An Approach of Optimal Path Generation using Ant Colony Optimization

Praveen Ranjan Srivastava*
Computer science and Information System Group (CSIS, Group)
Birla Institute of Technology and Science (BITS), Pilani
Rajasthan, India -333031
praveenr@bits-pilani.ac.in

Km Baby
Student, ME(SS), CSIS Group
Birla Institute of Technology and Science (BITS), Pilani
Rajasthan, India -333031
h2008124@bits-pilani.ac.in

G Raghurama
Electrical & Electronic and Instrumentation Group
Birla Institute of Technology and Science (BITS), Pilani
Rajasthan, India -333031
graghu@bits-pilani.ac.in

Abstract—Software Testing is one of the indispensable parts of the software development life cycle and structural testing is one of the most widely used testing paradigms to test various software. Structural testing relies on code path identification, which in turn leads to identification of effective paths. Aim of the current paper is to present a simple and novel algorithm with the help of an ant colony optimization, for the optimal path identification by using the basic property and behavior of the ants. This novel approach uses certain set of rules to find out all the effective/optimal paths via ant colony optimization (ACO) principle. The method concentrates on generation of paths, equal to the cyclomatic complexity. This algorithm guarantees full path coverage.

Keywords—Software Testing, Ant Colony Optimization (ACO), Path Testing, Optimal Path, Control Flow Graph (CFG), Cyclomatic Complexity (CC).

I. INTRODUCTION

Software Testing [1] forms one of the indispensable part of the software development life cycle [2] and heavily depends upon the software tester, guidelines of the organization which developed the software and various other factors. The diverse nature of the software Testing leads to standardization of the testing process. Some of the major methods [1, 2] of Software Testing include black box testing and glass box testing [2]. Glass box testing, also known as white box or structural testing depends upon generation of various control flow paths [2].

Generation of control flow paths [1, 2, 3] is a challenging problem in the domain of software engineering. Various methods like brute force, constraint based heuristics, symmetric matrix algorithms etc [4][5] have been suggested for generation of control flow paths, but none of those offer a simple and optimized solution for path prioritization. In glass box testing, the intent is to test thoroughly the internals of a particular program module. The ultimate goal is to write test cases that force the desired coverage of different structures. The three popular approaches to structural testing include control flow-based testing, data flow based testing and mutation testing [1][2]. The control flow based testing depends upon the control paths of the software. The total number of paths in a small program may be very large resulting in a very large number of test cases, which do not necessarily add value and hence, effective path selection becomes very important [3].

The current paper proposes a novel algorithm for the path prioritization in a simpler and effective manner. It has been identified that one of the software engineering areas with a more suitable and realistic use of artificial intelligence techniques is software testing [6, 7, 8, 9] and those techniques are known as a metaheuristic approach [10]. The ant colony optimization algorithm (ACO) [12, 15] is a probabilistic technique for solving computational problems which can be used to find optimal paths through graphs. The Algorithm is inspired by the behavior of ants in finding paths from the colony to food [13]. C. Peng Lam and others [11] use this ACO to generate test sequences for state-based software testing. One of the graph based algorithms that has been proposed by [14]. This algorithm is based on finding the shortest path between the starting node and the frontier nodes (nodes which are directly connected to the terminal node). This approach is thus able to cover all the nodes in the Execution State Sequence Graph (ESSG) but still not able to cover all the paths (Edges in between the nodes) in the ESSG, which is the weak level of testing.

This paper proposes an algorithm which uses an ACO technique to generate the optimal path suit. We use Control Flow Graph (CFG) diagrams [1] to generate optimal path. In computer science technology, a CFG describe the sequence in which the different instructions of a program get executed [1, 2]. In other word, a CFG describes how the control flows through the program. In order to draw the CFG, of a program, we need to first number all the statements of a program. The different numbered statements serve as the node of the control flow graph. Each node in the graph represents a basic block, i.e. a basic block has unique entry and exit points. These points are the first and last statement within a basic block. The control always enters a basic block at its entry point exit from its exit point. There is no possibility of exit or a halt at any points of a basic block except at its exit point [1].
The benefits of this approach are: (1) manual generated test prioritized paths are not always reliable while automatic test prioritized paths are reliable, because humans are the most dynamic and error introducing entity. (2) CFG based testing provides all statement/branch/loop/nodes coverage. (3) It provides all event coverage. (4) It provides all path coverage.

