Indian Journal of Science and Technology, *Vol 9(39), DOI: 10.17485/ijst/2016/v9i39/100733, October 2016*

ISSN (Print) : 0974-6846
ISSN (Online) : 0974-5645

Artificial Bee Colony based Test Data Generation for Data-Flow Testing

Sumit Kumar[1], D. K. Yadav[2] and D. A. Khan[2]

[1]Department of Information Technology, KIET, Ghaziabad – 201206, Uttar Pradesh, India; sumitkumarbsr19@gmail.com
[2]NIT, Jamshedpur - 831014, Jharkhand, India; dkyadav@nitjsr.ac.in, dakhan.ca@nitjsr.ac.in

Abstract

Objectives: It is a challenging task to generate and identify an optimal test set that satisfies a robust adequacy criterion, like data flow testing. A number of heuristic and meta-heuristics algorithms like GA, PSO have been applied to optimize the Test Data Generation (TDG) problem. The aim of this research work is to handle the automatic Test Data Generation problem. **Methods/Statistical Analysis**: This research work focuses on the application of Artificial Bee Colony (ABC) algorithm guided by a novel Fitness Function (FF) for TDG problem. The construction of FF based on the concept of dominance relations, weighted branch distance for ABC to guide the search direction. Ten well known academic programs were taken for experimental analysis. The proposed algorithm is implemented in C environment. **Findings:** To examine the effectiveness of ABC algorithm in Test Data Generation, ten academic programs were taken experiment. The effectiveness of proposed algorithm is evaluated using average number of generations and coverage percentages achieve parameters. The experimental results show that proposed ABC algorithm requires less number of generations in comparison to other algorithms. It is also noted that the proposed algorithm coverage almost all def-path for all programs. **Application/Improvements**: The experimental results depict that the ABC algorithm performs far better than other existing algorithm for optimizing test data.

Keywords: Artificial Bee Colony, Branch Testing, Data Flow Testing, Structural Testing, Test Data Generation

1. Introduction

Software testing is most vital phase in the process of making good quality software. The aim of the software testing is for the improvement in quality and reliability of the software product. It consumes more the 50% cost of making the software. The cost of software testing is proportional to the size of input search space. Hence automatically generating and identifying an optimal test data will definitely diminish the cost of software. The generation of automatic test data is still a research field and many testing tools are available with capture and playback features to automate the execution of test scripts. However, the test cases are manually selected by the human tester and may not be optimal. Hence to reduce human effort as well cost benefit, lots of research is going in this direction. The term "Software Testing" involves functional and structural testing. Structural testing is more efficient because of its capability of finding more defects in the software[1]. Structural testing involves testing the code in such a manner that each statement, branch, path and/or structure is tested at least once. From literature, it is observed that data flow test adequacy criteria is widely popular, efficient and effective method which is based on the "definitions" and "uses" of various data items[2]. TDG can be viewed as an optimization problem in software testing. Many heuristic algorithms have been applied for improving the Test Data Generation process. Some of these are Genetic Algorithm, Tabu Search, Ant Colony Optimization, Simulated Annealing, Particle Swarm Optimization (PSO) have been used with mixed results[3]. The detailed description of these algorithms is given in related work section. It is

**Author for correspondence*

also observed that in recent years, ABC algorithm attracts researchers due to its simplicity, capability of solving large complex problems, few control parameters and ease to use. In this research, an ABC based algorithm is proposed for test data optimization. ABC algorithm proves its competency in various research fields like design digital IIR filters[4] to estimate electricity energy demand[5] image processing and data clustering[6,7], Wireless Sensor Networks[8] and many more. Hence, the intent of this study is to investigate the potent of ABC algorithm for generating the optimal test cases. The main intent of this study is to examine the application of ABC algorithm in the area of software testing especially for optimal test suite generation. In the proposed approach, ABC algorithm leaded by a novel Fitness Function is proposed to generate test data using data flow coverage criteria. The Fitness Function is based on dominance relation between the nodes of a program's control flow graph with branch distance and branch weight. Experiments on 10 well known academic programs were performed and the results are promising and imply that the suggested ABC algorithm is more suitable to generate structural test cases as compared to Random, GA and PSO. The paper structure is as follow: Section 2 describes automated software TDG process and related work. Section 3 gives information about Data Flow Testing (DFT). Section 4 describes the ABC algorithm. The proposed methodology is defined in Section 5 and discussion on results of our approach is given in Section 6. Finally, outcomes of research work are concluded in Section 7.

