Test Case Generation for Data Flow Testing using Cuckoo Search Algorithm

Sanjiv Sharma, S.A.M. Rizvi, Vineet Kumar Sharma

Abstract: Software testing consumes the major portion of the total efforts required for software development. This activity is very time consuming and labor intensive. It is very hard to do testing in optimal manner. In this paper a new approach is proposed, which uses the nature inspired stochastic algorithm called Cuckoo Search Algorithm (CSA) for the automatic generation of test data for data flow testing. This approach considers all def-use as test adequacy criteria. For assistance to CSA in the state space a new fitness function is also proposed by using the concept of dominator tree and branch distance in a CFG. To validate the proposed approach experiments are carried out on 10 benchmarked programs and findings are contrasted with earlier work done in this domain. Further in order to prove that proposed approach performs better than the above mentioned approaches a statistical difference test (T-test) is also performed.

Index Terms: Software testing, Cuckoo Search Algorithm, Data Flow testing, Dominance tree, Branch Distance.

I. INTRODUCTION

Software testing is one of the most critical steps in the life cycle of product development and consumes the major portion of the total efforts required in the development of software. This phase includes the identification of defect as many as possible. Software testing requires approximately 50% of total efforts. These efforts are time, money and man power. Software testing can be partitioned into two classes: functional and structural testing. Functional testing is also referred to as black box testing whereas structural testing is referred to as white box testing [1][2]. Structural testing has better defect exposure capability compared to functional testing [3]. Automatic generation of test cases can reduce these efforts. Nature-inspired algorithms were used for this purpose in recent years [4]. The main idea behind this approach is automatic generation of test data, from the program input domain that satisfies the testing adequacy criterion. This testing adequacy criterion is represented in the form of fitness function [5]. In automatic generation of test cases most difficult task is the selection of test adequacy criterion from various criteria available in the literature [6].

This research article presents a new Cuckoo Search algorithm based approach for the automatic generation of test data for data flow testing. For assistance to CSA in the state space a new fitness function is also proposed by using the concept of dominator tree and branch distance in a CFG. The outcomes accomplished utilizing the proposed approach are contrasted and the random search approach as well as approaches proposed by the [7] [8] [9] on the 10 benchmarked programs [8] [3] [9] [10]. In the following sections, the remaining paper is arranged. Section II provides an overview of certain related work, section III gives background knowledge of the proposed work, section IV discusses the proposed approach, section V discuss the results gain from experiments and last section VI gives conclusive remarks on the work done and the results achieved.

Revised Version Manuscript Received on 10, September 2019.

Sanjiv Sharma,Computer Science & Engineering, KIET Group of Institutions, Ghaziabad, India.

S.A.M Rizvi, Computer Science, Jamia Millia Islamia University, New Delhi, India.

Vineet Kumar Sharma,Computer Science & Engineering, KIET Group of Institutions, Ghaziabad, India.

II. RELATED WORK

Automatic test case generation for a program is very challenging task. In last decade various researches worked in this field and used various meta heuristic based search algorithm. Simulated annealing is used by [11] for pair-wise testing , by [12] for automated program flow testing, by [13] for test case generation for path testing. Genetic Algorithm have been used in [14] for test case generation for path testing, in [15] for objected oriented program using UML modeling, in [16] for regression testing. For the prioritization of test cases it is used in [17]. Particle Swarm Optimization is used in [18][19][20] [21] for data flow testing. Ant Colony Optimization is used [22][23][24]. Firefly optimization algorithm have been used in [24][25] [26]. A CSA based frame work is also proposed in [27].

III. BACKGROUND

A. Control Flow Graph

A control flow graph (CFG) is a graphical depiction of computation flow of a program during the execution of the program. CFG is a directed graph in which node represents the basic blocks of the program and edge denotes the control flow paths. There are two special nodes are also there, these are entry node and exit node. Entry node is used to enter in the CFG and exit node is used to leave the CFG. Fig. 1 shows the CFG for the triangle classification program Fig. 2.

B. Data Flow Testing

Data Flow testing is one of the types of structural testing. A structural testing requires access to internal structure of the program. It center around the definition and uses of the variables defined and used at different places in program. This testing is used to examine the behavior of variables throughout the program and ensure that there is no error causes by the variables. Rapps [28] suggested various

criterion for data flow testing and subsumption hierarchy Fig. 3. In these criteria all uses criteria is most effective. As per Jorgensen [29] there may be following types of anomalies due to improper utilization of variables in the program. First is a variable is defined but never referenced in the program. Second is a variable is referenced but never defined and last is variable is defined multiple times before it is referenced. For data flow testing a program is converted into a CFG. In the CFG following types of nodes and paths are identified.

a) Definition Node: For a variable v, a node is called definition node if variable v is is defined in the corresponding node in the CFG.

b) Use Node: For a variable v, a node is called use node if variable v is used in the statement corresponding to that node. The use node may be either computation use node (c-use node) or predicate use node (p-use node).

c) Definition use Path: For a variable v definition use path (du path) is the path between the definition node v and the use node of v.

d) Definition Clear Path: A definition clear path for a variable v is a path where variable v is not defined again on any node on that path.

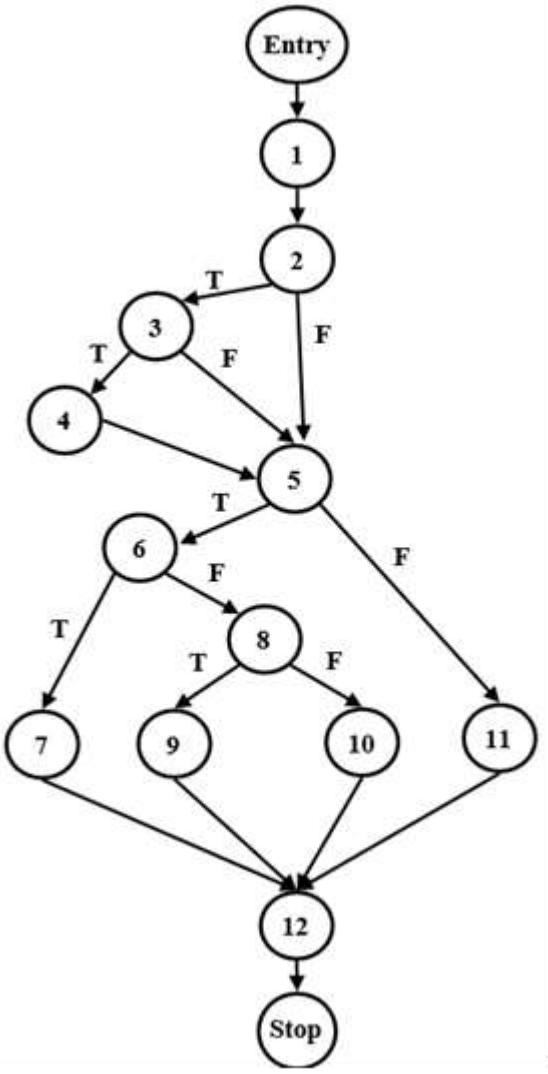

Fig. 1: CFG for Triangle Program Classifier

For data flow testing, testing adequacy criteria may be testing of all definition, testing of all uses and testing of all du paths. In testing of all definition for a variable v such paths are find out which include at least one use of variable v. In testing of all uses for a variable v , at least one path is find out for every definition of v to every use of v.

```
#include <stdio.h>
#include <conio.h>
1.      1 void main(){
2.      1 int a, b, c, valid;
3.      1 printf("\n Enter the values of three sides:");
4.      1 scanf("%d%d%d", &a, &b, &c);
5.      1 valid = 0;
6.      2 if ((a>=0)&&(a<=100)&&(b>=0)&&(b<=100)
              && (c>=0)&&(c<=100)){
7.      3       if(((a+b)>c)&&((c+a)>b)&&((b+c)>a)) {
8.      4               valid=1;
9.      5       }
10.     5 }
11.     5 if (valid==1){
12.     6       if((a==b)&&(b==c))
13.     7       printf("\n Equilateral Triangle");
14.     8       else if((a==b) || (b==c) || (c==a))
15.     9       printf("\n Isosceles Triangle");
16.     10      else
17.     10      printf("\n Scalene Triangle");
18.     10 }
19.     11 else {
20.     11      printf("\n Invalid Input");
21.     12 }
22.     12 }
```

