An Ant Colony Optimization Algorithm Based Automated Generation of Software Test Cases

Saju Sankar S[1(✉)] and Vinod Chandra S S[2(✉)]

[1] Department of Computer Engineering, Government Polytechnic College, Punalur, India
tkmce@rediffmail.com
[2] Department of Computer Science, University of Kerala, Thiruvananthapuram 695581, India
vinod@keralauniversity.ac.in

Abstract. Software testing is an important process of detecting bugs in the software product thereby a quality software product is developed. Verification and Validation (V & V) activities are the effective methods employed to achieve quality. Static and dynamic testing activities are performed during V & V. During static testing, the program code is not executed while in dynamic testing (Black Box and White Box), the execution of the program code is performed. Effective test cases are designed by both these methods. Tables are employed to represent test case documentation. The most beneficial representation - State table based testing, for generating test inputs is explained with the help of state graphs and state tables. This technique is mainly employed in embedded system software testing, real time applications and web application based software product testing where time constraints are a major criteria. Automatic generation of test cases will help to reduce time overhead in testing activities. Our study is to develop optimum test cases by a modified Ant Colony Optimization (ACO) technique in an automated method and it ensures maximum coverage. The prediction model used in this paper ensures better accuracy of the design of test inputs. A comparison of the similar optimization techniques was also discussed that is used in automated test case generation. A case study of the various states during the execution of a task in an operating system has been presented to illustrate our approach.

Keywords: Ant Colony Optimization · Software · Automated · Test cases · Pheromone

1 Introduction

A software product should be reliable and measurable. These qualities are evaluated by way of effective testing activities. Hence testing is an important stage of Software Development Life Cycle (SDLC). Like SDLC, Software Testing Life Cycle (STLC) is also a process which detects bugs or faults so as to rectify them before delivery of the software product to the customer. The testing of the various phases of STLC by way of manual testing is difficult due to the overhead of time, cost and schedule slippage. Hence automated testing is complemented for majority of the testing activities [1]. This automated generation of test cases is necessary when a set of tests are to be

Y. Tan et al. (Eds.): ICSI 2020, LNCS 12145, pp. 231–239, 2020.
https://doi.org/10.1007/978-3-030-53956-6_21

done repeatedly, compatibility testing, regression testing etc. For this purpose, artificial intelligence techniques are adopted with a Meta heuristic approach. The problem of generating sets of test cases for functional testing, structural testing etc. are achieved by automated test case generation techniques [2].

In this paper, we propose an optimal algorithm for the automated generation of test cases based on Ant Colony Optimization (ACO). State table based testing - a black box testing technique is employed, which assures a higher level of functional testing of the individual modules. Here the process is categorized as – software environment modeling, test case selection, test case execution, test metrics and automated test suite reduction techniques [3].

A software product can be optimized by various algorithms such as genetic algorithms, ACO, Particle Swarm Optimization (PSO), Artificial Bee Colony (ABC) etc. The ACO algorithm is a metaheuristic intelligent optimization method. The algorithm is used for finding the optimal path in a graph. ACO is also employed for the optimization of test case generation techniques namely functional testing, Structural testing, Regression testing etc. [4]. Praveen et al. proposed an ACO based automated software testing technique which shows the generation of optimal test cases in an automated environment [5].

This paper proposes an automated method of generating test cases in functional testing technique mainly state table based testing, optimized by an ACO algorithm. The results are compared with other optimization models for effectiveness [6].

2 State Table Based Testing

State table based testing is defined as the black box testing technique in which changes in input conditions cause state changes in the application under test [7]. It is a convenient method for testing software systems where states and transitions are specified. Tables are used for representing information relating to the design of test cases. We have represented them using state transition diagrams and state tables. A state table representation has a) Finite State Machine (FSM), a model whose output is dependent on both previous and present inputs. An FSM model represents the software behavior and serves as a guide to design functional test cases. b) State transition diagram or state graph, is a software system which has different states depending on its input and time. The nodes in a state graph represent states. The nodes are connected by links denoted as transitions. With the help of nodes and transitions, a state graph is prepared. The pictorial representation of an FSM is depicted by a state graph and it represents the states that the system can assume. The state graph is converted to a tabular form known as state tables. They specify states, input events, transitions and outputs. The information contained in the state graph and the state table is converted into test cases.

The state table contains cells of valid and invalid inputs. Valid inputs causes a change of state and invalid inputs do not cause any change in transition in the state of a task. Due to the tremendous overhead of time and cost exhaustive testing is not possible. The solution is to formulate a strategy for test case reduction without affecting the functional coverage of the system under test. The genetic algorithms employed currently have to be modified for the above [8].

