IJCSE International Journal of Computer Sciences and Engineering Open Access
Review Paper Volume-6, Issue-3 E-ISSN: 2347-2693

Application of ACO in Model Based Software Testing: A Review

Navneet Kaur*, Jaskaranjit Kaur[2], J.S.Budwal[3]

[1*]Dept. of Computer Science & IT, Lyallpur Khalsa College, Jalandhar, Punjab, India
[2]Dept. of Computer Science & IT, Lyallpur Khalsa College, Jalandhar, Punjab, India
[3]Dept. of Computer Science, GSSS Hazara, Jalandhar, Punjab, India

**Corresponding Author:* saininavneet@gmail.com, Tel.: 98157-76890*

Available online at: www.ijcseonline.org

Received: 18/Feb//2018, Revised: 21/Feb2018, Accepted: 23/Mar/2018, Published: 30/Mar/2018

Abstract— Software Testing is the process of testing the software in order to ensure that it is free of errors and produces the desired outputs in any given situation. Properly generated test suites may not only locate the defects in software systems, but also help in reducing the high cost associated with software testing. Model based software testing is an approach in which software is viewed as a set of states. There are a number of models of software in use today, a few of which make good models for testing. This paper introduces model-based testing and discusses its tasks in general terms with finite state models. Ant colony optimization (ACO) is best suited to model based software testing like finite state machines, state charts, the unified modeling language (UML) and Markov chains.

Keywords— Ant Colony, Optimization, Model Based Software Testing, Optimal Path, State Machine.

I. INTRODUCTION

Software testing remains the primary technique used to gain consumers' confidence in the software. Software quality remains a major problem for the software industry worldwide. Unfortunately, it is always a time-consuming and costly task to test a software system. Obviously, techniques that support the automation of software. A system *fails* when it does not meet its specification. The purpose of testing a system is to discover faults that cause the system to fail rather than proving the code correctness, which is often an impossible task. In the software testing process, each *test case* has an identity and is associated with a set of inputs and a list of expected outputs. Testing will result in significant cost saving. The application of artificial intelligence (AI) techniques in Software Engineering (SE) is an emerging area of research. Number of researchers around the world did the work on software testing using artificial inelegance; they examine the effective use of AI for SE related activities which are inherently knowledge intensive and human-centred. These issues necessitate the need to investigate the suitability of search algorithms, e.g. simulated annealing, genetic algorithms, and ant colony optimization as a better alternative for developing testing the data [6].

Ant Colony Optimization provides a general-purpose search methodology, which uses principles of the swarm intelligence [1]. In this paper, we discussed a new, computationally intelligent approach to improving the effectiveness of a given test set by eliminating "bad" test cases that are unlikely to expose any error, while increasing the number of "good" test cases that have a high probability of producing an erroneous output [7].

II. MODEL BASED SOFTWARE TESTING

A model of software is a depiction of its behaviour. Behaviour can be described in terms of the input sequences accepted by the system, the actions, conditions, and output logic, or the flow of data through the application's modules and routines. In order for a model to be useful for groups of testers and for multiple testing tasks, it needs to be taken out of the mind of those who understand what the software is supposed to accomplish and written down in an easily understandable form. It is also generally preferable that a model be as formal as it is practical. With these properties, the model becomes a shareable, reusable, precise description of the system under test. There are various examples of model based software testing are available, some of these are finite state machines, state charts, the unified modeling language (UML) and Markov chains.

The requirement common to most styles of testing is a well-developed understanding of what the software accomplishes, and MBST is no different. Forming a mental representation of the system's functionality is a prerequisite to building models. The following are some guidelines to the activities that may be performed in MBST.