Fig. 2: Triangle Classifier Program

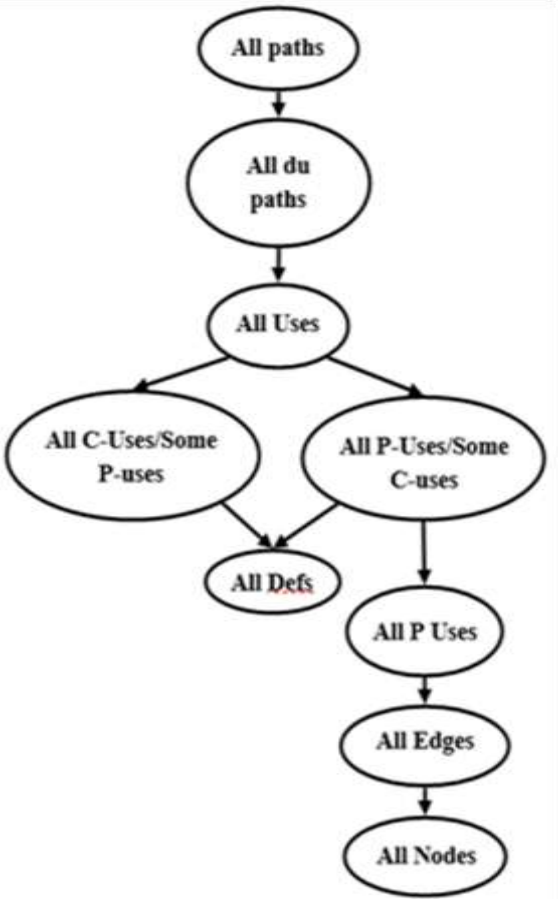

Fig. 3: Rapps Weyukar Subsumption Hierarchy [28]

The test all du path is the most important data flow testing technique. In this all du path for a variable v are tested. Table 1 shows the definition nodes and Table 2 shows the all du paths of the triangle classification program Fig. 2.

Table 1: Node Types in the Example Program

Variable	Def. Node	c-use Node
a	1	2, 3, 6, 8
b	1	2, 3, 6, 8
c	1	2, 3, 6, 8
valid	1,4	5

Table 2: Definition to use paths in Example Program

Variable	Du-path (Begin, end)	Definition clear?
A	1, 2	Yes
	1, 3	Yes
	1, 6	Yes
	1, 8	Yes
B	1, 2	Yes
	1, 3	Yes
	1, 6	Yes
	1, 8	Yes
C	1, 2	Yes
	1, 3	Yes
	1, 6	Yes
	1, 8	Yes
Valid	1, 5	Yes
	4, 5	Yes

C. Dominator Tree

In a CFG a node n1 dominates another node n2 if each path from start to n2 includes n1 node. By using above concept a dominance relationship can be established, which leads to a dominator tree of the CFG [30]. Fig. 4 Shows the example program's dominator tree.

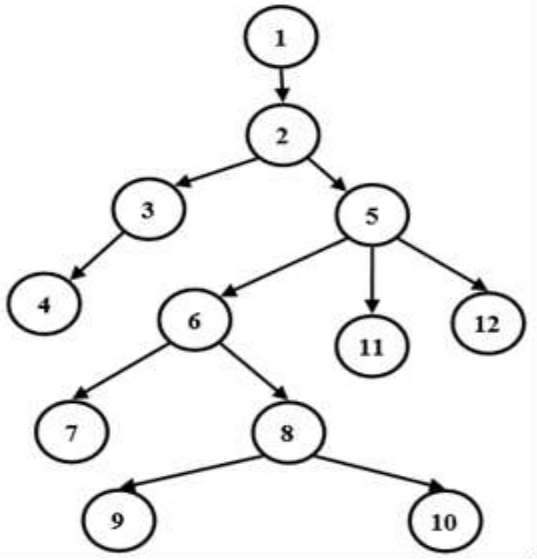

Fig. 4: Dominator Tree for Triangle Classifier Program

D. Cuckoo Search Algorithm

Cuckoo is one of those species, which used brood parasitism for reproduction. Cuckoo lay its egg in such nests in which host bird laid its egg recently and color and texture of host bird's egg's resembles with cuckoo's egg [31]. The eggs laid by the cuckoo might be distinguished by the host bird, in such case host birds either relinquish the nest and make another nest elsewhere or push out cuckoo's eggs from the nest. In a large portion of the cases, cuckoo eggs develop earlier than host bird's eggs, and once cuckoo chicks emerge from the egg, it forces other eggs out of the nest by following its instinct. This cuckoo chick activity increases the chances of survival and gives the host bird's larger share of food. This behavior of cuckoo species is simulated by the [32] in 2009 and converted into an algorithm which may be used for solution of optimization problems. In this algorithm a cuckoo choose a nest randomly to lay its egg from a pool of nests and lay only single egg at once. The nest which has highest quality of eggs will be used in future generations. There is a probability Pa [0,1] by which egg laid by the cuckoo is identified by the host bird. In this algorithm at any particular time of instance, eggs that are already in the nest represent the solutions of the problem and egg of the cuckoo laid recently represents the new solution. If the cuckoo solution is better than among available solutions in the nest, worst solution from the nest is replaced by the cuckoo solution. Fig. 5 shows the CSA algorithm and Fig. 6 shows its corresponding flow chart. From the above discussion it is clear that CSA is meta heuristic based optimization algorithm. For simplicity in CSA [32] used following three simple rules in the CSA algorithm.

1. In a randomly selected nest, each cuckoo lays only one egg at a time.
2. The nest having best quality of eggs from the set of available nests will be used in future generation.
3. The host bird can identify the cuckoo's egg by probability Pa [0, 1]. In this case, the host bird can leave the nest or may get rid of the cuckoo egg.

X.-S. Yang and S. Deb [32] suggested that CSA quality can be improved with Le´vy flight rather than random walking. Le´vy flight is more beneficial for the exploration of the state space because it has longer step length in long run [33].

Le´vy Flight

There is a foraging activity for various animals and insects in nature. It can be demonstrate using Le´vy flight. Le´vy flight is described as a random walk based on a heavily trailed probability distribution. It is an improvement over brawny movement. This behavior of Le´vy flight is very beneficial in exploration of the state space of various types of optimization problems [34][35]. Let us say a new solution is represented by X(t+1), by a cuckoo i, using Le´vy flight, using equation 1 can be written.

$$xi(t+1)= xi(t)+ \alpha \oplus Le´vy(\lambda) \quad (1)$$

Here α denotes the step size and its value must be positive and can be scaled as per requirement of the problem of

interest. In most of the cases its value is equal to 1. The symbol $\oplus$ is to represent entry wise multiplication like in PSO and a random walk is given by Le´vy flight. The random move is taken using the Le'vy distribution equation 2.

