A Non-Pheromone based Intelligent Swarm Optimization Technique in Software Test Suite Optimization

*D.Jeya Mala, M.Kamalapriya, R.Shobana,V.Mohan
Thiagarajar College of Engineering, Madurai, Tamil Nadu, India.
{djmcse@tce.edu, mkpriya@tce.edu, rsshobana@tce.edu}

Abstract - **In our paper, we applied a non-pheromone based intelligent swarm optimization technique namely artificial bee colony optimization (ABC) for test suite optimization. Our approach is a population based algorithm, in which each test case represents a possible solution in the optimization problem and happiness value which is a heuristic introduced to each test case corresponds to the quality or fitness of the associated solution. The functionalities of three groups of bees are extended to three agents namely Search Agent, Selector Agent and Optimizer Agent to select efficient test cases among near infinite number of test cases. Because of the parallel behavior of these agents, the solution generation becomes faster and makes the approach an efficient one. Since, the test adequacy criterion we used is path coverage; the quality of the test cases is improved during each iteration to cover the paths in the software. Finally, we compared our approach with Ant Colony Optimization (ACO), a pheromone based optimization technique in test suite optimization and finalized that, ABC based approach has several advantages over ACO based optimization.**

Keywords: Software Testing, Test Optimization, ABC (Artificial Bee Colony) Optimization, Agents, Test adequacy criterion, ACO (Ant Colony Optimization)

I. INTRODUCTION

The Major Objectives of Software Testing are to uncover as many as errors (or bugs) as possible in a given timeline, to demonstrate a given software product matching its requirement specifications, to validate the quality of a software testing using the minimum cost and efforts and to generate high quality test cases that perform effective tests, and issue correct and helpful problem reports (4).

As per Deb (2), many of the complex multi-variable optimization problems cannot be solved exactly within polynomially bounded computation times. Software testing is one of the multi-variable optimization problems in which generation and selection of efficient few test cases cannot be solved within bounded times. Meta-heuristic search algorithms are applied to solve these types of problems to find near-optimal solutions in reasonable running times. The non-pheromone based swarm intelligence algorithm described in this paper is a search algorithm capable of locating good solutions efficiently. The algorithm is inspired by the food foraging behavior of honey bees.

The paper uses path coverage (4) as the test adequacy criterion to find out the efficient test cases in covering the SUT within less time and cost. Finally, the proposed approach is compared with ACO (Ant Colony Optimization) and proved that, ABC based approach outperforms ACO.

A. EFFICIENCY OF TEST CASES

A test case is a set of conditions or variables and inputs that are developed for a particular goal or objective to be achieved on a certain application to judge its capabilities or feature. Test Cases are the formal implementation of a test case design. The goal of any given test case or set of test cases is to detect defects in the system being tested. A Test Case should be documented in a manner that is useful for the current test cycle and any future test cycles (4). Test Efficiency can be defined internal to the organization as how much resources were consumed how much of these resources were utilized. Usually, it is computed using the following formula (4):
Test Efficiency =Number of test cases executed / unit of time (generally per hour).

B. LITERATURE SURVEY

Many research works on Genetic Algorithms, Neural Networks (NN), ACO, and Simulated Annealing and Fuzzy logic have been applied for test case optimization.

1) Genetic Algorithms in Test Data Generation

As per (7), GA as a population based algorithm can be applied for test case generation. In their paper, they applied the basic Genetic operators to generate the test cases. And fitness is designed to find the efficient test cases to be used as parents for next generation of offspring. However, Genetic Algorithm has some disadvantages: strike up at local optima, thereby is difficult to find global optimization.

2) Ant Colony Optimization in Software Test Suite Optimization – (ACO)

ACO simulates the behavior of real ants. The first ACO technique is known as Ant System (3) and it was applied to the traveling salesman problem. Since then, many variants of this technique have been produced. ACO is a probabilistic technique that can be applied to generate solutions for combinatorial optimization problems. The artificial ants in the algorithm represent the stochastic solution construction

procedures which make use of (1) the dynamic evolution of the pheromone trails that reflects the ants' acquired search experience and (2) the heuristic information related to the problem in hand, in order to construct probabilistic solutions. However, it still has some shortcomings, for example: at initial stages it lacks pheromone, search slowly, and easily to trap into the local optimal solution if the quantity is large, which is to say premature convergence.

II. ARTIFICIAL BEE COLONY OPTIMIZATION (ABC)

Karaboga (2005) analyzes the foraging behavior of honey bee swarm and proposes a new algorithm simulating this behavior for solving multi-dimensional and multi-modal optimization problems, called Artificial Bee Colony (ABC) (1)(5) and (6). The main steps of the algorithm are:

- Send the employed bees onto the food sources and determine their nectar amounts;
- calculate the probability value of the sources with which they are preferred by the onlooker bees;
- stop the exploitation process of the sources abandoned by the bees;
- send the scouts into the search area for discovering new food sources, randomly;
- memorize the best food source found so far.

