Use of Genetic Approach for Test Case Prioritization from UML Activity Diagram

Wasiur Rhmann
Deptt. of Computer Science
B. B. Ambedker University
(A Central University)
Lucknow, U.P., India

Taskeen Zaidi, Ph.D.
Deptt. of Computer Science
B. B. Ambedker University
(A Central University)
Lucknow, U.P., India

Vipin Saxena
Deptt. of Computer Science
B. B. Ambedker University
(A Central University)
Lucknow, U.P., India

ABSTRACT
Software testing plays an important role for evaluation of the quality of the software. Quality of the software depends upon the kind of testing strategies applied by the software tester who generates valid and invalid test cases for evaluating the quality of software. For optimizing the testing procedure, testing paths are very valuable to judge the quality of the software. From the literature, it is revealed that it is big challenge to optimize the testing paths. In the present work, the concept of genetic algorithm is used for prioritization of test cases generated from Unified Modeling Language. Testing paths are generated from activity diagram is designed for generation of test cases from design specification which will further reduce the cost of software testing. Branch coverage technique and predicate coverage methods are used for prioritization of test cases by identifying the independent paths. Information flow metric and decision node based genetic function are used through a case study.

Keywords
Software Testing, Test Suites, Genetic Approach, Unified Modeling Language (UML), Activity Diagram

1. INTRODUCTION
Software testing is used to verify the quality of the software projects. Software testing helps to the customer to gain the confidence in software product. The process of testing of software is error prone and time consuming. Software testing is done with intent to find errors prior to deliver to the end user [1]. Different types of software testing are used to detect the bugs, faults and failures in the software. Manual testing is error prone and process of testing is automated. Recently software testing from different UML diagrams is gaining popularity. Unified Modeling Language is one of the important modeling languages which consist of the various kinds of static and dynamic diagrams for the designing of the complex research problems. Different types of UML diagrams are designed by Booch et al. [2-3] .These UML diagrams are used to represent static and dynamic view of the software system. Object Management Group [4-5] released different versions of platform independent Unified Modeling Language and approved UML as standard modeling language. Activity diagram is used for describing various activities for the software and hardware systems. Rambaugh et al. [6-7] have explained activity diagram to model the system for computational and organizational process. Activity diagram is used to model system behavior without going into internal details of the system. It contains states which represent performance of action and transition is triggered by completion of actions. Model used for testing remains valid even when there is small change in code. Software testing using design model may reduce the cost of software testing [8]. Mahali and Acharya [9] proposed a model based test case prioritization for regression testing and giving high priority to change activity and also given a prioritization formula for change activity. Independent paths are assigned to compute critical value for prioritization of test cases. Lamancha et al. [10] proposed test suite minimization by mutation analysis. Mutation score is considered for each test case and on the basis of score, test cases are selected. This procedure of test suite reduction is applied on different software programs. Fraser and Wotawa [11] proposed an approach to reduce the size of test suite and does not affect fault detection potential of test suite and test cases are created from model based checker technique. Test cases are discarded which have minimum fault detection potential. Rothermal et al. [12] described various techniques for test case prioritization for regression testing based on the fault detection potential of test cases. Three strategies are used for test case prioritization by code coverage of test case; code not previously covered by test case and estimated fault detection potential of a test case. These are prioritized and compared with randomly generated test cases and optimally ordered test suites. Fan et al. [13] point out that UML activity diagram can easily model parallel flow in system author considered two kinds of activity diagram atomic and compound activity diagram based on sub activity state in activity diagram. In this paper author deals with compound activity diagram. Functional decomposition, bottom up integration and round robin strategy are used to generate test cases from compound activity diagram. Purra and Khalition [14] explained a greedy approach for test suite reduction which is applied for test suite reduction. Test cases with maximum number of requirement coverage and minimum number of requirement overlap with others test cases are selected to form reduced test suites. McMaster and Memon [15] considered Graphical User Interface (GUI) testing which is done by considering call stack coverage. Call stack coverage for any program can be calculated by execution of program. UML state chart diagram is used for test case generation and authors transform conditional predicate on state transition. Test data is generated by using functional minimization method. Different coverage like full predicate coverage and transition path coverage are used by boundary value analysis to generate test case. Prasad et al. [16] identified redundant test cases based on several coverage criteria function call stack, branch coverage, etc. YU and Lau [17] analyzed modified condition/decision coverage and MUMCUT and several other coverage criteria. Fault detection potential for a test case by considering modified condition/decision coverage and other coverage criteria for logical decision for test suite reduction are described. Offut and Adurazik [18] pointed out that selection of test cases from UML is a challenging task. UML state chart diagrams are

used for test cases generation. Authors have defined test case based on specifications. Arora et al. [19] proposed semantics for automatic transformation of state chart diagram into finite state Automata. They used the finite state automata to generate regular grammar from finite state machine. This grammar is used to generate test cases against various test conditions.

