A Novel Approach for Test Path Generation and Prioritization of UML Activity Diagrams using Tabu Search Algorithm

Gargi Bhattacharjee, Priya Pati

Abstract— Software testing is highly vital to the development of any software. Testing ensures the quality of software and in turn increases its reliability and robustness. The prime focus lies in minimizing the expenses incurred in testing. To test software, the major problem lies in generation of test cases. Software specifications are the main sources for generating them. Software specifications may be UML diagrams, formal language specification, or natural language description. Testing professionals have now shifted their attention to UML. The software specification used in this paper is UML Activity Diagram. While testing the software, we need to identify all the paths in order to ensure complete testing. Test paths in software can be identified through existing algorithms. But the problem arises when we have to find out the most critical path among them as they are the ones most likely to be executed. In this paper, we have employed a Meta heuristic approach called tabu search to find out the path with highest priority so that it can be tested first. Identifying the most critical path increases, the testing efficiency.

Index Terms— Path Coverage, Software Testing,Tabu Search Algorithm, Test case generation, Test path prioritization and UML Activity Diagram.

1 Introduction

Software engineering is a discipline concerned with all aspects of software right from its inception to its acceptance. Software testing [1] plays a vital role in quality control of the software. Testing aims at detecting errors in the software and is carried out by executing the program on a set of tests and then comparing the actual outputs with the expected outputs. As software testing consumes about 50% of software development effort, test case generation plays a crucial role. Exhaustive testing is not possible because there are no limits on how much we can test. Thus, to limit the process of testing, the concept of testing criteria is used. Satisfying the testing criteria marks an end to testing process. As the complexity and size of software systems grow, more and more time and manpower are required for testing. Manual testing is so labor-intensive and error-prone that it becomes necessary to automate the testing techniques. The testing effort can be categorized under three heads: test case generation, test execution, and test evaluation. Since test case generation is the core of any testing process and the other two depend on it, construction of test cases is more challenging and difficult.

- *Gargi Bhattacharjee pursued her masters in software engineering from BIT Mesra, India. E-mail: gargib.07@gmail.com*
- *Priya Pati is a lecturer in Comp Sc & Engg Dept, NIT Srinagar, India. Email: priya.tulu@gmail.com*

Earlier test cases were designed based on the source code. Their inception started only after the codes had been written. Thus it made the testing process more time consuming. In order to reduce the time consumption, testers started shifting their attention to software specifications. Software specifications are the main sources for generating test cases. Software specification can be in the form of UML models, formal language specification or natural language description. Researchers and testing professionals focused on designing models using UML. Since model based testing can be planned at an early stage of the software development, testing and coding can be done concurrently, thereby producing the test cases in advance and reducing the development time. The early availability of test cases makes the test planning more effective. The Unified Modeling Language (UML) [2], [3] is a standardized, general-purpose modeling language. UML includes a set of graphic notation techniques to create visual models of object-oriented software. UML provides a number of diagrams to describe particular aspects of software artifacts. These diagrams can be classified depending on whether they are intended to describe the structural or behavioral aspects of systems. One such diagram which shows the dynamic nature of the system is the activity diagram. It is simple and easy to understand the flow of logic in the system.

In this paper, we use UML activity diagram as design specification. We have proposed a technique for prioritizing the test paths derived from UML Activity diagram using tabu search algorithm. Thus we are able to find out the most error prone path in the software. The paper is structured in the following way: In next section, we present the related works of activity diagrams used for generating test cases. Section 3, 4, and 5 presents a briefing about the related concepts used, the methodology used in finding out the most critical path and the experimental evaluation of the methodology used respectively. The last section gives an insight about the future aspect of the work.

2 Related work

This section reviews the existing work where activity diagrams are used for generating test cases. Different techniques are considered with different methods to generate test cases. Each of the techniques used has its own advantages and disadvantages.