The method of state table based testing is depicted with a case study, showing the different states of a task in an operating system. A task or job assigned to an operating system has the states – New, Ready, Running, Waiting, Terminated. The state graph is drawn as the first step. The state graph can be represented as a directed graph G = (N, E), where N denotes the states (nodes) of the system under test and E denotes the edges or transition between the states.

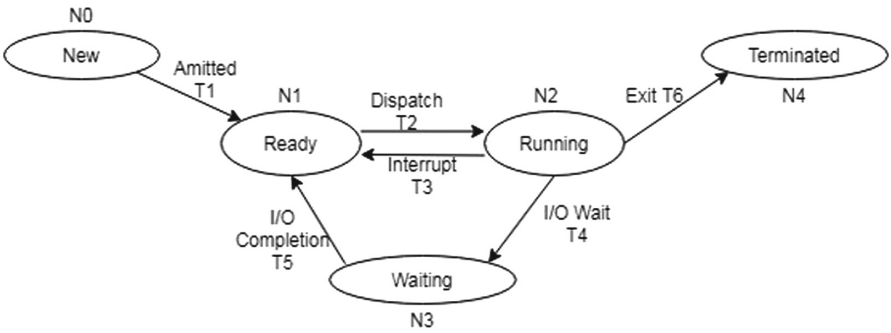

Fig. 1. State graph.

The transition events are Admitted T1, Dispatch T2, Interrupt T3, I/O Wait T4, I/O Completion T5 and Exit T6. From the state graph, a state table is prepared.

Table 1. State table

State/Input Event	Admit	Dispatch	Interrupt	I/O Wait	I/O Completion	Exit
New	Ready/T1	New/T0	New/T0	New/T0	New/T0	New/T0
Ready	Ready/T1	Running/T2	Ready/T1	Ready/T1	Ready/T1	Ready/T 1
Running	Running/T2	Running/T2	Ready/T3	Waiting/T4	Running/T2	Terminated/T6
Waiting	Waiting/T4	Waiting/T4	Waiting/T4	Waiting/T4	Ready/T5	Waiting/T4

Table 1 contains cells of valid and invalid inputs. From the state table, test cases are derived. The test cases generated are stored in a tabular form known as test case table and it contains six columns.

The test case table (Table 2) shows that 24 test cases are generated consisting of both valid and invalid test cases. We have to differentiate the valid and invalid test cases. Higher number of invalid test cases shows weak coverage. The solution can be obtained by reducing the test cases by prioritizing the test cases by following the rules.

a. Develop test cases consisting of valid test cases that cause a change of state.
b. Create test cases such that all test paths are executed at least once and the paths should be feasible.

Table 2. Test case table

Test case ID	Test source	Input		Expected Output	
		Current state	Event	Output	Next state
TC 1	Cell 1	New	Admit	T1	Ready
TC 2	Cell 2	New	Dispatch	T0	New
TC 3	Cell 3	New	Interrupt	T0	New
TC 4	Cell 4	New	I/O wait	T0	New
TC 5	Cell 5	New	I/O completion	T0	New
TC 6	Cell 6	New	Exit	T0	New
TC 7	Cell 7	Ready	Admit	T1	Ready
TC 8	Cell 8	Ready	Dispatch	T2	Running
TC 9	Cell 9	Ready	Interrupt	T1	Ready
TC 10	Cell 10	Ready	I/O wait	T1	Ready
TC 11	Cell 11	Ready	I/O completion	T1	Ready
TC 12	Cell 12	Ready	Exit	T1	Ready
TC 13	Cell 13	Running	Admit	T2	Running
TC 14	Cell 14	Running	Dispatch	T2	Running
TC 15	Cell 15	Running	Interrupt	T3	Ready
TC 16	Cell 16	Running	I/O wait	T4	Waiting
TC 17	Cell 17	Running	I/O completion	T2	Running
TC 18	Cell 18	Running	Exit	T6	Terminated
TC 19	Cell 19	Waiting	Admit	T4	Waiting
TC 20	Cell 20	Waiting	Dispatch	T4	Waiting
TC 21	Cell 21	Waiting	Interrupt	T4	Waiting
TC 22	Cell 22	Waiting	I/O wait	T4	Waiting
TC 23	Cell 23	Waiting	I/O completion	T5	Ready
TC 24	Cell 24	Waiting	Exit	T4	Waiting

c. The inputs which do not cause change in transition have to be identified from the state table.

As shown in Fig. 1, there are two paths which satisfy rule b.
1. N0 → N1 → N2 → N4
2. N0 → N1 → N2 → N3 → N1 → N2 → N4

3 ACO Algorithm for Test Case Generation

ACO is a probabilistic technique which searches for optimal path in a graph. Ant behavior is looking a shortest path between their colony and location of food source. The path is discovered by pheromone deposits when the ants move at random. More pheromone deposits on a path increases the probability of the path being followed. The path is selected based on the maximum pheromone deposit from start node and the path is analyzed for optimality [9].

