Path Oriented Test Case Generation for UML State Diagram using Genetic Algorithm

Jasmine Minj
Bilaspur, Chhattisgarh India

Lekhraj Belchanden
Pune, Maharastra India

ABSTRACT
This paper presents the technique to generate test cases from UML State diagram, that is based on path oriented approach. Genetic algorithm is used with stack based approach to get the optimized feasible test cases. Generated test cases are effective, efficient and optimized.

Keywords
Test Case Generation; UML Statechart Diagram; Genetic Algorithm; Extended Finite State Machine.

1. INTRODUCTION
Testing is one of the essential phase in software development life cycle. Its primary goal is to find the failure so that fault can be detected and corrected. Generating good and effective test cases are essential keeping in view the increased complexity of software. It ensures the quality and improves the efficiency of the software by performing software verification and validation. Testing is very expensive and requires lot of extra efforts. It is also very time consuming. Therefore the need of the hour is to reduce time, cost and extra efforts of testing for the better performance of the software. Testing can be carried out earlier in the development process so that the developer will be able to find the inconsistencies and ambiguities in the specification and hence will be able to improve the specification before the program is written [10]. Model based testing is specification and requirement based testing. By generating test cases based on requirement and specification, we are able to improve specification before the actual execution starts. Hence, it increase the reliability in testing and saves time, effort, cost and also reduces the number of errors and faults. Unified Modelling Language has become the de facto standard for modelling and design. It is widely accepted and used by industry [11]. Statechart represents the dynamic behavior of the object. UML Statechart gives the pictorial representation of both control flow as well as data flow.

Generating test cases from UML state chart method verifies the executability during path generation which prevents generating paths which will be discarded later. Effectiveness of test cases generated from UML Statechart is measured by state coverage, transition coverage and transition pair coverage criteria. This paper presents path-oriented test data generation which is an undecidable problem [12]. Path oriented test case generation aims to generate feasible test cases that covers every possible path in the program unit under test. It is difficult to get all path coverage, as due to the presence of loop in a program, we can have infinite number of paths. And if the program contain predicates then the path is exponential to the number of predicates and it may also lead to infeasible paths. Hence, path oriented testing can be called as a NP complete problem. In this paper we have presented an approach for test case generation using path oriented approach. Generated test cases are optimized using genetic algorithm. The paper is structured as follows. Section 2 gives a brief description about related work. Section 3 presents the proposed work. Section 4 presents a case study. Section 5 gives conclusions and future work.

2. RELATED WORK
Researchers have proposed many test case generation technique for state based testing. Chow T. S. [5] presented state machine diagram automatically converted into transition tree and test sequences obtained from transition tree satisfying specified criteria. Raluca Lefticaru et al. [12] proposed a method to generate feasible test sequence by applying genetic algorithms in transition containing guards condition as algebraic predicates. A. J. Offutt et al. [13] presented a method to get the test inputs based on state specifications and selecting test cases from formal criteria. Rajappa et al. [8] proposed the method to convert the state diagram into dual graph and Eular path has been guided by GA to produced the test sequences. Mahesh Shirole et al. [9] presented a methodology for generating test sequences by converting EFSMs into extended control flow graph and generating test cases and test data by applying GA and selection method guided by data flow. Bosman [24] proposed test sequence generation based on testing coverage criteria by using partial-w method. Kansomkeat S. et al. [25] presented an approach to convert UML Statechart into testing flow graph. Generate test sequence from testing flow graph guided by transition guards. Selection of test cases did by mutation analysis. Derderian K. et al [26] presented an approach for GA based test case generation of FSM. Fitness function is based on temporal constraints and guard ranking. Kung et al. [27] proposed a method for constructing state machine by using the symbolic execution. Test cases are generated using Chow's method. Selection technique proposed in literature for EFSM: [15, 16, 17, 18] proposed the control flow based selection technique and [19, 20 , 21, 22, 23] proposed the data flow selection technique for EFSM based test sequence.

3. PROPOSED WORK
UML Statechart is drawn from software specification.UML Statechart is used for modelling the behaviour of the software to show the control flow and dataflow. It consists of the initial state, final state, states, transition, guard function. It is converted into intermediate graph. Then the predicates are found in the intermediate graph. Predicates are represented in the form of binary bits which is taken as chromosome. Based on predicates, traverse the graph using DFS for test sequence generation. Cost of each path is calculate using McCabe's

formula :

$$Cost(C) = E - N + 2 \qquad (1)$$

E: Total number of edge in the path.
N: Total number of nodes in the path.

