PERFORMANCE ANALYSIS OF TEST DATA GENERATION FOR PATH COVERAGE BASED TESTING USING THREE META-HEURISTIC ALGORITHMS

MADHUMITA PANDA[1], PARTHA PRATIM SARANGI[2]

[1,2]Department of MCA, Seemanta Engineering College, Mayurbhanj, Odisha, 757086
Email: madhumita.panda3@gmail.com, ppsarangi@gmail.com

Abstract- This paper discusses an approach to generate test data for path coverage based testing using Genetic Algorithms, Differential Evolution and Artificial Bee Colony optimization algorithms. Control flow graph and cyclomatic complexity of the example program has been used to find out the number of feasible paths present in the program and it is compared with the actual no of paths covered by the evolved test cases using those meta-heuristic algorithms. Genetic Algorithms, Artificial Bee Colony optimization and Differential Evolution are acting here as meta-heuristic search paradisms for path coverage based test data generation. Finally the performance of the test data generation using three meta-heuristic optimization algorithms are empirically evaluated and compared.

Keywords- *Genetic Algorithms(GA), Differential Evolu-tion(DE), Artificial Bee Colony Algorithm(ABC), Path Cover-age Based Testing, Cyclomatic Complexity,Software test data generator.*

I. INTRODUCTION

Software testing is a necessary evil in the process of software development. It adds nothing to the functionalities of software but spends more than fifty percent of labor and money [1]. At the same time without thorough testing it is not possible to guarantee a product as error free. Software testing is performed for enhancing quality of the software by removing errors and fixing bugs prior to its delivery. Software testing includes three major activities, first selection of proper tests inputs, next execution of those inputs on the SUT, and finally evaluation of the correctness of the generated outputs. Out of these activities, selections of inputs and evaluation of outputs are the most error prone and time consuming tasks of the entire phase of testing [2]. Now a days gradually the complexity of the systems are increasing and these complex system needs to be error free safe secure and reliable from the users perspective. The competitive market is also forcing companies to define new approaches to reduce the development cost of systems. To increase the effectiveness and efficiency of the testing process and to reduce overall development cost of the software system new approaches are required for test automation. The automation of the process of test case generation is the most important aspect of test automation [1], [2]. very few powerful test data generation tools combining structural and functional testing are available in the market yet. Automatic model based testing tools usually accept two types of inputs, first of all a formal model of the software under test and a set of constraints guiding the tool for generating test data, test cases or test suites [1]. Automated software testing can speed up the testing process saving a lot of time and ensuring good quality of the SUT. A significant part of the telecommunications aerospace and microelectronics have been experimented for verification and test data generation since a decade [2]. Search based software test data generation is an constrained optimization problem. Now a days several approaches have been proposed by the researchers for search based test data generation, offering promising results. Test case optimization is an NP complete problem [3] and its common threads are test case selection, prioritization and minimization. The commonly used optimization techniques used by researchers include, evolutionary algorithms like genetic algorithms [17], [18] Genetic programming, Differential Evolution [11], swarm optimization techniques including Ant Colony Optimization [16], Particle swarm optimization [16], Artificial bee colony optimization [4], [5], Hybrid Genetic, Intelligent Search Agent [1], Graph based Intelligent Techniques, Hybridization of Soft Computing techniques [2] etc. Testing is a very expensive process and in this process redundant and irrelevant test cases increase the cost of testing unnecessarily. Therefore test case optimization is a must for saving manpower and cost of testing. Software test adequacy criteria are the rules to determine whether a software system has been adequately tested or not. In literature a number of test data adequacy criteria has been proposed and investigated. Some of them are control flow-based criteria, data flow based criteria, fault based, and error-based criteria. Control flow-based criteria includes statement coverage, path coverage, branch coverage, condition coverage, loop coverage [3] etc. Data-flow based criteria include all definitions criteria, all uses criteria. Fault-based adequacy criteria include error seeding and mutant killing score. Search based software testing appeared around five decades back and at that time it was mainly based on the symbolic execution and constraint solving approach

