International Journal of Intelligent Information Technologies
Volume 19 • Issue 1

On the Application of Quick Artificial Bee Colony Algorithm (qABC) for Attenuation of Test Suite in Real-Time Software Applications

Jeya Mala D., Vellore Institute of Technology, Chennai, India
Ramalakshmi Prabha, Anna University, Madurai, India

ABSTRACT

Software testing plays a vital role during the software development process, as it ensures quality software deployment. Success of software testing depends on the design of effective test cases. To achieve the optimization of generated test cases, the proposed approach combines both global and local searches by means of intelligent agents which exhibit the behaviour of employed bees, onlooker bees, and scout bees in the qABC algorithm. The proposed qABC algorithm has key improvements over the basic artificial bee colony algorithm (ABC) in test optimization by reducing redundancy, filtering of test cases in each iteration and parallel working of the bees. Further, the fitness evaluation of the test cases is done by employing two test adequacy metrics namely path coverage and mutation score. Further, the experimental evaluation of qABC, GA, and the basic ABC based test cases is done using several case study applications. The result shows that qABC outperforms the other algorithms in terms of effectiveness of test cases in revealing the faults with less time and a smaller number of test cases.

KEYWORDS

Project Management, Quick Artificial Bee Colony (qABC), Software Quality, Software Testing, Software Test Optimization, Test Cases, Validation

INTRODUCTION

Every software industry is expected to deliver quality products to satisfy their customers and to compete in the market. Software testing is one of the ways of improving software quality. As per 40-20-40 principle, software development devotes 40% of aggregate time for project analysis and design, 20% for programming and the remaining 40% for testing (Pressman, 2005).

According to IEEE, software testing is defined as the process of operating a system or component under specified conditions, observing or recording the results, and making an evaluation of some

DOI: 10.4018/IJIIT.318673

aspects of the system or component (IEEE Standard 610.12-1990). Which clearly indicates that a better testing approach ought to be emulated by the industries for producing better products. The quality of testing depends on the design of effective test cases.

The testing process is a time consuming one because it involves human resources more and more. Even though several testing tools are available in the market; the industries still continue the traditional manual writing test cases for the attainment of expected goal. So, the industries are in need to perform testing effectively within reasonable time.

The researchers usually preferred the evolutionary algorithm and swarm-based algorithm to find the solution for real world optimization problem- especially NP hard problems. Test case generation can also be non-deterministic (NP-hard) problem (Karaboga & Gorkemli, 2014). Hence this paper proposed a novel methodology for automatic generation of test data by using quick Artificial Bee Colony algorithm (qABC) which extends the functionality of the ABC algorithm.

Artificial Bee Colony (ABC) algorithm is a population based Swarm Intellgence (SI) algorithm designed by Karaboga (2014). ABC simulates the behaviour of foraging and waggle dance of the honeybees in nature (Karaboga & Basturk 2008). This algorithm consists of three types of artificial bees namely employed bees, onlooker bees and scout bees. ABC algorithm produces better results in many applications such as decision making, transportation problem etc. It is also used in test case generation. It generates the test cases effectively. (DJ Mala, 2017),

However, the proposed qABC improves the ABC algorithm in terms of generating effective test cases within a short period. qABC utilizes three bees as in ABC for generating test cases. The changes are introduced in Onlooker bees' behaviour and redundancy removal techniques are applied which is not available in basic or traditional ABC. In this way, ABC generates the test cases faster and accurately. Because coverage-based metrics will reveal errors effectively here, path coverage and mutation score are taken as test adequacy criteria. In this paper, qABC algorithm is explained in detail and then its performance is assessed by evaluating it on a set of test problems. Also, the performance of qABC is compared with other evolutionary algorithms. The results show that its performance is better than other evolutionary algorithms.

BACKGROUND

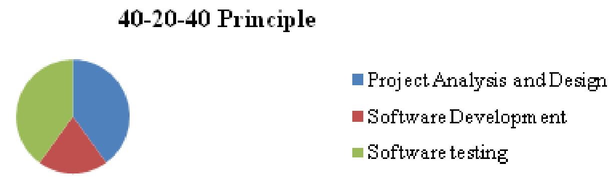
This section briefs the definition relevant to the key terms used in the proposed approach.

Test Case – A test case is a set of inputs, execution preconditions, and expected outcomes developed for a particular objective, such as to exercise a particular program path or to verify compliance with a specific requirement. [IEEEdo178b].

Test Optimization – To maximize the profit of finding more bugs and coverage and to minimize the total number of test cases needed.

Population Based algorithm – This type of algorithm begins with a population of initial solution and generates the population to produce the better solution by performing some operations such as Crossover, Mutation etc.

Figure 1. 40 -20-40 principle



Swarm Intelligence based algorithm – This type of algorithm is inspired by the collective behaviour of social insect colonies and other animal societies
Exploration - It is the process which refers to the ability of the algorithm to produce better solution by applying the knowledge of previous solutions.
Test Case Adequacy - A test case is adequate if it is useful in detecting faults in a program [8].
Mutation - It is a single syntactic change that is made to a program statement.
Mutation score – It is the percentage of non-equivalent mutants killed by the test data [8].
Mutation Score = 100 * D / (N - E)
D = Dead mutants
N = Number of mutants
E = Number of equivalent mutants

The rest of this paper is organised as follows: Section 2 highlights the related work; Section 3 describes the problem representation. Section 4 deals with the standard or basic ABC and qABC-Testcase Generator is outlined in Section 5. The case studies and the results are analysed in Section 6 and finally, Section 7, carries the conclusion.

LITERATURE SURVEY

Several research works have been proposed to solve hard optimization problems. Some of them are discussed in this section.

Karaboga and Gorkemlim (2019), have introduced new versions of ABC algorithm to solve Travelling Salesman Problem (TSP). One of these is the combinatorial version of standard ABC, called combinatorial ABC (CABC) algorithm. The other one is an improved version of CABC algorithm, called quick CABC (qCABC) algorithm. This work has produced some promising results in solving TSP.

Moradi et al. (2018) have proposed a clustering and memory-based chaotic artificial bee colony algorithm, denoted by CMCABC, for solving the dynamic optimization problems. A chaotic system has a more accurate prediction for future in the real-world applications compared to a random system, because in the real-world chaotic behaviours have emerged, but random behaviours have not been observed. In the proposed CMCABC method, explicit memory has been used to save the previous good solutions which are not very old. Maintaining diversity in the dynamic environments is one of the fundamental challenges while solving the dynamic optimization problems

Spieker et al. (2017) have introduced a new method for automatically learning test case selection and prioritization in Continuous Integration (CI) with the goal to minimize the round-trip time between code commits and developer feedback on failed test cases. This proposed method uses reinforcement learning to select and prioritize test cases according to their duration, previous execution and failure history

Bao et al. (2017) have proposed an improved adaptive genetic algorithm (IAGA) for test cases generation by maintaining population diversity. It uses adaptive crossover rate and mutation rate in dynamic adjustment according to the differences between individual similarity and fitness values, which enhances the exploitation of searching for global optimal solution.

Boopathy et al. (2017), have proposed a combination of Markov chain and Artificial Bee Colony (ABC) optimization techniques to attain the software code coverage. Initially, dd-graph is captured from the control flow graph of the source code and is represented as a Markov chain. Then, the number of paths is obtained based on linear code sequence and jump (LCSAJ) coverage. Further, ABC optimization is adopted to ensure software code coverage. The initial population is randomly selected from the test suite and populated for subsequent generations using the ABC algorithm. The test cases are generated for three mixed data type variables, namely integer, float and Boolean.

Aghdam and Arasteh (2017) proposed a method that uses Artificial Bee Colony (ABC) algorithm for solving the issue of test data generation and branch coverage criterion was used as a fitness function for optimizing the proposed solutions.

Sharma et al. (2016). have presented a set of methods that uses a Genetic Algorithm (GA) for automatic test-data generation in software testing. They have presented various Genetic Algorithm (GA) based test methods which will be having different parameters to automate the structural-oriented test data generation on the basis of internal program structure. The factors discovered are used in evaluating the fitness function of GA to select the best possible test method. These methods take the test population as an input and then evaluate the test cases for that program. This integration will help in improving the overall performance of genetic algorithm in search space exploration and exploitation with better convergence rate.