Selection: Fitness function is calculated by the cost of path and stack weight for each path. Stack weight(SW) for each

node is calculated by using algorithm presented in [26]. We use roulette wheel method for selecting individuals. Individual probability is calculated based on the fitness of the individual. cumulative probability is calculated based on individual probability. Random numbers are generated randomly for each individuals. Individuals are selected for next generation which are having random values smaller then final probability
Fitness function

F(X) = (C * C) + SW (2)

Individual Probability

$P(X) = F(X)/\sum_{X=1}^{n} F(X)$ (3)
where n is initial population size.

Cumulative Probability
$CP(Xk) = \sum_{X=1}^{k} P(X)$ (4)

Crossover: It recombines the selected pairs of individuals from previous generation with the specified probability known as crossover probability. It produces new individuals for next
generation. It is applied to get the better individuals from the existing ones. Many crossover operators are used in GA such as single-point, double-point, shuffle crossover etc. We are using single point crossover with crossover probability of 0.8.
Mutation: It works on bits. Bits are mutate based on mutation probability. Here we are using mutation probability as 0.2. Mutation helps in introducing diversity into the genetic pool. It adds new individuals randomly to the population and thereby avoids solution being struck in the local optima.

3.1 Proposed Approach

The Proposed method is as follows:
1) Draw the UML Statechart from given software specification.
2) Convert the UML Statechart into intermediate graph.
3) Find the predicates and based on predicate traverse the graph to generate test sequence.
4) Calculate the cost of each path.
5) Apply genetic algorithm until all paths are covered.
6) For each test path
a) Calculate the fitness value F(x)=(C*C)+stack weight.
b) Calculate the individual probability.
c) Calculate the cumulative probability value.
d) Bin range is specified based on cumulative probability.
e) Random number is generated for each individual/ chromosome and the specific bin in which it lies is found out.
f) Perform crossover operation.
g) Apply mutation operation.

4. CASE STUDY

For better understanding of the above discussed approach, we have taken ATM system's one transaction state chart into consideration as shown in Fig.1. From the state chart diagram we have the intermediate graph as shown in Fig.2. There are 4 predicates in our Intermediate graph i.e 2, 3, 4, 5. Each event which is shown as edge of the intermediate graph is represented by two bits. These predicates form the chromosome the length of which is determined by the total number of bits used to represent the events. We have chosen initial population as 01010101, 00000101, 01001001, 01010100 as shown in Table II. We calculate fitness value of each path using (2). Random numbers are generated between 0 and 1. In this example GA is run for 10 iterations. For the first chromosome 01010101, Path for corresponding chromosome is 1-2-3-5-6-7-8. Cost of this path is 6-7+2=1. The stack weight of this path is calculated using Table I. The stack weight is 8+7+6+9+7+20+15 72. Fitness value of this chromosome F(x)=(1*1)+72=73.

TABLE I
Stack weight of each node in ATM system's one transaction

Nodes	K	Size, S	weight=Max stack size-K
8	7	8	1
7, 8	6	7	2
6, 7, 8	5	6	3
5, 7, 8, 6	4	5	4
4, 5, 7, 8	3	4	5
3, 7	2	3	6
2	1	2	7
1	0	1	8

After executing Iteration I to X, in tenth iteration shown in Table VII, 01010100 individual has the highest fitness value as compared to other individuals. Therefore highest priority is assigned to this path. Corresponding path according to the chromosome is 1-2-3-5-6-7-8 which act as test case. It should be tested first in path testing.

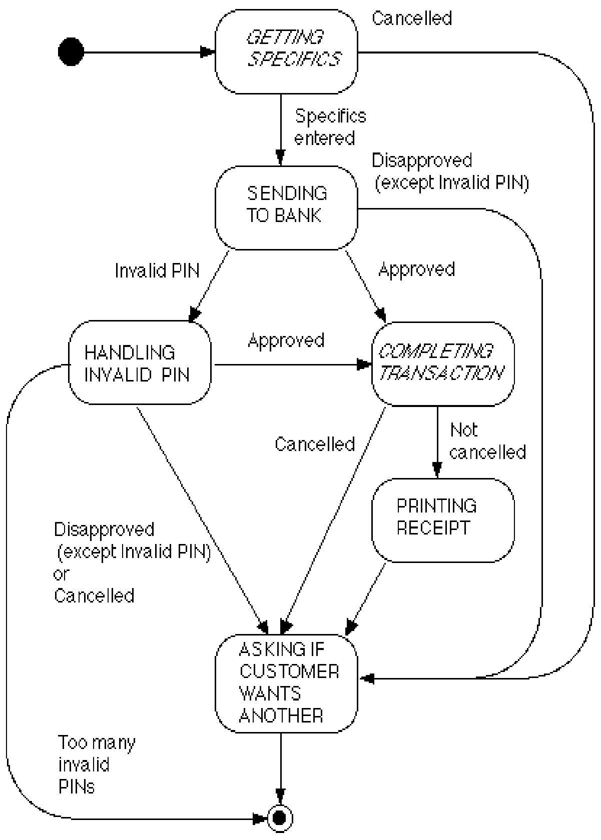

Fig. 1. UML state chart of ATM system's one transaction [29]

