World Academy of Science, Engineering and Technology
International Journal of Computer and Information Engineering
Vol:1, No:1, 2007

Software Test Data Generation using Ant Colony Optimization

Huaizhong Li and C. Peng Lam

***Abstract*—**State-based testing is frequently used in software testing. Test data generation is one of the key issues in software testing. A properly generated test suite may not only locate the errors in a software system, but also help in reducing the high cost associated with software testing. It is often desired that test data in the form of test sequences within a test suite can be automatically generated to achieve required test coverage. This paper proposes an Ant Colony Optimization approach to test data generation for the state-based software testing.

***Keywords*—** Software testing, ant colony optimization, UML.

I. Introduction

Software testing remains the primary technique used to gain consumers' confidence in the software. The process of testing any software system is an enormous task which is time-consuming and costly [1]. The development of techniques that will also support the automation of software testing will result in significant cost savings. The application of artificial intelligence (AI) techniques in Software Engineering (SE) is an emerging area of research that brings about the cross-fertilization of ideas across two domains. A number of published works, for examples [2] and [12], have begun to examine the effective use of AI for SE related activities which are inherently knowledge intensive and human-centred.

It has been identified that one of the SE areas with a more prolific use of AI techniques is software testing. The focus of these techniques involves the applications of genetic algorithms (GAs), for examples [8] and [10]. Other AI techniques used for test data generation included the AI planner approach [7] and simulated annealing [13]. Recently, Ant Colony Optimization (ACO) is starting to be applied in software testing [3, 10]. Namely Boerner and Gutjahr [3] described an approach involving ACO and a Markov Software Usage model for deriving a set of test paths for a software system, and McMinn and Holcombe [10] reported on the application of ACO as a supplementary optimization stage for finding sequences of transitional statements in generating test data for evolutionary testing. However, the results obtained so far are preliminary, and none of the reported results directly addresses specification-based software testing.

The authors are with the School of Computer and Information Science, Edith Cowan University, Perth, WA 6050, Australia (phone: 61-8-93706751; fax: 61-8-93706100; e-mail: {h.li,c.lam}@ecu.edu.au).

In this paper we propose to use UML Statechart diagrams and ACO for test data generation. The advantages of the proposed approach are: 1). our approach directly uses the standard UML artifacts created in software design processes; 2). the automatically generated test sequence is always feasible, non-redundant and achieves the all state coverage criterion. Section 2 briefly discussed about software testing and ACO. Section 3 presents the proposed ACO approach to test data generation, and the conclusion is found in Section 4.

II. Software Testing and ACO

Three main activities normally associated with software testing are: (1) test data generation, (2) test execution involving the use of test data and the software under test (SUT) and (3) evaluation of test results. The key question addressed in software testing is how to select test cases with the aim of uncovering as many defects as possible. Of the three activities mentioned above, test data generation and evaluation of test results are the most labour intensive and thus would benefit most from automation.

The process of test data generation involves activities for producing a set of test data that satisfied a chosen testing criterion. Horgan [6] has previously shown that test cases selected on the basis of test adequacy criteria are more effective at discovering defects in the SUT. While it is possible to manually generate an effective set of test cases, the more cost-effective approach is to automate the test data generation while ensuring that the given criterion is met.

A variety of techniques for test data generation have been developed previously and these can be categorised as structural and functional testing. Most existing work in automated test data generation involving AI uses GAs and is mainly in the areas of structural test data generation and temporal behaviour testing. The ultimate aim of using GAs for structural testing is to generate a set of test cases that provides the highest possible coverage of a given structural testing criterion. The test objectives are expressed numerically and are used subsequently to formulate a suitable fitness function that evaluates the suitability of the generated test cases.

ACO simulates the behavior of real ants. The first ACO technique is known as Ant System [4] and it was applied to the travelling salesman problem. Since then, many variants of this technique have been produced. ACO is a probabilistic technique that can be applied to generate solutions for combinatorial optimisations problems. The artificial ants in the algorithm represent the stochastic solution construction procedures which make use of (1) the dynamic evolution of

the pheromone trails that reflects the ants' acquired search experience and (2) the heuristic information related to the problem in hand, in order to construct probabilistic solutions.