1. Determine the components/features that need to be tested based on test objectives. No model is ideal to completely describe a complex or large system. Determining what to

model for testing is a first step in keeping MBST manageable.
2. *Start exploring target areas in the system.* If development has already started, acquiring and exploring the most recent builds with the intent of learning about functionality.
3. *Gather relevant, useful documentation.* Like most testers, model-based testers need to learn as much as possible about the system. Reviewing requirements use cases, specifications, miscellaneous design documents, user manuals.
4. *Establish communication with requirements, design, and development teams if possible.* Talking things over with other teams on the project can save a lot of time and effort, particularly when it comes to choosing and building a model.
5. *Identify the users of the system.* Each entity that either supplies or consumes system data, or affects the system in some manner needs to be noted.
6. *Enumerate the inputs and outputs of each user.* In some contexts, this may sound like an overwhelming task, all users considered, and it is tedious to perform manually. At this point, because automation is usually intended in MBST, the tester needs to begin investigating means of simulating inputs and detecting output.
7. *Study the domains of each input.* In order to generate useful tests in later stages, real, meaningful values for inputs need to be produced.
8. *Document input applicability information.* To generate useful tests, the model needs to include information about the conditions that govern whether an input can be applied by the user.
9. *Document conditions under which responses occur.* A response of the system is an output to one its users or a change in its internal data that affects its behaviour at some point in the future. The conditions under which inputs cause certain responses need to be studied.
10. *Study the sequences of inputs that need to be modelled.* This vital activity leads straight to model building and is where most of the misconceptions about the system are discovered (or worse, formed).
11. *Understand the structure and semantics of external data stores.* This activity is especially important when the system keeps information in large files or relational databases. Knowing what the data looks like and what it means allows weak and risky areas to be exposed to analysis.
12. *Understand internal data interactions and computation.* As with the previous activity, this under test and consequently the model's capabilities of generating bug-revealing test data.

III. ANT COLONY OPTIMIZATION

Ant Colony Optimization (ACO) is a paradigm for designing meta-heuristic algorithms for combinatorial optimization problems. The first algorithm which can be classified within this framework was presented in 1991 and, since then, many diverse variants of the basic principle have been reported in the literature. The essential trait of ACO algorithms is the combination of a priori information about the structure of a promising solution with a posterior information about the structure of previously obtained good solutions[3][4]. Meta-heuristic algorithms are algorithms which, in order to escape from local optima, drive some basic heuristic: either a constructive heuristic starting from a null solution and adding elements to build a good complete one, or a local search heuristic starting from a complete solution and iteratively modifying some of its elements in order to achieve a better one. The meta-heuristic part permits the low level heuristic to obtain solutions better than those it could have achieved alone, even if iterated. Usually, the controlling mechanism is achieved either by constraining or by randomizing the set of local neighbour solutions to consider in local search. The characteristic of ACO algorithms is their explicit use of elements of previous solutions [5].
The ACO meta-heuristic framework can be applied to discrete optimization problems having a finite set of components with connections between these components (with associated costs). More than this there is also a set of constraints on what components and connections compose a feasible solution, and each feasible solution is said to have a quality which is calculated by a function of all of the component costs.
During food hunting, ants leave pheromone on the travelled paths, and the shortest path will be discovered through teamwork and pheromone evaporation process [8][9]. Assume in the beginning of food foraging, ants choose their paths in random toward the direction of the food source, even when they come to a fork in the road, so any possible path would remain pheromone odour (Figure 1). In the return trip, since the pheromone evaporates in different evaporation rates according to the length of paths, it leads to different amount of residual pheromone. Therefore, the longer the path, the more pheromone evaporation, the less residual pheromones. The other follower ants will choose the shorter path according to the amount of residual pheromones. Through the time evolution of the group cooperation, ants will eventually choose the shortest path. the functioning of an ACO algorithm can be summarized as follows. A set of computational concurrent and asynchronous agents (a colony of ants) moves through states of the problem corresponding to partial solutions of the problem to solve. They move by applying a stochastic local decision policy based on two parameters, called *trails* and *attractiveness*. By moving, each ant incrementally constructs a solution to the problem. When an ant completes a solution, or during the construction phase, the ant evaluates the solution and modifies the trail value on the components used in its solution. This pheromone information will direct the search of the future ants. Furthermore, an ACO algorithm includes two more mechanisms: *trail evaporation* and, optionally, *daemon actions*. Trail evaporation decreases all trail values over time,

in order to avoid unlimited accumulation of trails over some component. Daemon actions can be used to implement centralized actions which cannot be performed by single ants, such as the invocation of a local optimization procedure, or the update of global information to be used to decide whether to bias the search process from a non-local perspective.