Chen Mingsong, Qiu Xiaokang, and Li Xuandong [4] used UML activity diagrams as design specifications, and presented an automatic test case generation approach. The approach first randomly generated abundant test cases for a JAVA program under testing. Then, by running the program with the generated test cases, they got the corresponding program execution traces. Last, by comparing these traces with the given activity diagram according to the specific coverage criteria, they got a reduced test case set which met the test adequacy criteria. The approach can also be used to check the consistency between the program execution traces and the behavior of UML activity diagrams. Debasish Kundu and Debasis Samanta [5] augmented the activity diagram with necessary test information. Then they converted the activity diagram into an activity graph. And finally form the activity graph they generated the test cases. Pakinam N. Boghdady et al. [6] proposed an automated approach for generating test cases from activity diagrams. The proposed model introduced an algorithm that automatically creates a table called Activity Dependency Table (ADT), and then uses it to create a directed graph called Activity Dependency Graph (ADG). The ADT is constructed in a detailed form that makes the generated ADG cover all the functionalities in the activity diagram. Finally the ADG with the ADT are used to generate the final test cases. Hyungchoul Kim et al [7] proposed a method to generate test cases from UML activity diagrams that minimizes the number of test cases generated while deriving all practically useful test cases. Their method first built an I/O explicit activity diagram from an ordinary UML activity diagram and then transformed it into a directed graph, from which test cases for the initial activity diagram were derived. This conversion was performed based on the single stimulus principle, which helped avoid the state explosion problem in test generation for a concurrent system. Supaporn Kansomkeat et al [8] in their paper proposed a Condition-Classification Tree Method (CCTM) that extended the Classification-Tree Method by using activity diagrams.

Initially, an activity diagram was analyzed to gather control flow information based on decision points and guard conditions. Then, this information was used to derive Condition-Classification Trees. Finally, the trees were used to generate a test case table and then test cases. Wang Linzhang, et al [9] proposed an approach to generate test cases directly from UML activity diagram using gray-box method. In this approach, test scenarios are directly derived from the activity diagram modeling an operation. Then all the information for test case generation, i.e. input/output sequence and parameters, the constraint conditions and expected object method sequence, was extracted from each test scenario. At last, the possible values of all the input/output parameters were generated by applying category partition method, and test suite could be systematically generated to find the inconsistency between the implementation and the design. A prototype tool named UMLTGF had been developed to support the above process.

Once the test cases were generated, focus laid on optimizing them. Researchers even used meta heuristic approach like genetic algorithm to optimize the generated test cases. Praveen R Srivastava et al. [10] in his paper presented a method for optimization of test cases by identifying the most critical path clusters in a program. They did this by developing a variable length Genetic Algorithm that selected the software path clusters which weighted high in accordance to the criticality of the path. Sangeeta Sabhwal, Ritu Sibal, and Chayanika Sharma [11] considered in their paper a novel approach where the testing efficiency was optimized by applying the genetic algorithm on the test data. For requirement change, a stack based approach for assigning weights to the nodes of the activity diagram was considered. The test paths were generated from the control flow graph which was derived from the activity diagram. In their study they found that the approach used was significant to identify location of the fault in the implementation and thus reduce the testing efforts. A.V.K. Shanthi [12] in her paper implemented genetic algorithm to prioritize the test cases derived from the activity diagram using the activity dependency table, to identify the error prone path in a software construct.

Not much work has been done using tabu search for prioritization of test cases. A.V.K Shanthi et al [13] proposed a methodology based on tabu search algorithm to prioritize the test cases. They used UML Activity Diagram, from it; they constructed the Activity Dependency table (ADT), and then generated the test paths. The test paths were then prioritized by using the Tabu search algorithm.

3 Related Concepts

Tabu Search Algorithm- Tabu search [14], [15] is a meta heuristic local search algorithm that can be used for solving combinatorial optimization problems (problems where an optimal ordering and selection of options is desired). Tabu search uses a local or neighborhood search procedure to iteratively move from one potential solution 'x' to an improved solution x' in the neighborhood of 'x', until some stopping criterion has been satisfied. It carefully explores the neighborhood of each solution as the search progresses. The solutions admitted to the new neighborhood, N*(x), are determined through the use of memory structures. Using these memory structures, the search progresses by iteratively moving from the current solution 'x' to an improved solution x' in N*(x). The method is illustrated with the help of a flow diagram in Fig 1.