In order to apply ACO to test case generation, a number of issues need to be addressed, namely, (1) transformation of the testing problem into a graph; (2) a heuristic measure for measuring the "goodness" of paths through the graph; (3) a mechanism for creating possible solutions efficiently and a suitable criterion to stop solution generation; (4) a suitable method for updating the pheromone; and (5) a transition rule for determining the probability of an ant traversing from one node in the graph to the next. In the next section, we present an ACO approach to automatically generate test data from UML Statechart diagrams for state-based software testing.

III. TEST DATA GENERATION USING ACO

State-based testing is a frequently used approach in software testing. There are two major problems associated with state-based software testing: (1) some of the generated test cases are infeasible; (2) inevitably many redundant test cases have to be generated in order to achieve the required testing coverage. To our knowledge, no systematic and efficient strategy has been reported to successfully deal with the automatic generation of feasible test cases for state-based software testing.

The "all state testing coverage" requirement is commonly used in state-based software testing. A test suite is said to achieve all states coverage if every state is accessed at least once by a test case within. A test suite for state-based software testing consists of a set of test cases in the following form:

$$S_A \rightarrow S_B \rightarrow S_C \rightarrow S_D \rightarrow S_A \rightarrow S_D \rightarrow S_A \rightarrow S_C \rightarrow S_B$$

or alternatively, $\{S_A, S_B, S_C, S_D, S_A, S_D, S_A, S_C, S_B\}$ for short notation, where S_A, S_B, S_C, S_D are the states in the corresponding UML Statechart diagram, and $\rightarrow$ represents a transition between the two states.

The proposed approach deals with the automatic generation of test suites from the UML Statechart diagrams for state-based software testing, and uses the all state testing coverage as test adequacy requirement. Specifically, the generated test suite has to satisfy three criteria:

1. *All state coverage*
2. *Feasibility* – Each test case represents a feasible path in the corresponding Statechart diagram
3. *Optimality* – Test suite contains non-redundant test cases which have the shortest possible test sequences

A directed graph is defined as G = (V, E) where V is a set of vertices of the graph and E a set of edges of the graph. A UML Statechart can be viewed as a directed graph where the vertices are the states of the Statechart diagram, and the edges are the transitions between the states. We have developed a tool to automatically convert a Statechart diagram to a directed graph. For example, a typical UML Statechart diagram, the Coffee and Cocoa Vendor Machine (CCVM), can be converted into a directed graph CCVM = (S, T), where S is the state set and T is the transition set. The original CCVM Statechart diagram and the converted graph are shown in Figure 1 and Figure 2 respectively. We will use the CCVM example to help demonstrating our approach.

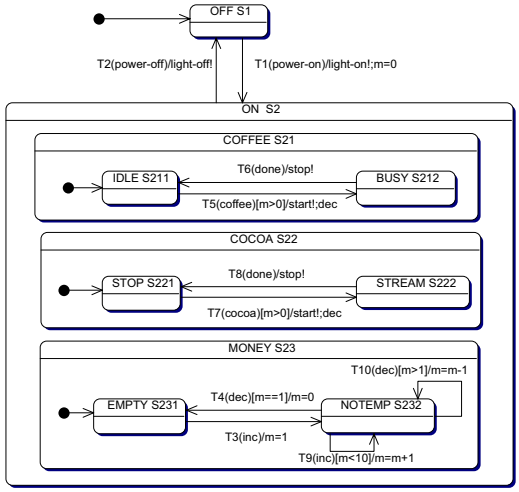

Figure 1. The UML Statechart diagram for CCVM

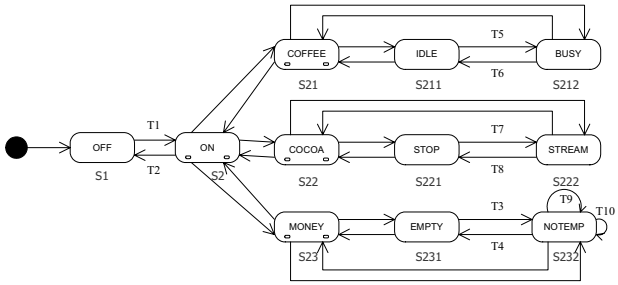

Figure 2. Converted CCVM graph CCVM = (S, T)

The converted graph is a directed, dynamic graph in which the edges (transitions in Statechart sense) may dynamically appear or disappear based on the evaluation of their guard conditions. Unlike the work in [3] and [10], it is difficult to apply the original ACO algorithms in [4] and [5] to this type of dynamic graphs to generate test data for the corresponding state-based software testing problems.

