Volume 4, Issue 8, August 2014 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

Evaluation of Test Cases Using Ant Colony Optimization with a New Heuristic Function: A Proposed Approach

Gulwatanpreet Singh
Assistant Professor
Dr BR Ambedkar NIT, Jalandhar, India

Poonam Rana
DAVIET, Jalandhar
Punjab, India

Parveen Kakkar
Asstt. Professor
DAVIET, Jalandhar, India

Abstract--A test case in software engineering is a set of conditions or variables under which a tester will determine whether an application or software system is working correctly or not. Before evaluation of test cases, the very first step of utmost importance is to select the appropriate test cases. This step is of great importance as it is difficult to cover all the test cases pertaining to a specific data. Evaluation of test cases is effort consuming and error-prone process. After selecting the appropriate test cases, next comes the various techniques used for evaluation of test cases. In my earlier research, I proposed and implemented the technique used for evaluation of test cases by Ant colony System. But that technique also has some limitations on the part of heuristic function such as pheromone stagnation and it does not cover the timely updation of heuristic function. This paper proposes an Ant Colony Optimization approach for evaluation of test cases but this time it will be with a different heuristic function which is described in the later part of this paper.

Keywords: software testing, Ant colony system , test case, heuristic function. Travelling Salesman Problem.

I. INTRODUCTION

Software testing is an activity aimed at evaluating difference between the actual requirements and the requirements that are met in a software product or project. The outcome of the software testing phase is generally errors that will be in the software. 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 b. Execution of the test cases 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 [5].Lot of time and effort is required in this process. 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 [6][7].

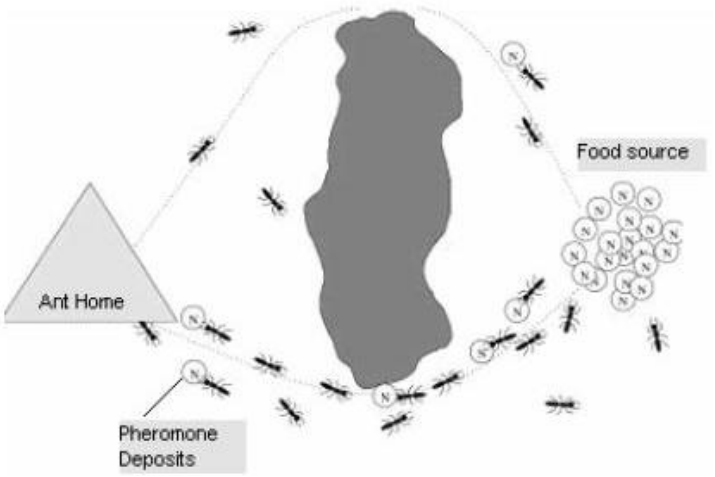
In this paper, we are going to present the technique which helps in evaluation of test cases using ACO but with a new heuristics function. The heuristic is a way of defining how far we are from our goals. The proposed technique will overcome the problems that occur in the earlier evaluation of test cases with ACO. In my earlier studies, the heuristic function was chosen according to the mathematical operation (in the case of test cases pertaining to calculator) the whole structure of the paper is divided into 4 sections: Section 1 contains the introduction about ACO and TSP problem and how the TSP is evaluated using ACO. Section 2 describes the earlier technique used for evaluation of test cases by using ant colony systems. Section 3 describes the new heuristic function and the proposed technique of evaluation of test cases by using this function. A complete algorithm (of the proposed technique) is presented in this section.

SECTION I:

Ant Colony Optimization belongs to family of ant colony algorithms, in swarm intelligence methods. It consists of some meta heuristic optimizations or heuristic which is designed for finding, generating or selecting a lower-level procedure or heuristic (partial search algorithm) that may provide a sufficiently good solution to an optimization problem particularly with incomplete information or limited computation capacity. Meta-heuristics [13] may make few assumptions about the optimization problem being re-solved and thus they may be utilizable or usable for a range of problems. The whole procedure can be best described with the fig. given below

The framework of a basic ACO algorithm

While considering the theoretical properties of ACO metaheuristics, first thing is to generalize the ACO definition. Here, The ACO is introduced in the form of an algorithm that was theoretically studied. It works as follows. At each iteration, ants probabilistically construct solutions to the combinatorial optimization problem under consideration, exploiting a given pheromone model. Then, optionally, a local search procedure is applied to the constructed solutions. The following are the steps for this:



Initialize Pheromone Values (T). At the start of the algorithm the pheromone values are all initialized to a constant value c > 0.