[27], [28]. In those techniques a version of SUT was executed and is guided by a cost function, now days known as fitness function. Inputs that were closer to executing a desired path through the program, were awarded with lower cost values and others were awarded with high cost values. Inputs with high cost value were normally discarded. This search based testing is gradually referred as a testing approach which mainly uses metaheuristic techniques. The simplest form of optimization algorithm is random search, which generates test data randomly until the goal of the test is fulfilled. Random search is very poor at finding solutions where the solution set occupies a very small part of the entire search space. Hill climbing and simulated annealing are search algorithms best at finding local optima, therefore these algorithms are described as local search approaches. Evolutionary algorithms (EA), use nature inspired techniques such as selection, mutation and crossover adopted from natural evolution to generate optimized solutions for problem under consideration. GA is an adaptive search procedure introduced by John Holland [18], [19] and extensively studied by Goldberg [17], De Jong and many other researchers. It uses a ”survival of the fittest” technique, where the best solutions survive and are varied until we get a good product [18]. Genetic algorithms are termed as global search algorithms as these cover the entire search space. Swarm Intelligence is the recent area introduced to the field of optimization. Recently many researchers have proposed and developed a number of meta-heuristics algorithms observing and simulating the foraging behavior of different animals and insects. These algorithms take into account both local as well as global optimal values. ABC is a swarm based optimization algorithm mimicking the behavior of bees while searching for food [5]–[7], [10].Differential Evolution (DE), proposed by Storn and Price in [11], is an efficient and simple evolutionary algorithm for real parameter optimization problems. In the last few years, the DE algorithm has been successfully applied to many science and engineering applications. Similar to other EAs, DE is a population based stochastic optimization method. In this paper we focus on the performance of binary genetic algorithm(BGA), differential evolution(DE) and Artificial Bee Colony (ABC) as test data generators. we have anayzed the efficiency of these three algorithms in path specific test data generation as well as exploration of the input domain with respect to time by taking one example program. The rest of the paper is orgnised as follows, section II includes concepts on meta-heuristic search based software test data generation, section III includes basic concepts of the three metaheuristic algorithms i.e. Genetic algorithms, Differential Evolution(DE) and Artificial Bee Coloney(ABC) Optimization Algorithms section IV includes related work performed in thesame field, section V includes our implementations section VI includes our experiments and results. Finally, we concludein Section VII with a discussion of future prespective of our proposed work.

II. META-HEURISTIC SEARCH BASED TEST DATA GENERATION

Search based software test data generation is explicitly defined as a constrained optimization problem. The meta heuristic search techniques are population based, iterative optimization algorithms.In each iteration a number of fitness function evaluation(here in this case programme run) using test data are performed. By exploiting the fitness values in for mation new test data are generated. Metaheuristics strategies obtain test inputs, employing the information available a run time.Then information is exploited to guide the search points for persuing the optimal regions defined by objective branches of the target control flow graph. Test data generation application is a two-step iterative process, first a branch o the control flow graph is traversed using input test data from population and then using fitness values target optimization problem is tackled. A control flow graph G = (V,E) is defined by a set V of vertices or nodes and a set E of edges or arcs Each vertex in V represents a program code block, two specia vertices labelled s and e refer to the begin and end entry o the control flow graph. An edge (v1, v2) ϵE, with v1 and v2 distinct from s and e, such that the control of the program flows from block v1 to v2 without crossing any other block The outdegree(V1) = 1 represents a simple vertex block o sequential statements. Every vertex v with outdegree(v) $\geq$ 1 represents a branch in a program code.

III. GENETIC ALGORITHM

Genetic Algorithms are metaheuristic search algorithms inspired from nature [17], [19]. These algorithms accept a pos sible set of solutions known as chromosomes and apply three nature inspired operators, selection, crossover and mutation fo selecting best solutions out of a possible set of solutions [18] Chromosome representation and correct fitness evaluation are the keys of the success in GA applications. The beauty o Genetic algorithms comes from their simplicity and elegance as robust search algorithms as well as from their power to discover good solutions rapidly for difficult high-dimensiona problems covering large solution spaces.

1) Steps Of Genetic Algorithm: 1) Randomly generate an initial population of potential solutions.
2) Evaluate the suitability of each solution with the help o fitness function.
3) Check whether the results obtained are satisfying stop ping criterion.
4) Select two solutions biased in favour of fitness using selection.

5) Apply crossover to the solutions at a random point on the selected strings to produce new solutions.
6) Mutate the new solutions based on a mutation probability.
7) Goto step 2.

IV. DIFFERENTIAL EVOLUTION ALGORITHM

Like other evolutionary algorithms, DE commences with a population of constant size of NP individual candidate solutions, where NP is the population size. Each member is a D-dimensional real-parameters vector representing a point in the solution space S [11]. The new candidate solutions of same population size are obtained generation by generation in the solution space. The subsequent generations are denoted by G = 0,1, , Gmax. However the vectors are changed over generation wise by a special kind of differential operator instead of classical crossover and mutation operators of genetic algorithms. DE is mainly based on a mutation operator, adding an amount to the difference of two individuals selected randomly from current population. The next important operator is the selection operator which selects the locally best solution thus avoiding the need of sorting or ranking the population [12]. DE generates new parameter vectors by adding the weighted difference between two population vectors to a third vector. These parameters are then crossed with the parameters of trial vector to yield trial vectors. Trial vector with better fitness value replaces target vector in the following generations which is known as selection. Each population vector has to serve at least once as target vector.