Le´vy ~ u = t- λ, (1< λ≤3) (2)

1. Set the objective function f(x), X= $(x_1, x_2, \ldots, x_d)^T$, initial population of host nests with size n, x_i (i=1,2,…,n), step size, valid range of inputs and maximum generation.
2. Initialize the population using initial randomly selected inputs.
3. Repeat from step 4 to 9 until goal is achieved or maximum generations of solutions are generated
4. Select a cuckoo randomly and generate a solution using Le´vy flight.
5. Figure out quality or fitness of cuckoo solution using the objective Function (F_i)
6. Select a nest randomly from available nest (j)
7. Compare F_i and F_j,
   a. if F_i is better than F_j then replace j by the new solution
8. Discard a fraction P_a of worse solutions and place a new one using Le´vy flight
9. Keep the best solutions, rank them and find the current best.
10. Return the best solution

Fig. 5: Algorithm of CSA

IV. PROPOSED APPROACH

The proposed approach works in two phases. In the primary stage static analysis of the program under test is done, designing of the fitness function, program is then instrumented and finally def-use paths are extracted from the program. In the second stage CSA is used to generate test cases for the program. The designed algorithm accepts the instrumented program in the form of CFG, CFG's dominant tree, def-use paths to be taken as inputs. Fig. 7 shows the proposed algorithm and Fig. 8 shows the corresponding flow chart.

A. Fitness function

In the search based optimization techniques fitness function is very important. It plays a very crucial role in and used to provide the guidance to search based technique and helps in exploration and exploitation of the problem's state space. This function depends upon nature of the problem and technique used and directly affects the performance of the technique used. The proposed fitness function is designed for data flow testing and uses criteria of all uses as the criterion for evaluating the data flow test. A du-path may not has a concrete path between definition node and use node un the CFG so it is considered as node to node function[36], so a du-path coverage is converted into two goals. The first goal is to reach the node of use and the second goal is to reach the node of use. The covered du-path must not contain any killing node. The proposed fitness function uses the concept of dominance tree, branch distance and concept of closeness level (CL).

For a du-path let u denotes the definition node and v denotes node of use, the fitness value of test case (tc) for a variable var is can be calculated using equation number 3 and 4.

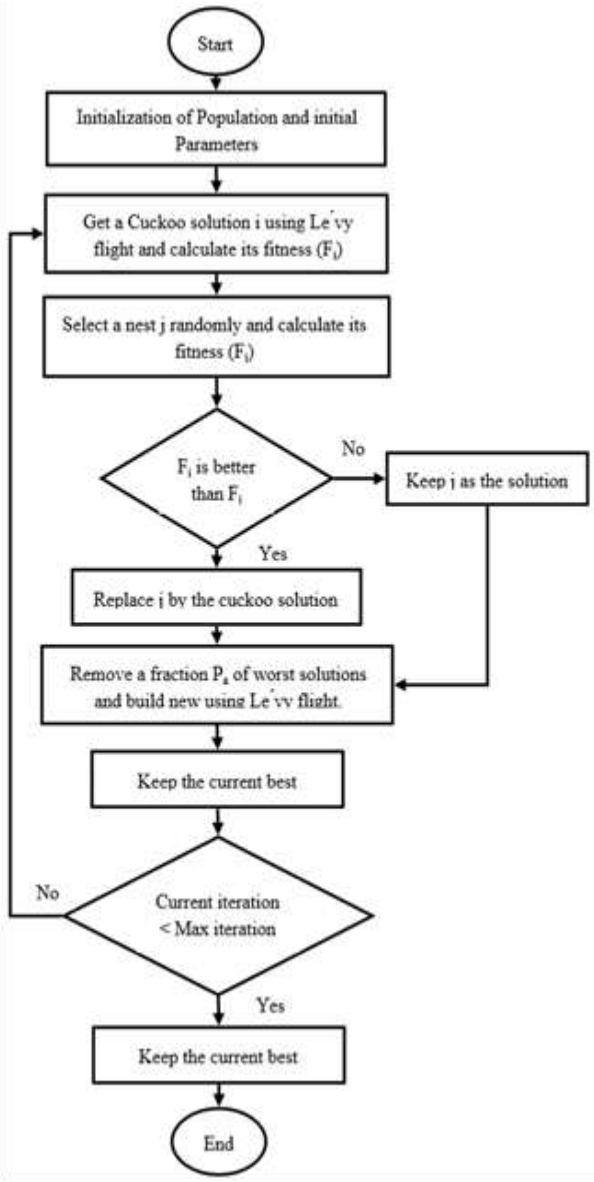

Fig. 6: Flow Chart of CSA

nodes u and v respectively by the test case tc using equation

number 5.

|fdom(u∨v)| denotes the nodes of dom(u) or dom(v) that are covered more than one time.

|udom(u∨v)| denotes not covered notes of dom(u) and dom(v).

$$bd(x, tc) = \begin{cases} 1 \text{ if execution path heads to the target node} \\ \frac{1}{f(c)} \text{Otherwise, here } f(c) \text{denoted the branch distance function from table} \end{cases} = (5)$$

B. Branch Distance

Branch distance is used to calculate the real path closeness from the planned path [37]. Branch distance is calculated on that node which has critical branch and it uses values of variable and constants involved in the predicates used at that node. Branch distance is a minimization function it gives the value zero if the target node is reached and in other cases it is calculated as shown in the Table 3 for different types of predicates value [38]. The distance of the branch is usually within the limit [0,1]. Modified branch distance bd(x,tc), where x corresponds the target node and tc denotes the test case in the current population is calculated using equation 5 [39].

V. EXPERIMENTAL SETUP AND ANALYSIS

The proposed approach is compared with the random test data generator, Genetic algorithm based approach proposed by the [7] and [8], and PSO based approach proposed by [9]. For the validation of the proposed approach widely used benchmarked programs [8], [3],[10], [9] are used. Table 4 shows the details of these programs..

A. Performance Evaluation Parameters

For the comparative analysis of efficiency and effectiveness of the CSA based approach are compared with random search technique and [7] [8] [9] following three evaluation parameters have been used and algorithm parameters setting are shown in the table 5. For comparative analysis of the above mentioned approaches, sizes of populations considered are 10, 15, 20 and 30. Table 5,6,7 and 8 represents the analysis for different population sizes on the benchmarked programs.

a) Average Number of Generation (ANG): This parameter denotes the average number of generations required to achieve 100% du-paths coverage. Although there is a cap of 103 iterations, is used for maximum number of generation for termination condition if 100% du-path is not achieved.

b) Average Success Rate (ASR): This is used to illustrate the possibility of reaching 100% du-path coverage per experiment.

c) Average Percentage of Coverage Achieved (APC): This parameter is used to denote the average of percentage of du-path covered in each experiment.

