Implementation Paper on Designing Efficient Test Suite Using Artificial Bee Colony Algorithm

Shruti Goel  
Gurgaon Institute of Technology and Management  
MDU University  
Rohtak, Haryana India  
Sachinabhi25@gmail.com

Ms. Sangeeta  
Assistant Professor  
Gurgaon Institute of Technology and Management  
MDU University  
Rohtak, Haryana India  
Sangeeta_Gautam@yahoo.co.in

Abstract  
For every quality software product it is important that its quality be improved and maintenance time be reduced as the product ages. The product should be reliable and should meet the standards as well. To do so proper testing strategies must be in place and they should be executed religiously. Here we recommend a algorithm on the principles of Artificial Bee Colony. Here the Bee is supposed to do the testing and tracking activity, which helps in saving time to find out optimal coverage test paths. This increases the confidence on product. Natural Bee hives have mainly three types of bees, namely scout bees, which perform undirected search of nectar. Onlookers’ bees determine which food source to explore next from the list given by scouts. Employed Bee finds new food source in the vicinity of depleted sources.

Keywords  
Software quality, Improvement, Testing, Path Coverage, Artificial Bee colony Optimisation, Shortest Path.

Introduction  
Other than writing code, a lengthy process of software development involves many other activities. Requirements need to be gathered from Customers by business analysts to understand and generate specifications of the software. Design of the software is done followed by programming. Quality checks need to be performed to eradicate errors and bugs to make sure the product meets its specification. As the product ages its maintenance tasks continues. Combining together all these phases make up a software development lifecycle. This process or cycle can be carried out by using one of the many available models for software development. With time enhancements are done to the product and defects are fixed which causes the product to change. This calls for testing activity to be performed as and when small/big changes are done to the code.

End to end analysis of the product is done via software testing to promote confidence and to make sure it meets the requirements specified by the client. Software testing comprises of executing the code with a set of input values and matching the output with the desired set of values. This set of conditions and variables under which a tester will determine whether a system under test satisfies the requirements and works correctly is called a test case. In this paper we use the term test case as the set of input to the program. Various test cases combined together are called a test suite or a test set. Professional testers generally maintain a diversified set of test cases to detect maximum defects. For every requirement mentioned in the specification document there exists a corresponding test case.

Every requirement needs to be verified or tested in the product for its correctness. This can be achieved via white box (code testing) or black box (giving input and checking against expected output) testing. Requirements testing may include covering up of simple statements, If-Then-Else statements, loops, and Function calls. This will be tested thru white box testing. If a special set of input values need to be tested against the expected values it will be verified via black box testing. Every Requirement can have a test
case corresponding to it or more than one requirement can be covered by a test case. Testing each and every requirement makes sure that the product is tested rigorously. For a particular statement or a decision clause a test case can be designed, if there does not exist any to address the code. M. Hutchins [1] stated in his paper that for each faulty program several thousand test sets were generated and examined the relationship between fault detection and coverage. Within the restricted domain of our experiments, test sets achieving coverage levels over 90% usually showed notably better fault detection than arbitrary chosen test sets of the same size.

Accompanying test suites grows and evolves as the software ages. New test cases are added to the suite as new functionality is added or changes are made to the product. Test cases may be added to guard against a live-with defect, which was tolerant till now but has to be fixed going forward. With time some test cases become redundant in the test suite in relation to the coverage criteria as the same test scenarios are covered by another test case. Notice that the attribute of a test case being useless(redundant) is related to a specific set of coverage requirements. A test case is said to be redundant if exercises a set of statements which are covered by another test case as well. Also if any test case is not finding any defect from a long time between various builds, then that test case can be removed from the test suite. Sometimes, the same test case may quite not be redundant connected to, for example when it used for definition – use pair, is not covered in any of the test case in the suite. It is crucial to note that redundancy of a test case is an attribute related to some specific set of requirements.

With time the test suites grows in size, they need more time for execution than is available for testing. That time it becomes desirable to reduce the number of test cases in the test suite. There are applications with extreme programming done on a daily basis, which are also tested end to end after every build. Test suite minimization is one common method that has been put forward to overcome the issue of extensively large test suites.