A. Pseudocode Of Differential Evolution

1) Initialize the generation $G = 0$ and a population PG of NP individuals
randomly in the uniformly distributed range$[X_{min}, X_{max}]$

2) While G < Gmax do

3) Randomly generate three integer numbers r1, r2, and r3 from [1, NP],
where $r1 \neq r2 \neq r3 \neq i$ For j = 1 to D do

   a) Mutation: Generate i^{th} donor vector V_i
   $v_{i,j} = x_{r1,j} + F * (x_{r2,j} - x_{r3,j})$

   b) Rearrangement: keep each parameters of donor vector in the range $[X_{min}, X_{max}]$

   c) Crossover: Generate i^{th} trial vector U_i
   If $rand(0, 1) < CR$, then $u_{i,j} = v_{i,j}$
   Else $u_{i,j} = x_{i,j}$

   d) End

4) Selection and replacement: If individual u_i fitness is better than individual x_i, then Replace individual x_i by u_i individual

5) End
6) Apply crossover to the solutions at a random point on the selected strings to produce new solutions.

7) Increment generation $G = G + 1$

8) End
9) End

V. ARTIfiCIAL BEE COLONY (ABC) ALGORITHM

The Artificial Bee Colony Algorithm is a swarm based optimization algorithm proposed for the first time by Karaboga [4]in 2005. There are three kinds of honey bees in ABC algorithm to forage food source. They are employed bees, onlookers and scouts bees. The tasks of these bees are to collect nectar around the hive. A bee waiting on the dance area for making decision to choose a food source is called an onlooker and a bee going to the food source previously visited by it is named as an employed bee. A bee carrying out random search is called a scout bee. In ABC, food searching and nectar foraging around the hive are performed by employed, onlooker and scouts bees collectively [4], [5], [8], [9]. In the ABC algorithm, the first half of the colony consists of employed artificial bees and the second half constitutes the onlookers. For every food source, there is only one employed bee. In other words, the number of employed bees is equal the number of food sources around the hive. The employed bee whose food source is exhausted by the employed and onlooker bees becomes a scout. The main steps of the algorithm are given below, in the algorithm ABC, for generating an initial solution for ith employed is used equation (2).

$$x_i^j = x_{min}^j + rand[0,1] * (x_{max}^j - x_{min}^j) \qquad (1)$$

i=1, 2,N and j=1, 2, 3D where x_i^j is a parameter to be optimized for the i^{th} employed bee on the dimension j of the D-dimensional solution space, N is the number of employed bees and x_{max}^j and $x_{min}^j$are the upper and lower bounds for x_i^j, respectively.
In both onlooker bee and employed bee phases, the food positions in the j^{th} Dimension is used in equation (3)

$$v_{i,j} = x_{i,j} + \varphi_{i,j}(x_{i,j} - x_{k,j}) j\epsilon 1, 2..D, k \neq Ii, k\epsilon 1, 2, N \quad (2)$$

where $x_{i,j}$is i^{th} employed bee,$v_{i,j}$ is the new solution for $x_{i,j}$ in the j^{th} dimension, $x_{k,j}$ is a neighbour bee of $x_{i,j}$ in employed bee population.
Hereϵ is a number randomly selected in the range of [-1, 1],D is the dimension of the problem,N is number of employed bees, and $j\epsilon 1, 2..D$ and $k\epsilon 1, 2, 3, N$ are selected randomly.
In order to generate a new food position, every onlooker bee memorizes the solution of n employed bee based on fitness values of the employed bees. The probability p_i of that onlooker bee will select the selection of the solution of the i^{th} employed bee is obtained as follows:

$$P_i = \frac{F(i)}{\sum_{j=1}^{n} F(j)} \qquad (3)$$

where F(i) is the fitness value of i^{th} employed bee and obtained as follow

$$F(i) = \begin{cases} \frac{1}{(1+obj(i))} & \text{if } (obj(i) \geq 0) \\ 1 + abs(obj(i)) & \text{if } (obj(i) < 0). \end{cases} \qquad (4)$$

where Obj(i)is the objective function specific for the problem. In addition, Equation (1) is used in order to generate new solution for the scout bee in the scout bee phase of the ABC and all the onlooker bees use Equation (2) so as to improve the solution.