The paper is organized as follows. In section II an overview on software testing and ACO are presented whereas section III and IV describe an approach of the basic functionality and proposed algorithm for test path prioritization. In section V the proposed evaluation model is discussed with an illustration and finally, section VI presents the conclusions and future work.

II. SOFTWARE TESTING AND ACO

In Software testing there are three key area: (1) test data generation and identification suitable path (2) test execution involving the use of test data and the software under test (SUT) and (3) evaluation of test results.[2,11]. According to them selecting the right test path is one of the top software-testing problems. Software testing is a key part of software development life cycle. Due to time, cost and other circumstances, exhaustive testing is not feasible, that’s why there is need to automate the testing process. Generation of the automated and effective test paths are a very difficult task in the software testing process. Effective test path(prioritize path) can decrease the overall cost of testing as well as increase the probability of finding defects in software systems. Testing effectiveness can be achieved by the path testing which is commonly used in, real time, embedded and web-based kind of software system.

No test path gives the guarantee of all defects detection. So most important part is how to select test path that can increase the probability of finding defects. In Structural testing the most important techniques is know as path testing [2]. Tester aim is to test all possible path which he/she identified from CFG. But according to software science [1,2 ] exhaustive testing is not possible, so in structural testing, path testing is most suitable technique, but the main problem is selection of optimal path, we can easily calculate the total number of path using cyclomatic complexity [16] but the problem is facing how you select optimal paths.

In this type of requirement, Ant's behavior is very useful. Blind Ants coordinate their activities via stigmergy [17] proposed Ant Colony Optimization, which is inspired by foraging behaviors of ant colonies, and target discrete optimization problem. The main idea is that the self organizing principles which allow the highly coordinated behavior of real ants can be applied with the artificial agents which collaborate to solve the complex computational problems. Real ants coordinate with each other by dropping and sensing a chemical level known as pheromone on the paths. Selection of path by ants depends on stochastic or probability theory. In ACO ants generate pheromone level on the path and due to this level they remember the path in future, it is described by heuristic knowledge. Ants have the visibility of path as well as the destination by the sense of pheromone level and heuristic knowledge. Selection procedure for a path is based upon the probability of path which has the highest pheromone level and heuristic knowledge.

In this Paper an Ant Colony Algorithm is purposes for finding the all optimal path in CFG of Software under test. This Algorithm is helpful for finding all Paths in between the nodes. Selection of path is depends on probability. The higher the probability means selection of that path is increase. The probability value of path is depends on feasibility of path, Pheromone value and heuristic information of path. This is an effective approach which can easily generate optimal paths.

Path Testing

A path represents the flow of execution from the start node to end node. Path testing designed to execute all or selected paths through a computer program [1, 2]. The nature of software represented through graph, and it is essential to test all nodes and edges under the graph, for this type of verification, path testing is more reliable and suitable. A method with N decision nodes, has 2^N possible paths, and if the method contains a loop, it may have an infinite number of paths; fortunately cyclomatic complexity [16] is an important method to reduce number of paths. Cyclomatic complexity used to generate number of linearly independent paths. A path is considered is linearly in depended path if it introduced at least one new node.

In path testing basically we ensure the adequacy criteria should be 100% under statement, branch as well as conditional coverage [1] for any software or module. In other words, if 100% statement or branch coverage, there is no guarantee, to achieve a proper adequacy level under path testing. A path under CFG is considered complete if it begins from start node and terminates with end node.

Below an example is given, which describe how to compute basic path on the basis of decision node and generate the CFG for the given program.

**Prog 1**
1. Begin
2. Int num, product, power ;
3. bool done;
4. product=1;
5. input(done);
6. while(!done){
7. input(num);
8. product= product*num;
9. input(done);
10. }
11. output(product);
12. end

It is clear from Figure 1 that there is one decision nodes, therefore cyclometric complexity (CC) = no. of decision nodes+1=2, it represents the number of independent paths.
Path-based software testing is the most preferred testing [2]. It ensures that all paths are coverage at least once. Though test sequences are generated on the basis of feasibility study of the various available paths from the current state especially for real time and complex systems. Therefore tester’s main job is to test all paths at least once.