2. Related Work

It has always been a challenging task to generate optimal test data based on any adequacy criteria. In past few decades, a lot of research was done on test data generation. This literature survey is mainly focused on Test Data Generation approach based on meta-heuristic algorithms in structural testing. In early 90s, Genetic Algorithm is mainly used for automatically generating test data using Branch Testing[9–12]. In[13] used control flow graph information for identifying the paths to be travelled and generated test data for such paths using GA. However, data flow coverage approach has not got the same attention because of its difficulties to find test cases around data flow dependencies of a program[14]. GA is mostly preferred in Data Flow Testing techniques[15–18], but GA has chances of obtaining local optima in the search space solution and the slow convergence rate. To overcome the demerits of GA, in[19] proposed PSO. PSO has also been applied for test data generation. In[20,21] applied PSO for Test Data Generation according to branch adequacy criteria and their research shows that PSO works efficiently compared to GA. But many times PSO also fall in local optima and premature convergence. In[22] applied the ABC algorithm for path testing. A new technique for TDG using DFT is presented. In this work, an improved Fitness Function is developed by including weight of a branch while calculating the branch distance thereby taking into account the nesting level and complexity of a branch. This provides a more robust fitness value.

3. Background

3.1 Data Flow Analysis

Data Flow Testing contains data flow information, extracted from control flow graph. Detailed information about these has been found. In Data Flow Testing, all the variables are defined either as "definition" occurrence (def) or "use" occurrence. A value associated with a variable is called "definition" whereas when a value is referred to a variable known as "use" occurrence. The "usage" of a variable can be beyond sub-categorized as a "computational use (c-use) or a predicate use (p-use)". The occurrence of "c-use and p-use" depends on usage of the value of variables for selecting execution path. If a statement is predicate statement, then it is p-use otherwise it is c-use. Figure 1 shows average of three subjects program and Figure 2 shows CFG of the program. Table 1 shows all def-use paths for an average of three subjects program.

3.2 Dominator Tree

In a CFG, node "a" is said to dominate node "b" if every path from starting node (source) to "b" contains "a", where each node's children are the nodes it immediately dominates. This forms a dominator tree[23]. The corresponding dominator tree (*DomT* (G)) for an average of three subjects program shown in Figure 3.

3.3 Artificial Bee Colony Algorithm

The ABC algorithm was developed by Dervis Karaboga in 2006 for solving constrained optimization problems such as task scheduling, knapsack, Travelling Salesman Problem (TSP), classification, clustering, path planning,

```
1.  1       #include<stdio.h>
2.  1       Void main()
3.  1       {
4.  1       int m1,m2,m3,avg,i,j;
5.  1       printf("Enter the marks of three subjects : ");
6.  1       scanf("%d %d %d",&m1,&m2,&m3);
7.  2       if(m1>100||m1<0||m2>100||m2<0||m3>100||m3<0)
8.  3       {
9.  3       printf("\nInvalid Marks! Please try again.");
10. 3       }
11. 4       else
12. 4       {
13. 4       avg=(m1+m2+m3)/3;
14. 5       if(avg<40)
15. 6       {
16. 6       printf("\nResult: Fail");
17. 6       }
18. 7       else
19. 7       {
20. 7       if(avg>=40&&avg<50)
21. 8       {
22. 8       printf("\nResult: Third Division");
23. 8       }

 24. 9       else
 25. 9       {
 26. 9       if(avg>=50&&avg<60)
 27. 10      printf("\nResult: Second Division");
 28. 11      }
 29. 11      else    //avg>=60
 30. 11      {
 31. 11      printf("\nResult: First Division.");
 32. 12      if((fprintf(f2,"12
                  "))&&(m1>=75&&m2>=75&&m3>=75))
 33. 13      {
 34. 13      printf("\tDistinction in all the subjects.");
 35. 13      }
 36. 14      else if((fprintf(f2,"14
                  )&&((m1>=75&&m2>=75)||(m2>=75
     &&m3>=75)||(m1>=75&&m3>=75)))
 37. 15      {
 38. 15      printf("\tDistinction in two subjects.");
 39. 15      }
 40. 16      else if((fprintf(f2,"16
                  "))&&(m1>=75||m2>=75||m3>=75))
 41. 17      {
 42. 17      printf("\tDistinction in one subject.");
 43. 18      }
 44. 19      }
 45. 20      }
 46. 21      }
 47. 22      }
 48. }
 49. printf("\n");
 50. }
```

Figure 1. Average of three subjects program.

