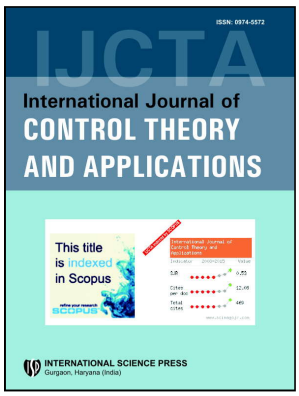

International Journal of Control Theory and Applications

ISSN : 0974-5572

Volume 10 • Number 19 • 2017

Generation and Optimization of Test Paths Using Modified ACO

Nisha Rathee[1] and Rajender Singh Chhillar[2]

[1] *Department of Information Technology, Indira Gandhi Delhi Technical University For Women, Delhi, India.*
[2] *Department of Computer Science and Application, M.D.U Rohtak, Haryana , India.*

Abstract: Model based testing has been the most preferred technique for generation of test cases by software practitioners. It helps in early detection of faults during the design phase, thus reducing development and testing cost. Various testing techniques are available for the generation of test paths using UML diagrams, but finding the optimized path is a challenge. Thus, optimization of test path at an early stage is required to find the best test suite which covers maximum faults with minimum time and minimum redundancy. In this paper, we have proposed an approach for the generation and optimization of test paths by updating the basic features of ACO algorithm using the concept of Backward Slicing and the basic properties of Graph Theory. Priority is set according to the proposed approach and test paths with highest priority are scheduled first. The Modified ACO algorithm has been shown to reduce the redundancy in the test paths, thus reducing the cost, time and effort for the generation of test cases. The modified ACO algorithm has been applied on a UML Activity diagram using the case study of a Library management system.

Keywords: Software testing, Cyclomatic Complexity, Automatic test path generation, Ant Colony Optimization, UML Diagram, Activity Diagram.

1. INTRODUCTION

Due to the extensive use of software designed systems and their increasing complexity, emphasis has been given on object oriented software design for the efficient development of software systems. The most difficult and most important component of software development life cycle is software testing [1,16,20]. The activity of software testing has become more cumbersome, time consuming and costly because of the increasing complexity and scope of software based systems. To cope with these complications, UML model-based testing approach for structural software testing has evolved. In model- based testing technique the test cases are derived from a model that describes the functionality of software systems. Unified Modeling Language (UML) is the standard modeling language for describing the features of a system visually and helps in understanding the behavior of the system clearly. Unified modeling language encompasses a diagram suite to represent different aspects of the system. UML transition sequences serve as a blueprint for the software systems [3,17]. The adaptation of UML specifications for testing purposes helps in early identification of faults and therefore minimizing the cost of testing efforts at later stages of the SDLC.

UML model based testing is well-liked by software developers for consistency and modelling of object oriented software systems [2]. Generation of test paths and corresponding test cases during the design phase of the Software Development Life Cycle (SDLC) enhances the confidence of a developer by detecting faults and

errors at an early stage in the development of software. Though, derivation of test cases from UML diagrams is very complex task. The size of test case suite increases with the size and complexity of software systems [4]. Thus, exhaustive testing covering the complete test suite becomes impossible. We should be able to remove the redundant test cases and also those which are not feasible. This leads to the problem of automatic generation and optimization of test paths. Till now, various testing techniques have been proposed and implemented for the automatic test paths generation using UML diagrams such as genetic algorithms [8], Ant Colony Optimization [6,7], Tabu search[8] based method, Cuckoo search [5] based methods and many more. All of these algorithms have been implemented successfully for the automatic generation of test paths, but the generation of adequate amount of test paths with the minimum number of redundant nodes from UML diagrams needs suitable techniques.

This paper presents an approach using Modified Ant Colony Optimization algorithm for the automatic generation of test paths with the minimum number of redundant nodes. The proposed approach deals with the case study of library management system using UML activity diagram. UML activity diagram is used for describing the functionality of the software system by representing the sequence of actions with the help of parallel and conditional activities [2,3].