**Test Suite Minimization**

Test suite minimization is optimization problem to find a minimally-sized subset of the test cases in a suite that exercises the same set of coverage requirements as the original suite. The redundant test cases in the test suite with respect to the coverage of some particular set of program requirements need to be removed. The test suite minimization problem is an illustration of the more general set-cover problem. The problem when given with a input of collection S of sets, each set covers a specific sort of entities, is to acquire a minimally-sized subset of S providing the same percentage of entity coverage as the original set S. The set-cover problem has been shown to be NP-Complete [2], and that is why there does not exist any known polynomial-time algorithm to desirably solve the minimization problem.

Nonetheless, in the area of computing optimally-minimized suites there has been some research work [3, 4]. Heuristics have been relied on for calculating near – optimal solutions. S Chvatál [5] presented a simple greedy heuristic for the set-cover problem in which each candidate set has a cost associated with it. Jones and Harrold [6] described two minimization heuristics that are designed specifically to be used in conjunction with the relatively strong modified condition/decision coverage criterion; one algorithm builds a minimized suite incrementally by identifying essential and redundant test cases, while the other algorithm is based on a prioritization technique that simply stops computing before all test cases in a suite have been prioritized. Agrawal’s work [8] implies a framework for minimization of suites using the notions of mega blocks and global dominator graphs. An algorithm based on a greedy heuristic for reducing the size of a test suite (referred to henceforth as the HGS algorithm) was developed by Harrold, Gupta and Soffa [7]

**Software Quality**

Software functional quality reflects how well it complies with or conforms to a given design, based on functional requirements or specifications. The product should meet the customer and users essential. It adheres to two types of norms. Internal norm: the end user is not concerned about it, it is code dependent and is for coder only. External norm: is related to the end user and his experience while using the interface of the product [1].

**a. Software Quality as a Layered Technology** A layered technology assists for constant improvement in
software growth process. For the software growth it has been divided into four layers and each layer tally to each other. These layers are represented in the figure 1.1. It displays all the four layers upon which the software engineering depend so as to develop superior software. If all the layers are considered during build process, it tends to fulfill maximum user needs and assists to move smoothly to whole the engineering process.

![Figure 1: Software quality process](image)

**1. Tools:** These are the software engineering tools which provide comfort up for the other two layers, that is, process and methods. The tools are either programmed or semi-programmed.

**2. Methods:** The methods regularly state how to develop software. They contain a type of phases of software development life cycle and offer special modeling techniques and the descriptive terminologies.

**3. Process:** It specifies the key process areas where the software is developed, organization of the deadlines, observing the supremacy of the software and focusing on maintenance.

**4. Quality focus:** Determines how well the software meets the functional and the non-functional needs and assists to develop the more improved development approaches.

**Related Work**

Artificial Bee Colony optimization that belongs to non-pheromone based swarm intelligence algorithms is considered suitable for many optimization based problems. Dervis Karaboga et.al applied Artificial Bee Colony (ABC) Optimization Algorithm for Solving Constrained Optimization Problems. In their paper, they showed superior performance of ABC in solving a set of constrained optimization problems [9].

Dervis Karaboga et.al, proposed the use of ABC in solving numerical function optimization. They compared the efficiency of ABC with other optimization algorithms such as GA, PCO and PS-EA and proved that ABC outperformed them [17].

Dusan Teodorovic et.al, proposed a bees algorithm to solve difficult combinatorial optimization problems. In their paper, in addition to proposing the Bee Colony Optimization (BCO) as a new meta heuristic, they also described two BCO algorithms called the bee system (BS) and the fuzzy bee system (FBS). In the case of FBS, the agents (artificial bees) use approximate reasoning and rules of fuzzy logic in their communication and acting. In this way, the FBS is capable to solve deterministic combinatorial problems, as well as combinatorial problems characterized by uncertainty [12].

Li-Pei Wong et.al, presented an improved Bee Colony Optimization algorithm with Big Valley landscape exploitation as a biologically inspired approach to solve the Job Shop Scheduling problem. They compared the experimental results of the proposed algorithm with Shifting Bottleneck Heuristic, Tabu Search Algorithm and Bee Colony Algorithm with Neighborhood Search on Taillard JSSP benchmark and they showed that it is comparable to these approaches [11].

Dušan Teodorovic et.al, proposed a bee colony optimization approach to complex transportation problems [10].

Adil Baykaslo Lu et.al, applied Artificial Bee Colony (ABC) Algorithm to Generalized Assignment Problem in the book on Swarm Intelligence: Focus on Ant and Particle Swarm Optimization [13].