III. ABC BASED SOFTWARE TEST SUITE OPTIMIZATION - PROPOSED APPROACH

In ABC model, we have the following types of bees (6)

1) **Employed,**
2) **Onlooker** and
3) **Scouts**

In our model, the system consists of three agents: Search Agent, Optimizer Agent and Selector Agent. Agents exhibit autonomy, social ability and inter operability to perform the task (8). It is understood that, there is exactly one test case needed for each test sequence or path. The test cases to be executed for the corresponding test sequences or paths are determined by following the path in the state space.

Initially, a population of test cases is generated. The Search Agent searches for an executable state in the SUT for each test case as it goes to an executable state in the test sequence /path as per the information in the knowledge source and determines the best next neighbor state.

This determination is done by analyzing all the neighbor states from the current state based on the selected test case's coverage (12). Based on that, it evaluates the happiness value (nectar amount) of each node surrounding the current node for the selected test case. Then the selection of the best node to transit is chosen. If the node is not feasible or not covered by the particular test case, then the node is removed from memory and the Selector Agent will start a new search for finding the node with higher feasibility in that path. Based on that, a happiness value or coverage measure is associated with each test case. A test case with highest happiness value or coverage measure is remembered and all the other test cases are removed from the memory.

The Optimizer Agent watches the Selector Agent and chooses the test cases depending upon the happiness value associated with each test case.

A. ABC BASED TEST OPTIMIZATION ALGORITHM

- Initial test cases are produced as random test cases for all test sequences
- REPEAT
  - The Search Agent monitor each test case as it goes to an executable state in the test sequence /path as per the information in the knowledge source and determines a neighbor state, then depending upon its feasibility, it updates the happiness value of each test case for each test sequence.
  - If the node is not feasible or not covered by the particular test case, then the node is removed from memory and the Selector Agent will start a new search for finding the node with higher feasibility in that path. Based on that, a happiness value or coverage measure is associated with each test case. A test case with highest happiness value or coverage measure is remembered and all the other test cases are removed from the memory.
  - The Optimizer Agent watches the happiness value of each test case and chooses one of them based on the happiness value and then goes to that node. After choosing a neighbor around that, it evaluates its happiness value.
  - Abandoned test cases are determined and then, they are replaced with the new test cases discovered by the Selector Bee
  - The best test case found so far is stored in the optimized test case repository for regression testing
- UNTIL (all the nodes have been visited at least once)

B. ABC BASED TEST SUITE OPTIMIZATION

1) Problem Formulation

Let TC={tc1,tc2,..tcn} be the set of test cases. And let hv(tci) represents the happiness value of test case tci. Let S={s1,s2,..sm} be the set of executable states in the software under test. Let TS={ts1,ts2,..tsn} be the set of test sequences in the software under test.

2) Pseudo Code for Search Agent:

Step1: Initialize TC ← rand(seed) for each tsi in TS.
Step 2: start-search(Search-Agent) with i←1
Step 3: search-point = s1
Step 3: Compute hv(tci) for each si in neighbor(search-point)
Step 4: if (feasible(si, tci)) = nil then
remove si; start-new search(Search- Agent)
Else remember tci ; remove all other tci

3) Pseudo Code for Optimizer Agent:

Step 1: Select-test-case(Optimizer-Agent)
Step 2: if (hv(tci)) > hv(tci+1)) for all i, optimal-test-suite← optimal-test-suite + tci

4) *Pseudo Code for Selector Agent* :

Step 1: if (hv(tci)) = 0 then Replace-test-case(Selector-Agent)
Step 2: new-testcase-gen(Selector-Agent)

IV. IMPLEMENTATION – ABC TESTER

Sample problem taken is finding biggest among three numbers. Here the decision making is done based on the values of the three numbers.

```
//Sample program is Compare.java
import java.io.*;
public class compare
{ public static void main(String [] args ) throws Exception
{ (1)int a ,b,c;
BufferedReader br = new BufferedReader(new
InputStreamReader (System.in));
System.out.println("Enter the value of A");
a=Integer.parseInt(br.readLine());
System.out.println("Enter the value of B");
b=Integer.parseInt(br.readLine());
System.out.println("Enter the value of C");
c=Integer.parseInt(br.readLine());
(2)if(a>b) {
(3)if(a>c){
System.out.println("The Biggest no is : " + a); }
(4)else{
System.out.println("The Biggest no is : " + c);}}
(5)else {
(6)if(b>c)
{System.out.println("The Biggest no is : " + b);}
(7)else
{System.out.println("The Biggest no is : " + c);}
(8)}}}
```

For the above program the cyclomatic complexity is 4 which indicate there are 4 independent paths.