1.1. Related Concepts

Ant Colony Optimization is a population based meta-heuristic technique that can be used to discover solutions for optimization problems [18]. ACO takes motivation from the tracking behavior of the ant species and their enhanced abilities such as memory of past actions and knowledge about the distance to other locations [18]. These ants while travelling on the paths leave a chemical substance known as pheromone. This pheromone attracts other ants, thus reinforcing the existing path and is also evaporated with time. The basic idea behind ACO is that shorter paths are better since the ants can travel these paths faster and the amount of pheromone would be more on these paths [9]. Thus, we estimate probability of each and every path with the help of which we assign priority to these paths [10]. The basic ant colony optimization algorithm depends upon the following factors [7]:

I. Feasibility of Path (Fij): It contains all the edges which are connected from node i to node j [16].
II. Pheromone value (τ [i,j]): It is the chemical substance which is released by an ant [16]. It allows an ant to form a conclusion in prospect.in the search of food. It keeps a trace of the path between two nodes. The value of pheromone is updated whenever an edge is traversed.
III. Heuristic value (η [i,j]): It indicates the visibility of a path for an ant at current vertex i to j. Visited Status (Vs): It shows the status of all nodes traversed by any ant for any state i.
IV. Probability: Probability value of the path depends upon two factors :(i) pheromone value τ [i, j] and heuristic information η [i,j] (p) of path for ant p [15].The choice of edge to be selected next in the path is depends upon the probabilistic value of the edge from node 'i' to node 'j', for an ant p.

We have also used the concept of cyclomatic complexity to identify the extreme number of independent paths in the input graph. Cyclomatic Complexity is a very popular technique proposed by McCabe in 1976 and is widely used for finding the maximum number of independent paths in a graph [11]. It is denoted by V(G). Here, V stands for the Cyclomatic number in graph theory and G signifies the complexity function of the graph [12]. V (G) can be calculated using the following formula:

$$V(G) = E-N+2 \quad (1)$$

Where 'E' is the total number of edges and 'N' is total number of nodes in the graph 'G' [21].

This paper proposes and presents a modified Ant Colony Optimization approach with the help of a UML activity diagram. This will enable us to prioritize the feasible paths on the basis of strength assigned and thus discard the less feasible test cases. This paper is described as follows: Section II represents an outline of the related work done in this field. In Section III, the proposed approach is discussed along with the calculations and observations. Section IV describes the conclusion of the paper and gives an outline of our future work.

2. RELATED WORK

Several algorithms have been proposed for the optimization of test paths and corresponding test cases.

In [7], Srivastava et.al have used the basic concept of Ant Colony Optimization algorithm for the generation of optimal test paths. After transforming the source code into a control flow graph, the number of feasible paths has been calculated using the simple principle of ACO. Calculated test paths have been verified using McCabe's cyclomatic complexity. This paper proposes a model for basis path testing technique using ant colony optimization algorithm. The proposed approach provides complete branch coverage for the generation of test cases.

Liping Li et. al. [13] proposed an extenics-based approach for the automatic generation of test cases using UML Activity Diagram. In the paper, they have proposed a Euler Circuit algorithm for the automatic generation of test cases. The Activity Diagram is transformed into a Euler Circuit by formalizing it using the concepts of n-dimensional matter elements. They have mainly focused on transition coverage between the activities in Euler Circuit algorithm for generation of test cases. This helps in a reduction in fault detection therefore reduces testing time and improves the quality of test cases.

Philip Samuel and Mall [14] have proposed a technique for the generation of test cases from UML Activity Diagrams using the concept of dynamic slicing and edge marking. A flow dependency graph (FDG) is formed from the UML Activity Diagram for the generation of dynamic slices. The transformation from one state of activity to another is represented by a corresponding node in FDG. Each and every edge of FDG is noted as 'stable' or 'unstable'. For each node in FDG, a dynamic slicing value for each conditional edge between activities (nodes) has been generated using the slicing criteria. Then by using function minimization method, test case data is created with respect to each and every generated sliced value. The proposed approach automatically generates the test cases corresponding to the sliced conditional value by using the case study of an ATM withdrawal system.

