Volume 5, Issue 1, January 2015 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

An Application of Genetic Algorithm for Extraction and Evaluation of Effectiveness of Paths for Testing from Activity Diagram

Rituraj Jain
Associate Professor, Department of Computer Science Engineering
Vyas Institute of Engineering and Technology, Jodhpur, Rajasthan, India

Abstract - Software testing is the process of validation and verification of the software product which in turn deliver the reliable and quality oriented software product to users. For this, an effective software testing process is required to generate effective test cases, which demand an effort, time and cost. In many software development organizations, the cost of testing can account for more than 40% of the total development cost for a software system. This necessity has increased the demand for techniques that can generate test data effectively as well as at low cost. This paper proposed an application of Genetic Algorithm (GA) for extraction of paths to be tested from UML Activity diagram and evaluation of its effectiveness. Activity diagram is transformed to activity flow graph and then assigning weights on it. On the weighted activity flow graph, GA operators will be applied to get path that must be tested first with the suitable test case.

Keywords: Activity Diagram, Activity Flow Graph, Genetic Algorithm, Software Testing, Test Data.

I. INTRODUCTION

For validating and verifying software system under development, testing is the most important activity of software development life cycle. Many software products fail in some or the other ways, just because of improper testing. It is the method to measure how much software system is conforming to its requirements specification specified by users and to demonstrate its correct operation [1, 2]. Testing techniques are classified as functional (black box) and structural (white box) testing. Functional testing is based on functional requirements whereas structural testing is done on code itself. Whole process of testing is comprises of three main activities which are test case generation, test execution, and test evaluation. Out of these three test case generation is the most critical and creative task. Results of the testing process is totally depends on the how effective test case were generated. Testing tools are available to do the testing either manually or automatically. Test cases can be derived from the models used for designing software specification as well architecture. Notation used for the models plays the key role during the test cases generation. Many types of models are used to derive test cases [3, 4, 5].

Unified Modeling Language (UML) models are one of the highly ranked types of models considered and used [6, 7, 8, 9] as an important source of information for test case design. If it is satisfactorily exploited, it will reduce testing cost and effort and at the same time improve the software quality. Several studies and approaches have been defined by many of the researchers during the last decade to use different UML models to generate test cases [9, 10, 11, 12, 13, 14, 15, 16, 17] for black box as well as white box testing. In this paper, we use UML activity diagrams as source to generate path which is to be tested first. The Activity Diagram can help to describe the flow of control of the target system, such as the exploring complex business rules and operations, also describing the business process.

Evolutionary Testing, which are based on evolutionary algorithms [18, 19] are now popular in the industry for test case generation. The most commonly used population-based evolutionary computation techniques are motivated from the evolution of nature. For years, many researchers have proposed different methods for developing test data/case generators [20, 21, 22, 23] using soft computing approaches. Genetic algorithms (GA), evolutionary programming, evolutionary strategies and genetic programming are four well-known examples [24], which have been used in many fields [25]. GA is well known form of the evolutionary algorithms conceived by John Holland in United States during late sixties [26].

This paper presents a new application of Genetic Algorithm to generate the first path for testing from the activity diagram and helps in generating the good test cases. Activity diagram will be used a building block for extracting the paths to be tested. Generated path will also evaluate to show the effectiveness of the proposed approach. The paper is divided into 6 sections. Section 2 & section 3 describe about the basic structure of Activity Diagram and GA respectively. In section 4, our proposed approach is discussed while section 5 describes the case study based on the proposed approach. Section 6 concludes the paper.

II. ACTIVITY DIAGRAM

The Activity Diagram can help to describe the flow of control of the target system, such as the exploring complex business rules and operations, describing the use case also the business process. Activity diagrams show the workflow from a start point to the finish point detailing the many decision paths that exist in the progression of events contained in

the activity [27]. The activity can be described as an operation of the system. The model elements consist of initial node, final nodes, nodes, edges. The nodes represent processes or process control, including action states, activity states, decisions, forks, joins and merge. The edges represent the occurring sequence of activities, objects involving the activity, including control flows, message flows and signal flows. Activity states and action states are denoted with round cornered boxes. Transitions are shown as arrows. Branches are shown as diamonds with one incoming arrow and multiple outgoing arrows each labeled with a boolean expression to be satisfied to choose the branch. Joins or forks are shown by multiple arrows entering or leaving the synchronization bar and merge shown by Diamond shaped box with multiple incoming flows. Figure 1 and figure 2 respectively show the main construct of activity diagram and example of an activity diagram which consists of most elements to describe an operation.