We consider the problem of simultaneously dispatching a group of ants to cooperatively search a directed graph G. The ants in our paradigm can sense the pheromone traces at the current vertex and its directly connected neighbouring vertices, and leave pheromone traces over the vertices.

An ant *k* at a vertex α of the graph is associated with a four tuple (S_k, D_k, T_k, P):

- *Vertex Track Set* $S_k = \{S_i\}$ keeps a vertex track of the ant's walking history
- *Target Set* D_k indicates those vertices which are always connected to the current vertex α. For the Statechart diagrams, target sets only exist for the super-states of the composite states, and the target set for a super-state contains the current status of the composite state. For example, the ON super-state for the CCVM graph contains the information ON(COFFEE, COCOA,

MONEY). Therefore, the target set for vertex ON is {COFFEE, COCOA, MONEY}. The target set for vertex COCOA is either {STOP} or {STREAM}, but not both because the super-state for these two sub-states only keeps the current status of the composite state

- *Connection Set* $T_k = \{T(V_i)\}$ represents the direct connections of the current vertex α with the neighbouring vertices. Direct connection means that there is only one directed edge from the current vertex to the destination vertex. T_k also documents all the edges spanning from the current vertex. $T(V_i) = 0$ means that the two vertices α and V_i are always connected, $T(V_i) = 1$ means that the two vertices appear to be connected for the current ant at the current vertex α, and $T(V_i) = -1$ indicates that the two vertices are not connected for the current ant, at the current vertex α and for the current time. For the corresponding UML Statechart diagram, the following situations appear:
  - $T(V_i) = 0$ means that either V_i is contained in the target set for α, or α is contained in the target set for V_i. This represents two vertices which are a super-state and its targeted sub-state;
  - $T(V_i) = 1$ means that the transition between the two states is evaluated to be feasible;
  - $T(V_i) = -1$ means that there is no transition between two states α and V_i, or the transition between the two states is infeasible, or V_i is not in the target set of the vertex α

  For examples, for an ant *k* at the state EMPTY (S_{231}) in the CCVM graph in Figure 2, $T_k = \{T(S_{23}), T(S_{232})\}$. If T_3 is feasible, $T_k = \{0, 1\}$, otherwise $T_k = \{0, -1\}$; for an ant *k* at the state COFFEE (S_{21}) with target set {IDLE}, $T_k = \{T(S_2), T(S_{211}), T(S_{212})\} = \{0, 1, -1\}$ since S_{212} is not contained in the target set for S_{21} at this stage
- *Pheromone Trace Set* $P = \{P(V_i)\}$ represents the pheromone levels at the neighbouring vertices which are feasible to an ant at the current vertex. The pheromone left by previous ants over the graph will not decrease, and the succeeding ants will use the remaining pheromone level to adjust their exploration.

Each ant keeps its own private sets S_k, D_k, and T_k, while the public set P is left on the graph for all of the ants to share. Ants can sense the pheromone levels on the graph, and modify P in the exploration of the graph. The following algorithm is proposed for an ant to explore the directed graph:

Algorithm for ant *k*

1. Evaluation at vertex α
   - *Update the Track* - Push the current vertex α into the track set S_k
   - *Evaluate Connections* - Evaluate all connections to the current vertex α to determine T_k. The procedure involves evaluation of all possible transitions from the current states α to other neighbouring states, using the state-transition table associated with the UML Statechart diagram
   - *Sense the Trace* - For the non-negative connections in T_k, the ant senses and gathers the corresponding pheromone levels P at the other ends of the connections
2. Move to next vertex
   - *Select Destination* - The following prioritized rules are used in ant's selection:
     - Select the vertex V_i with the lowest pheromone level $P(V_i)$ sensed from the current vertex α
     - If vertices V_i and V_j shares the same lowest pheromone level $P(V_i) = P(V_j)$, but $T(V_i) = 0$ and $T(V_j) = 1$, select V_i
     - If vertices V_i and V_j shares the same lowest pheromone level $P(V_i) = P(V_j)$ and $T(V_i) = T(V_j)$, randomly select one vertex

     Destination β is the vertex selected using the above rules
   - *Update Pheromone* - Update the pheromone level for the current vertex α to

     $P(\alpha) = \max(P(\alpha), P(\beta)+1)$ if $T(\beta) = 1$

     or

     $P(\alpha) = \max(P(\alpha), P(\beta)+1)+TP$ if $T(\beta) = 0$

     where TP is a high pheromone level which decays in one iteration of the two steps, namely, TP quickly decays to 0 before ant's next move at the end of Step 2
   - *Move* - Move to the destination vertex β, set α := β, and return to Step 1.