The present work deals with the test case prioritization through UML activity diagram and Genetic approach. A case study of book issue from library is considered to generate various test cases and testing paths are obtained by the use of Genetic approach.

2. CONCEPT OF GENETIC APPROACH

Genetic approach is used to find the optimal solution of the research problems. By the use of genetic approach, generation of solution to optimization problem using this technique is inspired by natural evolution like selection, crossover, and mutation. In this algorithm, population of individual solution of optimization problem is evolved to provide better solution. Evolution is an iterative process in which randomly generated population is evolved to give new generation. Fitness of each individual is calculated for objective function of optimization problem. Fittest individuals are selected and modified to form new generation. The new generation of solution is used in the next generation. The genetic algorithm operates in the following manner:

1. Representation of solution domain of optimization problem in genome called as chromosome;

2. Formulation of fitness function to evaluate solution domain;

3. Genetic operations (reproduction and crossover) are applied to generate new population;

When genetic representations of individual and fitness function are decided, then initial population (solution) is generated randomly from solution space. Selection of individual solution is based on fitness function. The different selection techniques are used and described below in brief:

2.1 Stochastic Sampling with Replacement

In this technique, expected occurrence of a chromosome in mating pool is calculated by dividing fitness of chromosome to sum of the fitness of all chromosomes in the population.

Let f_i is fitness function of selected chromosome, $\sum f_i$, where i=1(1)n, is sum of all fitness function of all chromosomes, p is probability of selecting the chromosome, e is expected number of occurrence and n is number of chromosomes' in population, then probability of selecting the chromosome is

$$p=f_i/\sum f_i \quad (1)$$

Expected occurrence in the mating pool is

$$e=p\times n \quad (2)$$

The above is a procedure called as **Route Wheel Selection** as space is allocated for each member on the wheel. When individual is chosen, a copy is created from original population and this genome is further available for the selection.

2.2 Stochastic Sampling without Replacement

In this method, one can monitor number of times of a chromosome which is selected for crossover or reproduction. The value of expected incidence of chromosome is stored as offspring and decrement each time 1 if it is selected for reproduction and 0.5 when it is for crossover and when offspring count falls to 0 chromosomes is not eligible for selection.

2.3 Tournament Selection

In this approach, two members are selected for entire population in tournament. These members compete with each other winner of this competition progress to next level. When tournament is over only one individual is left. Relative fitness of each individual is decided according to the level it is reached. Then crossover and mutation operations are applied.

In crossover, two individual genes are selected to form a new generation. In mutation, some changes are done in individual to form new chromosome. The process of selection, crossover and mutation continues till desired level of fitness is achieved or maximum number of generation reaches.

3. UML ACTIVITY DIAGRAM

Unified modeling Language is used to visualize design of an object oriented system. It provides a blueprint of system. There are several diagrams which describe structural, behavioral and implementation view of system. Activity diagram is one of the important diagrams of UML to describe dynamic aspect of the system. Activity diagram models the sequence of activity taking place in performing system operation. These sequences of activity may be serial or concurrent from start to end. The activity diagram models workflow. There may also be decision taking place at an activity which provides choice. Activity diagram captures high level view of system. This high level view is useful for nontechnical person to understand the business requirements. In the present work we create UML activity diagram for issuing the book from library.

4. METHODOLOGY

Test cases are prioritized which are generated based on independent path of activity flow graph. The steps for prioritization of test cases are

1. Construct the Activity Flow Graph (AFG) from Activity Diagram. An activity graph contains nodes and edges. Each node represent different nodes of activity diagram which are initial node, decision node, guard condition, fork node, join node, merge node;
2. Generate independent paths from AFG. These independent paths will be used as testing paths. An independent path consists of sequence of activity and independent path set consists of path with at least one new node;
3. For representation of test cases in 0 and 1 form we considered control node in Activity Flow Graph. Test case representation will consist of a number of bits equal to control nodes. There are two branches comes out from each control node we assign 1 and 0 to these branches. For representing a test path we use one branch from each control node;
4. Then we travel from start to end nodes containing these branches this will be test path represented by test case in control node form;
5. For formulating fitness function we considered the number of decision node occur in path and sum of information flow metric of node in path. Fitness function of each test case is given by

F=m+n; (3)

Where m=Sum of Information_flow_value of nodes in path

n=Decision nodes occurred in the path.