Mao et al. (2015), in their paper have reformed the basic ACO algorithm into discrete version so as to generate test data for structural testing. First, the technical roadmap of combining the adapted ACO algorithm and test process together is introduced. In order to improve algorithm's searching ability and generate more diverse test inputs, some strategies such as local transfer, global transfer and pheromone update are defined and applied. The coverage for program elements is a special optimization objective, so the customized fitness function is constructed in their approach through comprehensively considering the nesting level and predicate type of branch.

Wong and Choong (2015) 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 their algorithm with Shifting Bottleneck Heuristic, Tabu Search Algorithm and Bee Colony Algorithm with neighbourhood Search on Taillard JSSP benchmark and they showed that it is comparable to these approaches.

Tinggui, and Xiao (2014) have also enhanced artificial bee colony (ABC) algorithm with self-adaptive searching strategy and artificial immune network operators for global optimization.

Nokolic and Teodorovic (2013) proposed a bee's algorithm to solve difficult combinatorial optimization problems. In their paper, in addition to proposing the Bee Colony Optimization (BCO) as a new metaheuristic, they also described two BCO algorithms called the Bee System (BS) and the Fuzzy Bee System (FBS). In 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.

Lijuan et al. (2012), have proposed a practical model, which utilizes GA as the searching policy to generate software structural test data. To achieve higher performance, the issues such as encoding strategy, evolution operator, evaluation function construction and instrumentation were addressed in detail. A new method of initialization of population was introduced in order to make the initial population to have higher adaptability, and much emphasis is put on algorithms' operator evolution, which is a key factor that can highly affect algorithms efficiency.

Lam et al. (2012), combined both global search methods done by scout bees and local search method done by employed bees and onlooker bees in their modified ABC. The parallel behaviour of these three bees makes generation of feasible independent paths and software test suite optimization faster. Test Cases are generated using Test Path Sequence Comparison Method as the fitness value objective function.

Mala and Mohan (2010) have applied the general ABC algorithm for optimal test cases generation and they proved that, this ABC is outperforming the GA and Bacteriologic algorithms in test case generation and optimization by means of a number of case studies. But one of the weak areas in their work is the number of cycles is increased as the number of nodes is increasing in the software.

Karaboga and Basturk (2008) 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.

Karaboga and Basturk (2008) have proved that a most optimistic and an efficient bio-inspired algorithm is artificial bee colony (ABC) for optimization problems such as TSP etc.

Vanmali et al, (2002) identified that Tabu search requires more memory to reduce the local optima and searching while solving optimization problems.

PROBLEM REPRESENTATION

Optimization is the process of modifying a system to work more effectively or use fewer resources. Test Suit Optimization is the process of selecting or generating the test cases that cover the testing components within less time in other words to maximize the profit of coverage and minimize the test cases.

In this auto generation of test cases using qABC, path coverage is considered as test adequacy criterion for test suit optimization. Test cases are selected by coverage value associated with each path. The objective of the function is to maximize the coverage value.

The objective function of the proposed approach for test case optimization isMax.

Coverage_Value (Path) (1)

Sub to.

1 if $\sum_{i=1}^{n} fitenss$ (node$_i$,testcase)=100%

Coverage_value(Path)= (2)

0 otherwise

Eq. (1) is to maximize the coverage of path. The constraint (2) indicates that if the entire node in given path is covered by test case, the coverage value will be set into 1otherwise will be set in to 0. Nodes coverage is identified by calculating fitness of the node with the test case.

ABC ALGORITHM

Artificial bee colony (ABC) algorithm introduced by Dervis Karaboga on inspiration by the foraging behaviour and waggle dance of the honeybees. It is the most popular swarm intelligence-based optimization algorithm.

The ABC algorithm consists of three groups of bees, namely, employed, onlookers and scouts. The bees leave the hive and fly around the search space with food position in their mind and on finding which, they dance in the hive about their new food position.

The onlooker bees just watch this dance and decide the food sources to exploit. Regardless of any information about the food source, Scouts bees fly and choose the food sources randomly. If the new source has the higher quantum of nectar than that of the previous one in their memory, they memorize the new position and forget the previous one [10]. Initially all of the bees in the hive work as scouts and they all start with random solutions or food sources. In further cycles, the employee bee whose food source is abandoned becomes a scout and starts to find new food source.

Steps of ABC Algorithm

In ABC algorithm, a food source position is defined as a possible solution and the nectar amount or quality of the food source corresponds to the fitness of the related solution in optimization process.

Since, each employed bee is associated with one and only one food source, the number of employed bees is equal to the number of food sources [5].

The steps of the basic ABC algorithm are given below:

Step 1: Initialization of food sources

Step 2: REPEAT

- Employed bees leave the hive to the food source in their mind and finds a neighbour source, then evaluates its nectar amount and dances in the hive
- Onlooker bee watches the dance of employed bees and locates one of their sources guided by the dances, then goes to that source and finds the neighbour source around the food sources, and evaluates its nectar amount.
- Abandoned food sources are determined and then, they are replaced with the new food source by Scout bees.
- The best food source is memorised.

Step 3: UNTIL (requirements are met)

Initialization Phase

In the initialization phase, initial solution which is known as initial population of optimization problem is generated randomly. The initial population is represented as $x_{i.}$ Where i ε {1, 2..., SN}, SN is the randomly chosen index.

Food sources are randomly initialised with Eq. (3) in a given range.

$$x_i^j = x_{min}^j + \text{rand}(0,1)\left(x_{max}^j - x_{min}^j\right) \tag{3}$$

Where x_{min}^j lower bound, x_{max}^j upper bound of the variable j, x_i^j value of j for dimension i. Where j ε {1, 2… D}

Employed Bees Phase

The employed bees select the value x_i from initial population and find a neighbour food source v_i by using Eq. (2).

$$v_i^j = x_i^j + \text{rand}(0,1)\left(x_i^j - x_k^j\right) \tag{4}$$

Where$x_k$is a food source selected randomly. k ε {1, 2, . . ., SN}, Now, the fitness values of v_i^j and x_i^j are calculated. Based on the comparison of both the fitness values choose any of one the test case by applying Greedy selection process. The fitness value Fitness(x_i) is calculated from its objective function value Fit(xi)by using Eq. (5).

1/ (1 + Fit (x_i)) if Fit (xi) ^{3} 0

Fitness(x_i) =

1 + |Fit (x_i)| if Fit (x_i)< 0 (5)

Onlooker Bees Phase

The employed bees return to the hive, and share their information of food source with onlooker bees. An onlooker chooses her food source based on the probability of the fitness value. The probability value P_j can be calculated by Eq. (6).

$$P_j = \text{Fitness}\left(x_i\right) / \sum_{j=1}^{SN} \text{Fitness}\left(x_i\right) \tag{6}$$

Scout Bees Phase

Abandoned solution is determined. If it exists, the employed bee concerned becomes scout bee and it replaces the abandoned test case by new solution generated by using E

PROPOSED QABC-TESTCASE GENERATOR

The framework of qABC-Testcase Generator is shown in Figure 2. The qABC-Testcase- Test case Generator has two phases, the first of which is Scanning phase and the second is Generation phase. Initially the component to be tested is identified and it is scanned in scanning phase. This phase retrieves the features of the component such as number of paths and is stored in the repository of Component's Traits Data Base (CTDB). With the help of this information the Generation phase generates the test cases and store it in the repository Test Case Data Base (TCDB).

Scanning Phase

Figure. 3 shows the scanning phase where in, the component to be tested is scanned to get the traits of the component such as number of paths, node list and the data member present in the component. The number of paths in the component is calculated by using the metrics Cyclomatic complexity. Then, this information is stored in the repository known as Component's Traits DataBase (CTDB).