- Initially we have randomly generated test cases for each of these paths.
- Let Xij be the initial test cases where j stands for each of these paths and i denotes the variables. So ,in this case 1<=i<=3 and 1<=j<=4.
- Consider we have got {0,3,3} for path1; {4,4,4} for path2;{1,1,3} for path3; {3,0,0} for path4
- Xij represents a test case i related to path j . Here we have three variables and so i=1 means it is for a, similarly i=2 for b and i=3 for c.
- Qij=(-1,1) which is a random number generated during program execution.
- Now vij=xij+qij(xij-xkj) , where i=1 to 3 , j = 1 to 4 ,k = (i+1)%size of the test case in this case size=3
  V11=x11+q11(x11-x21)=0+0(0-4)=0;
  V12=x12+q12(x12-x22); V13= x13+q13(x13-x23)
- Compare the fitness of xij and vij. Apply greedy selection between xi and vi for each path j .
- Calculate the probability pi for the solution xi/vi by means of fitness values. Pi=fit/sum(1 to j)fit i.
- Based on them, test cases are selected based on their fitness value. Apply the greedy selection process between xi and vi. Unfortunately, both of these test cases have their fitness value as 0 and so we need to abandon these solutions and replace them with new solution by the Scout bees.
- Determine the abandoned solution and replace it with newly generated test cases by the scout using the formula
  Xij=minj+rand(0,1)*(maxj-minj)
- Again determine the fitness of this test case. Memorize the best cases achieved so far based on their fitness value.
- Run = Run +1.
- If (Run = max Run) read the minj and maxj value from the user.
- Repeat this until (path coverage =100 %) (In other words until all the paths are covered).

The above procedure is coded in java and we applied the said approach for each unit of the software under test. We found that the results were generated so fast because of the operation of the three agents are working in parallel way. Since the agents work in parallel, decision making is simple. Once a path is covered then it is indicated by the scout by a flag. Once this flag is set to true, it goes for new path. The generation of test cases using ABC is shown in Fig. 1, 2 and 3.

V. COMPARISON BETWEEN ABC Vs.ACO

From Li and Lam's methodology proposed in paper (14), we derived the following observations as its functionality and problems.

A. ACO – FUNCTIONALITY

(1) Pheromone based tracking of nodes - For the non-negative connections in the connection set, the ant senses and gathers the corresponding pheromone levels P at the other ends of the connections. (2) Moving to the next node based on pheromone level. (3) Updation of the pheromone value due to frequent evaporation of it. (4) The final optimal solution can be

obtained by examining all of the solution candidates created by ant exploration. (5) If the upper bound of search has been reached, then it is concluded that, the current group of ants fails to find a solution which achieves the required coverage. More ants will have to be deployed in order to find a solution.

B. PROBLEMS IDENTIFIED IN ACO

Since, the tracking is based on the pheromone values at each node, the values of them must be updated frequently to keep its current level due to its evaporation. This updation process presents substantial overhead in the optimization process. The final optimal solution can be obtained by examining all of the solution candidates created by ant exploration. Since, the process is a sequential one in which the solution selection is done only at the end, leads to computational overhead and memory limit problems. Suppose a group of ten ants have been deployed for the optimal solution generation, and if this group of ants fails, then a new group of ten other ants have to be deployed. The time spent for the initial process will be a mere waste and leads to substantial time overhead.

C. ABC – SOLUTIONS TO THE PROBLEMS

Since, ABC is a Non –Pheromone based technique, there is no need for the updation of pheromone values. The communication between the bees is by means of waggle dance; that is done by setting a status flag for each bee. Parallel behavior of group of bees (multi-threading) makes the algorithm faster in reaching near global optimal solution. The final optimal solution is the improvement done during each iteration of the bees' exploration process. There is no need to examine all the solution candidates generated from the beginning to the end at the final step. Hence, computational overhead and memory limit problems were balanced. If there is no improvement in the current solution, then the scout will start a new search and a new set of test cases are generated. There is no need to deploy more bees for this case. Hence, time overhead is reduced.