1. Select a Program under Test (PUT) and generate its corresponding CFG and find all def-use paths of it.
2. Construct fitness function for the guidance of CSA in the program state space.
3. Initialize values of population_size (Number of test cases), step_size, maximum_iteration, current_iteration, number_of_targets, number_of targets_achieved
4. Instrument the PUT
5. Randomly generate initial set test cases (initial population)
6. Execute the PUT using test cases generated in step 5
7. Remove the covered paths from the target paths and update the target path list
8. Randomly select a du-path from the target path list
9. While (current_iteration < maximum iteration) and number_of targets_achieved < number_of_targets
10. Randomly select a test case from the current population of test cases and find fitness function value (F_j) of it for selected path in the step 8. (old solution)
11. Randomly generate a new test case, run the PUT and find value of the fitness function (F_i). (new solution)
12. If $F_i > F_j$
13. Then replace old test case with new test case from current population
14. Else discard new solution
15. End if
16. Remove worst test case from the current population using fitness function values
17. Generate new test case using Le'vy flight on randomly selected test case from current population
18. Increment the value of current_iteration.
19. If selected du-path is covered
20. Then remove path from the target list, increment the targets_achieved and repeat step 8-19
21. Else repeat step 9-21
22. End while
23. If (number_of targets_achieved = number_of_targets)
24. Then show message all target achieved and show final test cases
25. Else if
26. Show the covered du-paths and corresponding test cases
27. End if

Fig. 7: Algorithm for Proposed Approach

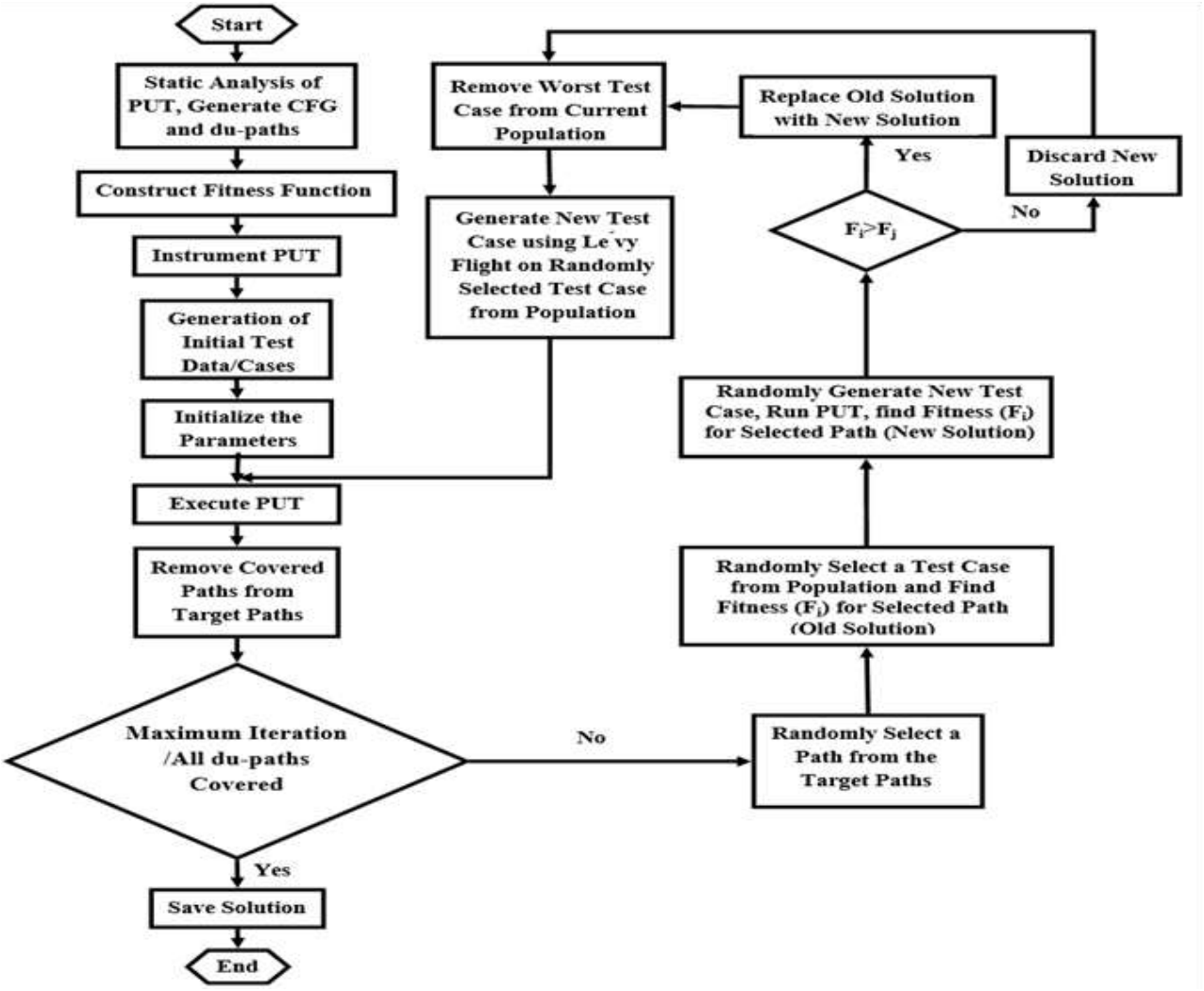

Fig. 8: Flow Chart for the Proposed Approach

Table 3: Branch Distance Functions for Predicates

S. No	Predicate (C)	Branch Distance Function $f(c)$
1	Boolean	If true then 0 else K
2	$x = y$	If $(x-y)=0$ then 0 else $abs(x-y)+K$
3	$x \neq y$	If $abs(x-y) \neq 0$ then 0 else K
4	$x > y$	If $(y-x)<0$ then 0 else $(y-x)+K$
5	$x \geq y$	If $(y-x) \leq 0$ then 0 else $(y-x)+K$
6	$x < y$	If $(x-y)<0$ then 0 else $(x-y)+K$
7	$x \leq y$	If $(x-y) \leq 0$ then 0 else $(x-y)+K$
8	$C_1 \&\& C_2$	$f(C1)+f(c2)$
9	$C_1 \| C_2$	$Min(f(c1), f(c2))$

K is a constant failure that is applied to the distance of the branch if predicate is incorrect.

Table 4: Benchmarked Programs used in Experimental Study

S. No.	Program name	No. of Variable	LOC	No. of du-paths
1.	Triangle Classifier	4	22	6
2.	Quadratic Equation	8	32	15
3	Day of the calendar	10	115	80
4	Income Tax Calculator	8	42	34
5	Prime Number	2	23	12
6	Average Marks of three subjects	4	42	15
7	Mid Value	4	30	19
8	Next Date	5	104	66
9	Line in a Rectangle	8	62	52
10	Factorial of a Number	2	21	8

B. Results&Discussion

Experimental results of the proposed work and the approaches used by the [7], [8], [9] are shown in the tables 5-8 and figures 9-12 and following study questions are answered here.

RQ1: What is the efficacy of the proposed approach in achieving 100% data flow coverage and generation of test cases?

From the experimental results of all the approaches it can be concluded that CSA performs better than the other approaches by using the new branch distance based fitness function. Although for small size population (size 10) CSA is not achieving 100% data flow coverage, but it gives 100% coverage in rest three population sizes. In other approaches random search's performance is worst.

RQ2: How beneficial is the recommended fitness in CSA?

The CSA-based approach reached 100% data coverage in a minimum of generation compared with other approaches on benchmarked programs. For the calculation of average number of generations 100 experiments are done for each benchmarked program.

RQ3: What is the effectiveness of the proposed approach in generation of optimized test suite?

For the approach based on CSA the average number of generations required for the optimal test suite are minimum. For the nested conditions CSA performs much better than other approaches.