A. Pseudocode Of ABC Algorithm

1) Generate the initial population x_i where $i = 1, 2, 3, .S_N$
2) Evaluate the fitness value of the population
3) Gen=1
4) While $Gen <= G_{max}$
5) For each employed bee
   Produce new solution
   $v_{ij} = x_{ij} + \varphi_{ij}(x_{ij} - x_{kj})$,
   Calculate the value f_i
   Apply greedy selection process
6) Calculate the probability value P_i for the solutions x_i.
7) For each onlooker bee
   Select a solution x_i depending on P_i
   Generate new solution v_i
   Calculate the value f_i
   Apply greedy selection process
8) If there is an abandoned solution then replace by with a new solution.
9) Gen=Gen+1
10) End

VI. RELATED WORK

Michael et.al [21] extended the work of dynamic test data generation by using function minimization method. They have used genetic algorithm to minimize the fitness function. They also examined the effect of program complexity on test data generation process. They suggested that satisfying individual test requirement is harder in large programs than in small ones. Moreover, as program complexity increases, non random test generation techniques become increasingly desirable. They also mentioned that most of the decisions are not covered when they contain a single Boolean variable, signifying a condition that can be either TRUE or FALSE. The technique they have used to define fitness function seems inadequate when the condition contains Boolean variables or enumerated types. But our approach will be able to cover conditions containg Boolean value and multiple paths as we are using the weighted approach i.e. we are assigning weights to each path covered.

P.R. Srivastava et.al [31] proposed a technique for generating test cases using 'path coverage testing criteria' and genetic algorithm. This technique was based on the criticality of the path. In this technique, the edges of the control flow graph are assigned weights. The fitness function is calculated by taking the sum of the weights of all the edges of a particular path. The path with maximum value of fitness function is the most critical. Ghiduk et.al [24] presented an automatic test-data generation technique. It uses genetic algorithm (GA) to generate test data satisfying data-flow coverage criteria

The technique applies the concepts of dominance relations between nodes to define a new multi-objective fitness function to evaluate the generated test data. Their technique is effective in achieving coverage of the test requirements, and in reducing the size of test suites, the search time, and the number of iterations required to satisfy the data-flow criteria.

It is too difficult to generate test cases for programs containing loops, arrays, and pointers using this method. Ramamoorthy [22], [28] Symbolic execution provides functional representation to program paths and assigns symbolic names for input values. It evaluates a path by interpreting the statements and predicates on the path in terms of these symbolic names, King [22]. Symbolic execution requires the systematic derivation of these expressions which can take much computational effort, Fosdick and Osterweil. Symbolic expressions are generated and show the necessary requirements to execute a certain path or branch, Clark. Many automated test data generators are based on symbolic execution, Howden [27].

The result of symbolic execution is a set of equality and inequality constraints on the input variables, which may be linear or non-linear. It defines a subset of the input space which will guide the execution of target path. If symbolic expression can be solved then the path is feasible otherwise the path is infeasible.

It is difficult to computationally manipulate algebraic expressions especially when number of paths is more. Programs containing pointers, loops, arrays reference values and calling modules are difficult to test, Korel [25]. If there are many constraints in the program which needs to be evaluated then they slow down the process of symbolic execution, [26]. Korel used a modified form of gradient descent. Initially the input values are changed slowly for getting the proper direction then the values are changed rapidly for getting good coverage.

This process continues till no change is encountered for a particular input value. Both Gradient descent and Korel method can get stucked in local minima. This problem of getting stucked in local minima has given rise to heuristic function minimization methods like simulated annealing [Kirup], tabu search [skorin] and genetic algorithm [26]. Rainer Storn et al. [11] introduced a new heuristic approach for minimizing possibly nonlinear and non-differentiable continuous space functions known as differential evolution (DE).

The Differential Evolution method (DE) was compared to Adaptive Simulated Annealing (ASA), the Annealed Nelder and Mead approach (ANM), the Breeder Genetic Algorithm (BGA), The EASY Evolution Strategy and the method of Stochastic Differential Equations (SDE). DE outperformed all the above described approaches in terms of number of function evaluations needed for reaching at global optima. A. K. Qin et al. [14] Presented a SaDE(Self adaptive Differential Evolution) algorithm, which eliminated the exhaustive search strategy and control parameters setting for the trial vector generation. In SaDE, trial vector generation strategies along with the control parameters, Scaling factor F and crossover rate (CR) will be probabilistically assigned to the target vector in the current population according to the probabilities learned from previous experience for generating improved solutions. They suggested that the success of DE in problem solving primarily depends on appropriate choice of trial vector generation strategies and setting of their associated control parameter values. They compared the performance of their proposed algorithm with the conventional DE and three adaptive DE with 26 constrained bound numerical optimized problems and proved that better quality solutions could be obtained using this algorithm. Landa Becerra et al. [12] applied DE for the first time to the problem of test data generation. They empirically evaluated the performance of a number of popular DE models with genetic algorithm. They observed that the results obtained by DE are better in most of the problems than those obtained by GA, at the same time they found that the proper setting of parameters plays a crucial role and DE is especially sensitive to the scaling factor F used in mutation. Karaboga et al. [13] simulated and compared the performance of DE algorithm with different popular versions of genetic algorithms like PGA, Grefensstette and Eshelman. They used De Jong's test functions for those simulations. They observed that the convergence speed of DE is significantly better than genetic algorithms and suggested DE to be a promising approach for engineering optimization problems. Karaboga et al. [4]In their work, they have used Artificial Bee Colony ABC Algorithm to find optimal weights for training of neural network. According to them ABC has good exploration and exploitation capabilities in searching optimal weights over traditional training algorithms as the traditional algorithms can get stuck in local minima and have high computational complexity.