III. AUTOMATIC PATH SEQUENCE GENERATION AND PRIORITIZATION BY USING ACO

Path-based software testing is the most preferred testing [2]. It ensures that all paths are coverage at least once. This paper suggests the approach to generate the automatic prioritized test path and provide a solution of above problem. Given approach deals with the automatic path prioritization generation from the CFG diagram. CFG is the form of directed graph \( G = \{V, E\} \), where vertices \( V \) are corresponding to states, throughout the paper state, vertex and node are used interchangeably. Edges \( E \) are corresponding to states, throughout the paper state, vertex and node are used interchangeably. Edges \( E \) are corresponding to the paths between the nodes. Node means a condition during the life of an entity in which it satisfies some condition, performs some action, or waits for some event. Event is an occurrence that may trigger a state path. Event types include an explicit signal from outside the system, an invocation from inside the system, the passage of a designated period of time, or a designated condition become true. Transition is the change of state of an object. Action is one or more actions taken by an object in response to a state change [18].

Purpose of ant colony optimization Algorithm is for covering the optimal path coverage at least once in CFG diagram of the Software under test. Selection of path is depends upon the probability of the path. Higher probability values means chances to select the path is also high. The probability value of path is depends upon: feasibility of path \( F_{ij} \), which shows direct connection between the vertices; pheromone value \( \tau_{ij} \), which helps other ants to make decision in the future, and heuristic information \( \eta_{ij} \) of the path, which indicates the visibility of a path for an ant at the current vertex. In some cases if there are equal probabilities of feasible path then by the following three steps the algorithm selects feasible path [12, 17, 19].

- An ant selects end-node as the next node, means ant will select path from current node to end_node.
- An ant will select the next state according to the value of visited status parameter (Vs). If current vertex V1 is direct connected to the vertex say V2 and it not visited yet by the ant, then ant will select V2 as the next state that means the path (V1→V2) traversed. This concept fulfills the criteria of all state coverage at least once.
- After all the above consideration if selection is not possible then the ant will select any feasible path randomly.

In proposed algorithm ant has ability to collect the knowledge of all feasible paths from its current state. An approach for feasibility check of the paths from current state is used. This approach is defined in feasibility set of path \( F_{ij} \).

![Flow graph representation of the Prog.1](image)

Path1: start → 2 → 3 → 4 → 5 → 6 → 11 → end
Path2: start → 2 → 3 → 4 → 5 → 6 → 7 → 8 → 9 → 10 → 2 → 3 → 4 → 5 → 6 → 11 → end

Now we have two paths, the key question is which path going first under testing process and why?

1. Feasible path set: \( F = \{F_{ij} (p)\} \) represents the direct connection with the current vertex 'i' to the neighboring vertices 'j'. Direct connection shows that the vertices are the adjacent to the current vertex 'i', i.e. a direct edge exist in between the current vertexes 'i' and the chosen vertex 'j'.
   - \( F_{ij} = 1 \) means that path between the vertex 'i' and 'j' is feasible.
   - \( F_{ij} = 0 \) means the path between the vertex 'i' and 'j' is not feasible.
2. Pheromone trace set: \( \tau = \tau_{ij} (p) \) represents the pheromone level on the feasible path (i→j) from current vertex 'i' to next vertex 'j'. The pheromone level is updated after the particular path traversed. This pheromone helps other ants to make decision in future.
3. Heuristic set: \( \eta = \eta_{ij} (p) \) indicates the visibility of a path for an ant at current vertex 'i' to vertex 'j'.
4. **Visited status set:** Vs shows information about all the states which are already traversed by the ant p. For any state ‘i’:
   - Vs (i) = 0 shows that vertex ‘i’ is not visited yet by the ant p.
   - Whereas Vs (i) = 1 indicates that state ‘i’ is already visited by the ant p.

5. **Probability set:** Selection of path depends on probabilistic value of path. Because it is inspired by the ant behavior. Probability value of the path depends upon the feasibility of path Fij (p), pheromone value τij (p) and heuristic information ηij (p) of path for ant p. There are α, β two more parameter which used to calculate the probability of a path. These parameters α and β control the desirability versus visibility. α and β are associated with pheromone and heuristic value of the paths respectively.

Proposed ant colony algorithm helps to get not only knowledge of present state but also all feasible paths from current state to next state and historical knowledge of already traversed paths and states by the ant.