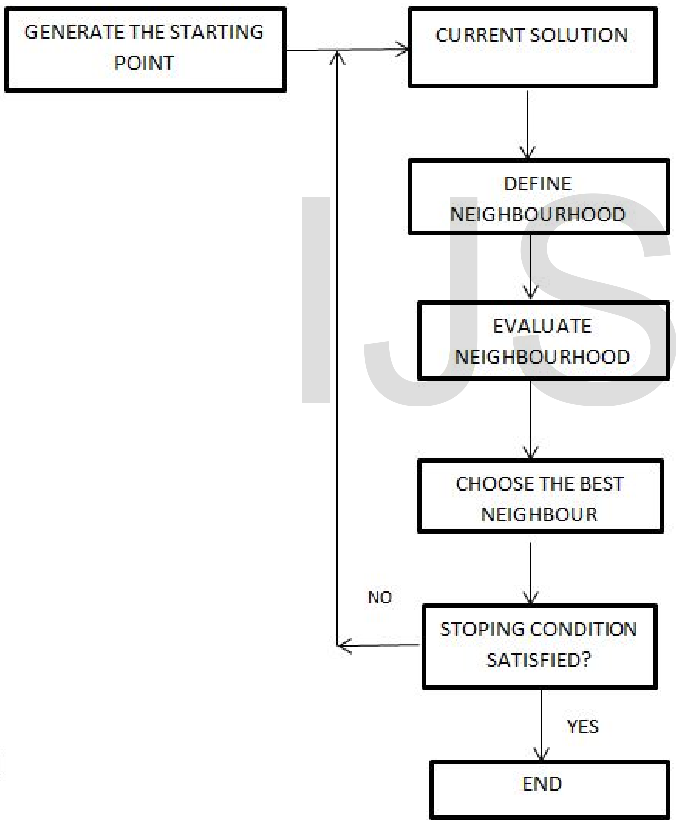

GENERATE THE STARTING POINT
CURRENT SOLUTION
DEFINE NEIGHBOURHOOD
EVALUATE NEIGHBOURHOOD
CHOOSE THE BEST NEIGHBOUR
NO
STOPING CONDITION SATISFIED?
YES
END

Figure 1: Flow diagram for Tabu Search

To evaluate the ***fitness function***, we make use of the Information Flow metrics. IF metrics are applied to the components of system design. Here we have considered the nodes as components. For a node A,

IF (A) = [FAN_IN (A) * FAN_OUT (A)]^2

Where "FAN_IN" is simply a count of the number of other nodes that can call, or pass control, to node A and "FAN OUT" is the number of components that are called by node A. The power component shows the non-linear nature of complexity. Thus the IF of a node gives us its fitness function.

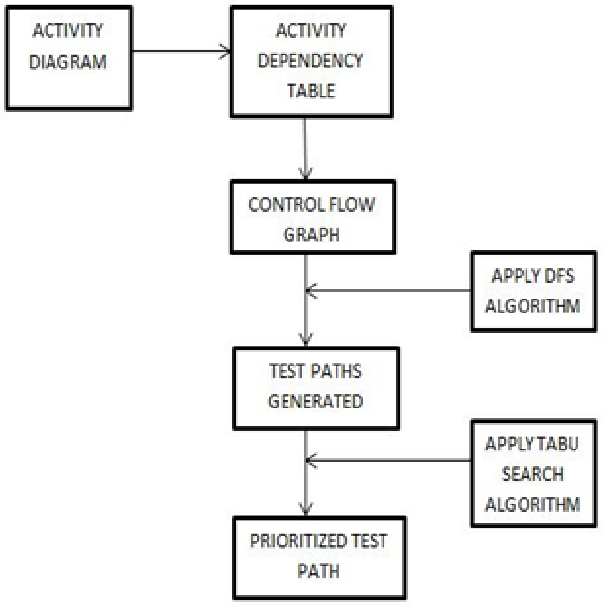
A ***control flow graph*** (CFG) is a representation, using graph notation, of all paths that might be traversed through a program during its execution. Since here we use an activity diagram, the nodes represent actions (activity, method execution) and the arcs indicate the flow of control from one action to another.