Chengying Mao et.al [9] has proposed an algorithm TDG_ACO by combining the features of Test Driver and ACO for test case generation with maximum branch coverage. The basic ACO algorithm has been redefined by reforming three rules: local transfer rule, pheromone update rule and global transfer rule. Five real world problems are used for the implementation and validation of the practicability and efficiency of the proposed algorithm. Results show that the updated algorithm outperforms the existing technologies.

Saurabh Srivastava et.al [10] has presented an extension of the Ant Colony Optimization which can be helpful in providing a better prioritization of test cases. It uses a popular way of structural testing known as Control Flow testing. A Control Flow Diagram is thus derived from the program source code and the independent paths are prioritized. The proposed approach allows the tester to find the probability of each path and then prioritize according to the calculated values.

3. PROPOSED METHODOLGY

This section illustrates the details of our proposed methodology for path prioritization by applying a modified ACO approach on UML activity diagram using the case study of a library management system. In our work, we have used behavioral UML models which are considered as a de facto standard in software industry to represent the requirements and dynamic nature of the system. The proposed methodology converts the UML Activity Diagram in to an Activity Diagram Graph with the help of the Activity Diagram Table. The modified ACO approach uses the concept of backward slicing for getting a clear visibility of an ant towards the food. Backward Slice of an ant calculates the number of nodes to be traversed between the current node and the end node in search of food. Cyclomatic complexity has been calculated to identify the maximum number of independent paths in the graph [19]. It is very difficult to generate test cases for all the test paths in graph. Therefore, optimization of test paths with minimum redundant nodes and maximum code coverage is required for the resourceful generation of test cases. The proposed approach is used for the generation and optimization of

independent test paths with minimum redundancy. We consider the scenarios, beginning with the start node, traversing through all the intermediate nodes, up to the end node. The steps involved are as follows:

i. Consider the case study of a library management system.

ii. Create the activity diagram for the system using Rational Rose suite as shown in figure 1.

iii. Generate an Activity Diagram Table (ADT) from the activity diagram. The table gives a symbolic representation of all the activities. Symbolic information of all the nodes is shown in Table 1.

iv. Generate an activity diagram graph (ADG) with the help of ADT as shown in figure 2. The ADG represents the activities as nodes and flow between activities as edges of the graph.

v. Calculate backward slice of each node in the graph as shown in Table 2.

vi. Calculate the cyclomatic complexity (CC) of the graph.

vii. Generation and optimization of test paths using the modified ACO algorithm.

Table 1
Symbolic representation of Nodes

Activity Name	*SymbolNumber*	*Activity Name*	*SymbolNumber*
Start node	1	Check numberof books issued to member	9
Enquiry aboutBooks	2	Book notIssued	11
Checkavailability ofbooks	3	Issue Book	12
DecisionNodes	4,7,10	Add member,book & Issue detail	13
Books notAvailable	5	Update bookStatus	14
ValidateMember	6	End Node	15
RegisterMember	8		