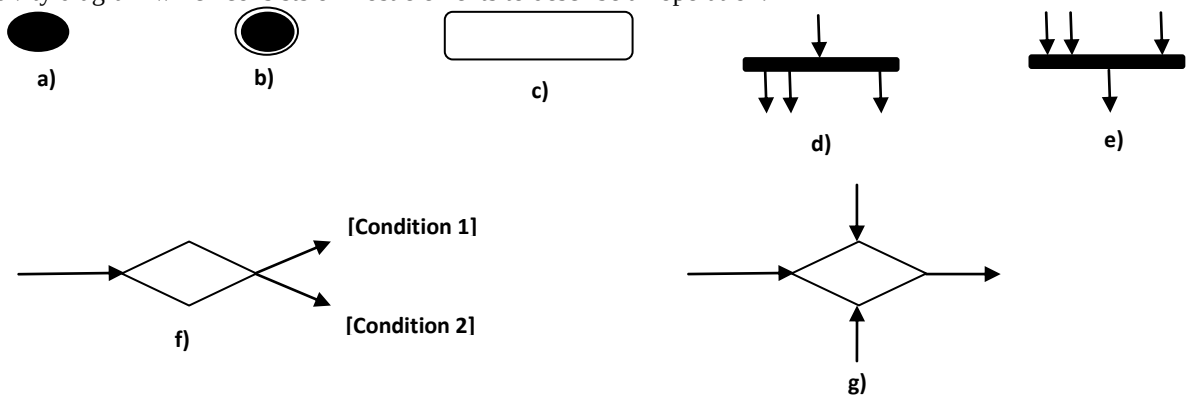

Figure 1: Main constructs of an Activity Diagram a) Initial Node, b) Final Node, c) Activity / Action Node, d) Fork Node, e) Join Node, f) Decision Node, and g) Merge Node

III. GENETIC ALGORITHM

Genetic Algorithms (GAs) are global optimization methods. GAs operates on a population of potential solutions, applying the principle of survival of the fittest to produce better and better approximations to a solution. GAs differing from conventional search techniques, start with an initial set of random solutions called population. Each individual in the population is called a chromosome, representing a solution to the problem at hand. A chromosome is usually, but not necessarily, a binary bit string. The chromosomes evolve through successive iterations, called generations. During each generation, the chromosomes are evaluated, using some measure of fitness. To create the next generation, new chromosomes, which called offspring, are formed by merging two chromosomes from current generation using a crossover operator or modifying a chromosome using a mutation operator. A new generation is formed by selecting, according to the fitness values, some of the parents and offspring and rejecting others so as to keep the population size constant. After several generations, the algorithm converges to the best chromosome, which hopefully represents the optimum or sub-optimal solution to the problem [26] [28] [29] [30]. Figure 2 shows the structure of a simple GA.

Operators of GA

Before applying the basic operations of the GA it is required to encoding a chromosome. A chromosome should in some way contain information about solution that it represents. There are different methods of representing a chromosome in the genetic algorithm, e.g. using binary, Gray, integer or floating data types. After encoding the chromosome, GA operators for reproduction will be applied on them. These operators, crossover and mutation, are the key to the power of GAs. They are the operators which create new individuals with the idea that the new individuals will be closer to a global optimum.