Generation Phase

The diagram in Figure 4 shows the Generation phase wherein, test cases are generated for the component to be tested by the proposed algorithm qABC which is developed from Artificial Bee Colony Algorithm (ABC).
Step 1: Initialization process begins. Test path is chosen from the database CTDB.
Each Employee bee do the following operations:
Step 2: By using initial test cases, the new test case is generated for the chosen path. Fitness value of test cases is evaluated at the first node in the path. By applying greedy selection process any one of the test cases which has a higher fitness value, is selected.

Figure 2. The framework of qABC-Testcase generator

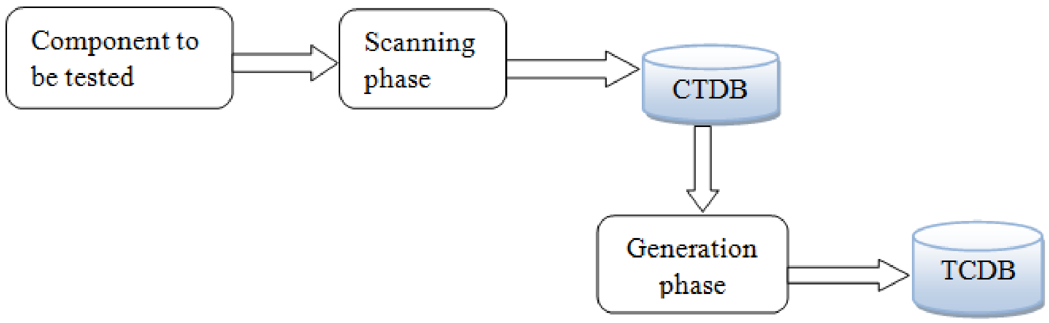
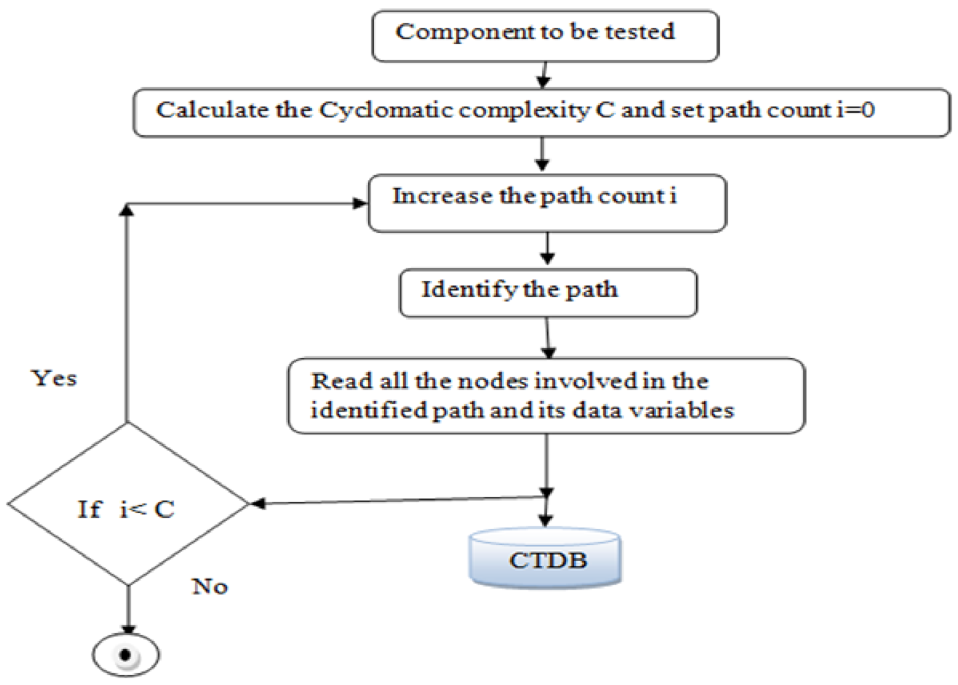


Figure 3. The scanning phase of qABC


Component to be tested
Calculate the Cyclomatic complexity C and set path count i=0
Increase the path count i
Identify the path
Read all the nodes involved in the identified path and its data variables
Yes
If i< C
No
CTDB

Step 3: By using the selected test case, the fitness value of each node in the selected path is calculated. If any one of the nodes is not covered by the test case, it will be considered as unfit, and the next uncovered path will be selected which does not have the unfit node.
Step 4: By using fitness values the probability is calculated.
Step 5: If the probability is 100%, the test case and the path will be stored in the database Test Case Data Base (TCDB). Then go to step 7
Step 6: Otherwise the test case will be marked as abandoned, and a new test case will be generated. Go to Step 3.
Step 7: Select the path from CTDB and the test case from TCDB for test case generation and follow step 2.

PROPOSED QABC ALGORITHM FOR TEST OPTIMIZATION

The phases of qABC algorithm are similar to those of ABC algorithm.

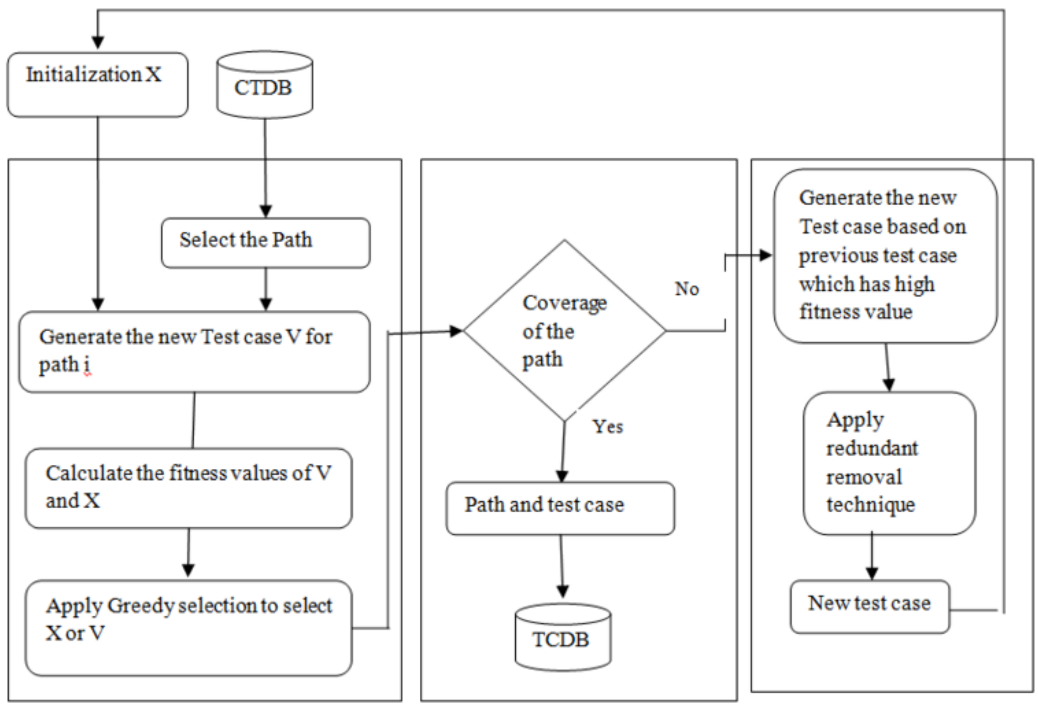
```
Initialization Phase
Repeat
Investigation Phase (Employee Bee)
Decision Phase (Onlooker Bee)
Replace Phase (Scout Bee)
Until (Maximum cycle)
```

Initialization Phase

In this phase, the following initialization can be done.

1. Control parameter Initialization:

Figure 4. The generation phase of qABC


Initialization X
CTDB
Select the Path
Generate the new Test case V for path i
Calculate the fitness values of V and X
Apply Greedy selection to select X or V
Coverage of the path
No
Yes
Path and test case
TCDB
Generate the new Test case based on previous test case which has high fitness value
Apply redundant removal technique
New test case

a. Number of paths by calculating cyclomatic complexity of the component to be tested. Number of employee bee is equal to the number of paths in the test component.
b. Total number of variables in each path, which is known as dimension D.
c. Maximum number of cycles.