Construct Solution (T). The basic ingredient of any ACO algorithm is a constructive heuristic for probabilistically constructing solutions. A constructive heuristic assembles solutions as sequences of elements from the finite set of solution components C. A solution construction starts with an empty partial solution S= (). Then, at each construction step the current partial solution Sp is extended by adding a feasible solution component from the set N(S) C \ {Sp}.

Algorithm: The framework of a basic ACO algorithm
input: An instance *P* of a CO problem model *P = (S, f,*Ω*)*.
Initialize_Pheromone_Values(*T*)
S_{bs}←NULL
whiletermination conditions not met **do**
G_{iter}← NULL
for *j*= 1, . . . , n_a**do**
S ←Construct_Solution(*T*)
ifS is a valid solution **then**
S ←Local_Search(s) {optional}
if*(f (*S*) < f(*S_{bs}*))* or *(*S_{bs}= NULL*)* **then** S_{bs}← s
G_{iter}← G_{iter}U {S}
end if
end for
Apply_Pheromone_Update(*T* ,G_{iter} ,S_{bs})
end while

This set is determined at each construction step by the solution construction mechanism in such a way that the problem constraints are met. The process of constructing solutions can be regarded as a walk (or a path) on the so-called construction graph G = (C,E), which is a fully connected graph whose vertices are the solution components in C and whose edges are the elements of E. The allowed walks on GC are implicitly defined by the solution construction mechanism that defines the set N (Sp) with respect to a partial solution Sp. The choice of a solution component C_{ji}∈N(Sp) is, at each construction step, done probabilistically with respect to the pheromone model.
The probability for the choice of C_{ji} is proportional to $[\ \tau_{ji}]^{\alpha} \cdot [\ \eta(C_{ji})]^{\beta}$, where η is a function that assigns to each valid solution component possibly depending on the current construction step—a heuristic value which is also called the heuristic information. This is the main point of consideration on which our current study is focused.
The value of parameters α and β, α > 0 and β > 0, determines the relative importance of pheromone value and heuristic information. The heuristic information is optional, but often needed for achieving a high algorithm performance.
In most ACO algorithms the probabilities for choosing the next solution component also called the transition probabilities are defined as follows:
$P(C_{ji} \mid Sp) = \{[\ \tau_{ji}\ ]^{\alpha} \cdot [\ \eta(C_{ji})]^{\beta}\ \} \ /\ \sum c_{\epsilon N(Sp)} [\ \tau_{jk}\ ]^{\alpha} \cdot [\ \eta(C_{jk})]^{\beta}$ V C_{ji}∈ N(Sp).
Note that potentially there are many different ways of choosing the transition probabilities. As an example of this construction mechanism let I denote the set of indices of the current decision variable and of the decision variables that have already a value assigned.
Let Ic denote the index of the current decision variable (i.e., the decision variable that has to be assigned a value in the current construction step). The solution construction starts with an empty partial solution Sp = (), with Ic ϵ {1, . . . , |V |} randomly chosen, and with I = {Ic}. Also, the index of the first decision variable is stored in variable if (i.e., I_f ← Ic).

Then, at each of the |V| − 1 construction steps a solution component C_{Iflc}ϵN(sp) is added to the current partial solution, where N(Sp) = {C_{kic} | k ϵ {1, . . . , |V |} \ I}. This means that at each construction step a domain value is chosen for the decision variable with index Ic. Once the solution component C_{jlc} is added to Sp, Ic is set to j.
The transition probabilities used in each of the first |V| − 1 construction steps are those of equation, where the heuristic information can, in the case of the ATSP, be defined as ŋ (C_{ji})=1/d_{ij} (this choice introduces a bias towards short arcs).
The last construction step consists of adding solution component C_{Ific} to the partial solution Sp, which corresponds to closing the Hamiltonian cycle.

