Technology*, 9.

Kumar, S., & Ranjan, P. (2017). ACO based test case prioritization for fault detection in maintenance phase. *International Journal of Applied Engineering Research*, *12*(16), 5578–5586.

Li, H., & Lam, C. P. (2004). Software Test Data Generation using Ant Colony Optimization. *International conference on computational intelligence*, 1-4.

Lv, X. W., Huang, S., Hui, Z. W., & Ji, H. J. (2018). Test cases generation for multiple paths based on PSO algorithm with metamorphic relations. *IET Software*, *12*(4), 306–317. doi:10.1049/iet-sen.2017.0260

Mann, M. (2015). Generating and prioritizing optimal paths using ant colony optimization. *Computational Ecology and Software*, *5*(1), 1.

Marijan, D., Gotlieb, A., & Sen, S. (2013, September). Test case prioritization for continuous regression testing: An industrial case study. In *2013 IEEE International Conference on Software Maintenance* (pp. 540-543). IEEE. doi:10.1109/ICSM.2013.91

Panthi, V., & Mohapatra, D. P. (2018). Firefly optimization technique based test scenario generation and prioritization. *Journal of Applied Research and Technology*, *16*(6), 466–483. doi:10.22201/icat.16656423.2018.16.6.745

Rhmann, W., Zaidi, T., & Saxena, V. (2015). Use of genetic approach for test case prioritization from UML activity diagram. *International Journal of Computers and Applications*, *115*(4).

Rothermel, G., Untch, R. H., Chu, C., & Harrold, M. J. (2001). Prioritizing test cases for regression testing. *IEEE Transactions on Software Engineering*, *27*(10), 929–948. doi:10.1109/32.962562

Sabharwal, S., Sibal, R., & Sharma, C. (2010, September). Prioritization of test case scenarios derived from activity diagram using genetic algorithm. In *2010 International Conference on Computer and Communication Technology (ICCCT)* (pp. 481-485). IEEE. doi:10.1109/ICCCT.2010.5640479

Sharma, B., Girdhar, I., Taneja, M., Basia, P., Vadla, S., & Srivastava, P. R. (2011, December). Software coverage: a testing approach through ant colony optimization. In *International Conference on Swarm, Evolutionary, and Memetic Computing* (pp. 618-625). Springer. doi:10.1007/978-3-642-27172-4_73

Shro & France. (1997). Towards a formalization of uml class structures in z. *Proceedings Twenty-First Annual International Computer Software and Applications Conference (COMPSAC'97)*, 646-651.

Singh, Y., Kaur, A., & Suri, B. (2010). Test case prioritization using ant colony optimization. *Software Engineering Notes*, *35*(4), 1–7. doi:10.1145/1811226.1811238

Srivastava, P. R. (2010). Automatic Test Sequence Generation for State Transition Testing via Ant Colony Optimization. In Evolutionary Computation and Optimization Algorithms in Software Engineering: Applications and Techniques (pp. 161-183). IGI Global. doi:10.4018/978-1-61520-809-8.ch009

Srivastava, P. R., Baby, K. M., & Raghurama, G. (2009, January). An approach of optimal path generation using ant colony optimization. In *TENCON 2009-2009 IEEE Region 10 Conference* (pp. 1–6). IEEE. doi:10.1109/TENCON.2009.5396088

Suri, B., & Singhal, S. (2011). Analyzing test case selection & prioritization using aco. *Software Engineering Notes*, *36*(6), 1–5. doi:10.1145/2047414.2047431

Suri, B., & Singhal, S. (2011). Implementing ant colony optimization for test case selection and prioritization. *International Journal on Computer Science and Engineering*, *3*(5), 1924–1932.

Touseef, M., Butt, N. A., Hussain, A., & Nadeem, A. (2015). Testing from UML design using activity diagram: A comparison of techniques. *International Journal of Computers and Applications*, *975*, 8887.

Wong, W. E., Horgan, J. R., London, S., & Agrawal, H. (1997, November). A study of effective regression testing in practice. In *Proceedings of the Eighth International Symposium On Software Reliability Engineering* (pp. 264-274). IEEE. doi:10.1109/ISSRE.1997.630875

XML-DEV. (2001). *Sax2*. https://www.saxproject.org/

Yang, S., Man, T., & Xu, J. (2014). Improved ant algorithms for software testing cases generation. *The Scientific World Journal*. doi:10.1155/2014/392309 PMID:24883391

Zhou, K., Wang, X., Hou, G., Wang, J., & Ai, S. (2012). Software Reliability Test Based on Markov Usage Model. *JSW*, *7*(9), 2061–2068. doi:10.4304/jsw.7.9.2061-2068