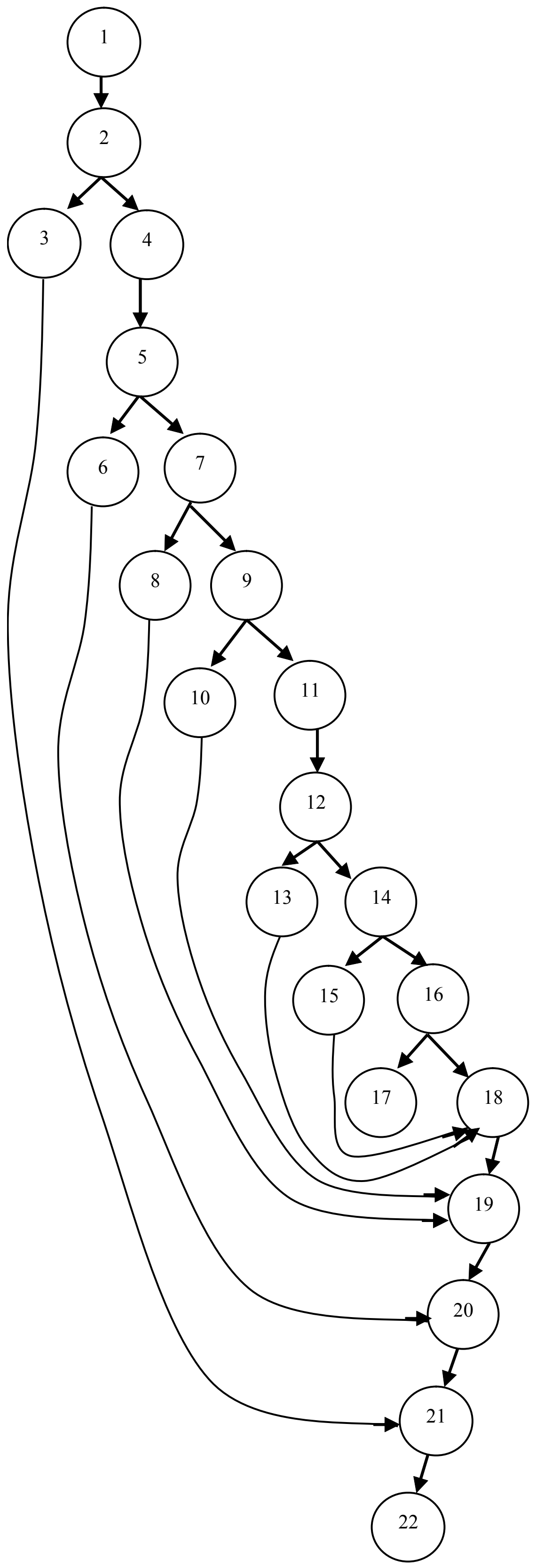

Figure 2. CFG for average of three subjects program.

Table 1. All def-use paths for average of three subjects program

Var	def node	use-node	def-use path
m1	1	2-3	1-2-3
m2		2-4	1-2-4
m3		4	1-4-1
		12-13	1-12-13
		12-14	1-12-14
		14-15	1-14-15
		14-16	1-14-16
		16-17	1-16-17
		16-18	1-16-18
Avg	4	5-6	4-5-6
		5-7	4-5-7
		7-8	4-7-8
		7-9	4-7-9
		9-10	4-9-10
		9-11	4-9-11

forecasting and many more. The main reason for this upsurge in usage of ABC algorithm is owing to the fact that it is simple to use and has very less number of user defined parameters. ABC is a meta-heuristic algorithm which consists of a population of individual vectors in a D-dimensional space it is defined in terms of food sources which represents an individual feasible solution for the problem. The position of food sources is evolved during the different phases of ABC algorithm which is divided into three main phases namely employed bees phase, onlooker bees phase and scout bee phase, where each of the different phases is signified by different types of bees namely employed bees, onlooker bees and scout bee, which operate during their respective phases of the overall algorithm. In Artificial Bee Colony algorithm, number of employed bees, onlooker bees and number of food sources, represented by the population vector is same. Each bee moves towards a particular food source and tries to improve the food source position in its adjacent neighborhood. While, there is only one scout bee in ABC.

The entire population consists of S_N food sources, that are to be evolved upon by three subroutines or three types of bees as stated earlier i.e. employee bees, scout bees and onlooker bees. These bees/subroutines work on the basis of the Fitness Function to evaluate the effectiveness of a food source which represents a prospective solution. The Fitness Function used is stated as:

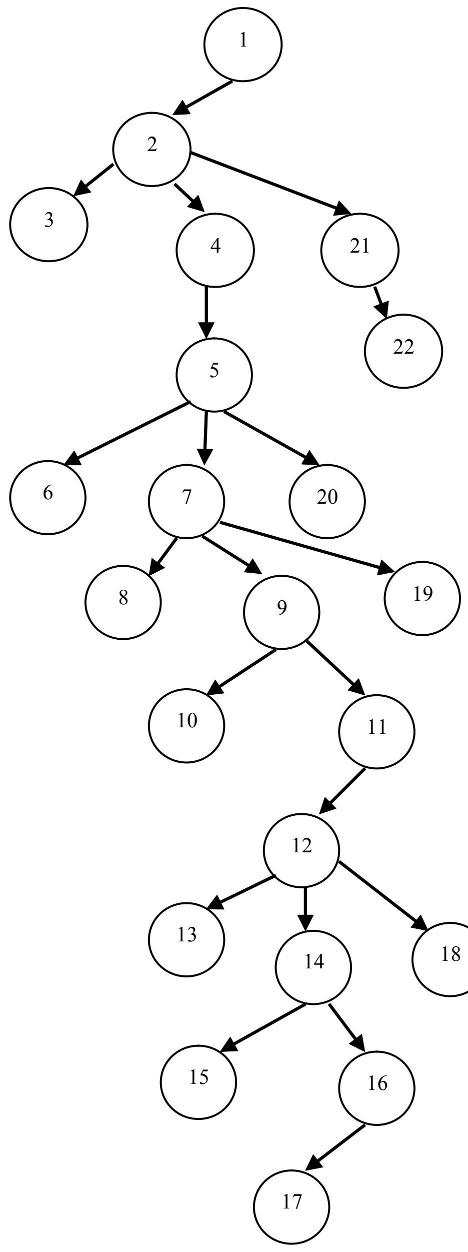

Figure 3. Dominator tree of an average of three subjects program.

