Evaluation of Test Cases Using ACO and TSP

Gulwatanpreet Singh, Sarabjit Kaur, Geetika Mannan
CTITR&PTU
India

Abstract:

A test case, in software engineering is a set of conditions or variables under which a tester will determine whether an application, software system or one of its features is working as it was originally established for it to do. The mechanism for determining whether a software program or system has passed or failed such a test is known as a test oracle.. After selection of the appropriate test cases for a particular product or project, next step is to evaluate the selected test cases. This phase incurs most of the time and effort. This paper proposes and implements a technique for evaluation of the software test cases 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 a process used to identify the correctness, completeness and quality of developed computer software. Software testing can be stated as the process of validating and verifying that a computer program/application/product and meets the requirements that guided its design and development. The main focus of software testing is to achieve the maximum level of a qualityin the product. The whole process of software testing is generally divided into three differentstages. a. Generation of test cases. b. Execution of the test cases. c.Evaluation of values generated by all the test cases The process of testing any software system is a time consuming process and pays much more oncost and time factor required in this process [2]. Testing of the system can also beachieved with automation techniques which help in reducing these two aspects. Software testingwhen gets combined with the artificial intelligence helps in making the work more efficient anderror free. Combination of AI and software testing provide different advantages as mentioned inpublished papers [1][3]. In this paper, we are going to present the technique which helps in making the software testingprocess more efficient and reduce the time and effort factors in testing process. To make thishappen, the whole paper is divided into two parts whose combined result will help us inachieving the desired objective. Part A will describe the Ant Colony Optimization technique for finding the travelling salesman problems and Part B will describethe implementation and results part.

PART A

A. Generation of test cases During Testing, different test cases are prepared. A test case is a set of conditions or variablesunder which a tester will determine whether a requirement is partially or fully satisfied. Selectingthe appropriate test cases for the complete evaluation of a software product is the highest prioritypart of software testing process. This thing can be achieved by the concept of equivalence classpartitioning. Equivalence class partitioning is the process of methodically reducing the huge setof possible test cases into a much smaller, but still equally effective set. This method tries todefine test cases that uncover classes of errors, thereby reducing the total number of test casesthat must developed. One test value is picked from each of the equivalence class, which helps incovering 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 automaticallygenerate the required test cases.

Ant colony optimization (ACO) was introduced by M. Dorigo [5] [6] and colleagues in 1992, as anovel 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 ofobjects. The core factor governing the ACO algorithm is pheromone model or parameterizedprobabilistic model. The pheromone model consists of a vector of model parameters T calledpheromone trail parameters. The pheromone trail parameters Ti _ T, which are usually associatedto 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 probabilitydistribution over the solution space;

• The candidate solutions are used to modify the pheromone values in a way that is deemed tobias future sampling toward high quality solution.

Evaluation of test cases using aco and tsp

1. Start
2. Initialize the parameters

2.1 Initialize the different test cases i.e. **T= {T1, T2, T3…..Tn}.**
3. Initialize the trail value of pheromone i.e. **ti=0**.
4. Each Ant is placed individually on starting state with empty memory Mk.
5. Do while (cycle loop gets completed)
6. {
Do up to (tour loop gets completed)
Update the value of **ti**. // (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 get executed at least once
// tour loop means that each ant should complete its tour.
// Here the **Ti** 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 begenerated while cycle loop R. hence, correspondingly, **R= {r1, r2, r3…rn).** Where **ri**willrepresent result of each test case.

C. Evaluating the output generated by the test cases
Once the results are generated by the above said algorithm, next step is to evaluate these resultsin order to check if any discrepancy exists among the obtained results and required results. Byfollowing the given approach we can easily identify if there exists any errors or not.