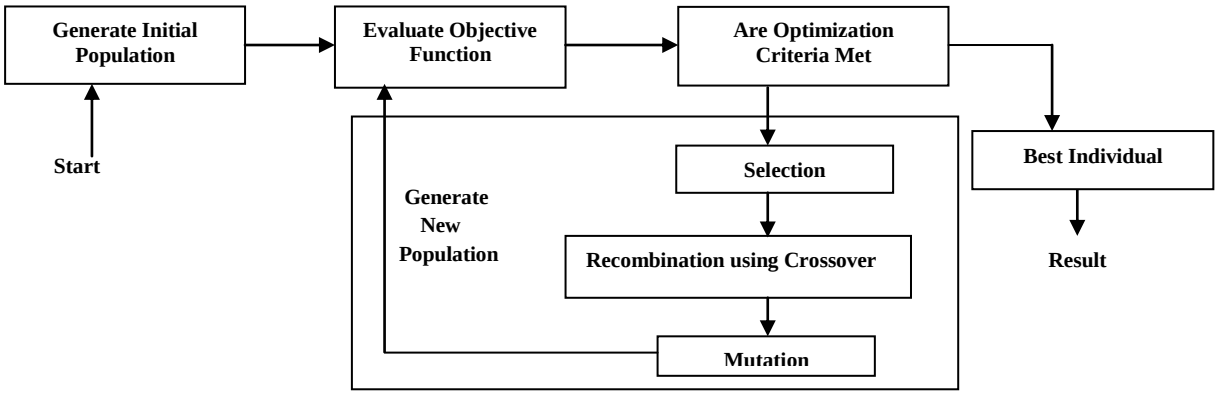

Figure 2: Structure of GA

a) Crossover Operator

Crossover selects genes from parent chromosomes and creates a new offspring. The simplest way how to do this is to choose randomly some crossover point and everything before this point copy from a first parent and then everything after

a crossover point copy from the second parent. The crossover operators search for better genes (building blocks) within the genetic material. The objective here is to create better individuals and a better population over time by combining material from pairs of (fitter) members from the parent population. Crossover occurs according to a crossover probability. Basic operation of crossover will be done as shown in figure 3:

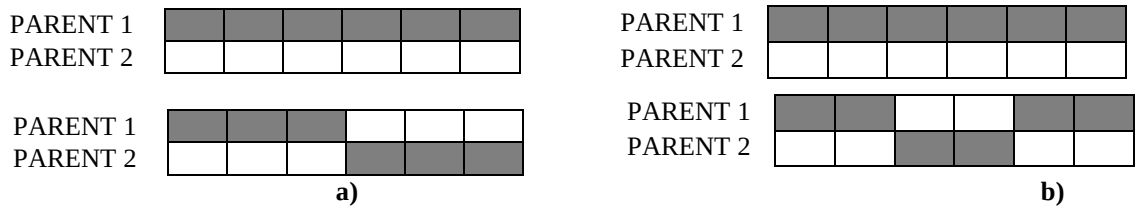

Figure 3: a) One Point Crossover, b) Two Point Crossover

b) Mutation Operator

After a crossover is performed, mutation takes place. Mutation is intended to prevent falling of all solutions in the population into a local optimum of the solved problem. Mutation operation randomly changes the offspring resulted from crossover. In case of binary encoding we can switch a few randomly chosen bits from 1 to 0 or from 0 to 1. Mutation can be then illustrated as shown in figure 4:

Before Mutation	1	1	0	0	1	0
After Mutation	1	0	0	1	1	0

Figure 4: Before and after Mutation

Crossover and Mutation Probability:

There are two basic parameters of GA - crossover probability and mutation probability. Crossover probability says how often will be crossover performed. If there is no crossover, offspring is exact copy of parents. If there is a crossover, offspring is made from parts of parents' chromosome. If crossover probability is 100%, then all offspring is made by crossover. If it is 0%, whole new generation is made from exact copies of chromosomes from old population (but this does not mean that the new generation is the same!). Crossover is made in hope that new chromosomes will have good parts of old chromosomes and maybe the new chromosomes will be better. However it is good to leave some part of population survive to next generation.

Mutation probability says how often will be parts of chromosome mutated. If there is no mutation, offspring is taken after crossover (or copy) without any change. If mutation is performed, part of chromosome is changed. If mutation probability is 100%, whole chromosome is changed, if it is 0%, nothing is changed.

IV. PROPOSED APPROACH

Proposed approach comprises 6 steps which are shown in the figure 5. Details of all the steps are as follows:

Step 1: Designing of an Activity Diagram based on the specification of the Use Case and Objects.