$$fit_m(x) = \begin{cases} \dfrac{1}{1 + fit_m(x)} if fit_m(x) \geq 0 \\ 1 + abs(fit_m(x)) if fit_m(x) < 0 \end{cases} \quad (1)$$

Employee bees form the first subroutine and each bee relates to a particular food source selected by it, the bee then exploits the solution to find a better candidate solution, according to the Equation given below. Afterwards, exploitation of the new food source is reported with onlooker bees.

$$u_{mi} = x_{mi} + \emptyset_{mi}(x_{mi} - x_{ki}) \tag{2}$$

The functioning of onlooker bees is rather different from the functionality of employed bees. Onlooker bees exploit the solution by roulette wheel mechanism, i.e. each onlooker bee selects a candidate solution based on probability, with respect to its quality to process the solution further. Onlooker bees perform this operation of probabilistic selection on the basis of the Equation given below.

$$p_i = \frac{fit_i}{\Sigma fit_m} \tag{3}$$

Sometimes, we use a different type of bee called scout bee to explore the search space. Scout bees come into effect when we are unable to improve a candidate solution after a number of iterations which have been arbitrarily defined earlier. Three scout bees are sent iteratively to search for a new candidate solution in the search space until we are able to satisfy a termination condition, i.e. the maximum number of generations is achieved. The number of employee bees and onlooker bees equals the number of candidate solutions in a particular search space.

4. Proposed Approach

4.1 ABC based Test Data Generation

The framework of automatic TDG mainly consists of two phases i.e. test environment construction and ABC algorithm phase. The first phase of this framework is the test environment construction. In this phase we perform the following:

- Static analysis on PUT.
- Construct fitness function.
- Instrument the PUT.
- Extract the def-use paths and dom paths.

In the ABC phase initialize the initial position of food sources randomly, no. of bees, colony size, ub and lb. The proposed ABC based Test Data Generation framework accepts the CFG of the program and data flow path. This information is computed by our self-developed tool reported in[24]. It is also used to remove infeasible paths. The proposed algorithm is shown by flow chart in Figure 4.

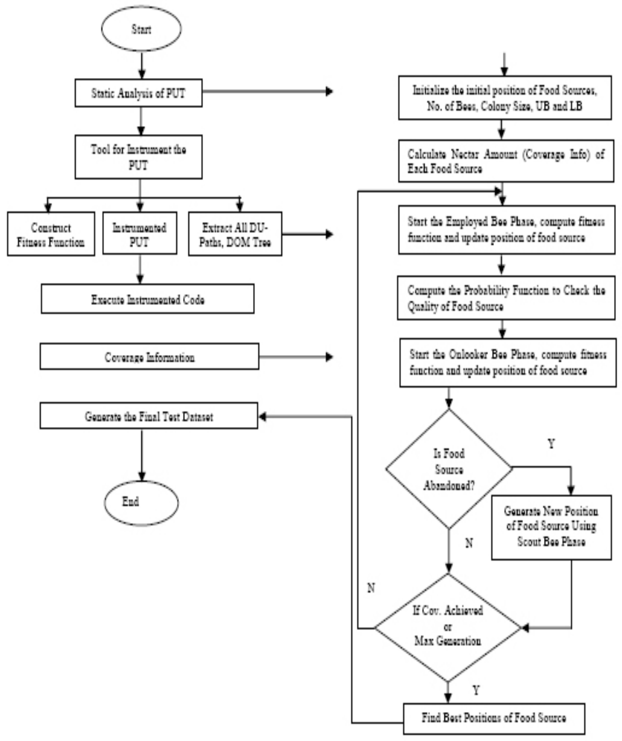
Start
Static Analysis of PUT
Tool for Instrument the PUT
Construct Fitness Function
Instrumented PUT
Extract All DU-Paths, DOM Tree
Execute Instrumented Code
Coverage Information
Generate the Final Test Dataset
End
Initialize the initial position of Food Sources, No. of Bees, Colony Size, UB and LB
Calculate Nectar Amount (Coverage Info) of Each Food Source
Start the Employed Bee Phase, compute fitness function and update position of food source
Compute the Probability Function to Check the Quality of Food Source
Start the Onlooker Bee Phase, compute fitness function and update position of food source
Is Food Source Abandoned?
Y
Generate New Position of Food Source Using Scout Bee Phase
N
N
If Cov. Achieved or Max Generation
Y
Find Best Positions of Food Source

Figure 4. Proposed ABC based Test Data Generation framework.

The steps of the proposed approach are given as:

Step 1: Assign the initial position of food sources randomly, no. of bees, colony size, ub and lb.