Test cases are prioritized according to fitness value. The prioritize test cases of test suite will cover full branch coverage and predicate coverage.

5. CASE STUDY

Let us consider a simple example of book issue from library. The activity diagram represents about book issue process from library and represented in figure 1. Users visit to the library with library card; librarian of library takes the card for issue of book, scan the card; check the validity of card if card is invalid librarian denies to issue the book else if card is valid librarian checks number of book issued on the card. If number of books issued are greater than 5 then librarian denies to issue the book else issues the book. The account will be updated with number of book.

The Steps are described below:

1. First we construct the activity diagram of book issue from library and Activity Flow Graph (AFG) from activity diagram as represented in figure 2;

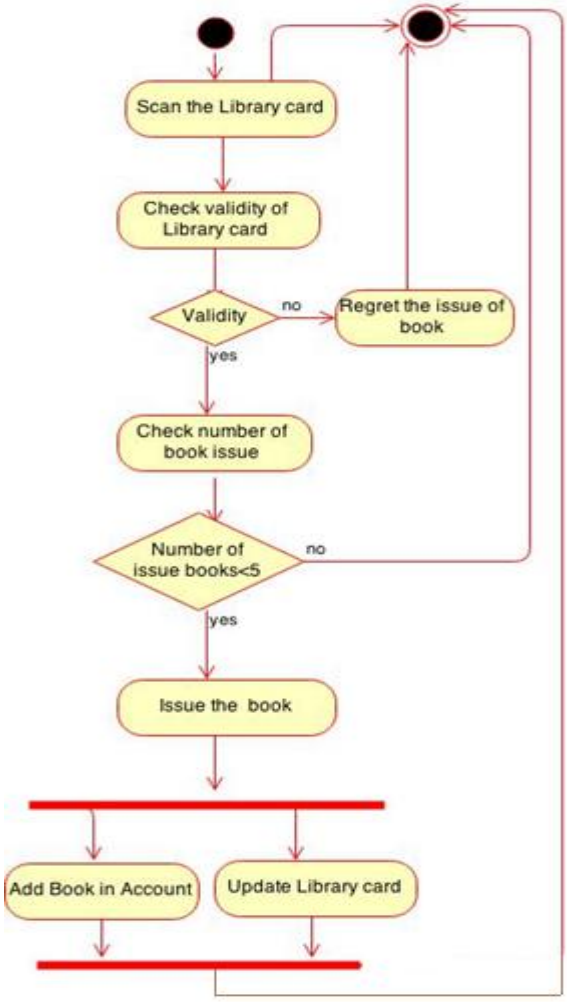

Fig. 1: UML Activity Diagram

2. Generate independent paths from AFG. These independent paths will be used as testing paths. An independent path consists of sequence of activity and independent path set consists of path with at least one new node;

3. Represent each independent path as test cases in the form of initial population of test cases generated from these paths we represent test cases in control node form. There are 3 control nodes which are 4, 6, and 8. From these nodes two branches come out, we take 0 for one branch and 1 for other branch. Therefore, Br_3=1, Br_2=0, Br_4=0, Br_5=1, Br_6=0, Br_7=1. We used three bit for representation of test path in genetic algorithm. Test case consisting Br_3 Br_4 Br_7 will be represented by 101 with travel path 1-2-3-4-5-6-7-8-9-11-13. Test case consists of Br_3 Br_4 Br_6 will be represented by 100 with travel path 1-2-3-4-5-6-7-8-10-10-13. Test case represented by Br_2 Br_5 Br_6 will be represented by 010 and with travel path 1-2-3-4-12-13. Test case represented by Br_3 Br_5 Br_6 will be represented by 110 with travel path 1-2-3-4-5-6-13;

4. Now we use this independent path in 1 and 0 forms to generate test cases in 1 and 0 forms. Compute fitness function using equation (3).

5. Then we applied crossover and mutation operator to give new generation of population;

Then we prioritize test cases of test suite according to fitness value of test cases.