Local Search(s). A local search procedure may be applied for improving the solutions constructed by the ants. The use of such a procedure is optional, though experimentally it has been observed that, if available, its use improves the algorithm's overall performance.

Apply_Pheromone_Update (T , G_{iter} , S_{bs}). The aim of the pheromone value update rule is to increase the pheromone values on solution components that have been found in high quality solutions.
Most ACO algorithms use a variation of the following update rule:
$\tau_{ji} \leftarrow [(1 - \rho) \cdot \tau_{ji}] + [\rho / G_{upd} \sum F(s)]$
fori = 1, . . . , n, and j = 1, . . . , |D_i |. Instantiations of this update rule are obtained by different specifications of G_{upd}, which in all the cases is a subset of $G_{iter}\{S_{bs}\}$, where G_{iter} is the set of solutions that were constructed in the current iteration, and S_{bs} is the best-so-far solution.
The parameter ρϵ[0, 1] is called evaporation rate. It has the function of uniformly decreasing all the pheromone values. From a practical point of view, pheromone evaporation is needed to avoid a too rapid convergence of the algorithm toward a sub-optimal region.

II. TSP ANT SYSTEMS

A number m of ants is positioned in parallel on m cities. The ants' start state, that is, the start city, can be chosen randomly, and the memory M_k of each ant k is initialized by adding the current start city to the set of already visited cities (initially empty). Ants then enter a cycle, which lasts NC iterations, that is, until each ant has completed a tour.
During each step an ant located on node i consider the feasible neighbourhood, reads the entries aij's of the ant-routing table Ai of node I, computes the transition probabilities, moves to the new city, and updates its memory. Once ants have completed a tour, they use their memory to evaluate the built solution and to retrace the same tour backward and increase the intensity of the pheromone trails τij of visited connections lij.
This has the effect of making the visited connections become more desirable for future ants. Then the ants die, freeing all the allocated resources .The pheromone trail information is changed during problem solution to reflect the experience acquired by ants during problem solving. Ants deposit an amount of pheromone proportional to the quality of the solutions they produced: the shorter tour generated by an ant, the greater amount of pheromone it deposits on the arcs which it used to generate the tour. This choice helps to direct search towards good solutions.
Ant's memory allows it to compute the length of the tour generated and to cover the same path backward to deposit pheromone on the visited arcs. The ant-routing table Ai=[aij(t)] of node i, Ni is the set of all the neighbour nodes of node i], is obtained by the following functional composition of pheromone trails **τij**(t) and local heuristic values ηij :
$a_{ij} = \{[\tau_{ji}]^{\alpha} \cdot [\eta(C_{ji})]^{\beta}\} / \sum_{c \epsilon N(Sp)} [\tau_{jk}]^{\alpha} \cdot [\eta(C_{jk})]^{\beta}$
Where α and β are two parameters that control the relative weight of pheromone trail and heuristic value. The heuristic values used ηij=1/Jcicj, where Jcicj is the distance between cities I and j. in other words, the shorter distance between two cities I and j, the higher heuristic value ηij. The probability *kijP*(t) with which at the t-th algorithm iteration an ant k located in city I chooses the city j Є Nk ii to move to is given by the following probabilistic decision rule[10]:
$\sum_{c \epsilon N(Sp)} [\tau^{j}_{k}]^{\alpha} \cdot [\eta(C^{j}_{k})]^{\beta}$
Where Nki Є Ni is the feasible neighbourhood of node i for k (that is, the set of cities ant k has not yet visited) as defined by using the ant private memory Mk and the problem constraints. If α =0, the closet cities are more likely to be selected: this corresponds to a classical stochastic greedy algorithm (with multiple starting points since ants are initially randomly distributed on the nodes).
If β =0, only pheromone amplification is at work: this method will lead to the rapid emergence of stagnation, that is, a situation in which all ants make the same tour which, in general, is strongly sub-optimal. An appropriate trade-off has to be set between heuristic value and trail intensity.
After all ants have completed their tour, each ant k deposits a quantity pheromone
$\tau_{ji}(t) \leftarrow \tau_{ji}(t) + \Delta \tau_{ji}(t)$
After pheromone updating has been performed by the ants, pheromone evaporation is triggered; the following rule is applied to all the arcs Lij of the graph G:
$\tau_{ji}(t) \leftarrow (1 - \rho) \cdot \tau_{ji}(t)$
Where ρ Є (0, 1] is the pheromone trail decay coefficient.