Step 2: Compute the nectar amount (coverage info) of every food source.

Step 3: Start the Employed Bee Phase, compute Fitness Function and update position of food source.

Step 4: Compute the probability function to check the effectiveness of food source.

Step 5: Start the Onlooker Bee Phase, compute Fitness Function and update position of food source.

Step 6: When a food source is abandoned, and then generates new position of food source using Scout Bee Phase.

Step 7: If convergence achieved or max generations reached, then stop and find best positions of food source. Otherwise repeat step 3 to 4.

4.2 Fitness Function

Fitness Function is significant in finding the optimal solution in TDG problem as fitness information is used to direct the search process. The performance of the particle in a PSO is judged on the basis of the fitness function. Every particle's fitness value is compared to every other particle's fitness value in order to calculate the optimal solution. DFT coverage criteria are picked to derive the fitness function. Def-use paths are the combination of use node and definition node. Def-use paths does not have concrete path between the nodes in control flow graph and thus represents a node-node fitness function. The Fitness Function proposed in improved by including branch weight to examine test data for DFT coverage criteria. The improved Fitness Function takes into account the difficulty of branch predicates, thereby assigning better fitness value to test data.

For a dcu-path (d, m), where d is the *def. node* and m is the *c-use node,* the fitness value f (d, m, t_i) of test case t_i (i=1...p) is given by Equation below:

$$ft(d, m, t_i) = \frac{1}{2} \times \left(\frac{|cdom(d, t_i)|}{|dom(d)|} \times wbd(d, t_i) + \frac{|cdom(m, t_i)|}{|dom(m)|} \times wbd(m, t_i) \right) \quad (4)$$

Where,

- dom (d):"dominance path of the def. node".
- dom (m): "dominance path of the c-use node".
- cdom (d, t_i) and cdom (m, t_i): "the covered nodes of dom (d) and dom (m), respectively".
- wbd_i (d, t_i) and wbd_i (m, t_i): "branch weight for the definition node and the c-use node respectively using eq.9".

Similarly, for a dpu-path (d, (m_1, m_2)), where d is the *definition node* and (m_1, m_2) is the *p-use edge,* the fitness value f (d, (m_1, m_2), t_i) of test case t_i (i = 1... p) is given by Equation below:

$$ft(d, (m_1, m_2), t_i) = \frac{1}{3} \times \left(\frac{|cdom(d, t_i)|}{|dom(d)|} \times wbd(d, t_i) + \frac{|cdom(m_1, t_i)|}{|dom(m_1)|} \times wbd(u_1, t_i) + \frac{|cdom(m_2, t_i)|}{|dom(m_2)|} \times wbd(m_2, t_i) \right) \quad (5)$$

Where,

- dom (d): "dominance path of the definition node".
- Dom (m_1) and dom (m_2): "the dominance paths of the p-use edge nodes respectively".
- Cdom (d, t_i), cdom (m_1, t_i) and cdom (m_2, t_i), "the covered nodes of dom (d), dom (m_1) and dom (m_2) respectively".
- wbd_i (d, t_i), wbd_i (m_1, t_i) and wbd_i (m_2, t_i) "branch weight for the definition node and the p-use node respectively using Equation 9".

The target is covered if the fitness value of t_i is 1.

Branch distance and branch weight are also used if only partial aim is acquired. Branch distance and branch weight are calculated using Equation 10 and 15. Weight branch distance wbd (x, t_i), where x is the intended node and t_i (i = 1...p) is an individual of the present population (test case) is evaluated as follows:

$$wbd(x, t_i) = \begin{cases} 1 & \text{if target node is covered} \\ \dfrac{1}{f(C) * wi} & \text{otherwise} \end{cases} \quad (6)$$

4.3 Branch Distance Computation

Branch distance of a node is calculated by Equation 7.

$$NodeDistance = ApproachLevel + v(branchdistance) \quad (7)$$

Where, approach level is the nearest point of the execution to the intended point. If the target node is not executed, branch distance is computed at the node, where control flow deviated on the basis of the values of the variables and constants included in predicates used at the node. If the predicate holds true branch distance is set to 0, otherwise k, where k refers to the penalty factor for deviating from its expected path to the real executed path. The value of the branch distance is normalized between range 0 and 1 using normalized function *v*. To compute branch distance for single and multiple predicates having logical and arithmetic expressions as per in[25].

4.4 Branch Weight Computation

Branch weight plays major role in calculating Fitness Function in the proposed study. In order to drive the efficient Fitness Function of a particle, it is required that a branch is assigned weight based on the reachable difficulty of the branch in execution. Branch weight is directly proportional to nesting weight and predicate weight.

It is difficult to reach the branch having deep nesting level. For branch $_i(bh_i)$ (1< = i< = z) nesting weight of the branch can be calculated as:

$$w_{nest}(bh_i) = \frac{nest_i}{nest_{max}} \quad (8)$$

Where, w_{nest} is the weight of i-th nested branch, $nest_i$ is the ith nesting level and $nest_{max}$ is the max nesting level.