Karaboga [5] applied the Artificial Bee Colony ABC Optimization Algorithm to train feed-forward neural networks which could correctly classify different variety of data sets widely used in the machine learning community. They investigated the performance of the ABC algorithm on benchmark classification problems and after comparison with the traditional and evolutionary algorithms their results indicate that ABC algorithm is very efficient in training feed-forward neural networks.

Arvinder et al. [6] have formulated and used the Bee Colony Optimization algorithm for fault coverage based regression test suite prioritization. They studied the food foraging behavior of natural bee and effectively implemented it in their algorithm using both the path exploration and exploitation phenomenon adapted by bees for test suite prioritization. They implemented it on cpp compiler but the disadvantage of the approach is it needs manual interface for providing the input data.

Bharati et al. [7]Presented a review on application of Artificial Bee Colony to the field of software testing. They compared and observed that currently Artificial bee colony algorithm has recently been applied to the field of software testing in test suite prioritization and test case optimization but much comparative study has not been made with other swarm based metaheuristic algorithms. Shivangi [8] Performed a literature review on the research activities based on ABC and BCO. She has summarized the studies on a tabular form on the basis of applications, implementations and results of different proposed algorithms. She has clearly identified the possible areas of applicability and techniques used in those papers. Mala et al [9] Presented an approach based on ABC(Artificial Bee Colony Optimization) for test suite optimization identifying the nodes with higher coverage and test cases using those high coverage nodes. They have shown that ABC is superior to GA based approach in generating global optimal solutions at the same time converges in less number of test runs. Bacanni et al. [10]Implemented an object oriented software system for ABC written in C sharp with GUI interface which could be used in optimization problems. It was found that the software is showing better performance on existing benchmark problems as it is using threads which increase the execution speed on multicore processors.

VII. IMPLEMENTATION

In our approach we have selected a weighted control flow graph as the program intermediate form. We are generating test data for path coverage based testing, as path testing provides the best code coverage leading to through testing of the entire SUT. Here the basic path set provides us the number of paths to be covered thus ensuring the number of test cases expected for complete path coverage. We have used two evolutionary algorithms DE algorithm, SGA,and one swarm intelligent based optimization algorithm, ABC algorithm for generating test data. At the same time we have also considered cyclomatic complexity number along with connection matrix for analyzing the basic paths present in our example program. A.

Metaheuristic algorithm for white box testing we have used the following steps, for generating test data performing path coverage based testing using three meta-

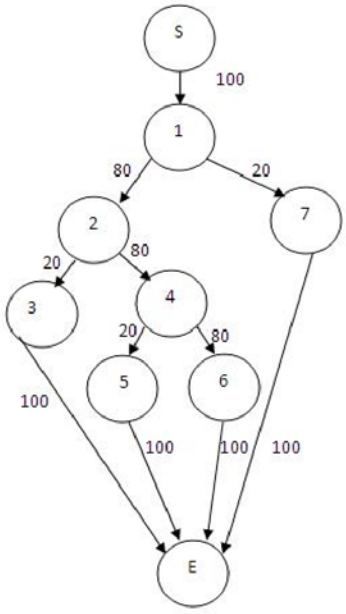

Fig. 1. Control Flow Graph for Triangle Classification Program

heuristic algorithms.
1) Accept program written in any programming language.
2) Instrument the program lines of code.
3) Generate control flow graph for the target program.
4) Prepare connection matrix from the control flow graph.
5) Find out all path sequences present in the graph.
6) Find out cyclomatic complexity of the graph.
7) Apply metaheuristic algorithms to automatically generate test data.
8) Compare traced paths with cyclomatic number.
9) Stop execution.

In our above algorithm the test data generation step using the three meta-heuristic algorithm is performed automatically, but the parameter setting and rest of the steps of the algorithm are performed manually.