SECTION II:

The whole process of evaluation of test cases with ACO is divided into 3 phases:

(i) Generation of test cases

During testing test cases are prepared. Selecting the appropriate test cases is the most difficult task as it has to cover all the data. So during testing we divide data into equivalent classes and the process is also called as equivalence partitioning. . This method is typically used to reduce the total number of test cases to a finite set of testable test cases, still covering maximum requirements. There can be automated test generation tools. **TesMa** a test case generation tool proposed by X.Zhang and T.Hashino[8]. It is a more accurate and high quality tool under acceptable cost.

(ii) Execution of test cases.

Once we are ready with the test cases, next step is to execute these test cases. The technique which was used for this execution is ACO by formulating the test cases in the form of TSP[9]Ant colony optimization algorithm (ACO) is a probabilistic technique for solving computational problems which can be reduced to finding good paths through graphs. . This technique helps in testing the test each variable of test case in best possible way. So that it can
locate errors in software system but also helps in reducing the high cost associated with testing. The algorithm used for this technique was:

ACO Algorithm for evaluating test cases

Initialization
a. Parameter initialization.
Initializing different test cases S={s1,s2,s3,……,sn}
b. Initializing pheromone trail value.
Ti=0;
c. Each ant is individually placed on initial state with empty memory
Do While (each test case is executed) – Cycle Loop
Do Until (Each Ant Completes a Tour) – Tour Loop
Local Trail Update
(Update value of Ti)
End Do
Analyse Tours
a. Construct ant solution.
b. Apply local search.
c. Best tour check.
d. Display results.
Global Trail Update
End Do

(iii)Evaluation of results

Once the results generated by ACO algorithms are obtained, next step is to evaluate these results. During evaluation of results discrepancy among obtained results and required results can be evaluated. Finally by observing these results we can check that whether there is an error in the software or not.

Results shown by this technique were quite impressive, as they try to evaluate to evaluate test cases in the best possible way. An example of evaluation of 14 test case variables is shown in figures below:

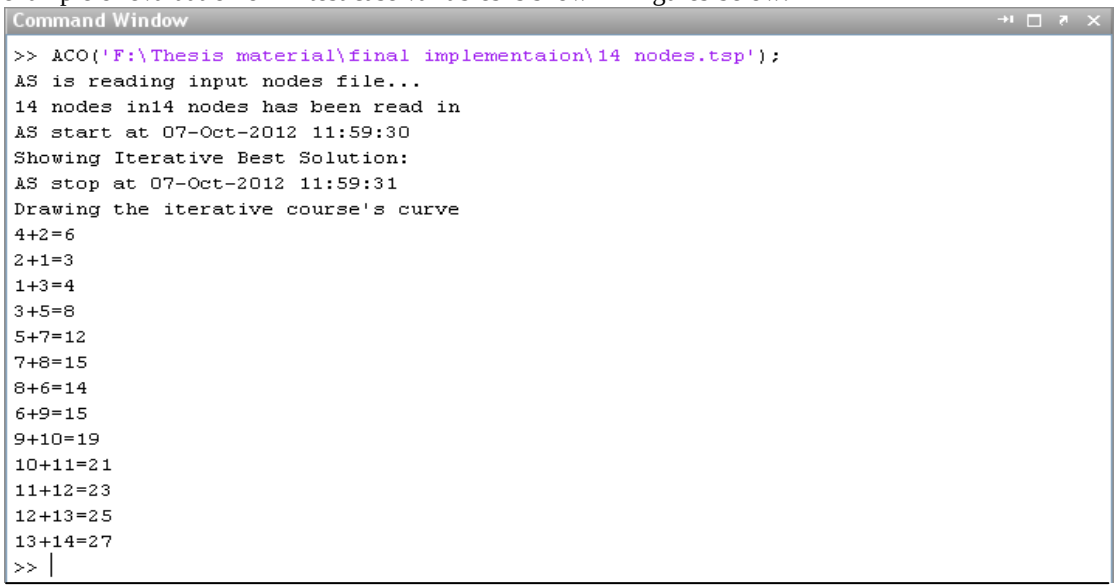


Fig.1 **S**howing the Output generated by ACO program for test case evaluation

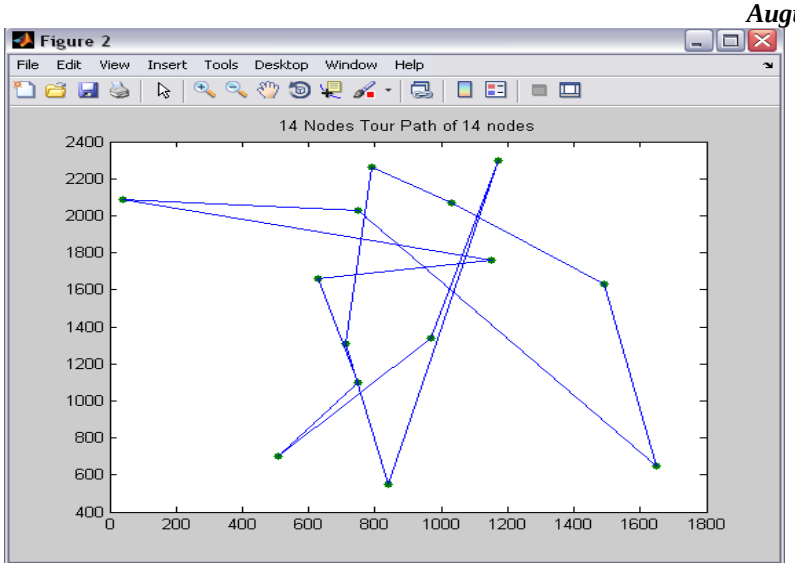


Fig. 2: Graph showing execution of nodes, each node represents the value of particular test case.

However, this technique has some limitations such as pheromone stagnation and not updating the heuristic function timely, due to which it still generates the sub optimal solutions. The Technique proposed in later sections of this paper will overcome these limitations.

SECTION III:

Ant Colony System (one of the best algorithms for solving NP-hard problems) suffers from pheromone stagnation problem when all ants congregate quickly on one sub- optimal solution. ACS algorithm utilizes the value between nodes as heuristic values to calculate the probability of choosing the next node. In case of technique used for evaluation of test cases, the heuristic function chosen according to the operation that was applied between values of any two nodes. However, this heuristic function was not updated any time throughout the whole process to reflect the new information realized by ants. The enhanced ACS technique for evaluation of test cases will integrate a new heuristic function [1] that will update the heuristic value each time the ants find an enhanced solution in the iteration. After the ant constructs its solution, a global update process will be applied for updation of the best-so-far solution. This event will change the environment for the next coming iteration. A function would be generated at this moment to reflect this change and hence a new heuristic value will be attained. The new information will be applied to the best-so-far edge or tour. The pseudo-code for the new heuristic function is shown in following algorithm[1]:

Step 0: for each path in the best tour do step 1 to 2
Step 1: if path i(i = 1, 2, n) is not updated before do step 2
Step 2: $\eta_i = \eta_i$+ (δ/ best-so-far tour) // δ is parameter from (0-10)
End

After applying this function, the values of heuristics will change according to the quality of best optimal solution. As we found the best solution, it will increase the heuristic value and vice versa. The parameter δ will reflect the influence of updating value that should be applied to heuristic value.