PART B
A. Implementation of algorithm and results
For checking the effectiveness of this algorithm, let's consider the case of a calculator. Wedecide to start with addition, try 2+0=. And get an answer of 2. That's correct. Then try 2+2=,and get 4. How far do we go? The calculator accepts a 9-digit number, so you must try all thepossibilities up to 1+99999999=
Once you complete that series, you can move on to 2+0=, 2+1=, 2+2=, and so on. Eventuallyyou'll get to999999999+999999999= next you should try all the boundary values: 1.0+0.1, 1.0+0.2, and so on. Once you verify thatregular numbers sum properly, you need to attempt illegal inputs to assure that they're properlyhandled.If we manage to complete all these cases, you can then move on to adding three numbers, thenfour 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 stillhave 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 testcase values first and evaluate them. The figure below shows the evaluation of selected test casescombined to form a test suite.

```
Command Window
>> ACO('F:\Thesis material\final implementaion\21 nodes.tsp');
AS is reading input nodes file...
21 nodes in 21 nodes has been read in
AS start at 07-Oct-2012 21:37:11
Showing Iterative Best Solution:
AS stop at 07-Oct-2012 21:37:13
Drawing the iterative course's curve
8+7=15
7+1=8
1+2=3
2+4=6
4+3=7
3+6=9
6+10=16
10+11=21
11+12=23
12+9=21
9+13=22
13+14=27
14+15=29
15+21=36
21+20=41
20+19=39
19+18=37
18+16=34
16+17=33
17+5=22
>>
```

Figure 1: shows the execution of 21 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 valuesalso increases.For executing the nodes ACO-TSP is used, so that each variable must be executed once. Thetravelling salesman problem (TSP) is an NP-hard problem in combinatorial optimization studiedin operations research and theoretical computer science. Given a list of node and their pair wisedistances, the task is to find the shortest possible route that visits each node exactly once andreturns to the initial node