The behavior of ants can be used for solving the problem. Selection of paths depends upon the theory of probability. Ants generate pheromone and deposits on the paths for further remembrance and it is a heuristic technique. The path visibility is accomplished by the level of pheromone intensity and heuristic information. The feasible path is selected based on highest pheromone level and heuristic information. The algorithm needs four parameters for selecting the valid transitions of the state graph – Probability, Heuristic information, Pheromone intensity and Visibility of the path [10].

Kamna et al. evaluated the performances of genetic algorithm (GA), Bee Colony optimization (BCO), ACO and modified ACO (m-ACO) for test case prioritization in terms of Average Percentage of Faults Detected (APFD) and Percentage of Test Cases Required (PTR) metric [6]. The metrics was evaluated by the case studies for triangle problem, quadratic equation problem etc. (Table 3).

Table 3. Comparison of GA, BCO, ACO and m-ACO for APFD and PTR.

	APFD				PTR			
	GA	BCO	ACO	m-ACO	GA	BCO	ACO	m-ACO
Triangle problem	0.88	0.95	0.93	0.97	18	12	16	12
Quadratic problem	0.86	0.91	0.89	0.93	20	18	18	16

The comparison shows that m-ACO algorithm shows better coverage than the other optimization algorithms.

In order to classify or to select the valid test cases, we used the modified algorithm 'Comprehensive Improved Ant Colony Optimization (ACIACO) for finding the effective optimization path and this path model is used to achieve highest coverage and reduce the number of iterations [11]. The establishment of transformation relationship makes effective use of ant colony algorithm for iterative optimization of test cases.

In this paper, generating the test cases from the state table is achieved by the modified ACO algorithm. The algorithm is for the effective traversing through the different states. Also the feasible test cases are generated so as to cover all transitions at least once. The procedure is given,

1. Initialize parameters.

   1.1 Set heuristic value (η).
   Initialize heuristic value of each transition in the state graph η =2.

1.2 Set pheromone intensity (ϒ).
Initialize pheromone value for each transition in the state graph (ϒ=1).
1.3 Set visited status (V_s).
Initialize $V_s = 0$ (the condition in which ant not visited any state).
1.4 Set probability (P).
Initialize $P = 0$.
1.5 Initialize $\alpha = 1$, $\beta = 1$, the parameters which controls the desirability and visibility.
1.6 Set count = maximum number of possible transitions.

2. While (count > 0)

2.1 Update the paths or visited status of the paths V_s [i] = 1.
2.2 Evaluate feasible path F (P), if any from the first node to the next node in the state graph. Else go to step 6.
2.3 Evaluate probability of the path.

$$P_{ij} = \sum_{i}^{k} ((\Upsilon ik)\hat{}\alpha * (\eta ik)\hat{} - \beta).$$

The probability has values between 0 and 1.

3. Move to the next node.

3.1 Select the destination node. Ant will follow the rules.

3.1.1 If there is self-transition from (i → i), select it.
3.1.2 Else select the transition not visited. (V_s=0).
3.1.3 Else if two or more transition having the same visited status $V_s[j] = V_s[k]$, then random selection of the node.

4. Update values of Pheromone and heuristic.

4.1 Update pheromone intensity, $\Upsilon_{ij} = (\Upsilon_{ij})^{\alpha} * (\eta_{ij})^{-\beta}$
4.2 Update heuristic, $\eta_{ij} = 2 * (\eta_{ij})$.
4.3 Update count, Decrement count by one each step.

5. Go to step 2.
6. Stop.

4 Results and Discussion

We have discussed how the test cases are minimized while maximizing coverage of testing. If there are ‘n’ states in a state graph, then there will be a maximum of ‘n*n’ test cases both feasible and infeasible generated thereby increasing the testing time overhead. For the automated generation of test cases, we used an open source testing

tool multidimensional modified condition/decision coverage which supports state graph based testing and generates test cases by traversing through the state transition graph [12].

The test cases given in Table 2, twenty four test cases both valid and invalid, which cause or do not cause change in transition from one state to another during the execution of a task has to be minimized.

4.1 Steps for Minimizing Test Cases

Wang's algorithm [13] was used for the effective reduction of test cases.

1. Calculate node coverage (NC) for each test case.
   Let NC (tc) = t_1, t_2 … t_n, where t_1, t_2, …. t_n are transitions.
2. If a number of set tc = 0, then tc is included in the effective set of test cases.
3. Final set of test cases is generated.