**IV. PROPOSED APPROACH AND PPTACO TOOL**

An algorithm is purposed for easily traversing all the states and prioritizes the all possible path which is the main purpose of the paper of all the path based coverage at least once.

Algorithm for ant p:

1. Initialize all parameter
   1.1 **Set heuristic Value (η):** for every path in the CFG initialize heuristic value η = 2.
   1.2 **Set pheromone level (τ):** for every path in the CFG initialize pheromone value τ = 1.
   1.3 **Set visited status (Vs):** for every state Vs = 0 (initially no state is visited by the ant).
   1.4 **Set Probability (P):** for each path initialize probability P = 0.

   1.5 α = 1, β = 1, here α and β are the parameter which controls the desirability versus visibility i.e. desirability means if an ant wants to traverse any particular path on the basis of pheromone value and visibility means the solution which ant has on the basis of prior experience regarding the path. These parameters are associated with pheromone and heuristic values of the paths respectively.

1.6 **Set count:** count = cyclometric complexity describe the different possible paths in CFG.

   CFG in the form of graph must be entered by the tester then tool automatically calculate the maximum number of possible paths depending upon the value of number of cc value.

1.7. **Set key:** key = end_node, it is a variable which store the value of end node.

2. **While (count>0)**

   Evaluation at vertex ‘i’

   2.1. **Initialize:** start = i and sum = 0, visit = 0.

   visit is a variable which used to discard a redundant path and sum used to calculate the value of strength of the path, which later used to prioritize the paths.

   2.2. **Update the track:** Update the visited status for the current vertex ‘i’ i.e. if (Vs[i] == 0) then Vs[i] = 1

   And visit = visit + 1. (increase the value of variable visit).

2.3 **Evaluate Feasible Set:** Means to determine F (p) for the current vertex ‘i’, this procedure evaluate the entire possible path from the current vertex ‘i’ to the all the neighboring vertices with the help of CFG diagram. If there is no feasible path then go to step 3.0.

2.4 **Sense the trace:** To sense the trace, evaluate the probability from the current vertex ‘i’ to all non-zero connections in the F (p), as discussed earlier ant’s behavior is probabilistic. For every non-zero element belongs to feasible set F (p), we calculate probability with the help of below formula.

\[
P_{ij} = \frac{(\tau_{ij})^\alpha \cdot (\eta_{ij})^{-\beta}}{\sum_k ((\tau_{ik})^\alpha \cdot (\eta_{ik})^{-\beta})}
\]

For every k belongs to feasible set F (p).

2.5. **Move to next vertex:** Using the below rule move to next vertex

   R1: Select paths (iÆj) with maximum probability (Pij).

   R2: If two or more paths (from iÆj and iÆk) have equal probability like (Pij = Pik) then select path according to below rule:

   R2.1. Compare each entry in the feasible set with the end_node i.e.

   If (feasible set entry == end_node) then select end_node as the next node otherwise follow R2.2.

   R2.2. Select that path which have next state not visited yet i.e. Visited status Vs = 0. If two or more state have same visited status i.e. Vs[j] = Vs[k] then follow R2.3.

   R2.3. Vs[j] = Vs[k] then select randomly

2.6. **Update the parameter:**

   2.6.1 Update Pheromone: Pheromone is updated for path (iÆj) according to the following rule

\[
(\tau_{ij}) = (\tau_{ij})^\alpha + (\eta_{ij})^{-\beta}
\]
2.6.2 Update Heuristic:
\[ \eta_{ij} = 2\times(\eta_{ij}) \]

2.7. Calculate Strength: It shows the values associated with each path
\[ \text{sum} = \text{sum} + \tau_{ij} \]
\[ \text{strength [count]} = \text{sum.} \]
\[ \text{start} = \text{next_vertex.} \]

2.8. if (start! = end_node)
then go to step 2.3.
else
if (visit==0) then discard the path it is the redundant path otherwise add new path.

2.9. Update count: decrement count by one each time.
\[ \text{count} = \text{count}-1. \]

3. End //end of algorithm
Count represents the cyclomatic complexity of a method, as count becomes zero; it shows all the decision nodes traversed.
Algorithm will stop automatically in two condition, firstly if there is no feasible path from the current node and secondly if the all feasible paths covered at least once.

