ACO BASED SOLUTION FOR TSP MODEL FOR EVALUATION OF SOFTWARE TEST SUITE

Gulwatanpreet Singh¹, Surbhi Gupta², Baldeep Singh³

¹, ². (Rayat institute of engineering and technology, Ropar, Punjab)
³. (SBBSIET, Jalandhar, Punjab)

Royal_surbhi@gmail.com
Singh_kpt@yahoo.co.in
Doad_baldeep@gmail.com

ABSTRACT

A test suite or validation suite is a collection of test cases that are intended to be used to test a software program to show that it has some specified set of behaviors. A test case is a set of conditions or variables under which a tester will determine whether a requirement is partially or fully satisfied. Selecting the appropriate test cases for the complete evaluation of a software product is the highest priority part of software testing process. After selecting the appropriate test cases and making a complete test suite, next thing is to evaluate this test suite. This phase is the most time consuming and tedious task. This paper proposes and implements a technique for evaluation of the software test suite using Ant Colony optimization based solution by formulating the problem in form of travelling salesman problem model, providing best optimal solution.

Keywords: artificial intelligence software testing, ACO, test case, equivalence classes, precision and recall, average node branching, iterative best cost

I. INTRODUCTION

Software testing is an activity aimed at evaluating the capability of a system and determining that it meets the required result. Testing conforms that the system or software works fulfilling the required needs. The main focus of software testing is to achieve the maximum level of a quality in the product. The whole process of software testing is generally divided into three different stages. a. Generation of test cases and creation of a test suite. b. Execution of the test suite. c. Evaluation of results of values of all the test cases in a test suite.
The process of testing any software system is a time consuming process and pays much more on cost factor [2]. Lot of time and effort is required in this process. Testing of the system can also be achieved with automation techniques which help in reducing these two aspects. Software testing when gets combined with the artificial intelligence helps in making the work more efficient and error free. Combination of AI and software testing provide different advantages as mentioned in published papers [1][3].

In this paper, we are going to present the technique which helps in making the software testing process more efficient and reduce the time and effort factors in testing process. To make this happen, the whole paper is divided into two parts whose combined result will help us in achieving the desired objective. Section 1 will describe the technique and Section 2 will describe the implementation and results part. Section 1 is further divided into two parts, which will tell us about the methods that how the test suite will be generated and how the different test cases in a test suite by formulating them in the form of TSP model and finding the solution through Ant Colony Optimization.

II. Section 1.

A. Generation of test cases and creation of a test suite.

During Testing, different test cases are prepared. A test case is a set of conditions or variables under which a tester will determine whether a requirement is partially or fully satisfied. Selecting the appropriate test cases for the complete evaluation of a software product is the highest priority part of software testing process. This thing can be achieved by the concept of equivalence class partitioning. Equivalence class partitioning is the process of methodically reducing the huge set of possible test cases into a much smaller, but still equally effective set. This method tries to define test cases that uncover classes of errors, thereby reducing the total number of test cases that must developed. One test value is picked from each of the equivalence class, which helps in covering all the possible inputs. Extracting Test Cases manually is a error prone process. TesMa is a kind of test case generation tool proposed by X.Zhang and T.Hashino[8]. It is a more accurate and high quality tool under acceptable cost. These kind of tools helps to automatically generate the required test cases.

B. Execution of the test suite using the proposed approach.

Once we are ready with the test suite preparation, very next step is to evaluate this entire test suite. We can perform the testing of this data in a test suite using ant colony optimization technique by formulating the problem in the form of tsp model. Ants traverse through the edges of the tsp graph and generate the shorter feasible tours by using information accumulated in the form of pheromone trails deposited on the edges of tsp graph. Ant colony optimization is a probabilistic technique for solving computational problems which can be reduced to finding good paths through graphs [4]. The basic idea behind this technique is the foraging behavior of real ants. This algorithm will help to test the every test case in a test suite in the form of a best optimal solution. So that it helps in fulfilling our basic need of testing i.e. locating errors and less cost will be there.
Ant colony optimization (ACO) was introduced by M. Dorigo [5][6] and colleagues in 1992, as a novel nature-inspired meta-heuristic for the solution of hard combinatorial optimization (CO) problems. Combinatorial optimization refers to finding an optimal object from a finite set of objects. The core factor governing the ACO algorithm is pheromone model or parameterized probabilistic model. The pheromone model consists of a vector of model parameters $T$ called pheromone trail parameters. The pheromone trail parameters $T_i$, which are usually associated to components of solutions, have values $\_i$, called pheromone values. The value of $i$ varies, with each tour, which is updated during the each tour loop. In general, the ACO approach attempts to solve an optimization problem by repeating the following two steps:

- Initially solutions are generated using a pheromone model i.e. a parameterized probability distribution over the solution space;
- The candidate solutions are used to modify the pheromone values in a way that is deemed to bias future sampling toward high quality solution.

**ACO based solution for TSP model for evaluation of software test suite.**

1. Start
2. Initialize the parameters
   2.1 Initialize the different test cases i.e. $T=$\{$T_1,T_2,T_3,...,T_n$\}.
3. Initialize the trail value of pheromone i.e. $t_i=0$.
4. Each Ant is placed individually on starting state with empty memory $M_k$.
5. Do while(cycle loop gets completed)
6. {
   Do upto (tour loop gets completed)
   Update the value of $t_i$.  // (Local trail update)
}
7. Evaluate the tours.
   7.1 Construct solution for every ant.
   7.2 Check the best tour in terms of higher density of pheromone.
   7.3 Display the results.
      (Global Trail update).
8. Finish

// cycle loop here means that each test case should gets executed at least once
// tour loop means that each ant should completes its tour.
// Here the $T_i$ will specify that which test case is going to be evaluated.
// During the tour loop value of pheromone is updated each and every time.
// trail value of pheromone is updated with each tour. The result for each test case will be generated while cycle loop R. hence, correspondingly, $R=$ \{$r_1, r_2, r_3...r_n$\}. Where $r_i$ will represent result of each test case.
C. Evaluation of results
Once the results are generated by the above said algorithm, next step is to evaluate these results in order to check if any discrepancy exists among the obtained results and required results. By following the given approach we can easily identify if there exists any errors or not.