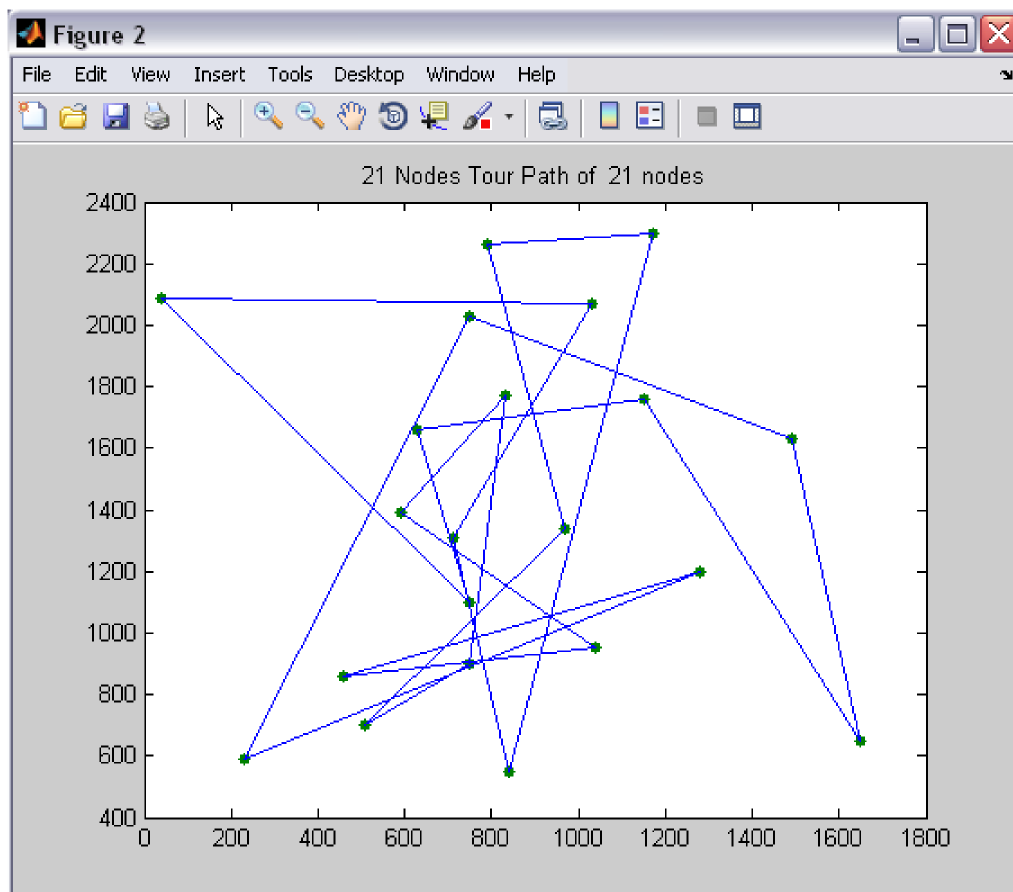

Fig 2 showsthe path followed to execute the test cases

B. **Evaluating the accuracy of algorithm**:
For evaluation of the accuracy of algorithm two different parameters are considered. These arePrecision and Recall.
a. **Precision and Recall**:
Precision is defined as fractions of retrieved variable, instances or conditions that are relevantand recall is defined as fraction of retrieved variables to the number of total variables. Morespecifically, recall is a measure of how many of the relevant documents were retrieved, whileprecision is a measure of how many of the retrieved documents were in fact relevant. Bothprecision and recall are therefore based on an understanding and measure of relevance.
Given, V1 = number of variables retrieved, V2 = number of relevant variables retrieved, Vn=number of relevant variables in the collection.Recall = V2 / Vn.80Precision = 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.
Precision and recall value for the proposed algorithm is 1.0. Which is the maximum accuracyvalue an algorithm can have, hence from these evaluations we can say that the proposedalgorithm is completely accurate. It does not contain any kind of logical error. It will give as100% accurate results.

C. Evaluating the efficiency of an algorithm:
For evaluating efficiency, we will consider two different parameters that are, **Checking best cost through multiple iterations** and average node branching.

a. Checking best cost through multiple iterations:
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 adesign, changes and refinements are made. This process is intended to ultimately improve thequality and functionality of a design.

Iterative best cost is given by number of path followed to generate the best cost path by numberof path total available. If 'p' is number of path followed to generate the best cost path and 'N' istotal number of available paths. Then, iterative best cost is given by p/N.

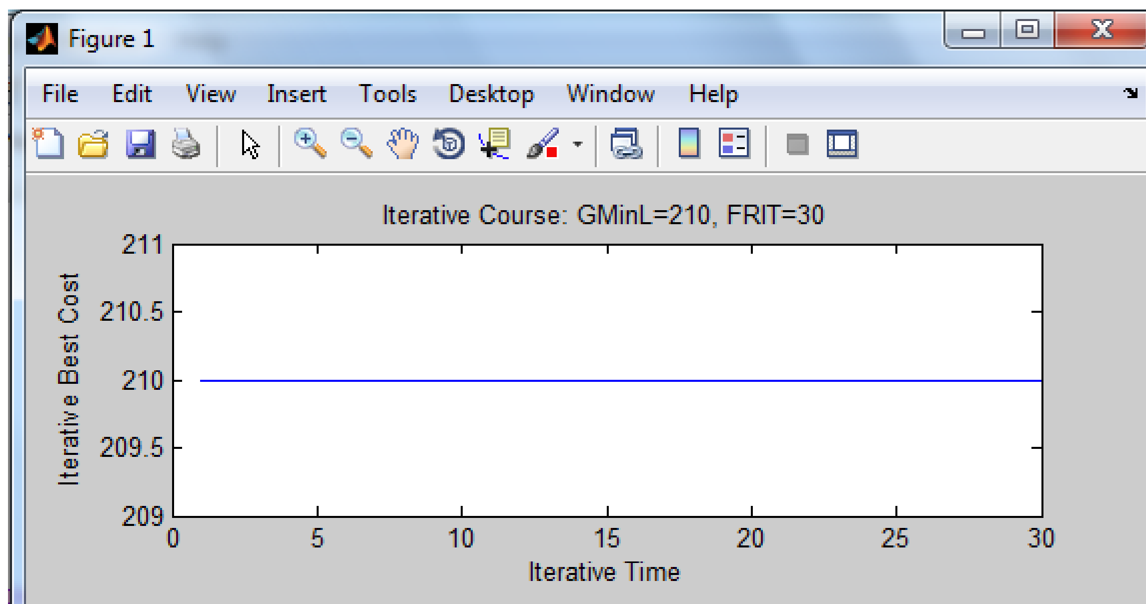

Figure 4: Graph representing iterative best cost vs. iterative time for 14 test case variables

Iterative best cost for evaluating **14 test case** variables using ant colony optimization is 210.Whereas traditional algorithms such as dijkstra, bell mom ford have much higher than ant colonyoptimization. Dijkstra algorithm will have iterative time 659 for evaluating 14 test cases so asbellmon ford, which is much higher than ant colony optimization. Higher the iterative cost loweris the efficiency. So, we can say that evaluation of a test suite using this approach will workmuch faster.

b. Average Node Branching:

In graph data structures, Node branching is given by the number of nodes connected to eachnode. 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 beabout 35. This means that, on average, a player has about 35 legal moves at his disposal at eachturn.Lower the average node branching factor more efficient is the algorithm**.** Higher average nodebranching factor results in more computational expensiveness and leading to combinatorialexplosion. Combinatorial explosion is the effect of functions that grow very rapidly as a resultof combinatorial considerations and results to excessive computation.