2. Test case Initialization: Initialize the test cases for each path.

Initialization task is important because the initial population can affect the convergence speed. Owing to the randomness and sensitivity dependence on the initial conditions, chaotic maps have been used to initialize the population so that the search space information can be extracted to increase the population diversity.

Initialization task can be performed in the following way:

- Random initialization
- The chaotic systems with affinity-based compression

The test cases are generated using random initialization by Eq. (1).

Chaotic map is used for generating the initial value to improve the global convergence. cm_n is sinus map and is calculated by using Eq. (7).

$$cm_{n+1} = 2.3(cm_n)^{2\sin(\pi cm_n)} \tag{7}$$

Where $cm_n = 0,1,2,\ldots..,\ N, n$ is the iteration counter and Nis the maximum number of chaotic iterations. Here, N is randomly chosen as 50

$$x_i^j = x_{min}^j + cm_n^j \left(x_{max}^j - x_{min}^j\right) \tag{8}$$

For initial test case generation, random method is applied, because it produces wide range of values within the boundary when compared to the chaotic method. The Figure5 shows that the 100 values generated within the range of 1 to 100 in random based and Figure6 shows that chaotic systems based. Figure5 displays initial values span across the boundary, but in Figure6 the most of values falls in range from 30 to 60. Hence for initialization random method is preferred.

Figure 5. Initialization using randomization

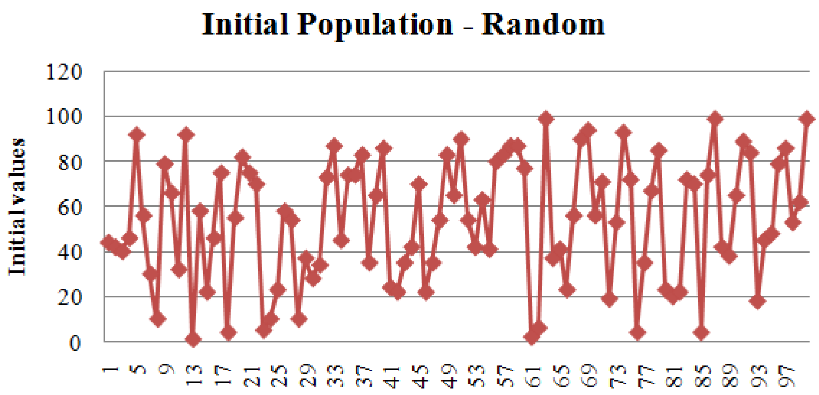
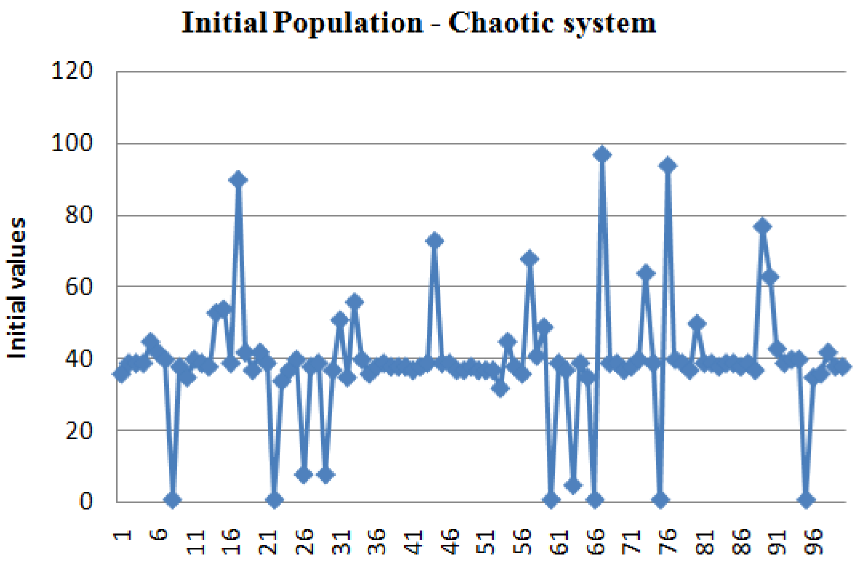


Figure 6. Initialization using chaotic method



Investigation Phase (Employee Bee)

In this phase each employed bee generates neighbour solution v_i of initial value x_i using Eq. (4). Then they find the fitness value of $v_i$and x_i using Eq. (5). They apply greedy selection method; either $v_i$or x_i will be selected based on their fitness value.

Suppose the number of paths in the component to be tested is five then five employed bees are involved.

1. Each employed bee memorises different test case from the initial solution and using this they generate new solution.
2. All employed bees work in parallel and initially all the nodes are considered as uncovered.
3. Each bee does the following:
   a. It calculates the fitness value for the first uncovered node of the first path by the memorised test case and newly generated test case. Then compare the fitness value and select any one of the test case using greedy selection process.
   b. The selected test case is used to calculate the fitness value of the next uncovered node in the path. If this value is fit for the node and the node is considered as covered node, the node name and the fitness value is memorized then it continues to calculate the fitness value for the next uncovered nodes in the selected path. If any node is not covered then it will be considered as unfit node and the bee moves to next path only it does not have the unfit node. It finds the fitness for the uncovered node in the new path and follows the step 3.b.
   c. If all the paths in the software are tested, then it moves to next phase. The test case, and the path is memorized when all the nodes in the path is covered.
4. The fitness value can be calculated by using Fitness (Xi), This function helps to check the node coverage and returns the coverage value. It reads the nodes from CTDB and finds the coverage value by using the following conditions.
   (i) For a>b, if a > b, Fiti = -1, else Fit_i =1or 0
   (ii) For a>=b, if a>=b,Fit_i = -1, else Fit_i =1or 0
   (iii) For a<b, if a <b, Fit_i = -1, else Fit_i =1or 0
   (iv) For a<=b, if a<=b, Fit_i = -1, else Fit_i =1or 0
   (v) For a==b, if (a==b), Fit_i = -1, else Fit_i =1or 0
   (vi) For a! =b,if a!=b, Fit_i = -1, else Fit_i =1or 0
   (vii) For a OR b, Fiti = a OR b
   (viii) For a AND b,Fit_i = a AND b

Decision Phase (Onlooker Bee)

1. Onlooker bee calculates the probability value of each test case which is informed by the employed bees.
2. The test case which has 100% probability for the path will be memorized.
3. If any path has not yet been covered, select the test case x_i which has high probability. Then find the neighbours of x_i.
4. Eq. (9) is used to find whether the generated test case is neighbour of $x_{i.}$

$$d\left(i,j\right) \le r * md_i \tag{9}$$

5. r is neighbourhood radius. Euclidean distance between x_i and x_j is calculated using Eq. (10). Eq. 11 is used to check the whether both values are same.

$$d\left(x_i^k, x_j^k\right) = \sqrt{\sum_{k=1}^{D}\left(x_i^k - x_j^k\right)^2} \tag{10}$$

$$A\left(x_i^j, v_i^j\right) = \frac{1}{1 + d\left(x_i^j, v_i^j\right)} < \alpha \tag{11}$$

Where α is a threshold value.α =1 indicates both test cases v_i^j and x_i^j are same. α = 0 shows that, there is no similarity between test cases v_i^j and x_i^j.

Hence, here 0.1 is considered as threshold value.

6. Euclidean distance between x_i and x_j is represented as d (i,j). Mean Euclidean distance for x_i is represented as md_i and is calculated by Eq. (12).

$$md_i = \sum_{j=1}^{SN} \frac{d\left(i, j\right)}{SN - 1} \tag{12}$$

7. Select the test case among the neighbours and generate new test case. Fitness value of the new test case is calculated for uncovered path. If it is covered now, it will be memorized.

Replace Phase (Scout Bee)

The scout bee determines the abandoned test case. If it is there, it replaces the test case by a new test case.

EXPERIMENTAL EVALUATION

A number of applications are taken to test the performance of the proposed quick Artificial Bee Colony algorithm (qABC). These case studies are of different sizes which are varied in number of class components.

Test cases are generated, and the time taken for test case generation is also measured. The test case is evaluated by path coverage and mutation score. Then, the qABC is compared with Genetic Algorithm (GA) and Artificial Bee Colony algorithm (ABC).