Table 5: Experimental Results for benchmarked programs

	Population Size =10														
	Performance Metric														
	Average No. of Generation					**Average Success Rate in %**					**Av. % of Coverage Achieved**				
Program No.	**CSA**	**Random**	**GA[7]**	**GA [8]**	**PSO [9]**	**CSA**	**Random**	**GA[7]**	**GA [8]**	**PSO [9]**	**CSA**	**Random**	**GA[7]**	**GA [8]**	**PSO [9]**
1	232	635	268	297	316	100	88	89	89	90	100	93	95	94	96
2	261	828	456	385	293	100	86	87	88	89	100	92	93	95	95
3	197	308	301	275	237	100	87	88	89	91	100	91	94	95	96
4	98	197	105	87	104	94	85	86	88	92	99	89	92	96	97
5	16	55	37	19	21	100	88	88	87	90	100	93	94	91	95
6	32	354	260	94	47	96	87	89	89	91	98	92	93	96	96
7	13	33	51	26	19	94	85	86	90	92	98	94	95	96	97
8	264	984	463	371	317	95	89	90	89	91	99	95	94	96	96
9	183	756	458	398	241	100	88	89	90	90	100	94	94	96	96
10	13	27	15	17	22	100	86	88	90	91	100	93	94	95	96

Table 6: Experimental Results for benchmarked programs

	Population Size =15														
	Performance Metric														
	Average No. of Generation					**Average Success Rate in %**					**Av. % of Coverage Achieved**				
Program No.	**CSA**	**Random**	**GA[7]**	**GA [8]**	**PSO [9]**	**CSA**	**Random**	**GA[7]**	**GA [8]**	**PSO [9]**	**CSA**	**Random**	**GA[7]**	**GA [8]**	**PSO [9]**
1	181	513	171	174	184	100	89	91	92	94	100	95	94	95	96
2	165	714	321	269	228	100	89	90	94	96	100	94	94	96	97
3	161	267	216	144	189	100	90	91	93	95	100	92	94	96	98
4	67	136	67	61	74	95	88	92	94	93	100	90	95	96	97
5	12	39	28	12	11	100	90	92	95	97	100	94	96	98	99
6	24	303	169	75	33	100	90	91	94	98	100	94	95	99	98
7	9	27	39	18	10	98	87	90	92	95	100	95	94	96	99
8	188	745	287	265	261	100	91	92	94	96	100	93	96	98	98
9	131	639	263	290	128	100	90	91	93	94	100	96	95	96	98
10	8	16	11	10	14	100	89	90	92	96	100	95	96	97	99

Table 7: Experimental Results for benchmarked programs

Program No.	Population Size =20														
	Performance Metric														
	Average No. of Generation					Average Success Rate in %					Av. % of Coverage Achieved				
	CSA	Random	GA[7]	GA [8]	PSO [9]	CSA	Random	GA[7]	GA [8]	PSO [9]	CSA	Random	GA[7]	GA [8]	PSO [9]
1	97	453	172	148	117	100	92	93	96	95	100	96	95	97	98
2	87	548	146	143	128	100	91	93	94	96	100	96	97	97	99
3	98	198	167	121	109	100	93	93	94	95	100	95	96	98	98
4	38	63	58	42	43	100	90	92	94	96	100	92	95	97	99
5	7	18	11	9	7	100	93	94	95	100	100	96	97	99	100
6	19	296	106	67	28	100	92	94	95	95	100	94	98	98	98
7	4	23	20	13	5	100	90	92	93	94	100	96	96	97	97
8	147	689	241	230	221	100	94	95	95	94	100	95	99	98	97
9	69	523	278	212	87	100	94	94	94	95	100	95	97	96	98
10	3	10	8	6	4	100	95	96	100	96	100	96	100	98	99

Table 8: Experimental Results for benchmarked programs

Program No.	Population Size =30														
	Performance Metric														
	Average No. of Generation					Average Success Rate in %					Av. % of Coverage Achieved				
	CSA	Random	GA[7]	GA [8]	PSO [9]	CSA	Random	GA[7]	GA [8]	PSO [9]	CSA	Random	GA[7]	GA [8]	PSO [9]
1	54	311	69	74	67	100	95	96	97	98	100	98	97	98	99
2	36	237	127	93	68	100	94	95	96	100	100	97	98	99	100
3	59	103	96	84	74	100	95	97	98	100	100	96	99	99	100
4	27	29	26	25	26	100	94	97	98	100	100	97	98	99	100
5	4	8	7	6	7	100	95	96	100	98	100	99	97	100	99
6	11	147	43	36	28	100	95	97	98	100	100	97	98	99	100
7	2	13	9	7	5	100	94	97	100	100	100	98	99	100	100
8	86	364	123	121	98	100	97	97	99	100	100	97	99	99	100
9	35	269	131	120	69	100	94	96	100	100	100	96	98	100	100
10	2	6	5	3	100	100	99	100	100	100	100	99	100	100	100