Nesting weight can be normalized by using Equation 12.

$$w'_{nest}(bh_i) = \frac{w_{nest}(bh_i)}{\sum\left([w_{nest}(bh]_i) \right)} \quad (9)$$

The predicate conditions are also accountable for satisfying the difficulty level. There could be many predicate conditions in a program, which will have different difficulties in satisfying it. The predicate condition '==' will be more difficult to satisfy as compared to '>', '<' or '! = '. Table 2 depicts the reference weights for different predicate conditions.

In any program, many conditions could be present in one branch predicate. Let's consider the branch predicate of bh_i (1< = i< = z) contain *n* conditions. For each condition con_j (1< = j< = n), its referral weight $w_{ref}(con_j)$ can be calculated as per Table 4. When predicate branch bh_i is the combination of multiple conditions *n* joined with *AND* operator, its predicate weight is square root of the sum of $w^2_{ref}(con_j)$ and if bh_i is the combination of multiple conditions *n* joined with *OR* operator, its predicate weight is set to minimum of the values in the weight set $w_{ref}(con_j)$.

$$w_{pd}(bh_i) = \begin{cases} \sqrt{\sum_{j=1}^{n} w^2_{ref}(con_j)} \text{ if condition is AND} \\ \min w_{ref}(con_j) \text{ if condition is OR} \end{cases} \quad (10)$$

Predicate weight can be normalized by using Equation 11.

$$w'_{pd}(bh_i) = \frac{w_{pd}(bh_i)}{\sum_{i=1}^{s} w_{pd}(bh_i)} \quad (11)$$

For each predicate of bh_i (1< = i< = z), the associated corresponding weight w_i can be calculated as the joint sum of $w'_{nest}(bh_i)$ and $w'_{pd}(bh_i)$.

$$w_i = \beta . w'_{nest}(bh_i) + (1-\beta) . w'_{pd}(bh_i) \quad (12)$$

Where β is the adjustment coefficient and it is set to 0.5 for the experiment.

5. Result and Discussion

5.1 Experimental Setup

To validate the proposed ABC based TDG approach, experiments are performed on some widely used 10 academic programs and details of these programs are shown in Table 3. The parameters tuning of ABC, PSO and GA are shown in Table 4.

Table 2. Reference weight of predicate condition

S. No.	Condition Type	Weight
1	= =	0.9
2	<,<=,>,>=	0.6
3	Boolean	0.5
4	!=	0.2

Table 3. Benchmark programs

Prog. No.	Program	#Vars	LOC	#def-use Paths
1	Quadratic Equation	5	37	20
2	Triangle Classifier Problem	4	41	11
3	Next Date	5	107	66
4	Calendar Problem	10	121	80
5	Line in a Rectangle	8	67	52
6	Avg. Marks of 3 Subjects	4	42	15
7	Income Tax Problem	8	45	34
8	Prime Number	2	27	12
9	MidVal	4	32	19
10	Factorial of a number	2	21	8

Table 4. Algorithmic parameter settings

Algorithm	Parameters	Value
Common Parameters	Population Size	10, 15, 20, 25
	Maximum number of generations	10^3
	Number of experiments for each program	100
	Fitness Function	As given by Equation 7 and Equation 8
ABC	Colony Size	20
	No. of Bee (Onlooker, employed Bee)	5
	Limit	10
PSO	Inertia weight	Varies from 0.4 to 0.9
	c1 and c2	c1=c2=2.0
	Vmax	Varies according to the program
GA	Population selection method	Roulette Wheel
	Crossover probability	0.8
	Mutation probability	0.15

5.2 Performance Evaluation Parameter

To compare the efficiency and the accuracy of proposed ABC with PSO, GA and random search, the following performance evaluation parameters are defined:

- Average number of generations: It is the average of evolutionary generations for achieving the 100% data flow paths coverage. The stopping condition for each program is either 100% def-use coverage or 10^3 iterations.

- Average percentage coverage achieved: It is the average of data flow paths coverage of all test input in each experiments.

5.3 Results

This sub section describes the results of proposed ABC based Test Data Generation based on performance evaluation parameters as explained above. To assess the performance of proposed approach ten benchmark programs are considered. The proposed approach is applied using C language on a core i3 processor with 2 GB RAM using window operating system. For every program each algorithm runs 1000 times individually to check the effectiveness of proposed approach. The proposed ABC-based approach is compared with GA proposed by Varshney and Mehrotra, PSO and random search. The overall experimental results for the ABC, PSO, GA and Random with respect to 10 benchmark programs on population size 10 are shown in Table 5 and Table 6. The experimental results depict that proposed ABC approach takes less number of generations and achieves higher average coverage as compare to PSO-based, GA-based and random Test Data Generation approach. On the basis of experimental results we can conclude that proposed ABC based Test Data Generation approach outperforms when compared to PSO-based, GA-based and random