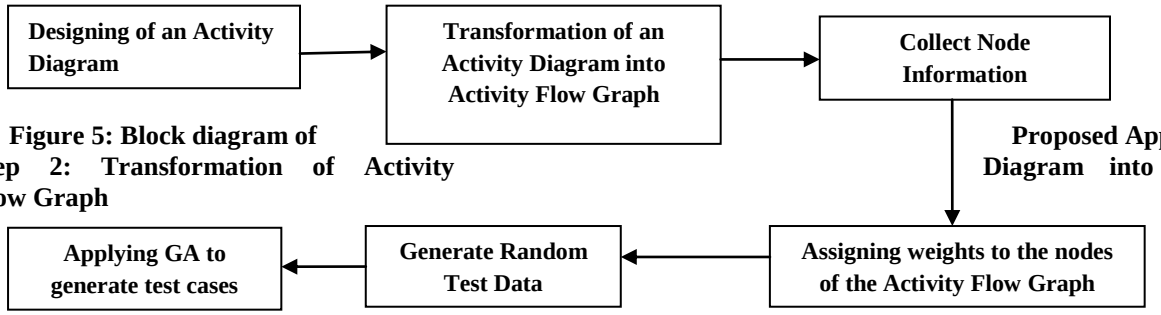

Figure 5: Block diagram of Proposed Approach

Step 2: Transformation of Activity Diagram into Activity Flow Graph

For each of the construct of the activity diagram, a node in the activity flow graph is created. Activity flow graph is directed graph where each node in the activity graph represents a construct and each edge of the activity graph represents the flow in the activity diagram. The mapping between activity diagram construct and activity flow graph is based on the conversion rules proposed in Table 1 [17].

Table 1: Conversion Rules

No.	Constructs of Activity Diagram	Node of Activity Flow Graph
1	Initial Node	Node of type *S* with no incoming edge
2	Activity Final Node	Node of type *E* with no outgoing edge
3	Flow Final Node	Node of type *E* with no outgoing edge
4	Decision Node	Node of type *D*
5	Guard Condition associated	Node of type *C* and associated with condition string. Its parent

	decision node	node is of type D
6	Merge Node	Node of type *M* and having single outgoing edge
7	Fork Node	Node of type *F* with single incoming edge
8	Guard condition associated with fork node	Node of type *C* and its parent node is of type *F*
9	Join Node	Node of type *J* and will have one outgoing edge.
10	Activity Node	Node of type *A*. Its associated string is activity name.

Step 3: Collect all the node information by traversing the Activity Flow Graph.

This step is to collect information of number of the nodes/activities calling a particular node i.e. FANIN as well as number of the nodes/activities called by that node i.e. FANOUT. These counts will help in to rank the criticality and rank of the node. Higher counts of the $FANIN_i$ and $FANOUT_i$ for a node i show that the node has critical processing and required to give higher rank during the testing. These counts will also used to assigned weights to each edge of the Activity Flow Graph. Additional to $FANIN_i$ and $FANOUT_i$, it is also required to keep the information of the distance ($DIST\_NODE_i$) of the node i from the initial node. Distance from the initial node is the count of the number of nodes needs to traverse to visit that node.

Step 4: Assigning weights to the nodes of the Activity Flow Graph.

In this step we start with the initial value of the weight which will be selected based on the Activity flow graph. Weights are assigned to the edges based on the rank of the nodes to which it is ending. Path having critical nodes will get higher weights. FANOUT, FANOUT will highlight the criticality of the node and edge originating to that node. At the node, weights of all the incoming edges will summed up and distributed to next level i.e. to the outgoing edges from that node. Distribution will be done using the 80-20 rule as proposed in [2]. Edge originating to higher rank node will get 80% of the total weight of the current node and rest 20% will be assigned to other edge. By this rule 80% wrights will be credited to fork, looping nodes as these nodes having higher rank and more critical to others and they required more emphasis during testing. If many edges which are exist from a node having looping or branching paths than 80% will be equally divided between them and rest of 20% will be given to sequential path.