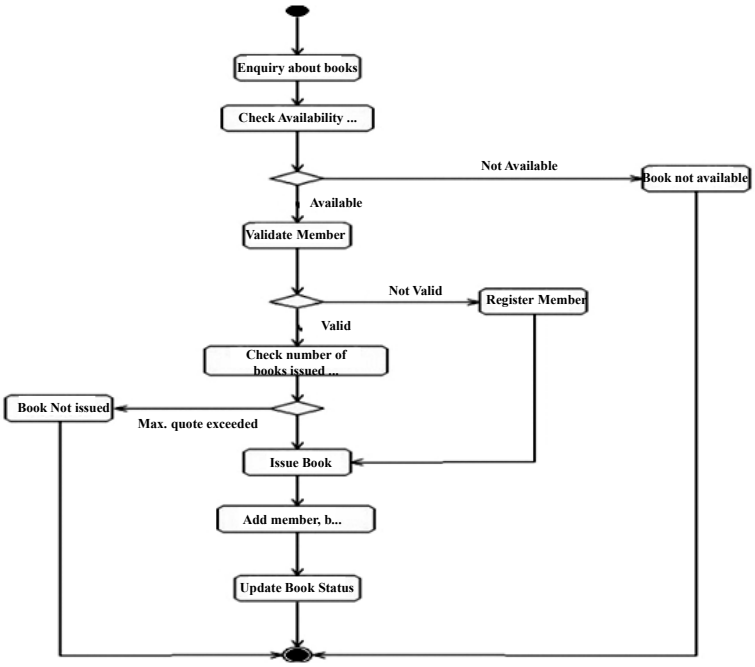

Figure 1: Activity diagram of library Management System

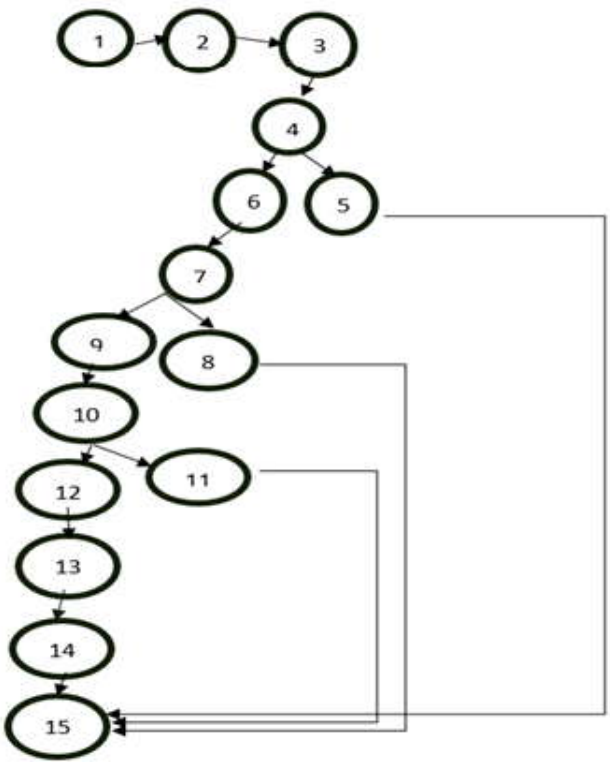

Figure 2: Activity Diagram Graph of library Management System

Table 2
Cost of Backward Slice of nodes

Node	BackwardSlice	Cost of BackwardSlice
1	1,2,3,4,5,6,7,8,9,10,11,12,13,14,15	15
2	2,3,4,5,6,7,8,9,10,11,12,13,14,15	14
3	3,4,5,6,7,8,9,10,11,12,13,14,15	13
4	4,5,6,7,8,9,10,11,12,13,14,15	12
5	5,15	2
6	6,7,8,9,10,11,12,13,14,15	10
7	7,8,9,10,11,12,13,14,15	9
8	8,15	2
9	9,10,11,12,13,14,15	7
10	10,11,12,13,14,15	6
11	11,15	2
12	12,13,14,15	4
13	13,14,15	3
14	14,15	2
15	15	1

3.1. Approach of the Algorithm

Algorithm_Modified_ACO (start, end_node, G):