D. COMPARISON CHARTS

ABC Algorithm has been applied to the following problem which is given in the table 1.We have Calculated the no of runs taken by each problem

TABLE1 Time Taken - ABC vs. ACO

S.No.	***Problem Description***	***Level***
1.	Factorial Program	Low
2.	Greeting Message	Low
3.	Marks Processing	Medium
4.	Discount Calculation	Medium
5.	Credit Card Validation	High
6.	Calculator	High

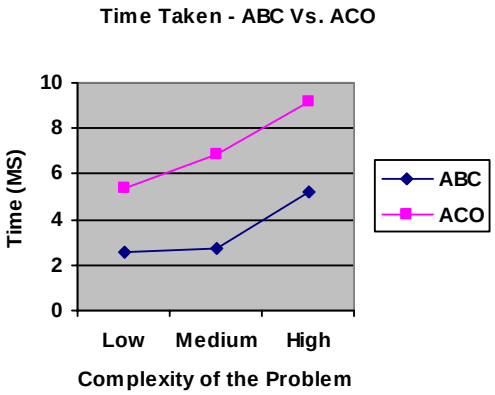

Figure. 4 – Time taken by ABC and ACO

1) Efficiency of the test cases (Improvement over iterations) – ABC vs. ACO

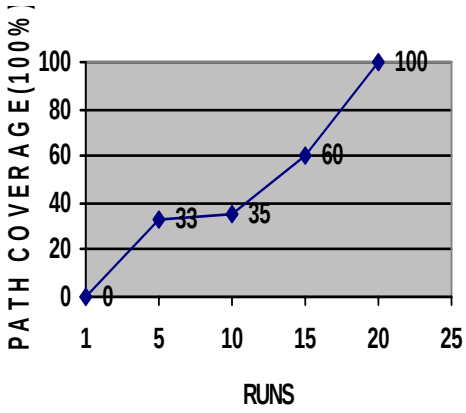

Figure. 5 –Path Coverage % using ABC

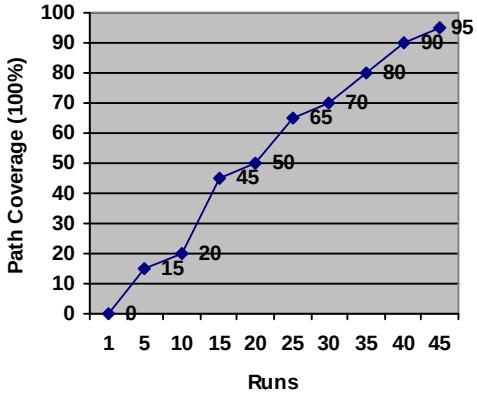

Figure. 6 –Path Coverage % using ACO

The graph shown in Fig. 4 describes the time taken by ABC and ACO algorithms. The graphs given in Fig. 5 and 6 shows the path coverage obtained in our case study by applying ABC and ACO.

VI. CONCLUSION

We discussed in this paper, how a non-pheromone based swarm intelligence algorithm is used in software test suite optimization. The size of the test suite is reduced by improving the quality of test cases during every iteration and keeping only the efficient test cases in the repository. When compared to ACO, the ABC based approach provides consistent results and does not have any problems in ACO like, continuous pheromone updation, computational, time and memory overheads. Hence, ABC model based test case optimization generates optimal results and it converges within less number of test runs.

REFERENCES

1. B.Basturk, Dervis Karaboga, "An Artificial Bee Colony (ABC) Algorithm for Numeric function Optimization", IEEE Swarm Intelligence Symposium 2006, May 12-14, 2006, Indianapolis, Indiana, USA.
2. Deb K. "Multi-Objective optimization using Evolutionary Algorithms". 2001, Chichester, UK: Wiley.
3. Huaizhong Li and C.Peng Lam, "Software Test Data Generation using Ant Colony Optimization", Transactions on Engineering, Computing and Technology, 2004, ISSN 1305-5313.
4. Roger S.Pressman, "Software Engineering: A practitioners Approach", 4th Edition,2007
5. D. Karaboga, B. Basturk, "On The Performance Of Artificial Bee Colony (ABC) Algorithm", Applied Soft Computing, Volume 8, Issue 1, January 2008, Pages 687-697. doi:10.1016/j.asoc.2007.05.007
6. D. Karaboga, B. Basturk, "Artificial Bee Colony (ABC) Optimization Algorithm for Solving Constrained Optimization Problems", LNCS: Advances in Soft Computing: Foundations of Fuzzy Logic and Soft Computing, Vol: 4529/2007, pp: 789-798, Springer- Verlag, 2007, IFSA 2007. DOI: 10.1007/978-3-540-72950-1_77.
7. Christoph C. Michael Gary E. McGraw Michael A. Schatz Curtis C. Walton, "Genetic Algorithms for Dynamic Test Data Generation", Proc. ASE'97.
8. Stuart Russel, Peter Norvig "Artificial Intelligence: A modern approach", 1995, Prentice-Hall Inc.

APPENDIX A

The Snapshots for the problem which we had taken into consideration