II. Section 2
A. Implementation of algorithm and results
For checking the effectiveness of this algorithm, let’s consider the case of a calculator. We decide to start with addition, try 1+0=. And get an answer of 1. That's correct. Then try 1+1=, and get 2. How far do we go? The calculator accepts a 9-digit number, so you must try all the possibilities up to 1+999999999=

Once you complete that series, you can move on to 2+0=, 2+1=, 2+2=, and so on. Eventually you'll get to

999999999 + 999999999 =

Next you should try all the boundary values: 1.0+0.1, 1.0+0.2, and so on. Once you verify that regular numbers sum properly, you need to attempt illegal inputs to assure that they're properly handled.

If we manage to complete all these cases, you can then move on to adding three numbers, then four numbers, and so on. There are so many possible entries that you could never complete them, even if you used a super computer to feed in the numbers. And that's only for addition. You still have subtraction, multiplication, division, square root, percentage, and inverse to cover.

Hence it becomes problematic for us to consider all the test cases, we just select some of the test case values first and evaluate them. The figure below shows the evaluation of selected test cases combined to form a test suite.

Figure 1 shows the execution of 14 different test case values by using ant colony optimization. As the number of test case values increases the iterative time for executing these test case values also increases.
For executing the nodes ACO-TSP is used, so that each variable must be executed once. The travelling salesman problem (TSP) is an NP-hard problem in combinatorial optimization studied in operations research and theoretical computer science. Given a list of node and their pairwise distances, the task is to find the shortest possible route that visits each node exactly once and returns to the initial node.

Fig 2 shows the path followed to execute the test cases

B. Evaluating the accuracy of algorithm:

For evaluation of the accuracy of algorithm two different parameters are considered. These are Precision and Recall.

a. Precision and Recall:

Precision is defined as fractions of retrieved variable, instances or conditions that are relevant and recall is defined as fraction of retrieved variables to the number of total variables. More specifically, recall is a measure of how many of the relevant documents were retrieved, while precision is a measure of how many of the retrieved documents were in fact relevant. Both precision and recall are therefore based on an understanding and measure of relevance.

Given, \( V_1 = \) number of variables retrieved, \( V_2 = \) number of relevant variables retrieved, \( V_n = \) number of relevant variables in the collection.

\[ \text{Recall} = \frac{V_2}{V_n}. \]
Precision = V2/V1.

Now, first of all consider the case of 5 variables (1, 2, 3, 4, 5) variables, here V1=5, V2=5, Vn=5. Hence precision and recall value is 1.0.

![Command Window](image)

Precision and recall value for the proposed algorithm is 1.0. Which is the maximum accuracy value an algorithm can have, hence from these evaluations we can say that the proposed algorithm is completely accurate. It does not contain any kind of logical error. It will give as 100% accurate results.

C. Evaluating the efficiency of an algorithm:

For evaluating efficiency, we will consider two different parameters that are, iterative best cost and average node branching.

a. Iterative best cost:

Iterative best cost is a process of generating best cost path by cyclic process of prototyping, testing and refining a process. Based on the results of testing the most recent iteration of a design, changes and refinements are made. This process is intended to ultimately improve the quality and functionality of a design.

Iterative best cost is given by number of path followed to generate the best cost path by number of path total available. If ‘p’ is number of path followed to generate the best cost path and ‘N’ is total number of available paths. Then, iterative best cost is given by p/N.
Iterative best cost for evaluating 8 test case variables using ant colony optimization is 72. Whereas traditional algorithms such as dijkstra, bellmon ford have much higher than ant colony optimization. Dijkstra algorithm will have iterative time 454 for evaluating 8 test cases so as bellmon ford, which is much higher than ant colony optimization. Higher the iterative cost lower is the efficiency. So, we can say that evaluation of a test suite using this approach will work much faster.

b. Average Node Branching:

In graph data structures, Node branching is given by the number of nodes connected to each node. If this value is not uniform, average node branching is used. For example, in chess, if a "node" is considered to be a legal position, the average branching factor has been said to be about 35. This means that, on average, a player has about 35 legal moves at his disposal at each turn.

Lower the average node branching factor more efficient is the algorithm. Higher average node branching factor results in more computational expensiveness and leading to combinatorial explosion. Combinatorial explosion is the effect of functions that grow very rapidly as a result of combinatorial considerations and results to excessive computation.
**Figure 4.8:** Graph representing average node branching vs iterative time for 8 test case variables.

Average node branching for evaluating 8 different test case variables using ant colony optimization remain between 2 to 5. This varies for different iterative time. We can say that for 8 nodes maximum branching factor is 5. Whereas for dijkstra algorithm it will be in between 1 to 7 for 8. So, we can say that evaluating test cases by ant colony optimization will be much efficient using the proposed algorithm.

**III. CONCLUSION**

The technique present in the given paper is much more efficient as compared to earlier techniques, in terms of time and cost. It provides emerging area of research that brings about the cross-fertilization of ideas across two domains i.e software testing and artificial intelligence This technique, when gets implemented in the software testing process, helps to generate the results of test cases with high accuracy and governing all other factors which finally results in making the software testing process more efficient.

**REFERENCES**