In our approach, we consider the path coverage criteria. The path coverage criterion ensures that all the activity paths in the graph have been traversed.

4 Proposed Methodology

This section illustrates the details of our proposed approach for test case generation and prioritization using Tabu Search Algorithm. Initially we create the activity diagram. Based on the Activity Diagram, we construct an Activity Dependency Table (ADT). We construct a CFG based on the ADT. We then apply DFS algorithm to generate the possible test paths and then by applying Tabu Search, we get the prioritized path. Our methodology is demonstrated through the flow diagram in Fig 2.

Figure 2: Flow diagram for Test case generation and prioritization



PROCEDURE

1. Create an Activity Diagram. It can be created by any UML Modeling tool like IBM's Rational Rose or MagicDraw.

2. From the Activity Diagram, generate the Activity Dependency Table. The table clearly shows the dependency among the activities. Thus, with its help, we can construct the control flow graph (CFG).

3. Construct the CFG 'G'.

4. Calculate fitness function *'fit_func'* of each node.

5. Apply DFS algorithm to generate the test paths.

6. To apply tabu search to the newly generated test paths, we create 2 memory structures and name it *'best_path'* and *'next_soln'*. *'best_path'* stores the path of prioritized nodes i.e. it stores the critical path and *'next_soln'* stores the possible nodes in the neighbourhood. We also create a counter variable *'count'* to keep track of the no of solutions of the neighbourhood. We then carry out the following steps:

    I. Visit the root node and store it in *best_path*.

    II. Check the neighbourhood and store it in *next_soln* and store the no. of solutions of the neighbourhood in *count*.

        a. If *count* has 1 solution, copy the node in *next_soln* to *best_path* and break.

        b. If *count* contains more than 1 solution, check for the fitness function *'fit_func'*

            i. If the nodes have unequal fitness values, move to the node with highest fitness value. Copy the node in *next_soln* to *best_path* and break.

            ii. If the nodes have equal fitness value,

```
  for each node
    while (count >=1)
          if (count = = 1)
          Move to the next node in next_soln
           else
Store the nodes traversed in best_path
                    Break;
               End if
          End while
      End for
```

    III. Move to the next node in the neighbourhood.

    IV. Repeat the process until it reaches the leaf node.

    V. *best_path* gives the prioritized path.

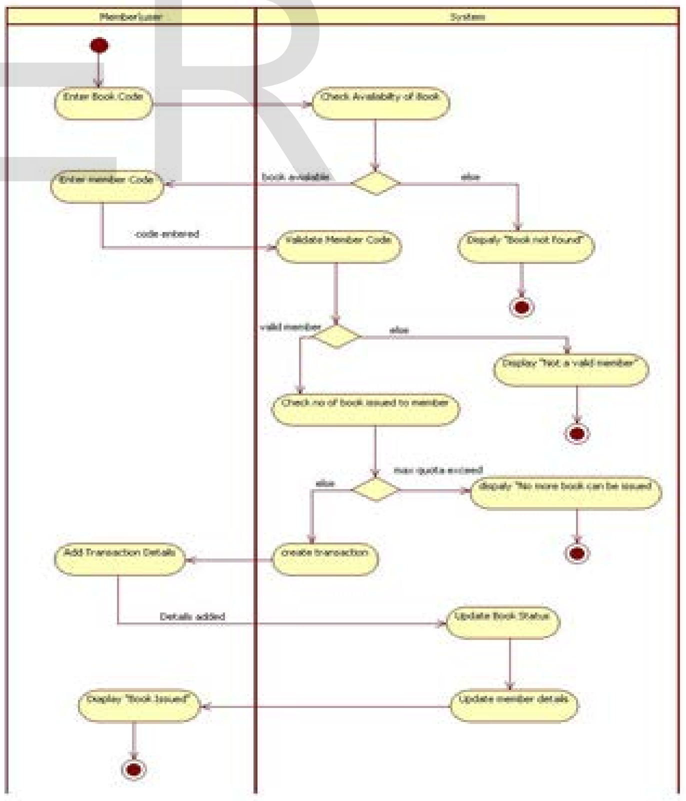
7. The generated test paths are checked against cyclomatic complexity.