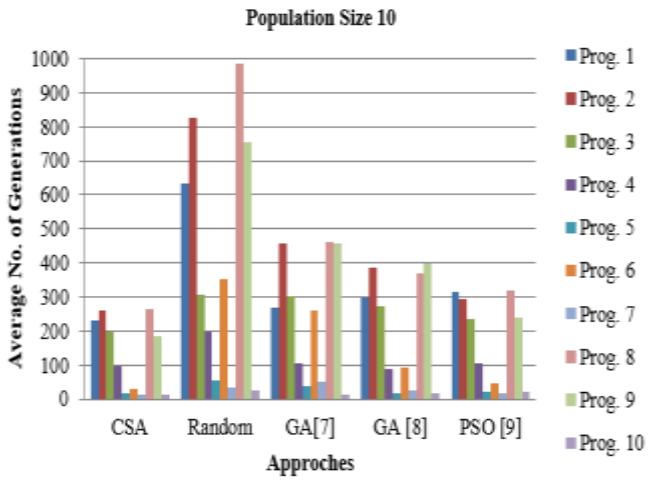

Fig. 9.1

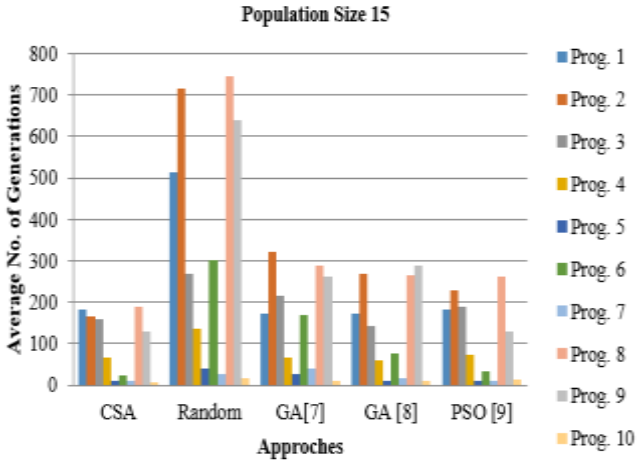

Fig. 9.2

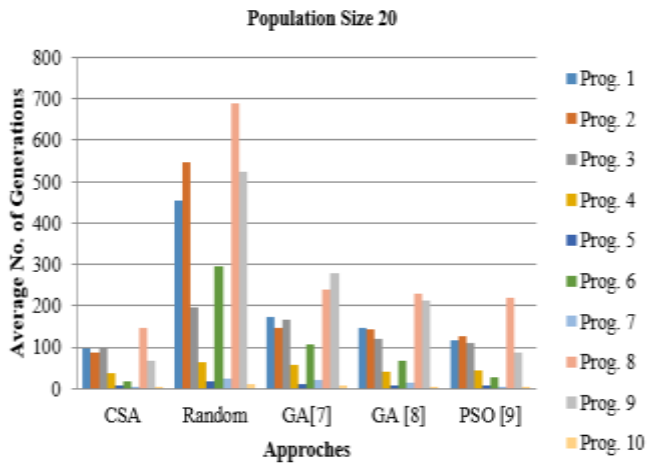

Fig. 9.3

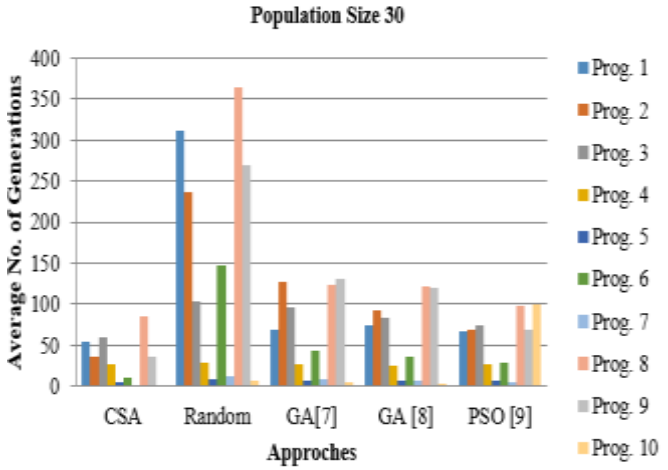

Fig. 9.4

Fig. 9 : Average Number of Generations with respect to population Size

Fig. 10.1

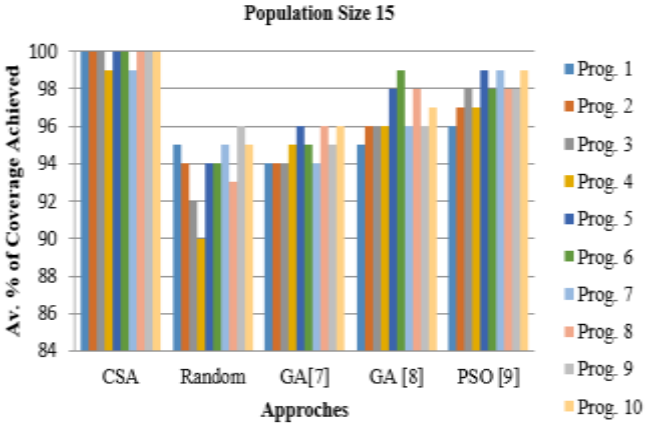

Fig. 10.2

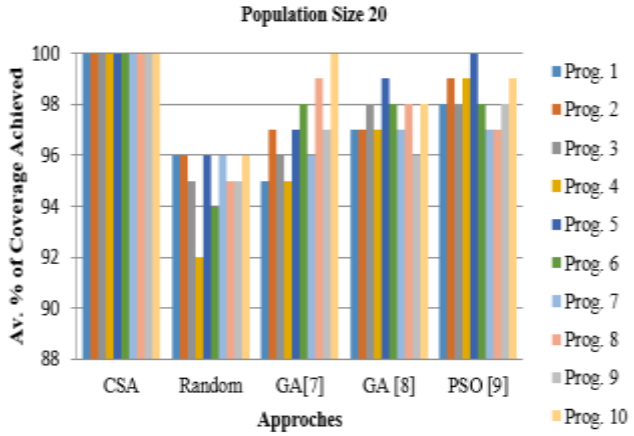

Fig 10.3

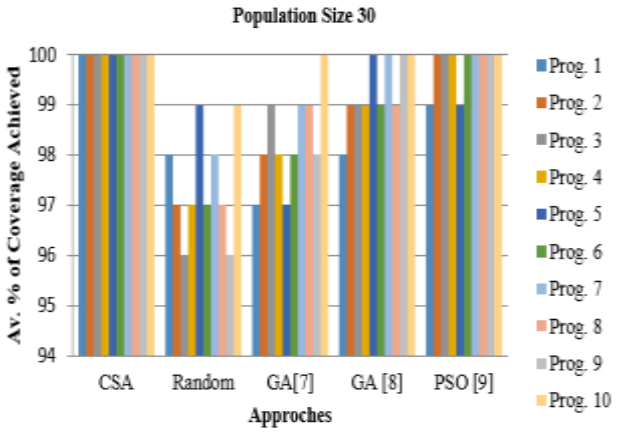

Fig. 10.4

Fig. 10: Average of % Coverage Achieved with respect to population Size

C. Statistical Analysis

To demonstrate the improved quality of the proposed solution over other approaches used in the [7], [8], [9], a statistical difference test called T-test has been used. In this analysis average number of generation, average percentage of coverage and average success rates are considered after performing 100 repeated trials on each benchmarked programs. For the T-test following three null hypotheses are framed and T-test results are shown in the tables 9-11.

H1: The approach based on the CSA is not much different than the random search and the methods used in the [7], [8], [9] with respect to ANG.

H2: The approach based on the CSA is not much different than the random search and the methods used in the [7], [8], [9] with respect to APC.

H3: The approach based on the CSA is not much different than the random search and the methods used in the [7], [8], [9] with respect to ASR.

For the null hypotheses H1, p value is smaller than 0.05 for all programs except than program 4 in the CSA Vs Random Search, so we reject the hypotheses and claims that performance of CSA based approach is better than the random search technique with respect to ANG. For CSA Vs [7] value of p is smaller than 0.05 for eight programs, in this case we can also rejects the hypotheses and claims that performance of CSA based approach is better than the [7] with respect to ANG. For CSA Vs [8] and CSA Vs [9] value of p is smaller than 0.05 for seven programs, in this case we can also rejects the hypotheses and claims that performance of CSA based approach is better than the [8] and [9] with respect to ANG.