For a simulative case study, the Mark Processing System is taken, and the steps involved in the proposed algorithm are exhibited with the test adequacy criteria for readers' understanding.

Case Study: Mark Processing System

Mark processing system is taken for explanatory purpose. This system consists of three predicate nodes. The conditions are made up of variables intmark and extmark and att. The sample code for student mark processing system is as follows.

```
void main()
{
1.          int intmark,extmark,att;
2.          read intmark,extmark,att;
3.          if(ext>50)
{
4.          printf("pass");
5.          If(att>90)
6.          int Bonusmark=10;
}
7.          else if ((int+ext)>40)
8.          printf("Pass");
9.           else
10.          printf("Fail ");
11.          }
```

Scanning Phase

Here the Cyclomatic value is 4. This value and the condition variables intmark, extmark and the 4 paths are stored in the database CTDB.

Generation Phase:

Step 1. The initial values of intmark, extmark, TotFitness for every path will be initialized. It is given in Table 1.

- Three variables intmark,extmark,att are initialized. Here TotFitness value indicates the maximum fitness value of the given path. If the TotFitness value is zero, it indicates that the path does not satisfy any condition.
- The number of paths of this program is 4.
- Repeat
- For all the paths, new test case is generated by using the following formula:

$$v_i^j = x_i^j + \ddot{o}\left(x_i^j - x_k^j\right)$$

- i indicates variables and j indicates path remove ', for example v11 indicates extmark, v12 is intmark and v13 att.'
- PATH 1.
- For i=1 to 3, j=1, k=1+ (i% number of variables used here)
  - i=1, k=2, $\varphi = 0.125$
    v11 = x11+ φ (x11-x12); v11= 65+(0.86667) (65-35) = 91
  - i=2, k=3
    v12=x12+ φ (x12-x13); v12=35+(0.74) (35-85) = -2
  - i=3, k=1
    v13=x13+ φ (x13-x11); v13=85+(0.75) (85-65) = 100

Table 1. Execution paths with fitness value

Path No	Path	Extmark	intmark	Att	TotFitness
P1	1-2-3-7-9-10-11	22	14	52	0
P2	1-2-3-4-5-6-11	65	35	85	4
P3	1-2-3-4-5-11	85	18	17	2
P4	1-2-3-7-8-11	3	69	89	2

- xi1= {65.35,85} and vi1= {91,-2,100} are applied to the first predicate node 3, that is fitness value is calculated by using the equations 5. The fitness value of xi1 is 2 and vi1 is 2. Both test cases have high fitness value. But TotFitness value of path1 is zero. Hence both test cases are not suitable to path1. Hence the process moves to the next path which includes the node3.

  For PATH2

- Greedy selection process is applied. Randomly xi1 is selected. Next this value is applied to next predicate node.
- The selected test case is already tested on node3, hence the fitness value calculated for the next predicate node5. Its fitness value is 0.5. Hence it is not fit for node5, now the process moves to next path which does not have the node5 in their path but include the node3. Here the next path is path3.

  For PATH3

- In path3, there is no further predicate node. Hence calculation of fitness value is stopped. By summing up of all the fitness values of node in the path3, total fitness value is 2. The probability is calculated as

Probability = (sum_of_fitenss value of all the nodes/ TotFitness of the path) * 100
Probability = (2/2) * 100= 100

- The probability is 100%, hence the test case, path and its probability values are stored.
- This indicates vi1 is suitable for the node3, hence it will be checked for the path which is not yet covered but includes the node3 in their path. If it covers any path it is also stored.
- The above process is also done by the entire employed bee in a parallel way. The test case and the coverage information are stored simultaneously in the database, hence the bees concentrate only on the uncovered path. This will in turn reduce the time.
- Finally, the onlooker bee finds any path is not yet covered, if it is there,
- The new test cases are generated for uncovered path by using the best test case generated so far.
- Suppose the best test case is x11= -20, x12=50, x13=50
- For i=1 to 3, j=1, k=1+ (i% number of variables used here)
  - i=1, k=2
    v11 = x11+φ (x11-x21); v11= -20+(0.625) (-20-50) = -63.75
  - i=2, k=3
    v12=x12+φ (x12-x31); v12=50+(0.625) (50-50) = 50
  - i=3, k=1
    v13=x13+φ (x13-x11); v13=50+(0.625) (50-(-20)) = 93.75
- The Euclidean distance is calculated between xi1= {-20,50,50} and vi1= {-63.75,50,93.75}.
- Based on the Euclidean value the newly generated test case is tested whether it is duplicated or not by using Eq11.
- Then Mean Euclidean value is calculated to check whether the test case in neighbour or not. If it is neighbour then it is applied to all the predicate nodes in the uncovered path. Now only one bee is acting as an employed bee. Otherwise new test case is generated by using Eq. (4).
- Cycle_count is increased.
- Until Cycle_count=Max_count

RESULT ANALYSIS AND DISCUSSION

A set of application programs are tested for experimental verification of the proposed method. Three factors such as number of cycles, path coverage and time taken in terms of seconds for generating test cases had been taken for verification. Test cases are generated by the proposed qABC algorithm, GA and ABC. Then their performances are compared. Table1 shows the path coverage of newly generated test cases, number of cycles and time consumed for test case generation of six case studies by GA, ABC and qABC. The result shows that, qABC is better in performance as it has higher path coverage with a smaller number of cycles and less consumption of time compared to other intelligent algorithm. Figure 7 shows the number cycles needed for test case generation by GA, ABC and qABC. Figure 8 shows the path coverage by the newly generated test cases. Figure 9 shows the Time taken for generating the test cases. The results are provided in Table2.

As Genetic algorithm (GA) generates the test cases in a local optimum way and the fitness measurement to decide the validity of test cases is not used for further generation, it falls down at local optima.

ABC follows the random based initialization and test cases are redundantly generated. Three bees alone work in parallel. After calculating the fitness values for all the nodes in the selected path, the coverage of the path will be decided. Then it moves to find the coverage of next path. So, the time consumption for test case generation is high.

Table 2. Path coverage and time taken for GA, ABC and qABC

	GA based			ABC			qABC		
Program	No. of Generation	Path Coverage	Time Taken	No. of Cycles	Path Coverage	Time Taken	No. of Cycles	Path Coverage	Time Taken
Marks Analysis System	190	80%	65	50	85%	35	20	100%	13
Hospital Management System	180	76%	65	50	82%	33	25	97%	12
Ticket Reservation System	170	81%	59	56	91%	30	30	98%	12
Library Management System	100	65%	40	75	90%	25	40	89%	5
Payroll System	110	61%	45	45	88%	28	25	99%	7
Tax Calculation	180	80%	56	60	100%	30	30	100%	7