1. For all v in V[G] do
2. Visited[v]=false;
3. End for.
4. Initialize stack S // Empty Stack
5. Initialize strength [start]=0;
6. For all v in V[G] do
7. Calculate BackwardSlice (v) // Cost of BackwardSlice of each vertex.
8. End for
9. Push (S, start); // Push the start vertex to the Stack s.
10. Weight[start] = 1.
11. Initialize count= cyclomatic complexity of Graph G
12. While count>0 && not empty(S)
13. do int i=pop(S);
14. If visited[i]==0 then set visited [i] = 1;
15. Evaluate feasibility set F[i] for current vertex i.
16. If F[i] ==0, then goto step 23.
17. Else Calculate weight of each node which are associated in the F[i] for node i.
    W[i,j]= Number of nodes incoming to node 'i' + Number of nodes outgoing from 'j'.
    Set weight of node 'j' associated to node 'i' as W[j] = W[i] + W[i,j];
18. Update pheromone $\tau[i,j] = (W[i] + W[i,j])/ W[i]$
19. Update visibility $\eta[i,j] = W[i] + W [i,j]$;
20. If there is single vertex in F[i] i.e only node 'j' then push (S,j) and go to step 22.
    Else calculate probability and cost of traversing (CTR) of every non- visited node 'j' in F[i] ;
    Pij $=(\tau[i,j]*\eta[i,j])/ \sum_k i (\tau [i,j]*\eta [i,j])$; // Calculate probability
    CTR = Pij * BackwardSlice(j); // Calculate cost of traversing.
21. push vertex in Stack S according to highest cost of traversing values.
    If CTR [i,j] > CTR [i,k] then push(S,j) and then Push(S,k)
    Else if CTR [i,j]= CTR [i,k] then
    a. Check if feasibility set entry equals end_node, then push (S,k) and push (S, end_node).
    b. If F[i] does not contain any end node, then select the path which has next state not visited yet. Push (S, j) and then push (S, k) // If 'k' is not visited yet.
    c. If same status then select any path randomly.
22. Calculate strength for all 'j' nodes in feasibility set of node 'i '.
    Strength[j] = strength[i] + $(\tau [i,j]*\eta[i,j])$
23. If (i != end_node) then goto step 12 ;
    Else update count= count-1 and print the path with its strength value and goto step 12.
24. End While.

3.2. Demonstration of proposed approach

The proposed approach utilizes the basic operations of stack and graph theory. Cost of backward slice of every node is calculated from the Activity Diagram Graph. Cost of backward slice of a node is the number of nodes associated to that node till the end node. It helps ants to identify the number of intermediate nodes between the current stage of the ant and the food. Cost of Traversing 'CTR', as defined in the algorithm above, is used for the optimization of test paths. The cost of traversing is a product of the probability of edges 'i' to 'j' and the cost of backward slice. An ant selects the edge that has the maximum Cost of Traversing, thereby proceeding further towards the food. Strength is calculated for optimization of test paths. The path having highest strength value is the most optimized path.

The algorithm takes the activity diagram graph with the start node and end node as input. In this graph start node and end nodes are '1' and '15' respectively. Initially the visited status of all the nodes in the graph is set to 'zero'. Strength of node'1' is set to zero. Backward slice of each and every node in the graph is calculated. Initially start node is pushed in the stack 'S'. Weight of start node is set to '1'. Total number of vertices in the graph is 15 and total number of edges in the graph is 16. Therefore, cyclomatic complexity of the graph is 17-15+2= 4.

The cyclomatic complexity of the graph is stored in the 'count' variable. While count is greater than zero and the stack is not empty, the proposed approach will pop an element that is node '1' from the stack and change its visited status to '1'. Feasibility set of node '1' is {2} i.e node '2' only. Weight of edge (1,2) is 0+1=1, as total number of nodes incoming to node 1 is 0 and total number of nodes outgoing from 2 is only 1. Set weight of node 2 is W [1] + W [1,2] i.e 1+1=2. Pheromone value τ [1,2] for edge (1,2) is 2 and visibility value η[1,2] is also calculated as 2. As there is only node 2 in feasibility set of node 1 therefore, there is no need to calculate the probability and directly push node 2 in to the stack S. Strength of node 2 is calculated as strength [1] + (τ [1,2]* η[1,2]), i.e 4. As node 1 is not equals to end node therefore, go back to step 12. Now the count is still greater than '0' and stack is not empty therefore it will again pop an element i.e node 2 from the stack. Feasibility [2] = {3}. Weight (2,3) = 2 and W[3]= 2+2=4. τ [2,3] = {(2+2)/ 2}= 2 and η[2,3]= (2+2)=4. As there is single node in feasibility set of node 2 therefore, push node 3 in to stack S. Strength [3]= {4+(2*4)}= 12. As node 2 is not equals to end node therefore again go back to step 12. Now again count is greater than zero and stack is not empty therefore, pop an element i.e node 3 from the stack 'S'.