In the above algorithm, TP is used to encourage an ant to perform forward exploration and to prevent an ant from immediately moving back to the previously stayed vertex. In the Statechart sense, TP prevents an ant from doing redundant moves between a super-state and its sub-state.

Multiple ants can be sent to explore the converted graphs simultaneously in order to accelerate the exploration process and to produce shorter test sequences. Each ant is assigned a unique ID which represents its priority in the cooperative team. The following rule is used in the cooperative exploration by the ants:

Rule: When there are two or more ants at a vertex α, the ants have to leave α according to their priorities in the team. An ant with a higher priority leaves and sets pheromone level for α first, followed by lower priority ants.

The final pheromone level left over α is the highest one set by all ants.

Similar to [14], it can be shown that all vertices can be visited in limited steps (upper bound). The details however are omitted due to space limitation. The algorithm for an ant terminates when one of the following two conditions is satisfied:

- The union of all track sets $\cup S_k$ contains all vertices of the graph which means the coverage criterion has been satisfied, i.e., all states have been visited at least once;
- The search upper bound has been reached. In this case, this group of ants fails to find a solution which achieves the required coverage. More ants will have to be

deployed in order to find a solution.

The final optimal solution can be obtained by examining all of the solution candidates created by ant exploration.

Next we demonstrate the proposed algorithm using the CCVM given in Figure 2. For illustration purpose, we send two ants, namely Ant 1 and Ant 2, to simultaneously walk the directed CCVM graph. Both ants start from the default state S_1. Using the proposed algorithm, two ants' traces are recorded in their trace set S_1 and S_2. Initially both ants have to make a random decision at vertex S_2 according to our algorithm. Assume that both ants randomly select to move from S_2 to S_{23}, one variation of their traces for the CCVM Statechart, which is illustrated using the dotted and the dashed traces respectively in Figure 3, turns out to be the optimal test suite which contains two test cases and satisfies all three required criteria:

Ant 1: {S_1, S_2, S_{23}, S_{231}, S_{232}, S_{23}, S_2, S_{22}, S_{221}, S_{222}}

Ant 2: {S_1, S_2, S_{23}, S_{231}, S_{232}, S_{23}, S_2, S_{21}, S_{211}, S_{212}}

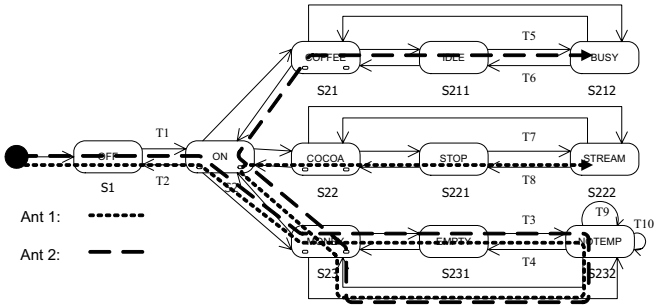

Figure 3 Optimal Solution

If in the random decision stage, Ant 1 decides to move from S_2 to S_{23}, but Ant 2 decides to move from S_2 to S_{21} instead, different traces are created for both ants. One solution candidate is illustrated in Figure 4 which also provides a feasible test suite for the CCVM graph:

Ant 1: {S_1, S_2, S_{23}, S_{231}, S_{232}, S_{23}, S_2, S_{22}, S_{221}, S_{222}}

Ant 2: {S_1, S_2, S_{21}, S_{211}, S_{21}, S_2, S_{22}, S_{211}, S_{22}, S_2, S_{23}, S_{231}, S_{232}, S_{23}, S_2, S_{21}, S_{211}, S_{212}}

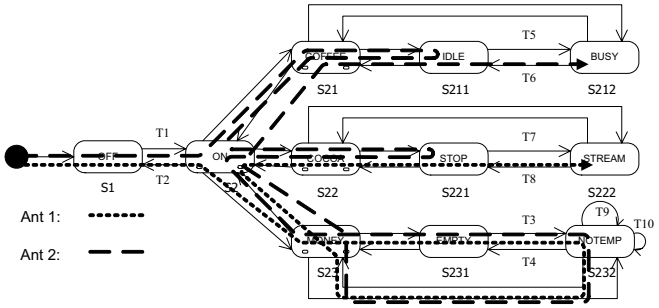

Figure 4 Solution Candidate

However, it can be easily observed that this alternative test suite doesn't satisfy the third criterion as the second test case in the suite contains many redundant nodes.