Figure 7. Number of cycles used by GA, ABC and qABC

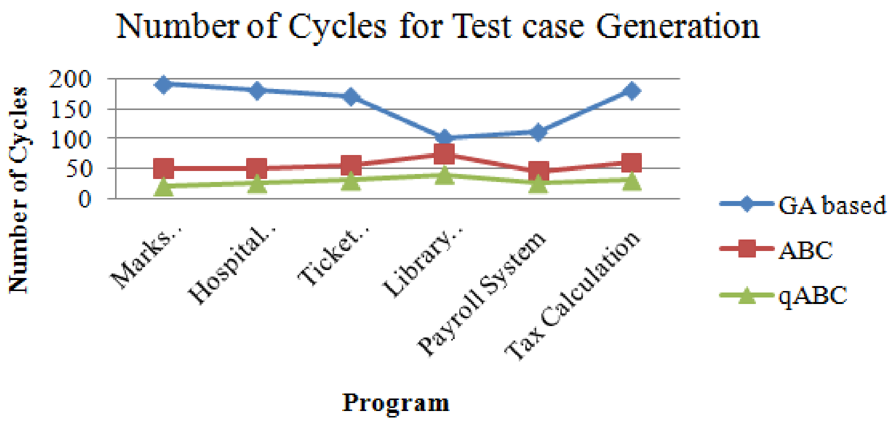


Figure 8. % of path coverage by GA, ABC and qABC

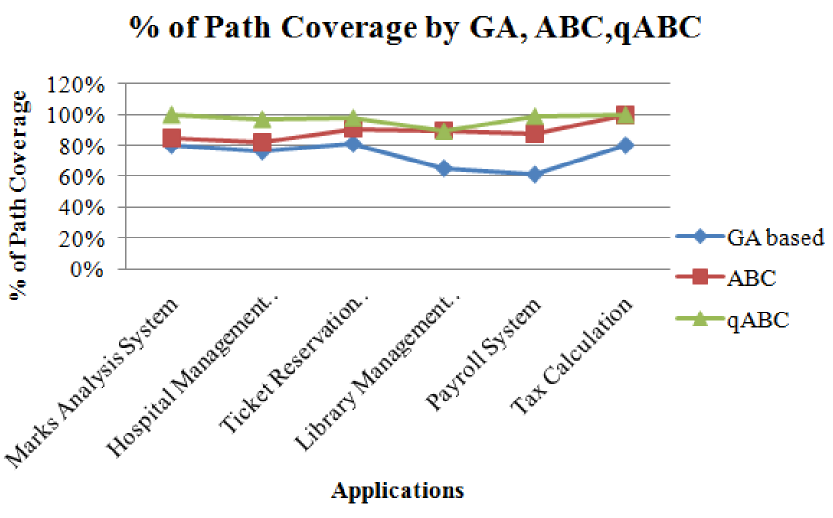
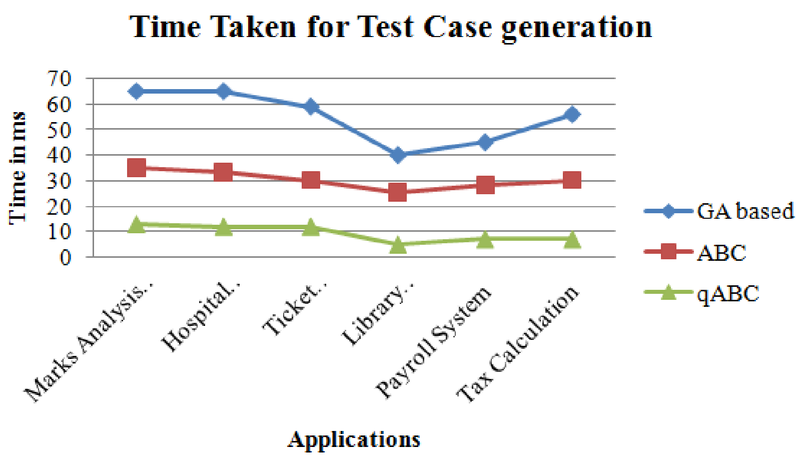


Figure 9. Time taken for test case generation by GA, ABC, qABC



The qABC finds the optimal solution with a smaller number of iterations and time because of the redundancy removal techniques. The redundant test cases are identified in the earlier stage and removed at once. In this proposed method more than one employed bee are employed and work in parallel and they update the path coverage status simultaneously. Each employed bee stops the fitness calculation process, when it encounters the node that is not covered by the test case in the selected path, and it moves to find the coverage of next uncovered path does not contain the recent uncovered node in its path. Hence it consumes less time and generated the test cases with minimum number of cycles.

Mutation Score Based Analysis

The effectiveness of the test case can also be measured by the mutation score. As per Offut's guidelines the defects are injected in the programs, and the faulty programs are called mutants. Here the mutants are generated using MuJava tool. Then the adequacy of the generated test case is checked. A test case is adequate if it is useful in detecting faults in a program. A test case can be shown to be adequate by finding at least one mutant program that generates a different output than does the original program for that test case. If the original program and all mutant programs generate the same output, then the test case is inadequate. The test adequacy is measured in terms of mutation score. The test case which has higher mutation score will be preferred. The mutation score of GA, ABC and qABC is calculated by using MuJava tool and it is shown in Table3 and Figure10. The mutation score of the qABC is high when compared with other evolutionary algorithms.

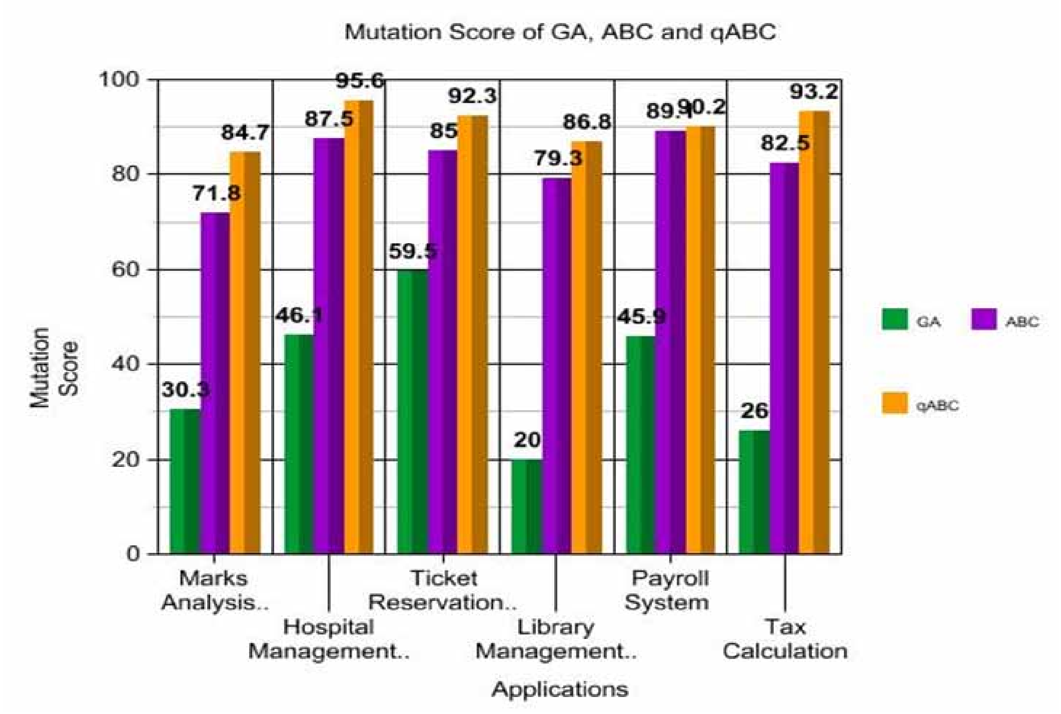
CONCLUSION AND FUTURE WORK

Successful Software Project Management includes the efficient quality assurance and testing activities done during the entire project management cycle. Although, several industries are now focusing their attention on achieving optimization in this labour intensive and time-consuming recurrent activity that

Table 3. Mutation score of GA, ABC and qABC

Program	GA	ABC	qABC
Marks Analysis System	30.30%	71.80%	84.70%
Hospital Management System	46.10%	87.50%	95.60%
Ticket Reservation System	59.50%	85%	92.30%
Library Management System	20%	79.30%	86.80%
Payroll System	45.90%	89.10%	90.20%
Tax Calculation	26.00%	82.50%	93.20%

Figure 10. Mutation score of GA, ABC and qABC



requires a lot of care to deliver the quality software product, they are still struggling on identifying the right techniques. Due to the enormous growth of AI in the software industry, several project management activities are also automated using such AI techniques.

This proposed work applied an improved version of an Artificial Bee Colony Algorithm (ABC) with each bee being represented as Intelligent Agent in their software implementation part, namely quick ABC (qABC). This algorithm has most important features such as (i) Parallel working behaviour (ii) Faster convergence (iii) improvement of quality in solution generation in each phase. Hence, it is applied in the most important, and cost and time impacted testing process to get the test cases generation in both automated and optimized manner.