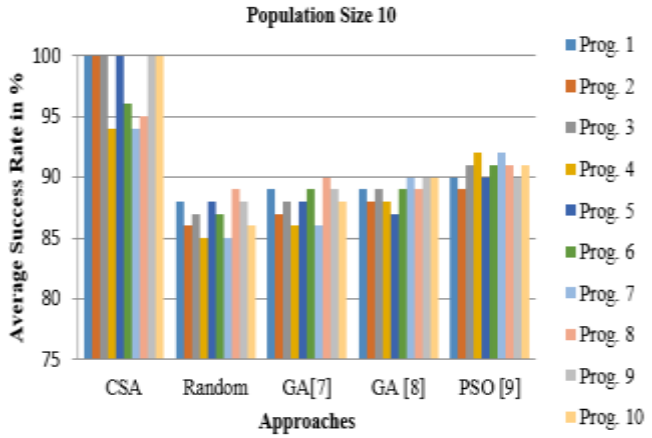

Fig. 11.1

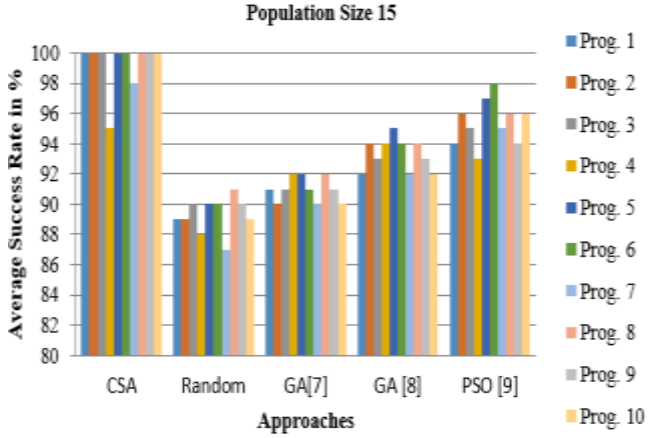

Fig. 11.2

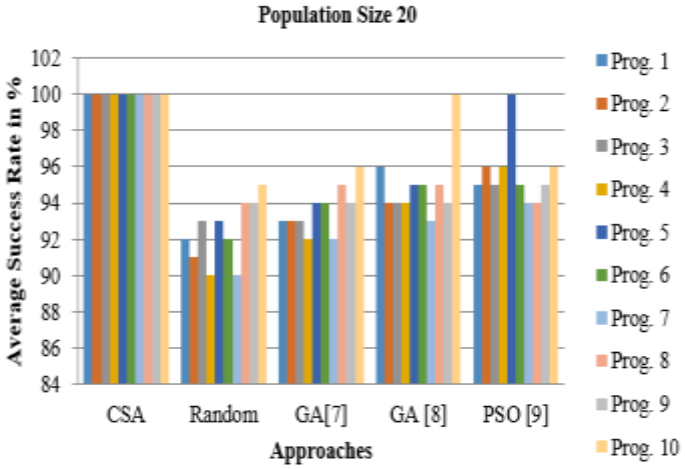

Fig. 11.3

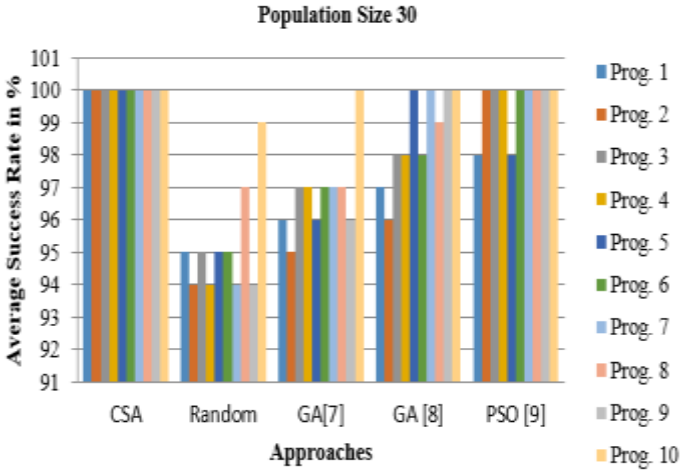

Fig. 11.4

Fig. 11: Average Success Rate with respect to Population Size

Program No.	p value CSA vs.Random Search	p value of CSA vs. [7]	p value CSA vs. [8]	p value CSA vs. [9]
Prog. 1	0.001635668	0.10264	0.069545	0.10015
Prog. 2	0.006671875	0.01339	0.007752	0.00524
Prog. 3	0.005117369	0.00926	0.12822	0.01882
Prog. 4	0.055490997	0.13588	0.161153	0.04947
Prog. 5	0.041353849	0.04503	0.034452	0.14666
Prog. 6	0.004120027	0.03039	0.004675	0.00450
Prog. 7	0.001814191	0.02328	0.005881	0.05118
Prog. 8	0.005268174	0.02492	0.007523	0.01767
Prog. 9	0.004609329	0.01056	0.005546	0.06469
Prog10	0.014617512	0.00703	0.015233	0.15364

Table 9: Statistical T-test results for hypothesis H1 for benchmarked programs

For the null hypotheses H2, p value is smaller than 0.05 for all programs, for the CSA Vs Random Search and CSA Vs [7], so we reject the hypotheses and claims that performance of The approach based on CSA is better than random search technique and [7] with respect to average coverage achieved. For CSA Vs[8] value of p is smaller than 0.05 for nine programs, in this case we can also rejects the hypotheses and claims that performance of CSA based approach is better than the [8] with respect average coverage achieved. For CSA Vs [9] value of p is smaller than 0.05 for seven programs, in this case we can also rejects the hypotheses and claims that performance of CSA based approach is better than the[9] with respect to APC.

Table 10: Statistical T Test results for hypothesis H2 for benchmarked programs

Program No.	p value CSA vs. Random Search	p value of CSA vs. [7]	p value CSA vs. [8]	p value CSA vs. [9]
Prog. 1	0.011404	0.002409	0.011003	0.017541
Prog. 2	0.008927	0.016213	0.015937	0.067708
Prog. 3	0.006033	0.018417	0.023103	0.045861
Prog. 4	0.008486	0.011404	0.007696	0.039802
Prog. 5	0.021204	0.005469	0.118993	0.106286
Prog. 6	0.002993	0.013631	0.006923	0.028834
Prog. 7	0.002993	0.015937	0.033138	0.126108
Prog. 8	0.005731	0.037913	0.008138	0.033138
Prog. 9	0.001089	0.011003	0.028834	0.045861
Prog. 10	0.001089	0.011003	0.028834	0.045861

For the null hypotheses H3, p value is smaller than 0.05 for all programs, for the CSA Vs Random Search and CSA Vs [7], so we reject the hypotheses and claims that performance of The approach based on CSA is better than random search technique and [7] with respect to average success rates. For CSA Vs[8] value of p is smaller than 0.05 for eight programs, in this case we can also rejects the hypotheses and claims that performance of CSA based approach is better than the [8] with respect average success rates. For CSA Vs [9] value of p is smaller than 0.05 for six programs, in this case we can also rejects the hypotheses and claims that performance of CSA based approach is better than the[9] in majority of cases with respect to ASR

Table 11: Statistical T Test results for hypothesis H3 for benchmarked programs

Program No.	p value CSA vs. Random Search	p value of CSA vs. [7]	p value CSA vs. [8]	p value CSA vs. [9]
Prog. 1	0.005375	0.006943	0.019507	0.020019
Prog. 2	0.004767	0.007696	0.013631	0.064674
Prog. 3	0.007696	0.013074	0.019507	0.040974
Prog. 4	0.001565	0.015888	0.032508	0.045861
Prog. 5	0.006012	0.010938	0.061009	0.091551
Prog. 6	0.002546	0.007696	0.009493	0.045861
Prog. 7	0.001812	0.006224	0.035497	0.057586
Prog. 8	0.008138	0.007316	0.016213	0.034452
Prog. 9	0.005442	0.008486	0.035594	0.041866
Prog. 10	0.038546	0.049683	0.092798	0.052192

VI. CONCLUSION

Use of nature inspired algorithms in the field of test case generation for program/software is now getting attention from the researcher community. Various nature inspired algorithms GA, PSO, ACO etc are used for test case generation and prioritization by considering different testing adequacy criterion. Data flow testing received a very little attention from the researchers. This paper uses Cuckoo Search Algorithm to generate optimal set of test suite for data flow testing. The test adequacy criterion selected here is all-uses criterion. For the guidance of the proposed approach in the search space a new objective function is designed which uses concept the dominance path in the CFG and branch distance. Experiments were carried out on 10 benchmarked programs to confirm the proposed approach and results are compared with earlier work done in this domain. For the comparison three performance parameters, average number of generations, average percentage of coverage achieved and average success rates are used. Further in order to prove that proposed approach performs better than the above mentioned approaches a statistical difference test (T-test) is also performed. Results of this test clearly indicate that proposed approach is significantly better than the others. In future this new approach can be applied on some industrial programs.