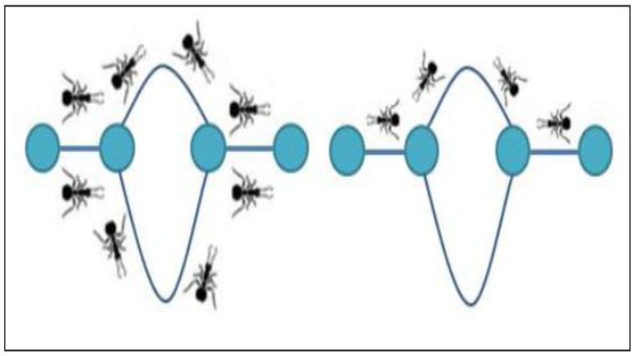
Figure 1:- Random path and shortest path

IV. MBST AND ACO

The difficulty of generating tests from a model depends on the nature of the model. Models that are useful for testing usually possess properties that make test generation effortless and, frequently, automatable [2] [14]. For some models, all that is required is to go through combinations of conditions described in the model, requiring simple knowledge of combinatory. In the case of finite state machines, it is as simple as implementing an algorithm that randomly traverses the state transition diagram. The sequences of arc labels along the generated paths are, by definition, tests. For example, in the state transition diagram below, the sequence of inputs “a, b, d, e, f, i, j, k” qualifies as a test of the represented system.[10]
It is not always feasible to generate all the possible test cases due to paucity of time, cost and other factors. Hence there is a need to automate the testing process that can generate the effective test paths (by prioritizing different paths) thereby reducing the overall cost of the testing process and increase the probability of finding the errors in the software systems.
A single test path cannot be used to detect all the possible defects in the software. ACO algorithm was initially applied to find a solution to the travelling salesman problem. The main idea of this paper is to incorporate the self organizing behaviour of ant with the artificial agents to solve the complex computational problem of finding the most optimal path among the different generated paths.