TABLE II
Iteration 1: Fitness of individual

S.No.	X	F(X)	IP	CP	BIN
1.	01010101	73	0.312	0.312	0-0.312
2.	00000101	53	0.226	0.538	0.3-0.538
3.	01001001	42	0.176	0.717	0.5-0.717
4.	01010100	66	0.283	1	0.7-1

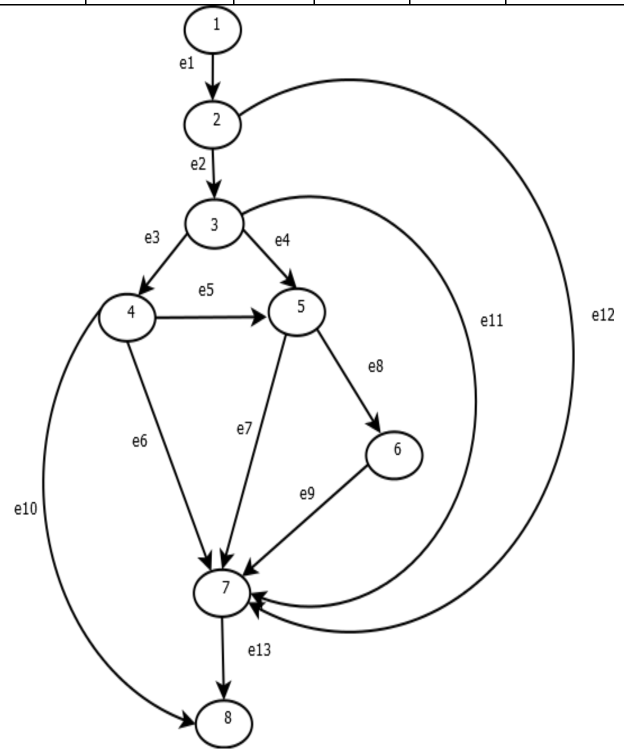

Fig. 2. Intermediate graph of ATM system's one transaction

TABLE III
Iteration 1: Selection of individuals

R	BIN	CROSSOVER	MUTATION	F(X)
0.4398	2	00000101	01010101	73
0.8147	4	01010100	01010100	66
0.9058	4	01010100	01010000	66
0.4576	2	00000101	00000000	53

TABLE IV
Iteration 2: Fitness of individual

S.No.	X	F(X)	IP	CP	BIN
1	01010101	73	0.283	0.283	0-0.283
2	01010100	66	0.256	0.539	0.2-0.539
3	01010000	66	0.256	0.195	0.7-1
4	00000000	53	0.205	1	

TABLE V
Iteration 2: Selection of individual

R	BIN	CROSSOVER	MUTATION	F(X)
0.9876	4	01010101	01010101	73
0.1973	1	00000101	00010101	43
0.7149	4	01010000	01010000	66
0.8858	1	00000000	00000000	53

TABLE VI
Iteration 10: Fitness of individual

S.No.	X	F(X)	IP	CP	BIN
1	01011000	73	0.316	0.316	0-0.316
2	01010101	73	0.316	0.632	0.3-0.632
3	01001001	42	0.182	0.814	0.6-0.814
4	00010101	43	0.816	1	0.8-1

TABLE VII
Iteration 10: Selection of individual

R	BIN	CROSSOVER	MUTATION	F(X)
0.3211	2	01010100	01010100	73
0.7956	3	01000101	01000101	61
0.8056	2	01001001	01001001	42
0.9872	4	00010101	00010101	43

5. CONCLUSION

Test case generation in path testing yields many infeasible paths which increase the time and cost of testing. We presented an approach for model based test case generation from UML State diagram using path oriented approach. Test cases generation method uses predicate coverage criteria. It is guided by genetic algorithm to get the feasible and optimal paths which act as test cases. Hence, all state coverage, all transition coverage is achieved.

6. REFERENCES

[1] G., Rumbaugh J., and Jacobson I., "The Unified Modelling Language User Guide", Addison-Wesley, 1999.

[2] Mahesh Shirole, Amit Suthar, Rajeev Kumar, "Generation of Improved Test Cases from UML State Diagram Using Genetic Algorithm", ACM, 2011.

[3] Offutt J. and Abdurazik A., "Generating Tests from UML Specifications", LNCS , vol. 1723/1999, issue 76, 1999.

[4] Kim Y.G., Hong H.S., Cho S.M., Bae D.B., Cha S.D, "Test case generation from UML State diagram", IEEE Proceedings- Software, Vol. 146, No 4, pp. 187-192, Aug, 1999.

[5] Chow T. S., "Testing software design modeled by Finite state machines", IEEE Transactions on Software Engineering, Vol 4, no. 3, pp. 178-187, May, 1978.