Feasibility [3] = {4}. W[3,4]= 3 and W[4] = 7. τ [3,4]={(4+3)/4}= 1.75 and η[3,4]=7. Pheromone value of an edge decreases and the visibility of an ant increases as the ant moves towards the food. Here pheromone value decreases to 1.75 and visibility increases to 7 as we move further towards the end node. As there is single node in F[3] therefore, push node 4 to Stack S. Strength[4] = {12+(1.75*4)}=24.25. Node 3 is not equals to end node therefore, again go back to step 12. Now it will again pop an element from the stack S i.e node 4. F[4] = {5,6}. W[4,5] = 2, W[4,6] = 2, W[5]=9, W[6] = 9. τ [4,5]={(7+2)/7}= 1.28 and η[4,5]= 9 , similarly τ [4,6]= 1.28 and η[4,6]= 9.

Now the feasibility set is containing more than one node therefore, we the ant will select higher priority node. Priority of a node is assigned according to the value of Cost of Traversing (CTR). Least CTR valued node is the most prior node. Highest priority is assigned is assigned to least CTR valued node, because this node is more close to the end node. The value of CTR is based upon two parameters, one is the probability of the edge (i,j) and other is backward slice of 'j' node. Here CTR [4,6] > CTR[4,5] therefore , the most prior node is node 5 and it should be pushed on the top of the stack. The ant will select the edge (4,5) for further traversing. Push (S, 6) and then Push(S,5). Strength [5] = 35.77 and Strength [6]= 35.77. Now 4is not equals to end node, therefore, it will again go back to step 12 and pop the node 5 from the stack. F {5} = {15}, W [5, 15] =1 and W [15] =10. τ [5,15]= 1.11 and η[5,15]= 10. As there is single node in feasibility set of node 5 therefore push (S, 15). Strength [15] = 46.9. As node 5 is not equals to end node therefore again go back to step 12 and pop node 15 from the stack S.

Now F{15}= empty, therefore go to step 23. It will check whether the node is end node or not. Now node 15 is the end node, therefore it will update the count parameter by deceasing its value by 1and print the path with its strength value. It will print the path 1: 1,2,3,4,5,15 with its strength value 46.9 and go back to step 12. Now the value of count is 3 and the stack is still not empty as it is containing node 6, therefore, it will pop node 6 from the stack and follow the similar steps as discussed above for rest of the paths.

In above graph, on 'x' axis we are having edges traversed by an ant p during the path 1and on 'y' axis we are having calculated values of 'η' and 'τ' during path 1 . As shown in above graph, the value of pheromone keeps on decreasing and the visibility of an ant towards the food keeps on increasing while traversing on path 1.

The proposed algorithm is implemented on the UML activity diagram using the case study of library management. The algorithm is implemented using C++ language. A strength parameter is calculated at the end of each path. Test paths are optimized using strength parameter. The path with highest strength value is optimized and should be tested at a higher priority. Figure 3, shows the results of the implemented algorithm. Test paths generated using the proposed methodology on the case study of a library management system are shown in table 3.

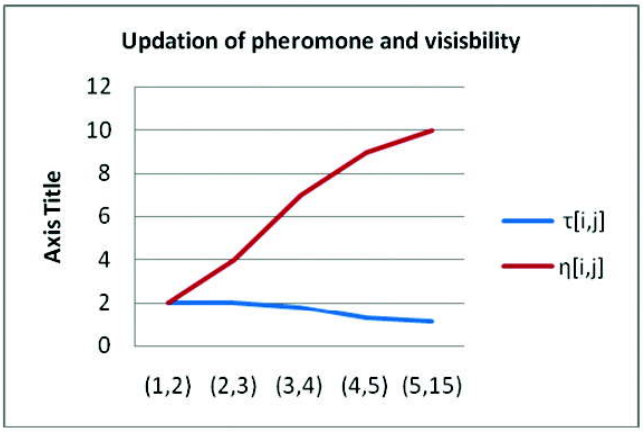

Figure 3: Updation of pheromone and visibility during path 1.