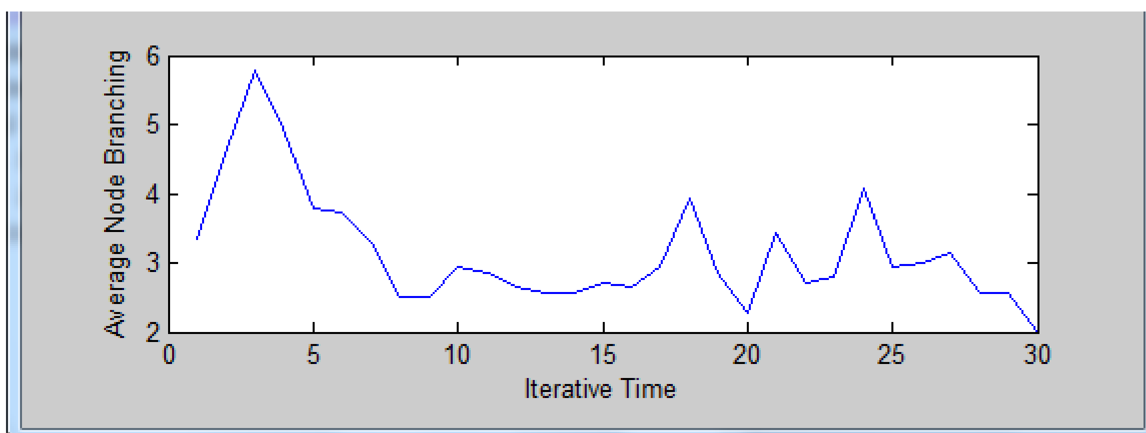

Figure 4.8: Graph representing average node branching vs iterative time for 14 test case variables.

Average node branching for evaluating 14 different test case variables using ant colony optimizationremain between 2 to 6. This varies for different iterative time. We can say that for 8 nodes maximumbranching factor is 6. Whereas for

dijkstra algorithm it will be in between 1 to 7 for 8. So, we can say thatevaluating 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, interms of time and cost. It provides emerging area of research that brings about the cross-fertilization ofideas across two domains i.e. software testing and artificial intelligence This technique, when getsimplemented in the software testing process, helps to generate the results of test cases with high accuracyand governing all other factors which finally results in making the software testing process more efficient. Furthermore, this technique is till now applied to only functionality based testing. It can also be extended to further types of testing.

REFERENCES

1. Briand, L. C., “On the many ways Software Engineering can benefit from KnowledgeEngineering”, Proc. 14th SEKE, Italy, pp. 3-6, 2002.
2. Binder, R. V., Testing Object-oriented Systems: Models, Patterns, and Tools, Addison Wesley.2000.
3. AldeidaAleti, Lars Grunske, IndikaMeedeniya, Irene Moser,” Software Test SequenceOptimization Using Graph Based Intelligent Search Agent” ,November 2009 pp. 505-509
4. Marco dorigo, Luca Maria Gambardella,” Ant Colonies for Travelling Salesman Problem”, 1997.
5. M. Dorigo, M. Birattari, and T. Stutzle, “Ant Colony Optimization: Artificial Ants as aComputational Intelligence Technique, IEEE computational intelligence magazine, November,2006.
6. M. Dorigo, G.Di Caro, and L.M. Gambardella, “Ant algorithm for discrete optimization”,Artificial Life, vol. 5, no. 2, pp. 137-172, 1999.
7. McMinn, P., “Search-based Software Test Data Generation: A Survey”, Software Testing,Verification and Reliability, Vol.14, No. 2, pp. 105-156, 2004.
8. X.Zhang and T.Hoshino. ”A trail on model based test cases extraction and test data generation”,in proceedings of third workshop on model based testing in practice, 2010.
9. Ron Patton “Software Testing” 2006.
10. Joecolelife‘s “Software engineering”2007.