Note that instead of two ants, a single ant or more than two ants can also be used to explore the CCVM graph. However, none of these cases provides solutions which satisfy the required criteria. For example, one of the best possible solution candidates created by single ant exploration is:

Ant: {S_1, S_2, S_{23}, S_{231}, S_{232}, S_{23}, S_2, S_{21}, S_{211}, S_{212}, S_{21}, S_2, S_{22}, S_{221}, S_{222} }

While this test suite contains the minimum number of test cases, the test case within the test suite has longer sequence length which violates the third requirement criterion. Similar observation can be drawn when the number of ants exceeds 3.

IV. Conclusion

This paper presented an ACO approach to test sequence generation for state-based software testing. A directed dynamic graph is created to represent the Statechart model structure of a software system under test. Using the developed ACO algorithm, a group of ants can effectively explore the graph and generate optimal test data to achieve test coverage requirement.

References

[1] Binder, R. V., *Testing Object-oriented Systems: Models, Patterns, and Tools*, Addison-Wesley. 2000.

[2] Briand, L. C.,"On the many ways Software Engineering can benefit from Knowledge Engineering", Proc. 14th SEKE, Italy, pp. 3-6, 2002.

[3] Doerner, K., Gutjahr, W. J., "Extracting Test Sequences from a Markov Software Usage Model by ACO", LNCS, Vol. 2724, pp. 2465-2476, Springer Verlag, 2003.

[4] Dorigo M., Maniezzo, V., Colorni, A., "Positive Feedback as a Search Strategy", Technical Report No. 91-016*,* Politecnico di Milano, Italy, 1991.

[5] Dorigo M., Maniezzo, V., Colorni, A., "The Ant System: Optimization by a Colony of Cooperating Agents", *IEEE Transactions on Systems, Man, and Cybernetics-Part B,* Vol. 26, No.1, pp.29-41, 1996.

[6] Horgan, J., London, S., and Lyu, M., "Achieving Software Quality with Testing Coverage Measures", *IEEE Computer*, Vol. 27 No.9 pp. 60-69, 1994.

[7] Howe, A. E., Mayrhauser A. V., and Mraz, R. T., "Test Case Generation as an AI Planning Problem", *Automated Software Engineering, Vol. 4, pp 77-106, 1997.*

[8] Li, H., Lam, C.P., "Optimization of State-based Test Suites for Software Systems: An Evolutionary Approach", International Journal of Computer & Information Science, Vol. 5, No. 3, pp. 212-223, 2004.

[9] McMinn, P., "Search-based Software Test Data Generation: A Survey", Software Testing, Verification and Reliability, Vol.14, No. 2, pp. 105-156, 2004.

[10] McMinn, P., Holcombe, M., "The State Problem for Evolutionary Testing", Proc. GECCO 2003, LNCS Vol. 2724, pp. 2488-2500, Springer Verlag, 2003.

[11] Pargas, R. P., Harrold, M. J., and Peck, R., "Test-Data Generation Using Genetic Algorithms", Software Testing, Verification and Reliability, Vol. 9, pp. 263 - 282, 1999.

[12] Pedrycz, W., Peters, J. F., Computational Intelligence in Software Engineering, World Scientific Publishers, 1998.

[13] Tracey, N., Clark, N., .Mander K., and McDermid, N., "A Search Based Automated Test Data Generation Framework for Safety Critical Systems", in Systems Engineering for Business Process Change (New Directions), Henderson P., Editor, Springer Verlag, 2002.

[14] Wagner, I. A., Lindenbaum, M., Bruckstein, A. M., "ANTS: Agents, Networks, Trees, and Subgraphs", Special issue on Ant Colony Optimization (M. Dorigo, G. Di Caro, T.Stützle (eds)), *Future Generation Computer Systems*, Vol. 16, No. 8, pp. 915-926, North Holland, June 2000.