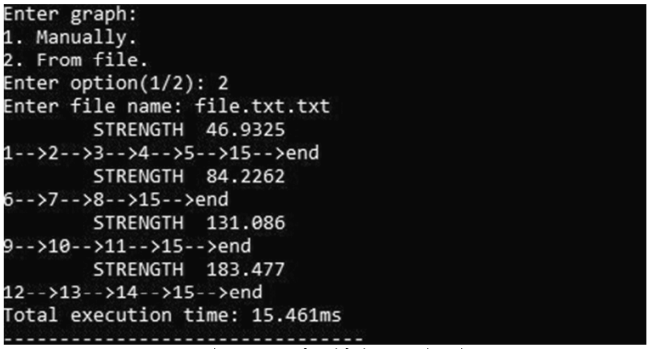

Figure 4: Implementation of modified _ ACO algorithm

Table 3
Test paths generated using the proposed approach

Paths	*Strength*	*priority*
Path 1: 1->2->3->4->5->15->end	46.9325	4
Path 2: 6->7->8->15->end	84.2262	3
Path 3: 9->10->11->15-end	131.086	2
Path 4: 12->13->14->15->end	183.477	1

As shown in table above: Path 4 has highest priority with maximum coverage.

Let us compare this proposed methodology with existing technique as defined in [7]. They presented an approach based upon ACO for the generation of test paths. The existing technique converts the source code into a control flow graph and then implements the ACO algorithm for the generation of optimal test paths. Test paths generated using the existing approach are as follows:

Path1:1,2,3,4,5,15,end
Path2 1,2,3,4,6,7,8,15,end

Path3:1,2,3,4,6,7,9,10,11,15,end
Path4 1,2,3,4,6,7,9,10,12,13,14,15,end

Path 1 generated from the existing approach is same as that generated by our proposed methodology. But other paths generated from their approach have various redundant nodes. 1, 2,3,4,6 and 7 are redundant nodes in paths 2,3 and 4. Therefore, the existing approach will cost more in time due to the generation of test cases for these redundant nodes. While designing test cases for these test paths, each time we have to generate test cases for theses redundant nodes. Therefore, proposed methodology is very useful and fruitful as compared to the existing approach for the generation of test cases as it has minimum redundancy as shown in figure 3.

4. CONCLUSION AND FUTURE WORK

The paper has proposed and implemented a modified approach for the generation and optimization of test paths using the basic concepts of graph theory and ACO algorithm on a UML Activity Diagram.

This approach has also used the concept of cyclomatic complexity and backward slicing for the generation and optimization of test paths. Backward slice of a node can be calculated from the Activity Diagram Graph to find the number of intermediate nodes between the food and the current stage of an ant. The proposed approach generates independent test paths with maximum coverage and minimum redundancy. Test paths generated from the proposed approach have no redundant nodes thus reducing the cost, time and testing efforts for the generation of test cases and also help in early identification of faults in the system software. In future work, we will try to integrate this approach with other evolutionary techniques for the optimization of test paths.

REFERENCES

[1] Ian Sommerville, Software Engineering ,eight Edition, Pearson Edition,2009.

[2] Ajay Kumar Jena, Santosh Kumar Swain, Durga Prasad Mohapatra, "A Novel Approach for Test Case Generation from UML Activity Diagram", IEEE International Conference On Issues and Challenges In Intelligent Computing Techniques (ICICT), pp. 621-629, 2014.

[3] Anbunathan , Anirban Basu, "Dataflow test case generation from UML Class diagrams", Computational Intelligence and Computing Research (ICCIC), 2013 IEEE International Conference ,pp 1-9, 2009

[4] Monalisa Sarma, Debashish Kundu, Rajib Mall, "Automatic Test Case Generation from UML Sequence Diagram", 15th International Conference on Advanced Computing and Communications.