Step 5: Generate random test case

A activity flow graph has 2^n paths where n is number of branches. Traversing each path may be very time consuming therefore it is required to select a meaningful paths as target paths. The entire selected path must cover all the branches, statements and each construct of the actual program going to be developed. All the test cases will be defined based on the path selected for testing. So we start with selecting a random generated test case which will be represented by binary bits. Number of bits used for the representation will be the number of decisions nodes presented in the activity flow graph. Bit 0 represented false edge execution while 1 represented true edge of the of decision node. This randomly generated test case is able to execute all the nodes exactly once. From this initial populated test case new test cases will be generated in the next step.

Step 6: Applying GA to generate the test cases for all the testing scenarios.

This last step is repetitive process in which random Genetic Algorithm will be applied to initial populated test case generated in the previous step. The GA operators are Selection, Fitness Function, Crossover, Mutation and Reinsertion. Each of these operators plays important role in generation of test data.

Following sub steps will be applied until evaluation objectives not satisfied.

a) Selection: The selection of parents for reproduction is done according to a probability distribution based on the individual's fitness values. First the fitness value is calculated using the Fitness function. Weights are used to determine the relative contribution of a path to the fitness calculation. Thus, more weight is assigned to a path which is more "critical". Criticality of the path to test data generation is based on the fact that predicate, loop and branch nodes are given preference over sequential nodes during software testing. The fitness value of each chromosome is calculated by using the formula given below:

$$F = \sum_{i=0}^{n} w_i \qquad \textbf{(Eq. 1)}$$

where, w_i is weight of i^{th} node in a path under consideration and n is number of nodes in a current path. Weight of i_{th} node is calculated by using FANIN, FANOUT and distance_node information of the path as shown by equation given below.

$$w_i = (FANIN_i * FANOUT_i) * (DIST\_NODE_i)^2 \qquad \textbf{(Eq. 2)}$$

Probability of selection SEL_PROB is given by

$$SEL\_PROB = F_m / \sum F_m \qquad \textbf{(Eq. 3)}$$

where m = 1 to n and n is population size.

b) Crossover: Before the crossover operator may be applied, the individual can be converted into a binary representation. There are number of techniques of crossover, but all require swapping of sequence of bits in the chromosome. It involves swapping between two individuals or test data. Crossover happens according to a crossover probability CROSS_PROB. For each parent selected, generate a random real number R in the range [0, 1]; if R < CROSS_PROB then select the parent for crossover. CROSS_PROB is assumed to be 80%. Two point crossover operation is applied when the parents are selected based on the CROSS_PROB.

c) Mutation: Mutation always works after the crossover operator. This kind of mutation is called here normal mutation which operates on chromosomes created during crossover, and flips each bit with the pre-determined probability. In mutation the bits are flipped from 0 to 1 and vice versa. Mutation occurs according to a mutation probability

MUTA_PROB. To perform mutation, a real number R is generated in the range {0, 1} for each chromosome in the offspring and for each bit within the chromosome. If this generated number R is less than the MUTA_PROB then mutation will be performed.

Same procedure is carried out for the new data set obtained for further crossover and mutation until we start getting better values of the fitness function F.

V. CASE STUDY OF PROPOSED APPROACH

Step 1: For experiment the proposed approach an Activity Diagram for Library Book Issue process available at [30] will be used. This diagram is shown in figure 6. As defined in the second step, this activity diagram will be transformed to activity flow graph as shown in figure 7. As per the step 3, information related to each node will be identified as shown in table 2.

Table 2: FANIN, FANOUT and DIST_NODE Information for each node

Node No.	FANIN	FANOUT	DIST_NODE	FANIN * FANOUT * (DIST_NODE)^2
1	0	1	Initial Node	0
2	1	1	1	1
3	1	1	2	4
4	1	2	3	18
5	1	1	4	16
6	1	1	4	16
7	1	2	5	50
8	1	1	6	36
9	1	1	6	36
10	1	2	7	98
11	1	1	8	64
12	2	1	7	98
13	1	1	8	64
14	1	1	9	81
15	3	0	Ending Node	0