[6] Rajappa V., Biradar A., and Panda S., " Efficient Software Test Case Generation Using Genetic Algorithm Based Graph Theory", IEEE in Proceedings of the 1st International Conference on Emerging Trends

in Engineering and Technology (ICETET 08), pp. 298-303, May, 2008.

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

[8] Rajappa V., Biradar A., and Panda S, “Efficient Software Test Case Generation Using Genetic Algorithm Based Graph Theory”, In Proceedings of the 1st International Conference on Emerging Trends in Engineeringand Technology (ICETET 08) , pp. 298-303, Nagpur, 2008.

[9] Mahesh Shirole, Amit Suthar, Rajeev Kumar, “ Generation of Improved Test Cases from UML State Diagram Using Genetic Algorithm”, ACM, 2011.

[10] Kansomkeat, S. and Rivepiboon, W., “Automated-generating test case using UML statechart diagrams”, ACM, pp.296 300, 2003

[11] Binder, R. V., “Testing object-oriented software: a survey”, Software Testing Verification Reliability, pp.125 252, 1996.

[12] Raluca Lefticaru, Florentin Ipate, “Automatic State-Based Test Generation Using Genetic Algorithms”, IEEE Ninth International Symposium on Symbolic and Numeric Algorithms for Scientific Computing, 2008.

[13] A. J. Offutt, S. Liu, A. Abdurazik, and P. Ammann, ”Generating test data from state-based specifications, Software Testing Verification Reliability, pp.2553, 2003.

[14] Byoungju Choi, Hoijin Yoon, Jin-Ok Jeon , “A UML-based Test Model for Component Integration Test”, Workshop on Software Architecture and Component ,pp.63-70, 1999.

[15] J. C. Fernandez, C. Jard, T Jeron, L Nedelka and C. Viho, “using on the fly verification techniques for the generation of test suites ”,Computer Aided Verifiction Lecture Notes in Computer Science, springer-verlag, vol. 1102, pp. 348-359, 1996.

[16] J. Grabowski, D. Hogrefe, R. Scheurer and Z. R. Dai, “Applying SAMSTAG to the B-ISDN protocol sscope”, in Testing of Communicating Systems, Vol. 10, Chapman and Hall, 1997.

[17] S. Huang, D. Lee and M Staskauskas, “Validation based test se- quence generation for networks of EFSMs”, in Proceedings of SDL Forum, pp. 135-151, 1996.

[18] A. Kerbrat, T. Jeron and R. Groz, “Automated test generation from SDL Specifcations”, in Proceedings of IFIP FORTE/PSTV, 1996.

[19] R. E. Miller and S. Paul, “Generating conformance test Sequences for combined control and data flow of communication protocols”, in Proceedings of PSTV’ 92, pp. 13-27, 1992.

[20] B. Sarikaya, G. V. Bochmann and E. Cerny, “A test design methodology for protocol testing”, IEEE Transactions on Software Engineering, Vol. 13, No. 5, pp. 518-531, May, 1987.

[21] H. Ural, “Test sequence selection based on static data flow analysis”, Computer Communications, Vol. 10, No. 5, pp. 234-242, Oct, 1987.

[22] H. Ural and B. Yang, “A Test Sequence Selection Method for Protocol Testing”, IEEE Transactions on Communications, Vol. 39, No. 4, pp 514-523, Apr, 1991.

[23] H. Ural and A. Williams, “Test Generation by Exposing Control and Data Dependencies within System Specifcations in SDL”, in Proceedings International Conference on Formal Description Techniques, pp. 339-354, Oct 1993.

[24] Bosman O, “Object test coverage using finite state machine”, In Technology of Object-Oriented Languages and Systems, pp. 171-178, 1995.

[25] Kansomkeat S., Rivepiboon W, “Automated-Generating Test Case Using UML Statechart Diagrams”, In Proceedings of SAICSIT, pp. 296-300,2003.

[26] Derderian K., Merayo M. G., Hierons R. M., and Nunez M., “Aiding Test Case Generation in Temporally Constrained State Based Systems Using Genetic Algorithms”, In Proceedings of the 10th InternationalWork-Conference on Artificial Neural Networks, 2003.

[27] Kung D.C., Suchak N., Gao J., Hsia P, “On object state testing”, In Proceedings of Computer Software and Applications Conference, pp. 222-227, 1994.

[28] Sangeeta Sabharwal, Ritu Sibal, Chayanika Sharma, “Applying Genetic Algorithm for Prioritization of Test Case Scenarios Derived from UML Diagrams”, IJCSI International Journal of Computer Science Issues, Vol.8, Issue 3, No. 2, May, 2011.

[29] http://www.mathcs.gordon.edu/courses/cps211/ATMExample/Statecharts.html