By generating different test cases and applying those cases on every execution path of the class components simultaneously, the tester will get the efficient and suitable test cases associated with each path quickly. As the test adequacy criteria are including both coverage and mutation score, the test cases generated will be both efficient and successful ones as they can reveal a greater number of injected defects in the software. Various applications are tested by the test cases generated using qABC and other evolutionary algorithms namely GA and ABC. Based on the results, it has been identified that, the proposed qABC based test optimization approach yields better results in terms of fast generation of test cases, higher percentage of path coverage and mutation score of revealing the injected errors when compared to GA and ABC.As a future work, more factors will be considered to increase the mutation score, in order to improve the efficiency of the proposed qABC algorithm-based test optimization approach.

REFERENCES

Bao, X., Xiong, Z., Zhang, N., Qian, J., Wu, B., & Zhang, W. (2017). Path-oriented test cases generation based adaptive genetic algorithm. *PLoS One*, *12*(11), e0187471. doi:10.1371/journal.pone.0187471 PMID:29136028

Banharnsakun, A., Achalakul, T., & Sirinaovakul, B. (2011). The best-so-far selection in artificial bee colony algorithm. *Applied Soft Computing*, *11*(2), 2888–2901. doi:10.1016/j.asoc.2010.11.025

Bestoun S. A. (2016). Test case minimization approach using fault detection and combinatorial optimization techniques for conFigureuration-aware structural testing. *Engineering Science and Technology, an International Journal, 19*(2), 737-753

Chen, T., & Xiao, R. (2014). Enhancing artificial bee colony algorithm with self-adaptive searching strategy and artificial immune network operators for global optimization. *TheScientificWorldJournal*, *2014*, 2014. doi:10.1155/2014/438260 PMID:24772023

Mao, C., Xiao, L., Yu, X., & Chen, J. (2015). Jinfu Chen c (2015), Adapting ant colony optimization to generate test data for software structural testing. *Swarm and Evolutionary Computation*, *20*(February), 23–36. doi:10.1016/j.swevo.2014.10.003

Spieker, H., Gotlieb, A., Marijan, D., & Mossige, M. (2017). *Reinforcement Learning for Automatic Test Case Prioritization and Selection in Continuous Integration, ISSTA'17*. ACM. doi:10.1145/3092703.3092709

Shah, H., Tairan, N., Garg, H., & Ghazali, R. (2018). Global Gbest Guided-Artificial Bee Colony Algorithm for Numerical Function Optimization. *Computers*, *7*(4), 1–17. doi:10.3390/computers7040069

Karaboga, D., & Gorkemli, B.Karaboga and Beyza Gorkemlim. (2019). Solving Traveling Salesman Problem by Using Combinatorial Artificial Bee Colony Algorithms. *International Journal of Artificial Intelligence Tools*, *28*(01), 1950004. doi:10.1142/S0218213019500040

Karaboga, D., & Basturk, B. (2008). On the performance of artificial bee colony (ABC) algorithm. *Applied Soft Computing*, *8*(1), 687–697. doi:10.1016/j.asoc.2007.05.007

Wong, L.-P., & Choong, S. S. (2015). A Bee Colony Optimization algorithm with Frequent-closed-pattern-based Pruning Strategy for Traveling Salesman Problem. *Conference on Technologies and Applications of Artificial Intelligence (TAAI).* IEEE. doi:10.1109/TAAI.2015.7407122

Mala, D. J., & Mohan, V. (2009). ABC Tester-Artificial bee colony based software test suite optimization approach. *International Journal of Software Engineering*, *2*(2), 15–43.

Vanmali, M., Last, M., & Kandel, A. (2002). Using a neural network in the software testing process. *International Journal of Intelligent Systems*, *17*(1), 45–62. doi:10.1002/int.1002

Pressman, R. S. (2005). *Software engineering - A practitioner 's Approach* (International Edition). McGraw-Hill.

Nikoliж, M. (2013, September). DuљAn Teodoroviж (2013), Empirical study of the Bee Colony Optimization (BCO) algorithm. *Expert Systems with Applications: An International Journa*, *40*(11), 4609–4620. doi:10.1016/j.eswa.2013.01.063

Moradi, M., Nejatian, S., & Hamid, P. V. R. (2018). CMCABC: Clustering and Memory-Based Chaotic Artificial Bee Colony Dynamic Optimization Algorithm. *International Journal of Information Technology & Decision Making*, *17*(04), 1007–1046. doi:10.1142/S0219622018500153

Narasimman, S. (2017, August). Quantification of Software Code Coverage Using Artificial Bee Colony Optimization Based on Markov Approach. *Arabian Journal for Science and Engineering*, *42*(8), 3503–3519. doi:10.1007/s13369-017-2554-7

Lam, S. S. B., Raju, M. L. H. P., M, U. K., Ch, S., & Srivastav, P. R. (2012). Automated Generation of Independent Paths and Test Suite Optimization Using Artificial Bee Colony. *Procedia Engineering*, *30*, 191–200. doi:10.1016/j.proeng.2012.01.851

Sharma, A., Patani, R., & Aggarwal, A. (2016). Software Testing Using Genetic Algorithms. *International Journal of Computer Science & Engineering Survey.*, *7*(2), 21–33. doi:10.5121/ijcses.2016.7203

Wagner, I. A., Lindenbaum, M., & Bruckstein, A. M. (2000). ANTS: Agents, Networks, Trees, and Subgraphs. In M. Dorigo, G. Di Caro, & T. Stützle (Eds.), *Special Issue on Ant Colony Optimization, Future Generation Computer Systems* (pp. 915–926). ACM, North Holland.

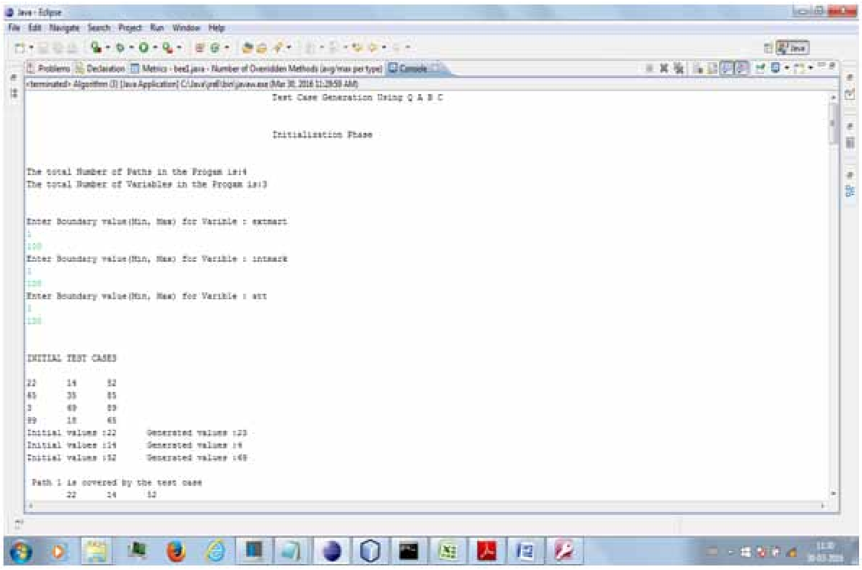
Wang, L., Yue, Z., & Hou, H. (2012). Genetic Algorithms and Its Application in Software Test Data Generation. 2012 International Conference on Computer Science and Electronics Engineering .