5 EXPERIMENTAL RESULTS

CASE STUDY: LIBRARY MANAGEMENT SYSTEM

1. We explain our proposed method by taking the activity diagram (Fig 3) of Library Management System. This diagram was made using IBM's Rational Rose.

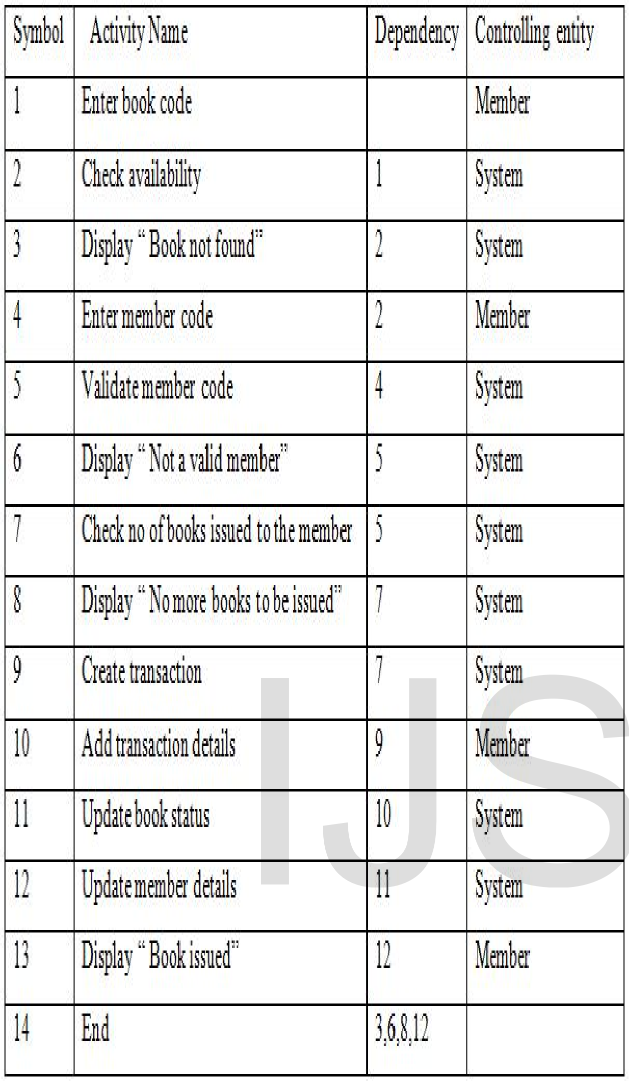
Figure 3: Activity diagram of library management



system

2. With the help of the activity diagram, we generate a dependency table (table 1).

Table 1: ADT of library management system



Symbol	Activity Name	Dependency	Controlling entity
1	Enter book code		Member
2	Check availability	1	System
3	Display " Book not found"	2	System
4	Enter member code	2	Member
5	Validate member code	4	System
6	Display " Not a valid member"	5	System
7	Check no of books issued to the member	5	System
8	Display " No more books to be issued"	7	System
9	Create transaction	7	System
10	Add transaction details	9	Member
11	Update book status	10	System
12	Update member details	11	System
13	Display " Book issued"	12	Member
14	End	3,6,8,12	

The table consists of the following fields: - Symbol, Activity Name, Dependency and the Controlling Entity. *Symbol* is used to represent the activities in the diagram; *Activity Name* suggests the name of the activity; *Dependency* indicates the dependencies of the activity on the previous activities and *Controlling Entity* shows the entity which controls the corresponding activity.

3. The CFG 'G' (fig 4) is generated from the dependency table. With the help of the dependencies, we map the nodes in their respective sequence.