Alok Singh applied Artificial Bee Colony (ABC) Algorithm to the leaf constrained minimum spanning tree problem. Given an undirected, connected, weighted graph, the leaf-constrained minimum spanning tree (LCMST)
problem seeks on this graph a spanning tree of minimum weight among all the spanning trees of the graph that have at least ‘l’ leaves. In this research work, they proposed an artificial bee colony (ABC) algorithm for the LCMST problem. They compared the performance of ABC approach against the existing best approaches. They demonstrated the superiority of the new ABC approach over all the other approaches based on computational results. The new approach obtained better quality solutions in shorter time [14].


Nurhan Karaboga proposed a new design method based on artificial bee colony algorithm for digital IIR filters [16].

Mohammad et.al. have applied honey-bee mating optimization algorithm on clustering in data mining. They applied honey-bee mating technique to avoid local optima in cluster analysis. They also compared the efficiency of the approach with other approaches such as GA, SA, TS, and ACO, on several well-known data sets. Their finding shows that the proposed algorithm works well than the best one [17].

The analysis on software testing issues has been focused principally on the Independent Path Generation and Software Test Suite Optimization. Many intelligent swarm computing software test procedures are enormously proposed and each has its pros and cons. The crucial findings obtained as part of the environment/framework research are:

Neural Networks has a black box data processing structure and has difficulties in programming. It also deals with algorithmic complexity.

Tabu search faces challenges in the amount of memory needed for the prevention of stuck up at local optima and it has short term memory to remember all the test cases in the current search.

Genetic Algorithms has obstacles in giving stable results (stuck up at local optima); the coupling is slow and has inexplicit memorization of best individual result.

Ant Colony Optimization has disadvantages of lengthier test series and duplication of nodes within the same sequence. In the beginning phase, it begins with searching gradually, deprives pheromone and certainly inclines to local optimum. In case of high quantity ACO has disadvantages of early convergence.

Proposed Method
We suggest an ABC based search algorithm to create test data. In this research work, the responsibility of the bee is extended to do the evaluation and supervision job which eventually decreases the manual task and built the reliance on the product by verifying it with the description of the given software. Bee colony consists of three types of bees, namely scout bees, which unmethodically explores for the food sources, onlooker bee determines which food sources to be investigated next from the list food sources given by scout bees, and lastly employee bees will search for new food source in vicinity of depleted food source. Moreover detailed description of bee colony is given in proposed scheme.

A. Proposed Method Flowchart
There are three groups of artificial honey bees in ABC algorithmic rule; employed bees, onlookers and scouts. The previous portion of the community contains of the employed/busy artificial bees and the other portion contains the onlookers/observers. There is an employed bee for every food source. In one way the number of employed bees is equal to the number of food sources around the hive. Scout bees are those which were employed bee at a time but their food source has depleted. The sequence chart of the ABC algorithmic rule is shown.
The pseudo code of the ABC algorithmic rule as mentioned:

1. set default values for group of solutions
2. Assess the group of solutions.
3. Iteration = 1.
4. Reiterate.
5. construct new results for the employed bees and assess them
6. Exercise the greedy selection procedure.
7. Determine the probability value pi for the solutions
8. Build the new solutions for the onlookers depending on pi values and analyze them.
9. Exercise the selection process.
10. Find out the uninhabited solution for the scout, if exists and substitute it with a recent unmethodical produced solution.
11. Retain the perfect solution obtained so far.
12. Iteration = Iteration+1.
13. Iterate until cycle = MCN.

B. ABC Algorithm

Advantages of ABC algorithm:

Strengths The major capabilities of the ABC algorithm are presented as below:

a) One most important facility can be obtained while applying this kind of optimization method for solving combinatorial and numerical optimization problems is that this method is very much flexible and can easily be modified according to the problem. Moreover, this method can easily be applied to various problems.

b) The neighborhood (local) search is performed at two levels by busy bees and observer bees in each iteration and universal search is done by scout bees when the process stuck to local optima make the search process very strong. For this reason, the ABC algorithm can give optimal (or nearly optimal) solution to almost any kind of unimodal or multimodal type of problem (or combination of both)
having linear or any kind of nonlinear objective function or constraints.

c) The ABC algorithm is quite robust. Unlike the other gradient-based techniques, it has no tendency to blow up if it starts too far from the optima. It does not require the user to specify starting point.

d) This algorithm also performs quite well when there are lots of local optima to avoid.

e) The positive feedback in the ABC algorithm results in speedy detection of capable result.

f) The ABC algorithm uses distributed computation, which circumvents early merging.

g) The insatiable heuristic applied in ABC algorithm assist in reaching to a satisfactory result in the advance stages of the search process

C. ABC Optimization Algorithm for Test suit optimization

Load the test Cases by preliminary test data created by the tool. Equivalence partitioning and boundary value analysis techniques are used to generate preliminary test data.