V. DEMONSTRATION OF PROPOSED ALGORITHM
An ant can start from any nodes and it can generate a sequence of test cases. Test sequence depends upon the feasibility of path from the current vertex to other vertices and accordingly it will take decision for further proceeding and in the end it give the optimal test sequence in CFG diagram of software under test. Here optimal means all decision nodes traversed at least once. We have developed a prototype tool called PPTACO using the proposed algorithm to automatically generate path sequences and prioritization for given CFG diagrams. This tool takes input as a CFG; tester will put the ant at start node and get optimal path sequence and prioritization. Tool also gives output analysis in file format. Tester can see the internal values generated by ant like heuristic, pheromone values, probability calculation and describe selection of best path according to algorithm.

The brief description about how this sequence of path is generated for the CFG for Prog2 is mentioned in figure 2. Representation of paths which are chosen by an ant is also given in figure 2 and numeric value shows the sequence of paths traversed by an ant. Tabular representation for all paths are given in the Table 1.

Prog.2:
1. Start
   1. Int bottom=0, top=Array.length-1, mid ;
   2. r.found =false ;
path2 tester put an ‘p’ ant at node start, for start node, tool generate the feasible set F(start) = up to node ‘4’ it keep on moving and updating the parameters as there is no decision node. At node 4 feasible set i.e. F[4] = {5,end} with probability P(p6,5) > P(p4,end) so according to R1 ant select node ‘5’ as the next node then update parameter along with calculation of strength. At node six feasible set have two nodes i.e. F[6] = {7,10} with equal probability and visited status P(p6,7) = P(p6,10) and V[7] = V[10] = 0, so according to R2 ant select next node randomly say next node = ‘7’ now ant move till it get ‘end’ node as there is no further any decision node and will trace path2.Algorithm will follow similarly for path3 and path4. Path1 will not generate desire result.

As it is a binary search program in which has the lowest strength with low priority, in path2 there is probability to get result hence it has higher strength but path3 and path4 are executed most likely as to match the key value in first trial not likely as it in case of entering in the while loop. Below Table II is shown, which describe the different unique paths and associated data for second ant.

<table>
<thead>
<tr>
<th>Path</th>
<th>Node</th>
<th>Strength</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>path1</td>
<td>start,1,2,3,4,5,6,7,8,9,10,11,4,5,6,7,8,9,end</td>
<td>28.785</td>
<td>1</td>
</tr>
<tr>
<td>path2</td>
<td>start,1,2,3,4,5,6,7,8,9,10,11,4,5,6,7,8,9,end</td>
<td>24</td>
<td>2</td>
</tr>
<tr>
<td>path3</td>
<td>start,1,2,3,4,5,6,7,8,9,10,11,4,5,6,7,8,9,end</td>
<td>18.25</td>
<td>3</td>
</tr>
<tr>
<td>path4</td>
<td>start,1,2,3,4,5,6,7,8,9,10,11,4,5,6,7,8,9,end</td>
<td>28.785</td>
<td>1</td>
</tr>
</tbody>
</table>

Proposed algorithm display all the possible unique paths and help to prioritize various paths with respect to each other on the basis of path strength parameter. Algorithm is also has the capability to recognize redundant paths, if ant found any redundant path then ant will discard the redundant path as per step 2.9.2. Time Complexity of the proposed algorithm is of O (n^2) in the worst case. This is another fruitful result of the proposed .Though the proposed PPTACO approach provide the good solution for path prioritization testing over the existing approaches but further improvement may also possible. The proposed approaches have following limitation such as: The repetition of states which have the multiple paths. The advantage of proposed PPTACO tool is easy to use for any software tester. The strength of PPTACO tool is based upon high level of software coverage because all path as well as path coverage at least once during testing process.

VI. CONCLUSION

This paper proposes a model for path prioritization technique by using ant colony optimization. The result that we got by applying proposed method is very encouraging. We have taken the directed graph approach to model the system and shown the different paths of the model during the execution. After successful execution of algorithm, it automatically selects a best path sequence which covers the maximum coverage at least once. Soft computing techniques can evaluate path based testing which may ultimately help software industry to a greater extent. A number of extensions and applications of this model may be possible by using the different metaheuristic techniques.

REFERENCES