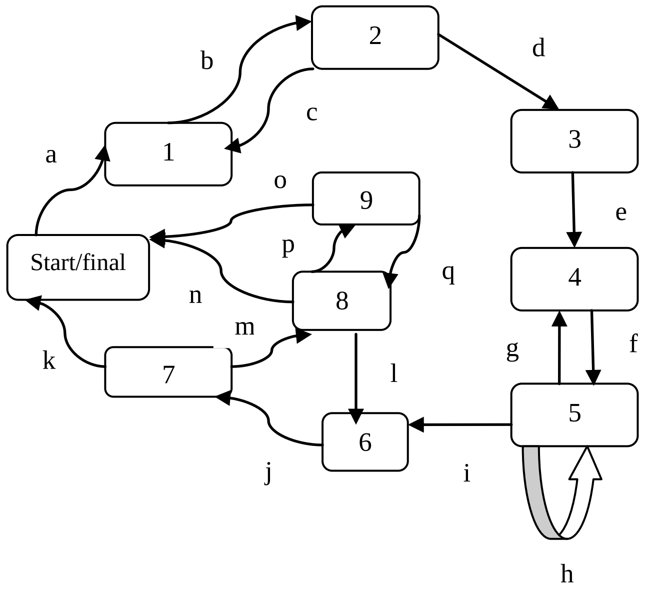

Figure 2- A state machine

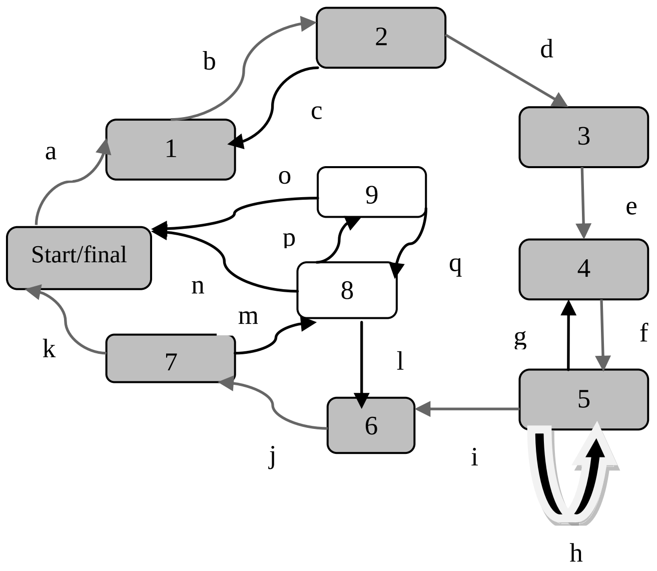

Figure 3:- Test path in state machine

Real world ants, wander randomly, upon finding food return to their colony while laying down pheromone trails. If other ants find such a path, they will move on that path rather than move randomly [11][12]. Thus, when an ant finds a good (shorter) path from the colony to the food source, the pheromone level will be high in that path and all the ants will eventually follow that single path. Therefore, the selection procedure for a path is based on the probability of finding a path that has the highest pheromone level and heuristic knowledge.

V. ATTRIBUTES USED FOR GENERATING AND PRIORITIZING THE PATHS

In this algorithm ant has ability to collect the Knowledge of all feasible paths (a path from source to destination) from its current state. This approach is defined in feasibility set of path (Fij).The ant also has other information about path: pheromone level on path (τij), Heuristic information for the paths (ηij), Learned Value (Lij), visited states with the help of visited status (Vs) and last is probability parameter P. When transitions between two nodes are exits and ant explored that transition then ant will update the pheromone level as well as heuristic value. Pheromone level is increased according to last pheromone level and heuristic information but heuristic information, is update only on the basis of previous heuristic information[13].

An ant p at a vertex i and another vertex j which is directly connected to "i", it means there is a path between the vertices i and j i.e. (i->j). In the graph this path associated with six tuple Fij (p), τij (p), ηij (p), Vs(p) and Pij (p), Lij(p) where (p) shows that values of tuple associated with ant p. All description about these attribute is given below: Prioritization of the paths is done based on the pheromone level of the edges of the corresponding path. The edges are associated with Fij (p), τij (p), ηij (p), Lij (p) and Pij (p) where (p) shows the values of tuple associated with ant p. Description about the attributes:

- Feasible path set: If there is a direct edge from vertex i to vertex j then Fij = 1, otherwise Fij = 0
- Pheromone τij: τij represents pheromone level from vertex i to vertex j and is being constantly updated as the paths are traversed.
- Heuristic ηij: represents the visibility of the path for an ant and is used to calculate the probability for the ant to take a particular path
- Learned Lij: Lij indicates the possibility of finding new paths if the vertex j is chosen for traversal.
- Probability Pij: Pij indicates the probability of ant to choose vertex j for traversal from vertex i. Probability depends on the feasible set, heuristic and the pheromone level of the corresponding path.

VI. CONCLUSIONS

Ant Colony Optimization has been and continues to be a fruitful paradigm for designing effective combinatorial optimization solution algorithms. This paper presented an ant colony optimization approach to test sequence generation for state-based software testing. Using this algorithm, a group of ants can effectively explore the UML State chart diagrams and automatically generate test sequences to achieve the test adequacy requirement. The discussed technique has proven to be effective in generating optimal set of test cases from a Markov chain based usage model. Especially with the growth in software with extensive graphical interfaces, usage modeling will prove to be very productive.

REFERENCES

[1] Huaizhong LI, C. Peng LAM, "*An Ant Colony Optimization Approach To Test Sequence Generation For State Based Software Testing",* Fifth International Conference on Quality Software (QSIC'05), pp. 255-264, 2005.