In our case study, the algorithm is implemented as
N = {N0, N1, N2, N3, N4} representing the nodes.
ID = {ID1, ID2, …. ID6} representing input data.
OD = {OD1, OD2, …. OD6} representing output data.
T = {T1, T2, …. T6} representing transitions.
Ti is a transition from source node to the destination node.
Ti = {Np, Nq}, where Np is the source node and Nq is the destination node.
The next step is to extract each transition,
T1 = {N0, N1} to T6 = {N2, N4}.

After completely extracting all the transitions, the next step is to generate the test cases from TC1 to TC24.

The last step is to reduce the set of generated test cases by calculating the node coverage for each test case and determining which test cases are covered by other test cases. i.e. from NC(TC1) to NC(TC24). The test cases are valid if the node coverage is empty. All other test cases are invalid and can be ignored.

Hence the six valid test cases which causes state transitions in the execution of a task in an operating system are Test Suite, TS = {TC1, TC8, TC15, TC16, TC18, TC23} as shown in Table 4.

Table 4. Cells containing valid test cases.

State/Input Event	Admit	Dispatch	Interrupt	I/O Wait	I/O Completion	Exit
New	Ready/T1					
Ready		Running/T2				
Running			Ready/T3	Waiting/T4		Terminated/T6
Waiting					Ready/T5	

This reduction in test suite helps saving of time in resource constrained software projects. The ACIACO algorithm effectively covered the criteria such as reduced test suite size, improved fault detection capability, reduced time, cost and highest coverage criteria. A comparison of the three kinds of coverage – statement coverage, branch coverage and modified condition/decision coverage shows that the modified ACIACO algorithm obtained better values as shown in Table 5. It is evident that the ACO optimization algorithm will effectively improve the quality of the test cases generated.

Table 5. The coverage of different ant numbers on ACIACO

ACIACO	Statement coverage	Branch coverage	Modified condition/Decision Coverage
8 ants	82.37%	64.30%	17.17%
32 ants	94.38%	80.23%	35.55%
56 ants	97.89%	87.77%	45.25%
88 ants	100.00%	100.00%	76.13%

5 Conclusion

We have proposed a modified ACO based approach for the automated and effective generation of valid test cases for the state transition based software testing. An FSM model is prepared. From the model, a state graph followed a state table and a test case table was generated. Also a test suite reduction algorithm is implemented thereby the optimization is achieved. ACO is a promising methodology for test case generation, selection and prioritization problem. ACO algorithm fully satisfies software coverage without having any redundancy. The power of nature inspired models in software engineering area is emerging because of its optimization outcome. The swarm based optimization is evolving in many of the software optimization fields. It is highly recommended, agent and swarm fused optimization algorithms can be used to generate efficient automated test environments.

References

1. Setiani, N., et al.: Literature review on test case generation approach. In: ICSIM 2019 Proceedings of the Second International Conference on Software Engineering and Information Management, pp. 91– 95, January 2019
2. Tahbildar, H., Kalita, B.: Automated software test data generation: direction of research. Int. J. Comput. Sci. Eng. IJCSES **2**, 99–120 (2011)
3. Swain, T.R., et al.: Generation and optimization of test cases for object oriented software using state chart diagram, CS & IT – CSCP (2012)
4. Singh, G., et al.: Evaluation of test cases using ACO with a new heuristic function: a proposed approach. Int. J. Adv. Res. Comput. Sci. Softw. Eng. IJARCSSE (2014)

5. Srivastava, P.: Structured testing using ant colony optimization. In: Proceedings of the First International Conference on Intelligent Interactive Technologies and Multimedia, pp. 203–207. IITM, ACM India, December 2010
6. Kamna, S., et al.: A Comparative evaluation of 'm-ACO' technique for test suite prioritization. Indian J. Sci. Technol. **9**(30), 1–10 (2016)
7. Li, H., Lam, C.P.: An ant colony optimization approach to test sequence generation for state based software testing. In: Proceedings of the fifth international conference on quality software, QSIC. IEEE (2005)
8. Thakur, P., Varma, T.: A survey on test case selection using optimization techniques in software testing. Int. J. Innov. Sci. Eng. Technol. IJISET **2**, 593–596 (2015)
9. Vinod Chandra, S.S.: Smell detection agent based optimization algorithm. J. Insti. Eng. **97**(3), 431–436 (2016). https://doi.org/10.1007/s40031-014-0182-0
10. Srivastava, P.: Baby: automatic test sequence generation for state transition testing via ACO, IGI, Global (2010)
11. Shunkun et al.: Improved Ant algorithm for software testing cases generation. The Sci. World J. (2014)
12. Maheshwari, V., Prasanna, M.: Generation of test case using automation in software systems – a review. Indian J. Sci. Technol. **8**(35), 1 (2015)
13. Linzhang, W., et al.: Generating test cases from UML activity diagram based on gray-box method. In: Proceedings of the 11th Asia-Pacific Software Engineering Conference (APSEC04) (2004)