B. Working of the algorithm

we have taken the triangle classification problem as our example program, as it has been widely used in the literature for test data generation. This program accepts three input edge values and determines what type of triangle could be constructed out of these edge values, it also checks wheather the input values are satisfying the basic condition of triangle or not. Here the meta-huristic algorithms are accepting the program as input and automatically generating input test data(edge values) for path coverage based testing of the above problem. In our implementation first we manually instrumented our program and prepared the control flow graph of example program. Then we identified the path numbers in our example triangle classification problem,using McCabe's cyclomatic complexity as four. Figure 1 shows the control flow graph of our example triangle classsification problem.

1) Path 1: 1-2-3(sclene)
2) Path 2: 1-2-4-5(Isosceles)
3) Path 3: 1-2-4-6(Equilateral)
4) Path 4: 1-7(Not a Triangle)

We have used a weighted control flow graph to identify the paths correctly on the basis of weights.For this we have used the 80-20 rule [31]. The main objective of the 80-20 rule is to assign more weights to those nodes of the control flow graph which hold more number of branches. In this approach initial 100 credit is asssigned to sequential edges and 80 credit is supplied to the edges of branches and loops satisfying true condition and rest 20 is supplied to the edges of branches satisfying false condition of a specific node. Applying the above rule, an initial credit is taken 100, then weight 100 is assigned to each sequential statement covered, weight 20 is assigned to every false edge encountered and weight 80 to every true edge encountered during traversal.

Then the total weight of every path from start node to end node is calculated as this weight is taken as the objective value for each path of the example program. DE algorithm is a recently developed algorithm and introduced to the field of test data generation by R.Landa [12].It's a very robust algorithm already proved itself in constrained optimization and it requires less number of function evaluations. We have used Roulette wheel selection and one-point (or single) crossover for our genetic algorithm. Here we have performed mutation on a bit- by-bit basis and the mutation occurred according to a mutation probability pm, which is an adjustable parameter. Artificial Bee Coloney is an algorithm simulating the food foraging behaviour of honey bees, where the colony contains three group of bees, employed bees, onlooker bees and scout. First the scout bee carries out random searches for discovering new food sources, in our case it is the test data covering target path. The position of food source represents possible set of solution and the necter amout of each food source represents the fitness of the associated solution. First a swarm of virtual bees is generated and a randomly distributed solution set withen the range of (1-D) dispersed over d-dimentional solution space is taken as the initial set of solutions. An employed bee performs local search to determine the position of best solution by comparing the necter amount(fitness value). Employeed bees exchange this information with onlooker bees on the dance area. After visiting a food source predefined number of steps employed bee turns to a scout bee. Then

reproduction followed by replacement and selection is performed. Here the algorithm gives best solution as both local search(performed by employed and onlooker bees) as well as global search(performed by scouts) is made by the bees.

VIII. EXPERIMENTS AND RESULTS

We have used Intel Core 2 Duo CPU with 1.8GHz with 2GB RAM. We used MATLAB version R2008a for simulating our program. We have used binary genetic algorithm(BGA), Differential evolution algorithm(DE) and Artificial Bee Colony(ABC) optimization algorithm for our program implementation.

A. Fitness function

$f(x) = \sum_{i=1}^{n} p_{(i)} \ \ldots \ (2)$
Where, $p_{(i)}$ = fittest path.

$p_{(i)} = \sum_{i=1}^{m} W_{(i)} \ \ldots \ (3)$
Where, $W_{(i)}$ = weight assigned to the fittest path.

B. Parameter settings for DE

Given below are the parameter setting for DE algorithm.

1) We have taken different variants of the scaling factor, $F(.25, .75, 1.0 and 1.6)$.
2) CR we have taken$(.5, .6, .7, .8)$.
3) we keep our population size fixed from 100 to 500.
4) Generation size from 50 to 200.

we obtained better results in the above parameter values of CR and F as proposed by [12]

C. Parameter settings for GA

Following are the parameter setting for GA.

1) Population size: Initially 100 then 500.
2) No.of generations: 50 to 500.
3) Chromosome Length pn bits. Where P in number of parameters and n is number of bits, where $n = 0, 15$.
4) Cross over probability 0.7.
5) Random probability 0.5.
6) Mutation probability 0.01.

D. Parameter settings for ABC

Following are the parameter setting for ABC.

1) The colony size is the sum of the number of employed bees and onlooker bees.
2) Number of food sources are equal to the half of colony size.
3) limit of visit to the food source by employed bee is 10.
4) population size 200,400,500.
5) No.of generations: 50 to 500.