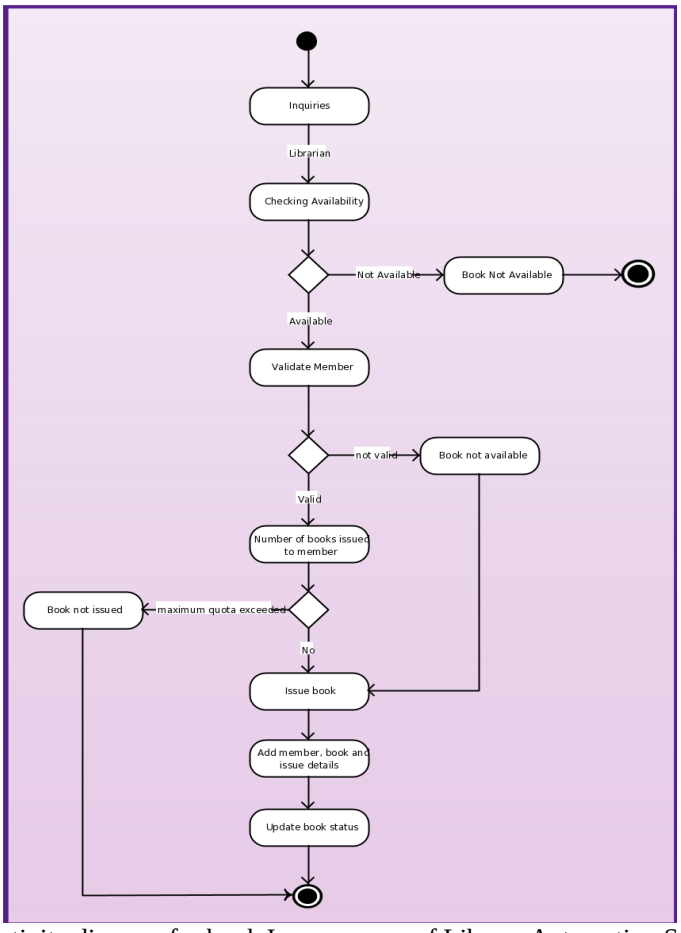

Figure 6: Activity diagram for book Issue process of Library Automation System [30]