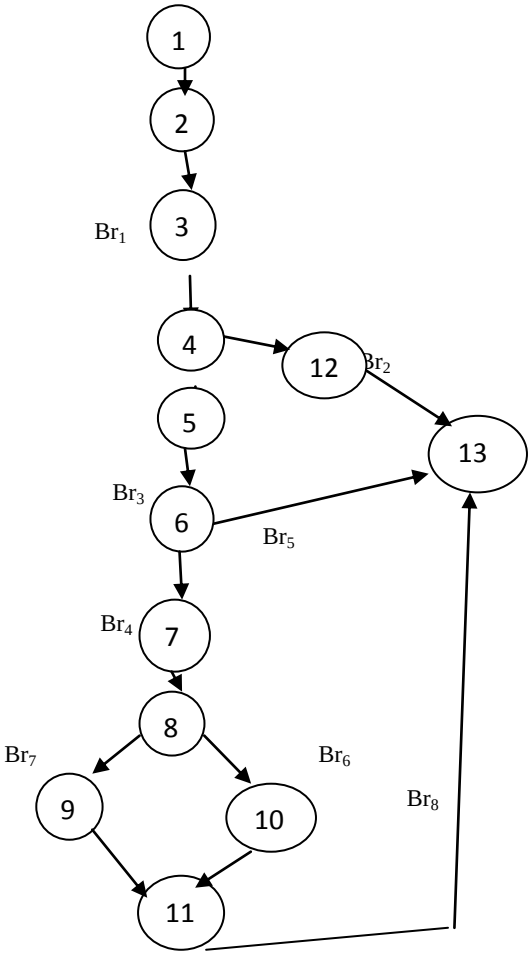

Fig. 2: Activity Flow Graph (AFG) from UML Activity Diagram

Independent paths from Activity Flow Graph are-

1-2-3-4-12-13

1-2-3-4-5-6-13

1-2-3-4-5-6-7-8-9-11-13

1-2-3-4-5-6-7-8-10-11-13

Information_flow_value of node=Fan-in × Fan-out

Where Fan-in of node is number of node that call to that node and Fan-out of a node is number of nodes that are called by that node. Table 1 represents node number taken from AFG and Information flow value which is computed by multiplying fan-in and fan-out values for corresponding nodes according to equation(4). For node 1 there is no branch incoming to this node so fan-in is 0 while one branch is coming out so fan-out is 1 and information flow value is computed as 0. Thirteen nodes are represented in the table according to AFG.

Table 1: Computation of Fitness Value

Node number	Information Flow
1	0
2	1
3	1
4	2
5	1
6	2
7	1
8	2
9	1
10	1
11	2
12	1
13	0

Table 2 shows the fitness value which is calculated for test data T1, T2, T3, T4 and given by

Fitness Value =Sum of Information_flow_values of nodes + covered decision nodes

f_1=0+1+1+2+1+1=5+1=6;

f_2=0+1+1+2+1+2+0+1=7+1=8;
f_3=0+1+1+2+1+2+1+2+1+2+0+2=13+2=16;

f_4=0+1+1+2+1+2+1+2+1+2+0+2=13+2=16; probability p_i is calculated using $p=f_i/\sum f_i$ then we calculated cumulative probability (c_i) and generated random number and find out the mating pool of initial population.

Table 2: Fitness of Initial population

S. No.	x	f(x)	p_i	c_i	Random No.	Mating Pool
T1	100	16	.347	.347	.62	T3
T2	010	6	.130	.477	.33	T1
T3	110	8	.173	.65	.82	T4
T4	101	16	.347	1	.91	T4

In table 3 we take mating pool and applied crossover and mutation.

Table 3: New population after crossover and mutation

S. No.	Mating Pool	Crossover	Mutation
T1	100	101	100
T4	101	100	110
T3	110	101	101
T4	101	110	101

In table 4 we used new generated population and applied same procedure as applied on initial population to get new mating pool.

Table 4: Fitness of new population

S. No.	x	f(x)	p_i	c_i	Random No.	Mating Pool
T 1	100	16	.285	.285	.9	T 4
T 2	110	8	.142	.427	.3	T 2
T 3	101	16	.285	.712	.8	T 4
T 4	101	16	.285	1	.4	T 2

In table 5 we applied crossover and mutation operation to get new generation of population.

Table 5: New population after second crossover and mutation

S. No.	Mating Pool	Crossover	Mutation
T1	101	110	101
T2	110	101	101
T3	110	101	100
T4	101	110	100

In table 6 fitness value is calculated for population generated from table 5

Table 6: Fitness of new generated population

S. No.	x	f(x)
T1	101	16
T2	101	16

T3	100	16
T4	100	16

Test cases corresponding to the 100 and 101 have highest fitness values. Test paths are given below for 100 and 101 values, respectively

1-2-3-4-5-6-7-8-9-11-13

1-2-3-4-5-6-7-8-10-11-13

These test paths should be tested first in order to get earlier faults in the software.