Step 1: Artificial bees are employed to the test case of the first feasible module of the program. Employed bees receive the fitness value of the test cases and the node information.

Step 2: Onlooker bees receive the module information and fitness value of the test cases as the input and gauges the fitness of every test case received from the busy bees. Then the test case with the maximum fitness value is memorized in the database

Step 3: The adjacent modules of the visited modules are explored by the employed bees and the fitness value of the current test case is assessed in comparison to the neighboring nodes by the selector bee.

Step 4: The module with the maximum fitness value is opted and marked with the current node which defines the new test path. Optimal Test repository stores all this information.

Step 5: Temporary abandoned repository collects all the remaining modules and test cases apart from the ones selected.

Step 6: Steps 3-6 are repeated until the test path is complete. Else the modules and test cases from the abandoned database are selected for the next test path creation by the busy bees.

Step 7: The scout bee pilots new group of test cases and substitutes the test cases of the abandoned repository with the new test cases if the selector bee finds the shortlisted test cases are inefficient.

The above steps are repeated until the end user established end condition is met. Every algorithm needs to be converted to its pseudo code before it can be implemented. Only then a program to develop its application can be developed. The descriptive pseudo code of the Test Case Optimization algorithm using ABC method is defined below.

The following is the detailed algorithm

1. Set the test cases which are executed by the search bee.  
2. Determine an executable state and assess the test nodes  
3. Set the current traversal path as Iteration = 1  
4. Repeat  
5. Create new test cases $z_{ij}$ in the vicinity of $x_{ij}$ for the selector bee with the help of formula  
   \[ X_{ij} = \min_j + \text{rand}(0, 1)(\max_j - \min_j) \]  
   - $X_{ij}$ is the initial test case  
   - $\min_j$ is the minimum no. of test cases  
   - $\max_j$ is the maximum no. of test cases  
   - $\text{rand}()$ is a random number generation function which selects either 1 or 0 randomly.

6. Create the test data using equivalence partitioning and boundary value analysis.

7. Apply greedy selection process for the generated test data.

8. Traverse the SUT with the generated Test Data and calculate the fitness value.

9. The onlooker selects the test cases with the highest fitness value and abandons the rest
10. The traversal process is carried out till a particular test data with 100% fitness value and 0% fitness value is produced

\[ P_i = \frac{\text{fit}_i}{\sum \text{fit}_i} \]

Where \( P_i \) is the probability function which signifies the probability with which the \( i_{th} \) test data traverses an independent test path successfully.

11. Add the test case to the optimal repository.

12. In the next iteration scouts generate the new test data and go to step 5

**Result and Evaluation**

To execute the above algorithm, our technique is to use the ABC Test Suite Optimization tool to optimize the Test Cases by adapting the ABC algorithm. The tool takes as input a program to create independent routes. Using the created independent routes Test Cases are traversed along the paths with the help of ABC algorithm. By doing so the test cases with highest coverage (High fitness Value) are discovered which leads to the creation of final optimal Test Suite.

---

![Figure 3: Optimized test suit path](image)

**Conclusion**

In our proposed method test case enhancement is achieved using ABC to upgrade quality. The suggested algorithm uses ABC methodology to work towards rapid and systematic ways for creating efficient test suite. Also the algorithm is contrasted with earlier algorithms and previous ABC methods, and the outcome confirms that the suggested algorithm is defeating other algorithms and methods. In the suggested ABC method, efficient test suite is produced for each independent path of the program where each path will have two types of data.

In the suggested method every test case would be equal to a food source and the target of the method would be to discover best food sources i.e. test cases with maximum coverage. Every food source position depicts to a capable solution of the optimization problem and the nectar quantity of the food source relates to the quality (fitness) of the respective solution.

**References**