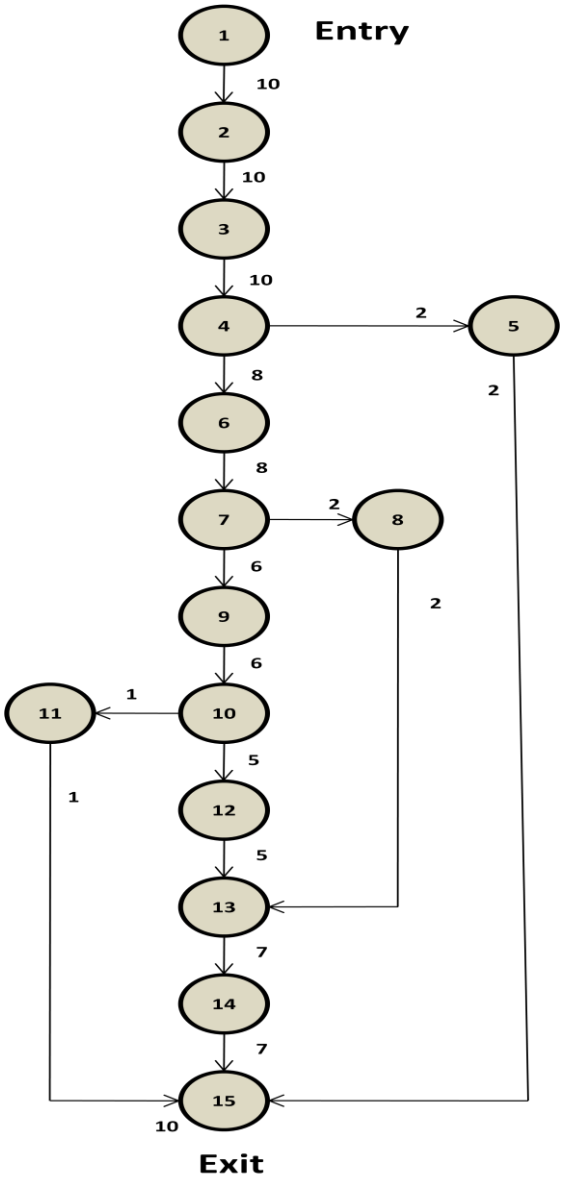

Figure 7: Transformed weighted activity flow graph

Table 3: Iteration I

Sr. No.	T	F(T)	SEL_PROB	R
1	010	39	0.044674	0.267
2	100	368	0.421535	0.354
3	110	466	0.533792	0.654

After getting the node information in step 3, weights of all the edges will identified and add these weights to activity flow graph in step 4. These weights are also mentioned with the all the edges to activity flow graph in figure 7. As the generated graph is sparse in nature so initial weight taken as 10. If it is dense then we can take initial weight as 100. Now in step 5 a random test case is generated. Initial population is selected is 010, 100, 110. Path relevant to initial population 010 will be 1,2,3,4,6,15. Path relevant to initial population 100 will be 1,2,3,4,6,7,8,12,13,14,15. Path relevant to initial population 110 will be 1,2,3,4,6,7,9,10,12,13,14,15. Fitness for each of the initial test 010, 100, 110 data is 39, 368, and 466 respectively. Table 3 shows information related to I iteration of GA process. This process will be repeated for number of iteration to get the test data with the highest fitness value. Path corresponding to this test data will be tested first. For the chosen example, chromosome 110 was found with the highest fitness value so the path corresponding to chromosome 110 must be tested first. Path corresponding to this test data is 1,2,3,4,6,7,9,10,12,13,14,15.

VI. CONCLUSION

In software testing, the generation of testing data is one of the key steps which have a great effect on the automation of software testing. In this paper an approach is presented which is used activity diagram as a main input construct to identify the test case from where to start i.e. propose approach is trying to find the path that must be tested first and the

test cases relevant to that path be prepared first. Main input to the approach is the activity diagram which is transformed to activity flow graph to applying GA on it. Genetic algorithms are often used for optimization problems in which the evolution of a population is a search for a satisfactory solution given a set of constraints. The greatest merit of genetic algorithm in program testing is its simplicity. By identifying the best path for testing first, it can be achieved more effective way of testing which in turns help in project management for effort & cost estimation and their distribution and quality enhancements.

REFERENCES

[1] Dr. Velur Rajappa, Arun Biradar, Satanik Panda, Efficient software test case generation Using Genetic algorithm based Graph theory" International conference on emerging trends in Engineering and Technology, pp. 298-303, IEEE (2008).

[2] Praveen Ranjan Srivastava and Tai-hoon Kim, Application of Genetic algorithm in software testing, International Journal of software Engineering and its Applications, vol.3, No.4, pp. 87-96 (2009).

[3] B. Berenbach, D. Paulish, J. Kazmeier, A. Rudorfer, Software and Systems Requirements Engineering in practice, The McGraw-Hill Companies Inc., USA, 2009.

[4] S.R. Dalal, A. Jain, N. Karunanithi, J.M. Leaton, C.M. Lott, G.C. Patton, B.M. Horowitz, Model-Based Testing in Practice, Proceedings of the 21st international conference on Software engineering, New York, USA, 1999.

[5] A.C. Dias-Neto, R. Subramanyan, M. Vieira, G.H. Travassos, A Survey on Model-based Testing Approaches: A Systematic Review, Proceedings of the 1st ACM international workshop on Empirical assessment of software engineering languages and technologies in conjunction with the 22nd IEEE/ACM International Conference on Automated Software Engineering (ASE), New York, USA, 2007.

[6] Object M. Group, OMG Unified Modeling Language (OMG UML), Superstructure,V2.1.2, ttp://www.omg.org/spec/UML/2.1.2/Superstructure/PDF/

[7] S.K. Swain, D.P. Mohapatra, R. Mall, Test Case Generation Based on Use case and Sequence Diagram, International Journal of Software Engineering, IJSE 3 (2010).