For all the above described algorithms our input range was (-10,000 to 10,000)

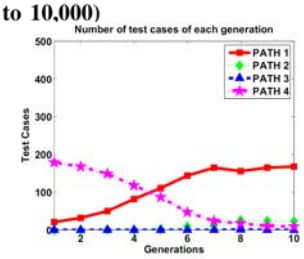

Fig. 2. Path coverage by GA

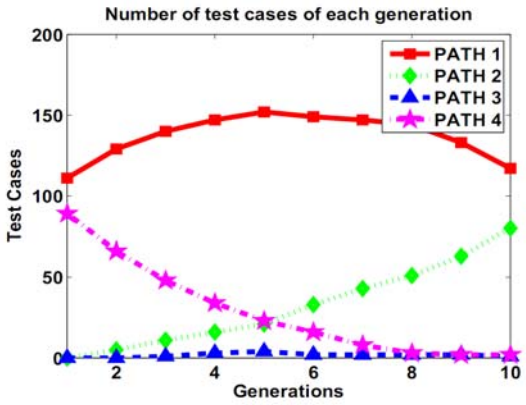

Fig. 3. Path coverage by DE

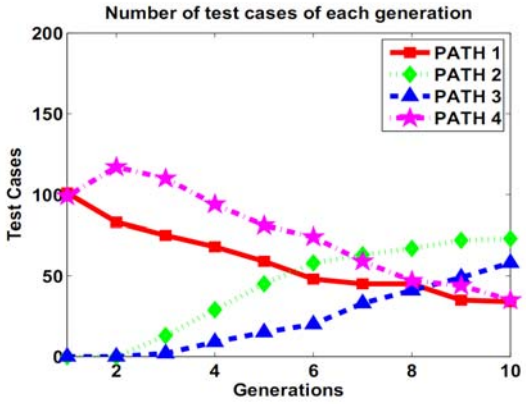

Fig. 4. Path coverage by ABC

E. Comparision of the results

when we focused on the best results obtained using the different algorithms, we found that ABC explores better in the search space for all the four paths in comparision to DE and BGA as shown in figure 4. In addition to that DE also gives better exploration for path1, path2 and path4 as shown in figure 3 and BGA's exploration is better only for path1 and path4 as shown in figure 2. In terms of time compexity the execution time for DE algorithm is less than BGA and ABC algorithms. The three algorithms were executed ten times with same number of generation(50) and population size(200) and the average exection time for DE was 0.8192 secs, for BGA 0.8448 secs and for ABC it was 1.0659 secs.The given below figures show the total number of input values generated for each path covering the input range from -10,000 to 10,000.

IX. CONCLUSION

Here in this paper we have studied three metaheuristic algorithms, Differential Evolution, Genetic Algorithm and Artificial bee coloney optimization algorithm to find a better solution for path coverage based testing, by generating test data automatically. We have compared our results and we found that DE outperforms BGA and ABC in

generating test data within less time showing better coverage and in fewer generations. We have also found the parameter setting plays a major role in DE, specially the scaling factor F and CR rate. In our future approach we are planning to implement the above concept of test data generation (TDG) using other nature inspired algorithms and Hybrid approaches of the evolutionary algorithms.

ACKNOWLEDGMENT

The authors would like to thank Seemanta Engineering College, Mayurbhanj, Odisha for providing computational facilities and financial assistance.

REFERENCES

[1] M S Geetha Devasena, M L Valarmathi, ”Multi Agent based Framework for Structural and Model based Test Case Generation”, International conference on Modeling, optimization and Computing(ICMOC), Elsevier Publications, doi,10.1016/j.procng.2012.06.440.

[2] Manoj Kumar, Arun Sharma, Rajesh Kumar,”Optimization of Test Cases using Soft Computing Techniques: A Critical Review”, wseas Transactions on Information Science and Applications, ISSN:1790-0832, Issue 11, Volume 8, November 2011.

[3] P. B. Sharma, Ruchika Malhotra and Mohit Garg, ”Empirical Validation of an Efficient Test Data Generation Algorithm Based on Adequacy based Testing Criteria”, Software Engineering : An International Journal (SEIJ), Vol. 2, No. 1, March 2012.

[4] Dervis Karaboga, Bahrije Akay, Celal Ozturk, ”Artificial Bee Colony (ABC) Optimization Algorithm for Training Feed-Forward Neural Networks”, International Conference proceedings on Modeling Decisions for Artificial Intelligence, Volume 4617, pp 318-329, Springer 2007.

[5] Karaboga, Dervis, ”Neural Networks Training by Artificial Bee Colony Algorithm on Pattern Classification”, International journal on Neural and Mass parallel Computing and Information System, Volume-19, Springer May-2009.