Table 5. Experimental results on average no. of generation

S. No.	Program	ABC	PSO	GA	Random
1	Quadratic Equation	138	157	271	659
2	Triangle Classifier Problem	184	217	341	862
3	Next Date	226	275	409	843
4	Calendar Problem	183	205	263	498
5	Line in a Rectangle	107	102	257	743
6	Avg. Marks of 3 Subjects	34	51	108	668
7	Income Tax Problem	39	45	62	75
8	Prime Number	6	9	12	15
9	MidVal	4	4	5	17
10	Factorial of a number	4	5	7	9

Table 6. Experimental results on average coverage

S. No.	Program	ABC	PSO	GA	Random
1	Quadratic Equation	99	98	95	88
2	Triangle Classifier Problem	97	97	96	84
3	Next Date	99	98	94	82
4	Calendar Problem	100	98	95	86
5	Line in a Rectangle	98	99	96	92
6	Avg. Marks of 3 Subjects	100	100	100	98
7	Income Tax Problem	100	100	99	97
8	Prime Number	100	100	100	100
9	MidVal	100	100	100	100
10	Factorial of a number	100	100	100	100

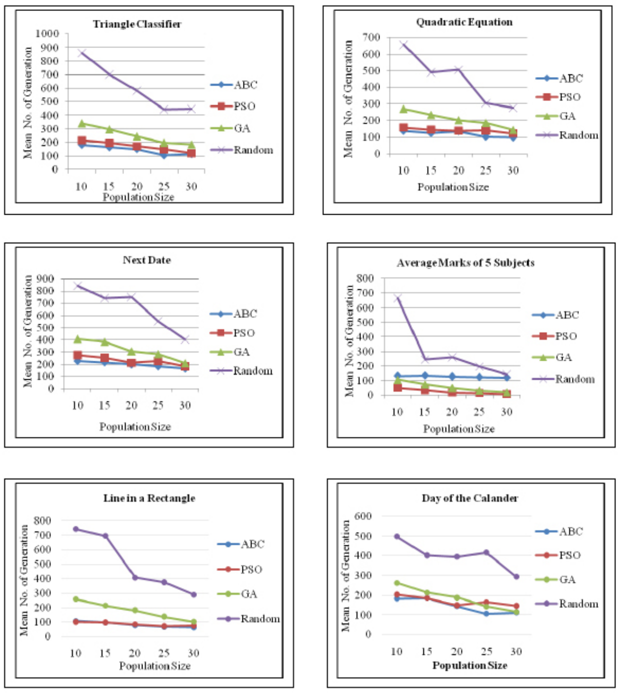

Figure 5. Graphs for average generation vs. population size.

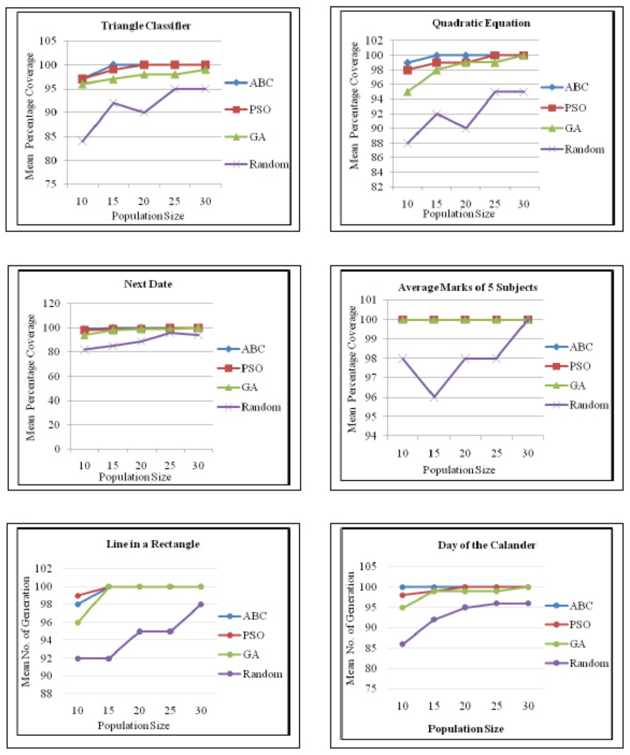

Figure 6. Graphs for average coverage on benchmarks program.

Test Data Generation approach. The graphical comparison of the proposed algorithm and other algorithm being compared using average number of generations and coverage percentage achieve parameters is shown in Figure 5 and Figure 6 respectively. From these figures, it is clearly stated that proposed algorithm take less number of generation to coverage the maximum def-path.

6. Conclusion

In this study, ABC based algorithm is applied for test data generation. It is an NP-hard problem and conventional methods are not given accurate results for such type of problems. It is also noted that these methods have no capability to change itself according to problem domain. Hence, to generate the optimize suite of test data and also to overcome limitations of conventional methods, an ABC based algorithm is adopted to deal this problem. ABC algorithm is characterized by the bee behavior and this algorithm has strong exploration and exploitation capabilities among same class of algorithms. In this work, ten benchmark problems of different dimensions are used to examine the performance of the proposed ABC algorithm. The performance of the proposed algorithm is measured on two performance matrices and also compared other well know algorithms exist in literature such as random search, GA and PSO for data-flow coverage. It is seen that proposed algorithm coverage most of def-path for all benchmark programs. It is also concluded that proposed algorithm outperforms than other algorithm being compared. In future, we intend to execute the proposed algorithm on more real and industrial programs.