[8] M. Riebisch, I. Philippow, M. Götze, UML-Based Statistical Test Case Generation, Revised Papers from the International Conference NetObjectDays on Objects, Components, Architectures, Services, and Applications for a Networked World, Springer-Verlag London, UK, 2003

[9] Rituraj Jain, Bipin Pandey, Importance Of Unified Modeling Language For Test Case Generation In Software Testing, Int. J. Computer Technology & Applications,Vol 5 (2),345-350

[10] Hartmann, J., Imoberdof, C., Meisenger, M., UML-Based Integration Testing, in ISSTA 2000 conference proceeding, Portland, Oregon, 22-25 August 2000, pp. 60-70.

[11] M. Katara and A. Kervinen, Making model-based testing more agile: a use case driven approach, In Proc. Haifa Verification Conference 2006, number 4383 in Lecture Notes in Computer Science, pages 219-234. Springer, 2007.

[12] Alan Hartman, Mika Katara, and Sergey Olvovsky, "Choosing a Test Modeling Language: a Survey, In Proceedings of the Haifa Verification Conference 2006, LNCS, Springer.

[13] Nebut, C., Fleurey, F., Le Traon, Y., Jézéquel, J.M.: Automatic Test Generation: A Use Case Driven Approach. IEEE Transactions on Software Engineering 32(3) (March 2006)

[14] M. Prasanna and K.R. Chandran, Automatic Test Case Generation for UML Object diagrams using Genetic Algorithm, Int. J. Advance. Soft Comput. Appl., Vol. 1, No. 1, pp. 19-32, July 2009.

[15] Ashalatha Nayak, Debasis Samanta: Automatic Test Data Synthesis using UML Sequence Diagrams, in Journal of Object Technology, vol. 09, no. 2, March-April 2010, pp. 75-104.

[16] Minj, Jasmine, and Lekhraj Belchanden. Path Oriented Test Case Generation for UML State Diagram using Genetic Algorithm, International Journal of Computer Applications 82 (2013).

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

[18] André Baresel , Hartmut Pohlheim , Sadegh Sadeghipour, Structural and functional sequence test of dynamic and state-based software with evolutionary algorithms, Proceedings of the 2003 international conference on Genetic and evolutionary computation: PartII, July 12-16, 2003, Chicago, IL, USA

[19] O. Buehler and J. Wegener. Evolutionary functional testing of an automated parking system. In International Conference on Computer, Communication and Control Technologies and The 9th International Conference on Information Systems Analysis and Synthesis, Orlando, Florida, USA, 2003.

[20] P.R. Srivastava, V. Ramachandran, M. Kumar, G. Alukder, V. Tiwari, P. Sharma, Generation of test data using Meta heuristic approach, In the Proceedings of IEEE TENCON, Nov 2008..

[21] B. Korel. Automated software test data generation. IEEE Transactions on Software Engineering, 16(8), August 1990.

[22] B.F. Jones, H.-H. Sthamer and D.E. Eyres. Automatic structural testing using genetic algorithms. Software Engineering Journal, pages 299-306, September, 1996.

[23] Ritruaj Jain, Bipin Pandey, Soft Computing Based Approaches for Software Testing: A Survey, International Journal of Soft Computing and Engineering (IJSCE) ISSN: 2231-2307, Volume-4, Issue-2, May 2014

[24] Wegener J, Baresel A, Sthamer H, Evolutionary Test Environment for Automatic Structural Testing, Information and Software Technology, 2001, Vol 43, pp: 841-854

[25] Yuhui Shi, Russell C. Eberhart, Parameter Selection in Particle Swarm Optimization , Lecture Notes In Computer Science; Vol. 1447 Proceedings of the 7th International Conference on Evolutionary Programming VII, 1998 pp: 591-600.
[26] Goldberg, D.E, Genetic Algorithms: in search, optimization and machine learning, Addison Wesley, M.A, 1989.
[27] http://www.visual-paradigm.com/VPGallery/diagrams/Activity.html (Access on 1^{8th} Dec ,2014)
[28] M. Melanie, An Introduction to Genetic Algorithms, Massachusetts, MIT Press, 1999.
[29] K. F. Man , K. S. Tang and S. Kwong Genetic algorithms: Concepts and applications, IEEE Trans. on Industrial Electronics, vol. 43, no. 5, pp.519 -534 1996
[30] http://creately.com/jupiter/diagram/image/i3a1pdbk1 (Access on 20th Dec, 2014)