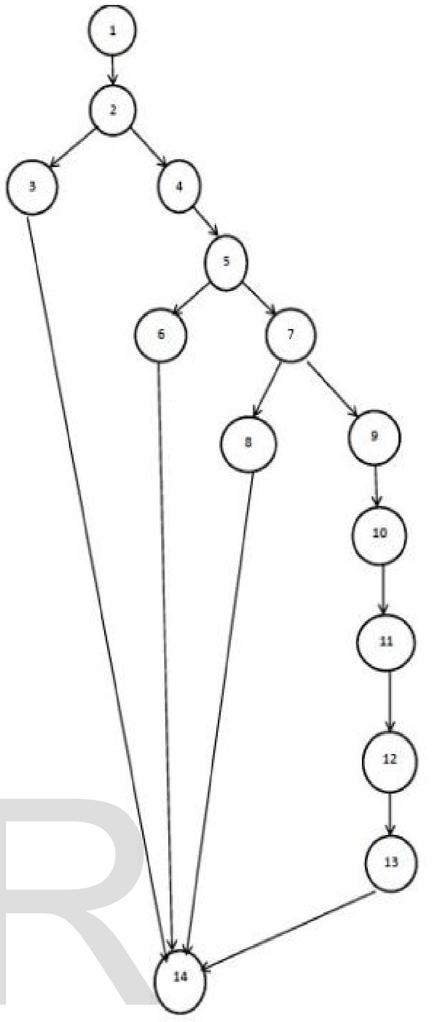
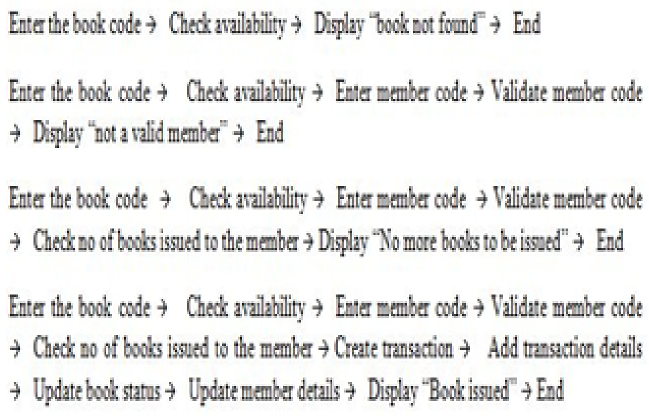


Figure 4: Control Flow Graph 'G'

4. On applying DFS algorithm, the possible test paths (fig 5) are generated.



Enter the book code → Check availability → Display "book not found" → End

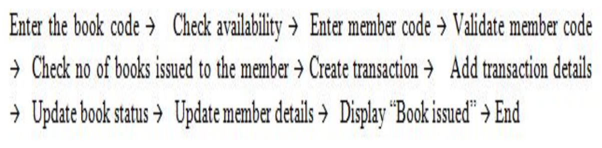
Enter the book code → Check availability → Enter member code → Validate member code → Display "not a valid member" → End

Enter the book code → Check availability → Enter member code → Validate member code → Check no of books issued to the member → Display "No more books to be issued" → End

Enter the book code → Check availability → Enter member code → Validate member code → Check no of books issued to the member → Create transaction → Add transaction details → Update book status → Update member details → Display "Book issued" → End

Figure 5: Generated Test Paths

5. Once on obtaining the test paths, by applying Tabu Search, we get the prioritized path (fig 6). The prioritized path shows the path which is most likely to be executed, thus making it the most critical one.



Enter the book code → Check availability → Enter member code → Validate member code → Check no of books issued to the member → Create transaction → Add transaction details → Update book status → Update member details → Display "Book issued" → End

Figure 6: Prioritized Test Path

6. The generated test paths are checked against cyclomatic complexity.

   Cyclomatic complexity for a CFG is calculated by:
   V (G) = E-N+2; where E is the no of edges and N is the no of nodes in the CFG respectively.
   Or V (G) = P+1; where P is the no of predicates in the CFG.
   Based on the equations; V (G) = E-N+2;
   Here E= 16, N= 14
   V (G) = 16-14+2 =4
   Or V (G) = P+1; here P= 3
   V (G) = 3+1= 4

   The cyclomatic complexity of the graph is 4 and the no of test paths generated is also 4. Thus, it validates the test paths generated.