6. CONCLUSIONS

In the present work, we presented an activity diagram and genetic algorithm for test case prioritization. In this paper we investigated the method for generating test case from testing path of activity diagram which is efficient and is used for prioritization of test cases with full branch coverage and decision coverage by applying genetic algorithm.

Test paths are prioritized through genetic approach and these prioritized test paths can be used to reduce testing cost by executing test cases which have high probability of detection of faults. We can apply this procedure on large activity diagram and will get prioritized test suites. The case study prioritizes the test cases using the said technique. We may generate test data more efficiently in future research and we can use this approach in generation of test cases for comparatively large application.

7. REFERENCES

[1] Roger S. Pressman, Software Engineering: A Practitioner Approach, Mc. Graw Hill, 2009.

[2] G. Booch, J. Ramaugh and I. Jacobson, The Unified Modeling Language User Guide: Addison Wesley, 1999.

[3] G. Booch, J. Rumbaugh and I. Jacobson, The Unified Modelling Language User Guide, Twelfth Indian Reprint Pearson, 2004.

[4] OMG, Unified Modelling Language Specification, Available Online http://www.omg.org, 2001.

[5] OMG, XML Metadata Interchange (XMI) Specification, Available online via http://www.omg.org, 2002.

[6] J. Rambaugh, I. Jacobson and G. Booch, The Unified Modeling Language Reference Manual Addision-Wesley, 1999.

[7] OMG, UML Revision Task Force OMG Unified Modeling Language Specification, Version 1.4(final draft) February 2001.

[8] P. Samul and R. Mall, Boundary Value Testing Based on UML Models, in: Proceeding of 14th Asian Test Symposium (ATS), 2005, Pages 94-99.

[9] P. Mahali and A. A. Acharya, Model Based Test Case Prioritization Using UML Activity Diagram And Evolutionary Algorithm, International Journal of Computer Science and Informatics, 2013, Vol. 3, Issue 2, Pages 42-47.

[10] B. P. Lamancha, P. R. Mateo and M. P. Usaolo, Reduction of Test suites using Mutation, Fundamental Approaches to Software Engineering, Lecture Notes in Computer Science, 2012 ,Volume 7212, Pages 425-438.

[11] G. Fraser and F. Wotawa, Redundancy Based Test Suite Reduction, In Proceeding of the 10th International Conference on Fundamental Approaches to Software Engineering, 2007, Pages 291-305.

[12] G. Rothermal, R. H.Untch and M. Harrold, Prioritizing Test Cases for Regression Testing, IEEE Transaction on Software Engineering, 2001, Vol. 27, No. 10, Pages 929-948.

[13] Xin Fan, Jian Shu, Linlan Liu and Qi Jun Liang. Test Case Generation from UML Subactivity and Activity Diagram. Electronic Commerce and Security, 2009, Second International Symposium, Volume 2, Pages 244-248.

[14] S. Purra and A. Khalition, On the Optimization Approach Towards Test Suite Minimization, International Journal of Software Engineering and its Application, 2010, Vol. 4, No. 1, Pages 15-18.

[15] S. McMaster and A. Memon, Call-Stack Coverage for GUI Test Suite Reduction, IEEE Trans. Software Eng., 2008, Vol. 34, Pages 99-115.

[16] S. Prasad, M. Jain, S. Singh and C. Patvardhan, Regression Optimizer A Multi Coverage Criteria Test Suite Minimization Technique, International Journal of Applied Information Systems, April 2012, Volume 1, No. 8, Pages 5-11.

[17] Y. T. Yu and M. F. Lau, A Comparison of MC//DC, MUMCUT and Several Other Coverage Criteria for Logical Decisions, Journal of System and Software, 2006, Vol. 79, No. 5, Pages 577-590.

[18] J. Offut and A. Adurazik, Generating Test from UML Specification, In Proceeding of 2nd International Conference on Unified Modeling Language, 1999, Pages 416-429.

[19] Deepak Aroara, Bramah Hazela and Vipin Saxena, Semantics for UML Model Transformation and Generation of Regular Grammar, ACM SIGSOFT Software Engineering Notes, 2012, Volume 37, Issue 3, Pages 1-5.