7. References

1. Zhu H, Patrick AV, Hall John HR. Software unit test coverage adequacy. ACM Computing Surveys. 1997 Dec; 29(4):366–427.
2. Rapps S, Weyuker EJ. Selecting software test data using data flow information. IEEE Transactions on Software Engineering. 1985 Apr; 11(4):367–75.
3. Harman M. The current state and future of search based software engineering. Proceedings of the 29th International Conference on Software Engineering; Minneapolis, USA. 2007 May. p. 342–57.
4. Karaboga N. A new design method based on Artificial Bee Colony algorithm for digital IIR filters. Journal of the Franklin Institute. 2009 May; 346(4):328–48.
5. Varshney S, Mehrotra M. Search based software Test Data Generation for structural testing: A Perspective. ACM SIGSOFT Software Engineering Notes. 2013 Jul; 38(4):1–6.
6. Sahoo G, Kumar Y. A two-step Artificial Bee Colony algorithm for clustering. Neural Computing and Applications; 2015 Nov. p. 1–15.
7. Horng MH, Jiang TW. Multilevel threshold selection based on the Artificial Bee Colony algorithm. Artificial Intelligence and Computational Intelligence. 2010 Oct; 6320:318–25.
8. Hashim A, Ayinde BO, Abido MA. Optimal placement of relay nodes in Wireless Sensor Network using Artificial Bee Colony algorithm. Journal of Network and Computer Applications. 2016 Apr; 64:239–48.
9. Harman M, McMinn P. A theoretical and empirical study of search-based testing: Local, global and hybrid search. IEEE Transactions Software Engineering. 2010 Mar-Apr; 36(2):226–47.
10. Jones BF, Sthamer HH, Eyres DE. Automated structural testing using Genetic Algorithms. Software Engineering Journal. 1996 Sep; 11(5);299–306.
11. McMinn P. Search-based Software Test Data generation: A Survey. Journal of Software Testing, Verification and Reliability. 2004 Jun; 14(2):105–56.
12. Pargas RP, Harrold MJ, Peck R. Test-Data Generation using Genetic Algorithms. Journal of Software Testing, Verification and Reliability. 1999 Dec; 9(4):263–82.

13. Ahmed MA, Hermadi I. GA-based multiple paths test data generator. Elsevier Computers and Operations Research. 2008 Oct; 35(10):3107–24.
14. Kumar S, Yadav DK, Khan DA, Varshney S. A comparative study of automatic Test Data Generation for Data Flow Testing using GA, PSO and BPSO. International Journal of Applied Engineering Research. 2015; 10(55):2329–36.
15. Ghiduk A S, Harroldand MJ, Girgis MR. Using Genetic Algorithms to Aid Test-Data Generation for Data-Flow Coverage. Proceedings of IEEE 14th Asia-Pacific Software Engineering Conference; 2007 Dec. p. 41–8.
16. Girgis MR. Automatic Test Data Generation for Data Flow Testing using a Genetic Algorithm. Journal of Universal computer Science. 2005 Jun; 11(6):898–915.
17. Vivanti M, Mis A, Gorla A, Fraser G. Search-based Data-Flow Test Generation. IEEE International Symposium on Software Reliability Engineering (ISSRE); 2013 Nov. p. 370–9.
18. Mahajan M, Kumar S, Porwal R. Applying Genetic Algorithm to increase the efficiency of a data flow–based Test Data Generation approach. ACM SIGSOFT Software Engineering Notes. 2012 Sep; 37(5):1–5.
19. Karaboga D, Akay B. A comparative study of Artificial Bee Colony algorithm. Applied Mathematics and Computation. 2009 Aug; 214(1):108–32.
20. Windisch A, Wappler S, Wegener J. Applying Particle Swarm Optimization to software testing. Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation (GECCO“07); 2007 Jul. p. 1121–8.
21. Mao C. Generating test data for software structural testing based on Particle Swarm Optimization. Arabian Journal of Science and Engineering. 2014 Jun; 39(6):4593–607.
22. Mala DJ, Mohan V. ABC tester - Artificial Bee Colony based software test suite optimization approach. International Journal of Software Engineering. 2009 Jul; 2(2):15–43.
23. Varshney S, Mehrotra M. Search-based Test Data Generator for data-flow dependencies using dominance concepts, branch distance and elitism. Arabian Journal for Science and Engineering. 2016 Mar; 41(3):853–81.
24. Kumar S, Yadav DK, Khan DA, Srivastava A. A tool to generate all DU paths automatically. IEEE Conference on Computing for Sustainable Global Development (IndiaCom); 2015 Mar. p. 1780–5.
25. Tracey N. A search-based automated test-data generation framework for safety-critical systems. Systems Engineering for Business Process Change; 2002. p. 174–213.