6 Conclusion and Future Work

In this paper, we showed that Tabu Search Algorithm is applied to Activity Diagram. This can also be applied to other UML Behavioral Models like Use case, Sequence, Collaboration, State Chart diagrams for generating and prioritizing test cases. The approach used is capable of detecting faults, like faults in loops, synchronization faults, etc. Other heuristic approaches (particle swarm optimization, ant colony optimization, simulated annealing etc.) can also be applied in prioritizing the test paths. A hybrid algorithm can also be developed combing some features of the existing heuristic algorithms with tabu search algorithm to find out the critical path in a more efficient way.

Acknowledgment

The authors wish to thank Dr. K. S. Patnaik who is currently working as an Assosiate Professor in Birla Institute of Technology, Mesra in the Department of Computer Science and Engineering for his valuable guidance during this work.

References

[1] B.Bezier, *Software Testing Techniques*, 2nd edition, Dreamtech Press, 2004.

[2] J. Rumbaugh, I. Jacobson, and G. Booch. *The Unified Modeling Language Reference Manual*, Addison-Wesley, 2001.

[3] J. Rumbaugh, I. Jacobson, and G. Booch. *The Unified Modeling Language User Guide*, Addison-Wesley, 2001.

[4] C. Mingsong, Q. Xiaokang, and L. Xuandong,. *Automatic test case generation for UML activity diagrams*, in 2006 international workshop on Automation of software test, pp. 2-8, 2006.

[5] Debasish Kundu, Debasis Samanta,. *A Novel Approach to Generate Test Cases from UML Activity Diagrams*, Journal of Object Technology, Vol. 8, No. 3, pp.65 -83, May-June 2009.

[6] Pakinam N. Boghdady, Nagwa L. Badr, Mohamed Hashem and Mohamed F.Tolba. *A Proposed Test Case Generation Technique Based on Activity Diagrams*, International Journal of Engineering & Technology IJET-IJENS Vol: 11 No: 03 June 2011.

[7] H. Kim and S. Kang and J. Baik and I. Ko. *Test Cases Generation from UML Activity Diagrams*, 8th ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/ Distributed Computing(ACIS-SNPD), IEEE Computer Society, July 2007.

[8] Kansomkeat, S., Thiket, P., Offutt, J. *Generating Test Cases from UML Activity Diagrams Using the Condition-Classification Tree Method*. In: Proceedings of the 2nd International Conference on Software Technology and Engineering (ICSTE 2010), pp. V1-62–V1-66. IEEE Computer Society, Los Alamitos, CA (2010).

[9] W. Linzhang, Y. Jiesong, Y. Xiaofeng, H. Jun, L. Xuandong, and Z. Guoliang. *Generating test cases from UML activity diagram based on gray-box method* , In 11th Asia-Pacific Software Engineering Conference (APSEC04), pp. 284-291, 2004.

[10] Praveen Ranjan Srivastava, and Tai-hoon Kim. *Application of genetic algorithm in software testing* International Journal of Software Engineering and its Applications 3, no. 4 (2009): 87-96.

[11] Sangeeta Sabharwal, Ritu Sibal, Chayanika Sharma. *Applying Genetic Algorithm for Prioritization of Test Case Scenarios Derived from UML Diagrams*, International Journal of Computer Science Issues, vol.8, issue 3, no.2, May 2011, pp. 433-444.

[12] A.V.K. Shanthi and G. Mohan Kumar. *A Heuristic Technique for Automated Test Cases Generation from UML Activity Diagram,* Journal of Computer Science and Applications. Volume 4, Number 2 (2012), pp. 75-86

[13] A.V.K.Shanthi and G.Mohan Kumar. *A Novel Approach for Automated Test Path Generation using TABU Search Algorithm,* International Journal of Computer Applications (0975 – 888) Volume 48– No.13, June 2012.

[14] Fred Glover. *Tabu Search - Part 1*. ORSA Journal on Computing **1** (2): 190–206. 1989.

[15] Fred Glover. *Tabu Search - Part 2*. ORSA Journal on Computing **2** (1): 4–32. 1990.