Proposed algorithm for Evaluation of Test Cases using ACO with a new heuristic function

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.et_i= 0
4. Each Ant is placed individually on starting state with empty memory M_k.
5. Do while (cycle loop gets completed)
{
5.1. Do upto (tour loop gets completed)
5.2. Update the value of t_i//(Local trail update)
}
6. Evaluate the tours.
6.1. Construct solution for every ant.
6.2. Check the best tour in terms of higher density of pheromone.
6.3. Display the results.
6.4. (Global Trail update)
7. Apply new heuristic function

7.1. For each path in the best tour do step 1 to 2
7.2. If path i (i = 1, 2, n) is not updated before do step 2
7.3$\eta_i = \eta_i + (\delta/$ best-so-far tour) // δ is parameter ranging from (0-10)
8. Finish

III. CONCLUSIONS

The proposed technique will prove to generate better results as compared to the previous technique used for evaluation of test cases using ACO. This technique avoid being get trapped in the local optimal solution, hence generates the global optimal solution. This technique can also helps remove the defects of earlier technique such as pheromone stagnation. If this technique would gets combined with the automatic generation of test cases than it will result in automated software product for testing. If we come up with better exploration mechanisms we will generate even more better results. In the succession this paper, we will present the implementation part of this technique and will show the results.

ACKNOWLEDGEMENT

We would like to acknowledge the authors "Mustafa Muwafak Alobaedy" and "Ku ruhana Ku-mahamud' for their significant contribution in designing of this new heuristic function which we have applied in our technique.

REFERENCES

[1] Alobaedy, Mustafa Muwafak, and Ku Ruhana Ku-Mahamud, "*New Heuristic Function in Ant Colony System for the Travelling Salesman Problem "* , IEEE,Computing and Convergence Technology (ICCCT), 2012.
[2] MarcoDorigo, Mauro Birattari, and Thomas Stutzle. " Ant colony optimization ", IEEE, Computational Intelligence Magazine ,pp.28-39, 2013
[3] EricBonabeau, Marco Dorigo, Guy Theraulaz *" Swarm intelligence "*, MIT Press, 1999.
[4] J.L. Deneubourg, S. Aron, S. Goss, J.M. Pasteels , *" The self organizing exploratory patternof the argentine ant "*, Journal of Insect Behavior, Vol. 3, No. 2, pp.159-168, 2000.
[5] Binder, R. V., Testing Object-oriented Systems: Models, Patterns, and Tools, Addison Wesley. 2000**.**
[6] Briand, L. C*.," On the many ways Software Engineering can benefit from Knowledge Engineering"*, Proc. 14th SEKE, Italy, pp. 3-6, 2002.
[7] AldeidaAleti, Lars Grunske, IndikaMeedeniya, Irene Moser*," Software Test Sequence Optimization Using Graph Based Intelligent Search Agent"* ,November 2009 pp. 505-509
[8] X.Zhang and T.Hoshino. *"A trail on model based testcases extraction and test data generation"*, in proceedings of third workshop on model based testing in practice, 2010.
[9] G.Singh, S.Gupta and B.Singh *" ACO based solution for tsp model for evaluation of software test suite"* in IAEME Volume 3, Issue 3, October - December (2012), pp. 75-82.
[10] McMinn, P., " *Search-based Software Test Data Generation: A Survey"*, Software Testing, Verification and Reliability, Vol.14, No. 2, pp. 105-156, 2004.
[11] Ron Patton "Software Testing" 2006.
[12] Joecolelife's "Software engineering" 2007.
[13] Blum C., Roli A.(2003) "*Metaheuristics in combinatorial optimization: overview and conceptual comparison"*, ACM Computing Surveys 35 (3), pp.268–30
[14] A. Aljanaby,K.R Ku-Mahamud and N.M Norwawi. *"Optimizing Large scale combinatorial problems using multiple ant colony algorithms based on pheromone evaluation techniques"* , International Journal of computer science an network security, vol 8(10), p 54-58, 2008