Aghdam, Z. K. & Arasteh, B. (2017). An Efficient Method to Generate Test Data for Software Structural Testing Using Artificial Bee Colony Optimization Algorithm. International Journal of Software Engineering and Knowledge Engineering, 27(06), 951–966

APPENDIX – A

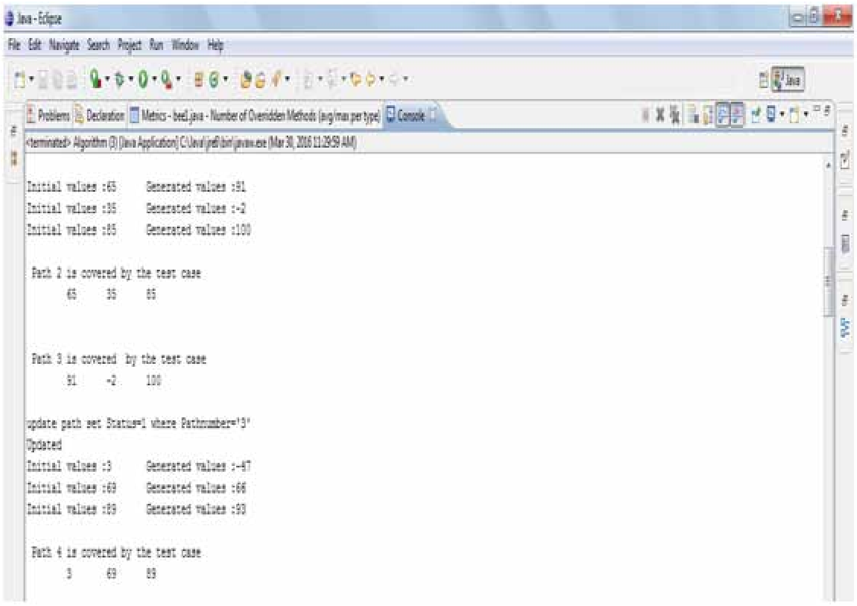
It provides the screenshots of the tool developed in-house. They are provided in Figure 11, Figure 12 and Figure 13.
(i) Test case generator using qABC
(ii) Sample: Mutation Score of the test cases of student Mark analysis system

Figure 11. Test case generation using qABC



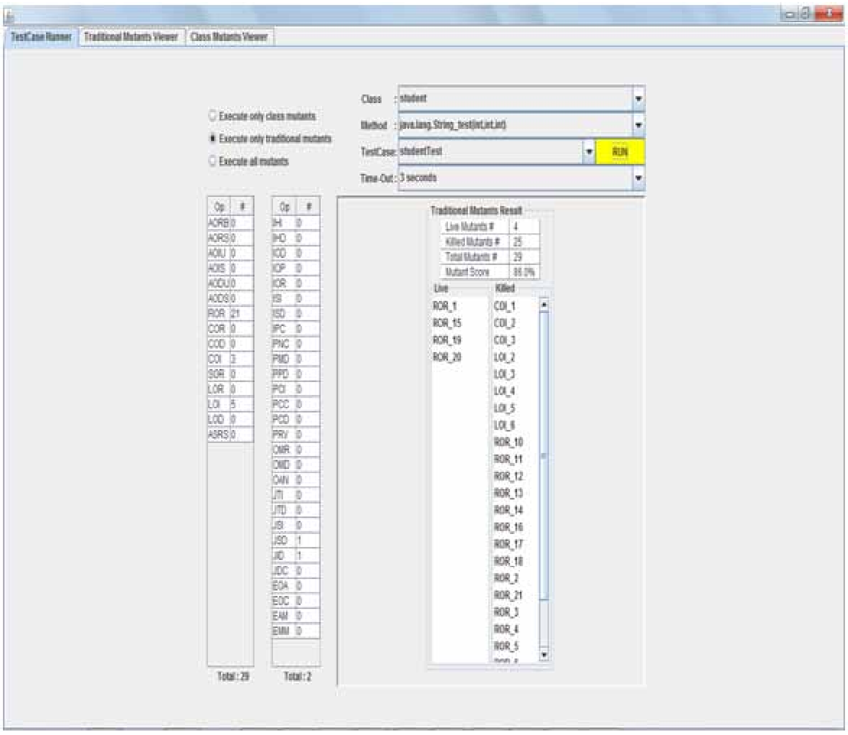
```
<terminated> Algorithm (3) [Java Application] C:\Java\jre8\bin\javaw.exe (Mar 30, 2016 11:29:59 AM)
                                        Test Case Generation Using Q A B C


                                        Initialization Phase


The total Number of Paths in the Progam is:4
The total Number of Variables in the Progam is:3


Enter Boundary value(Min, Max) for Varible : extmart
1
100
Enter Boundary value(Min, Max) for Varible : intmark
1
100
Enter Boundary value(Min, Max) for Varible : att
1
100


INITIAL TEST CASES

22      14      52
65      35      85
3       69      89
89      18      65
Initial values :22      Generated values :23
Initial values :14      Generated values :4
Initial values :52      Generated values :69

 Path 1 is covered by the test case
        22      14      52
```

Figure 12. Population generation using qABC



```
<terminated> Algorithm (3) [Java Application] C:\Java\jre8\bin\javaw.exe (Mar 30, 2016 11:29:59 AM)

Initial values :65      Generated values :91
Initial values :35      Generated values :-2
Initial values :85      Generated values :100

 Path 2 is covered by the test case
        65      35      85



 Path 3 is covered  by the test case
        91      -2      100

update path set Status=1 where Pathnumber='3'
Updated
Initial values :3       Generated values :-47
Initial values :69      Generated values :66
Initial values :89      Generated values :93

 Path 4 is covered by the test case
        3       69      89
```

Figure 13. Mutation score based analysis for effectiveness of test cases



TestCase Runner | Traditional Mutants Viewer | Class Mutants Viewer

( ) Execute only class mutants
(●) Execute only traditional mutants
( ) Execute all mutants

Class : student
Method : java.lang.String_test(int,int,int)
TestCase: studentTest
RUN
Time-Out : 3 seconds

Op	#
AORB	0
AORS	0
AOIU	0
AOIS	0
AODU	0
AODS	0
ROR	21
COR	0
COD	0
COI	3
SOR	0
LOR	0
LOI	5
LOD	0
ASRS	0

Total : 29

Op	#
IHI	0
IHD	0
IOD	0
IOP	0
IOR	0
ISI	0
ISD	0
IPC	0
PNC	0
PMD	0
PPD	0
PCI	0
PCC	0
PCD	0
PRV	0
OMR	0
OMD	0
OAN	0
JTI	0
JTD	0
JSI	0
JSD	1
JID	1
JDC	0
EOA	0
EOC	0
EAM	0
EMM	0

Total : 2

Traditional Mutants Result

Live Mutants #	4
Killed Mutants #	25
Total Mutants #	29
Mutant Score	86.2%

Live	Killed
ROR_1	COI_1
ROR_15	COI_2
ROR_19	COI_3
ROR_20	LOI_2
	LOI_3
	LOI_4
	LOI_5
	LOI_6
	ROR_10
	ROR_11
	ROR_12
	ROR_13
	ROR_14
	ROR_16
	ROR_17
	ROR_18
	ROR_2
	ROR_21
	ROR_3
	ROR_4
	ROR_5

D. Jeya Mala is currently working as Associate Professor Senior in the School of Computer Science and Engineering (SCOPE), Vellore Institute of Technology, Chennai. She has more than 20 years of teaching and research and 4 years of industrial experience. She received her doctorate degree in the area of "Artificial Intelligence in Software Engineering". She is listed in the Who's Who list of SEBASE repository of University College of London, UK for her research work in the area of Search Based Software Engineering and been invited as a member of Machine Intelligence Research Labs (MIR Labs). She is selected as an Expert Member of AICTE-NEAT evaluation Committee 2022. To her credits she has received funding support for sponsored research projects, collaborative and number of consultancy projects. She has published a patent and has published more than 60 papers in reputed, refereed; SCI and Scopus indexed journals and conferences and book chapters. She has published a MOOC course for Udemy that has more than 2200 learners' enrollment. Also, a text book for McGraw Hill publishers and two edited books for IGI Global USA, more than two course books for universities were published and some are ongoing. She is the Recipient of "Best Techno Faculty Award 2016" from ICTACT, A Govt Initiative; 'Poster Awardee for the year 2016" from Indian Science Congress Assiciaation (DST, Govt.of India), "Best Oral Presentation Award" from GCGSD 2020, "Certificate of Excellence" and "Top Performer Award" from IIT Bombay etc. She has formed the reviewer board of several international journals and conferences. She is a member of editorial boards and technical programme committees of several reputed journals and conferences. Her research interests include Artificial Intelligence, Software Engineering, Software Test Optimization, Cyber Security and Block Chain.

Ramalakshmi Prabha was a research scholar of Anna University, Chennai. She has the area of interests as Software Engineering, Software Testing, Intelligent Techniques.