[6] Arvinder Kaur, Shivangi Goyal, ”A Bee Colony Optimization Algorithm for Fault Coverage Based Regression Test Suite Prioritization”, International Journal of Advanced Science and Technology Vol. 29, April, 2011.

[7] Bharti Suri, Snehlata, ”Review of Artificial Bee Colony Algorithm to Software Testing”, International Journal of Research and Reviews in Computer Science (IJRRCS) Vol. 2, No. 3, June 2011.

[8] Shivangi Goyal ”The Applications Survey: Bee Colony”, IRACST - Engineering Science and Technology, An International Journal (ESTIJ),ISSN: 2250-3498, Vol.2, No. 2, April 2012.

[9] D. Jeya Mala, V. Mohan, ”ABC Tester - Artificial Bee Colony Based Software Test Suite Optimization Approach”, International Journal of Software Engineering, IJSE Vol.2 No.2 July 2009.

[10] Nebojsa Bacanin, Milan Tuba, and Ivona Brajevic, ”Performance of object-oriented software system for improved artificial bee colony optimization”, International Journal of Mathematics and Computers in Simulation”, Issue 2, Volume 5, 2011.

[11] R.Storn, Kenneth Price, ”Differential Evolution - A Simple and Efficient Heuristic for global Optimization over Continuous Spaces”, Journal of Global optimization, Volume 11, Issue 4, pp 341-359, Dec 1997.

[12] R. Landa Becerra, R.Sagarna and X.Yao, ”An evaluation of Differential Evolution in Software Test Data Generation”, IEEE congress, Evolutinary Computation 2009.

[13] Dervis KARABOGA, Selcuk OKDEM, ”A Simple and Global Optimization Algorithm for Engineering Problems: Differential Evolution Algorithm”, Turk J ElecEngin, VOL.12, NO.1 2004.

[14] A. K. Qin, V. L. Huang, and P. N. Suganthan, ”Differential Evolution Algorithm With Strategy Adaptation for Global Numerical Optimization”, IEEE Transactions On Evolutionary Computation, Vol. 13, No. 2, April 2009.

[15] A. Joglekar, M.Tungare, ”Genetic algorithms and their use in the design of evolvable hardware”, IEEE 10th regional Conference,2001.

[16] Emad Elbeltagi,Tarek Hegazy and Donald Grierson, ”Comparison among five evolutionary-based optimization algorithms”, Conference on Evolutionary Programming VII, p.611-616, March 25-27, 1998.

[17] Goldberg, ”Genetic Algorithms in search, optimization and machine learning”, Addison-Wesley, Massachusetts,1989.

[18] Holland”Adaptation in Natural and Artificial Systems” 2nd ed. MIT Press, MIT, Cambridge.

[19] Mathew ”Genetic Algorithm” IIT,Bombay,Mumbai400076.

[20] Rajib Mall”Fundamentals of Software Engineering” Third Edition, PHI Publications, New Delhi.

[21] Christophc, Michael”Genetic Algorithm for dynamic test data generation” Waton, Technical report,Rstr-003-97-11.

[22] Harman and King ”Automated Test Data Generation using Search Based Software Engineering” ISBN: 978-0-7695-2971-2, doi 10.1109/AST.2007.

[23] Diaz, Tuya, and Blanco”Automated software testing using a metaheuristic technique based on Tabu search” page 310 - 313, doi10.1109/ASE.2003.1240327, October 2003.

[24] Ghiduk, Harrold, Girgis ”Using Genetic Algorithms to Aid Test-Data Generation for Data-Flow Coverage” doi10.1109/ASPEC.2007.73, Dec. 2007.

[25] Korel ”Automated Software Test Data Generation”IEEE Transaction on software engg. Vol. 16, August 1990.

[26] Coward ”Symbolic execution systems - a review” Software Engineering Journal, November 1988.

[27] Howden ”Reliability of the Path Analysis Testing Strategy” IEEE Transaction on software engg., Vol. SE-2, No-5, May 1976.

[28] Ramamoorthy, Ho, Chen ”On the Automated Generation of Program Test Data” IEEE Transaction on software engg., Vol. SE-2, No. 4, December 1976.

[29] P. Bhuyan and D.P.Mohapatra”Automated Test Case Generation and Its Optimization for Path Testing” U10.1109/ICIE.2009.22, August 2009.

[30] Pargas, Harrold, Peck”Test -Data generation using Genetic Algorithms” Journal of software testing verification and realiabilit, ywiley, 1999.

[31] Praveen Ranjan Srivastava and Tai-hoon Kim”Application of Genetic Algorithm in Software Testing” Journal of Software Engineering and Its Applications,Vol. 3, No.4, October 2009.