[2] Dan Liu,Xuejun Wang, Jianmin Wang, "*Automatic Test Case Generation Based On Genetic Algorithm*", Journal of Theoretical and Applied Information Technology, Vol. 48 No. 1, pp. 411-416, 2013.

[3] Praveen Ranjan Srivastava1, Nitin Jose, Saudagar Barade, Debopriyo Ghosh, "*Optimized Test Sequence Generation From Usage Models Using Ant Colony Optimization*", IJSEA, Vol.1, No.2, pp. 14-28, 2010.

[4] Navneet Kaur, Jaspreet Singh Budwal, "*Hybrid Approach to Retrieval of Reusable Component from a Repository Using Genetic Algorithms and Ant Colony*", International Conference on Genetic and Evolutionary Method, Las Vegas, Nevada, USA , pp.147-152, 2008.

[5] Rafael S. Parpinelli1, Heitor S. Lopes1, And Alex A. Freitas2, "*Data Mining With An Ant Colony Optimization Algorithm", IEEE Transactions on Evolutionary Computation"*, Vol. 6, Issue: 4, pp. 321 – 332, 2002.

[6] Praveen Ranjan Srivastava1 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, October 2009.

[7] Navneet Kaur, Jaspreet Singh Budwal ,"*Intelligent Web Search Optimization with reference to Mutation Operator of Genetic and Cultural Algorithms Framework*", 2014 IEEE International Conference on Advanced Communication, Control and Computing Technologies (ICACCCT), pp. *619-623,* 2014.

[8] Vittorio Maniezzo, Luca Maria Gambardella, Fabio de Luigi, "*Ant Colony Optimization*", Studies in Fuzziness and Soft Computing book series STUDFUZZ, Vol 141, pp 101-121.

[9] Dorigo, M., Di Caro, G. & Gambardella, L. M. *"Ant algorithms for discrete optimization"*. Artificial Life Vol 5, No. 2, 137-172, 1999.

[10] Huaizhong Li and C. Peng Lam, "*Software Test Data Generation using Ant Colony Optimization*", International Journal of Computer, Information Science and Engineering Vol:1 No:1, pp 126-129, 2007.

[11] M. Dorigo, A. Colorni and V. Maniezzo, *"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.

[12] L.M Gambardella and M. Dorigo M, "*Solving Symmetric and Asymmetric TSPs by Ant Colonies*", Proceedings of the IEEE Conference on Evolutionary Computation, ICEC96, Nagoya, Japan, May 20-22, pp. 622-627, 1996.

[13] Ahmed S. Ghiduk, "*A New Software Data-Flow Testing Approach via Ant Colony Algorithms*", Universal Journal of Computer Science and Engineering Technology, pp 64-72, 2010.

[14] Neha Pahwa, Kamna Solanki, "*UML based Test Case Generation Methods: A Review*", International Journal of Computer Applications, Vol 95– No.20, pp 1-6, 2014.

Authors Profile

Ms.Navneet Kaur did Master of Information Technology from Apeejay College of Fine Arts, Jalandhar in 2003. She did her Master of Engineering in Software Engineering from Thapar Institute of Engg. & Technology, Patiala in 2005. She has teaching experience of 12 years. She is currently working as Assistant Professor in Department of Computer Science & IT, Lyallpur Khalsa College, Jalandhar, Punjab,

India. Her main research work focuses on Software Engineering, Software Reuse, Genetic Algorithms and ACO.

Ms. Jaskaranjit Kaur did Master of Technology in Computer Science & Engineering from DAV University Jalandhar in 2015. She has teaching experience of 2 years. She is currently working as an Assistant professor in Department of Computer Science & IT, Lyallpur Khalsa College, Jalandhar, Punjab, India.

Mr. J.S.Budwal did Bachelor of Computer Application from Appejay Institute of Management, Jalandhar. He did Master of Computer Science from MDU, Rohtak, Haryana, India. He has teaching experience of 12 years. He is currently working as Computer Faculty in GSSS Hazara, Jalandhar,Punjab, India.