[5] Praveen Ranjan Srivastava, Monica Chis, Suash Deb, Xin-She Yang, " Path optimization for software testing: an intelligent approach using cuckoo search", in proceedings of the 4th Indian International Conference on Artificial Intelligence, IICAI 2011, Bangalore, India, pp. 725–732, ISBN:978-0-9727412-8-6.

[6] Li Huaizhong C. Peng Lam, An ant colony optimization approach to test sequence generation for state based software testing, i n: Proceedings of the Fifth International Conference (IEEE) on Quality Software, QSIC, 2005,pp. 255–264.

[7] P.R. Srivastava, K. Baby, G. Raghurama, "An approach of optimal path generation using ant colony optimization", in proceedings of the TENCON 2009 IEEE Region 10 Conference, Singapore, pp. 1–6.

[8] Yong Chen, Yong Zhong, " Automatic path-oriented test data generation using a multi-population genetic algorithm", in proceedings of the Fourth International Conference (IEEE) on Natural Computation, ICNC, vol. 1, 2008, pp. 566–570.

[9] Chengying Mao, Xinxin Yu, and Jifu Chen, Jinfu Chen "Generating Test Data for Structural Testing based on Ant Colony Optimization", 12th International Conference on Quality Software (IEEE), 2012.

[10] Saurabh Srivastava, Himanshi Raperia, Jastej Badwal, " Extended ACO Algorithm for Path Prioritization", International Journal of Computer Applications (0975-8887) , Volume 67-No.1, April 2013.

[11] Du Qingfeng, Dong Xiao , "An Improved Algorithm for Basis Path Testing", 2011 IEEE

[12] Zhang Zhonglin and Mei Lingxia, "An Improved Method ofAcquiring Basis Path for Software Testing", 5thInternational Conference on Computer Science and Education, ICCSE 2010, pp.1891-1894.

[13] Liping Li, Xingsen Li, Tao He & Jie Xiong. "Extenics-based test case generation for UML Activity Diagram". Proceedings of Elsevier Information Technology and Quantitative Management, pp. 1186 – 1193, 2013.

[14] Philip Samuel, Rajib Mall, "Slicing Based Test Case Generation from UML Activity Diagrams". ACM SIGSOFT Software Engineering Notes, vol. 34, No. 6, 2009.

[15] Bhuvnesh Sharma, IshaGirdhar, Monika Taneja, Pooja Basia, Sangeetha Vadla, and Praveen Ranjan Srivastava, "Software Coverage : A Testing Approach through Ant Colony Optimization", B.K. Panigrahi et al. (Eds.): SEMCCO 2011, Part I, LNCS 7076, pp. 618–625,.© Springer-Verlag Berlin Heidelberg 2011.

[16] Agarwal, Komal, Manish Goyal, and PraveenRanjan Srivastava. "Code coverage usingintelligent water drop (IWD)", International Journal of Bio- Inspired Computation, 2012.

[17] Shiorle, Mahesh, and Rajeev kumar. "UML behavioral model based test case generation: a survey", ACM SIGSOFT Software Engineering Notes, 2013.

[18] Tan, Xiaoyong, and Yongmei Ren. "Study on the Emergency Rescue VRP Based on AntColony Optimization andGeneralized Distance", 2013 Fourth Global Congress onIntelligent Systems.

[19] Vivekanandan, K., T. Megala and P.Chandini. "Automatic generation of basis test path using clonal selection algorithm", International Conference on Information Communication and Embedded Systems(ICICES), 2016.

[20] Agarwal, Komal, Manish Goyal, and Praveen Ranjan Srivastava. "Code coverage using intelligent water drop (IWD)", International Journal of Bio-Inspired Computation, 2012.

[21] Linda M. Laird, M. Carol Brennan, Software Measurement and Estimation: A Practical Approach, John Wiley & Sons, 05-Jun-2006.