REFERENCES

1. B. Beizer, Software Testing Techniques (2Nd Ed.). New York, NY, USA: Van Nostrand Reinhold Co., 1990.
2. A. P. Mathur, Foundations of Software Testing, 1st ed. Addison-Wesley Professional, 2008.
3. C. Mao, "Generating Test Data for Software Structural Testing Based on Particle Swarm Optimization," Arab. J. Sci. Eng., vol. 39, no. 6, pp. 4593–4607, 2014.
4. S. Sharma, S. A. M. Rizvi, and V. K. Sharma, "Research on use of Nature Inspired Algorithms in Software Testing," Int. J. Innov. Technol. Explor. Eng., vol. 8, no. 11, pp. 3446–3452, 2019.
5. M. Harman, "Software Engineering Meets Evolutionary Computation," Computer (Long. Beach. Calif)., vol. 44, no. 10, pp. 31–39, Oct. 2011.
6. S. Jiang, J. Chen, Y. Zhang, J. Qian, R. Wang, and M. Xue, "Evolutionary approach to generating test data for data flow test," IET Softw., vol. 12, no. 4, pp. 318–323, 2018.
7. M. R. Girgis, "Automatic Test Data Generation for Data Flow Testing Using a Genetic Algorithm," J. Univers. Comput. Sci., vol. 11, no. 6, pp. 898–915, 2005.
8. A. S. Ghiduk, M. J. Harrold, and M. R. Girgis, "Using genetic algorithms to aid test-data generation for data-flow coverage," Proc. - Asia-Pacific Softw. Eng. Conf. APSEC, pp. 41–48, 2007.
9. S. S. Kumar, D. K. Yadav, and D. A. Khan, "An Adaptive PSO Algorithm Based Test Data Generator for Data-Flow Dependencies using Dominance Concepts," 2016.
10. S. Jiang, J. Shi, Y. Zhang, and H. Han, "Automatic test data generation based on reduced adaptive particle swarm optimization algorithm," Neurocomputing, vol. 158, pp. 109–116, 2015.
11. M. Patil and P. J. Nikumbh, "Pair-wise Testing Using Simulated Annealing," Procedia Technol., vol. 4, pp. 778–782, 2012.
12. N. Tracey, J. Clark, and K. Mander, "Automated program flaw finding using simulated annealing," Proc. 1998 ACM SIGSOFT Int. Symp. Softw. Test. Anal. ISSTA 1998, pp. 73–81, 1998.
13. N. Mansour and M. Salame, "Data Generation for Path Testing," pp. 121–136, 2004.
14. P. B. Nirpal and K. V Kale, "Using Genetic Algorithm for Automated Efficient Software Test Case Generation for Path Testing," Int. J. Adv. Netw. Appl., vol. 915, no. 6, pp. 911–915, 2011.
15. M. Prasanna, K. R. Chandran, and K. Thiruvenkadam, "Automatic test case generation for UML collaboration diagrams," IETE J. Res., vol. 57, no. 1, pp. 77–81, 2011.
16. A. Kaur and S. Goyal, "a Genetic Algorithm for Regression Test Case Prioritization Using Code Coverage," Int. J. Comput. Sci. Eng., vol. 3, no. 5, pp. 1839–1847, 2011.

17. W. Jun, Z. Yan, and J. Chen, "Test Case Prioritization Technique Based on Genetic Algorithm," in 2011 International Conference on Internet Computing and Information Services, 2011, pp. 173–175.
18. S. Singla, D. Kumar, H. M. Rai, and P. Singla, "A hybrid PSO approach to automate test data generation for data flow coverage with dominance concepts," Int. J. Adv. Sci. Technol., vol. 37, pp. 15–26, 2011.
19. N. Nayak and D. P. Mohapatra, "Automatic test data generation for data flow testing using particle swarm optimization," Commun. Comput. Inf. Sci., vol. 95 CCIS, no. PART 2, pp. 1–12, 2010.
20. S. Varshney and M. Mehrotra, "Search Based Software Test Data Generation for Structural Testing: A Perspective," SIGSOFT Softw. Eng. Notes, vol. 38, no. 4, pp. 1–6, Jul. 2013.
21. S. Kumar, D. K. Yadav, and D. A. Khan, "An accelerating PSO algorithm based test data generator for data-flow dependencies using dominance concepts," Int. J. Syst. Assur. Eng. Manag., vol. 8, no. 11, pp. 1534–1552, 2017.
22. A. S. Ghiduk, "A New Software Data-Flow Testing Approach via Ant Colony Algorithms," Univers. J. Comput. Sci. Eng. Technol., vol. 1, no. 1, pp. 64–72, 2010.
23. S. Biswas, M. S. Kaiser, and S. A. Mamun, "Applying Ant Colony Optimization in software testing to generate prioritized optimal path and test data," in 2015 International Conference on Electrical Engineering and Information Communication Technology (ICEEICT), 2015, pp. 1–6.
24. C. Mao, L. Xiao, X. Yu, and J. Chen, "Adapting ant colony optimization to generate test data for software structural testing," Swarm Evol. Comput., vol. 20, pp. 23–36, 2015.
25. R. K. Sahoo, D. P. Mohapatra, and M. R. Patra, "A Firefly Algorithm Based Approach for Automated Generation and Optimization of Test Cases," Int. J. Comput. Sci. Eng., vol. 4, no. 8, pp. 1–6, 2016.
26. R. Sharma and A. Saha, "Optimization of object-oriented testing using firefly algorithm," vol. 2667, no. October, 2017.
27. S. Sharma, S. A. M. Rizvi, and V. Sharma, "A Framework for Optimization of Software Test Cases Generation using Cuckoo Search Algorithm," in 2019 9th International Conference on Cloud Computing, Data Science & Engineering (Confluence), 2019, pp. 282–286.
28. S. Rapps and E. J. Weyuker, "Selecting Software Test Data Using Data Flow Information," IEEE Trans. Softw. Eng., vol. SE-11, no. 4, pp. 367–375, 1985.
29. P. C. Jorgensen, Software Testing Fourth Edition A Craftsman's Approach. 2014.
30. T. Lengauer and R. E. Tarjan, "A Fast Algorithm for Finding Dominators in a Flowgraph," ACM Trans. Program. Lang. Syst., vol. 1, no. 1, pp. 121–141, Jan. 1979.
31. R. B. Payne and M. D. (Michael D. Sorenson, The cuckoos. Oxford University Press, 2005.
32. X.-S. Yang and S. Deb, "Cuckoo Search via Levey Flights," 2009 World Congr. Nat. Biol. Inspired Comput. (NaBIC 2009), pp. 210–214, 2009.
33. R. A. Vazquez, "Training spiking neural models using cuckoo search algorithm," 2011 IEEE Congr. Evol. Comput. CEC 2011, pp. 679–686, 2011.
34. M. F. Shlesinger, G. M. Zaslavsky, and U. Frisch, Eds., Lévy Flights and Related Topics in Physics, 1st ed. Springer-Verlag Berlin Heidelberg, 1995.
35. M. F. Shlesinger, "Search & Research," Nature, vol. 19, no. 3, pp. 27–28, 2006.
36. J. Wegener, A. Baresel, and H. Sthamer, "Evolutionary test environment for automatic structural testing," Inf. Softw. Technol., vol. 43, no. 14, pp. 841–854, 2001.
37. B. Korel, "Automated Software Test Data Generation," IEEE Trans. Softw. Eng., vol. 16, no. 8, pp. 870–879, Aug. 1990.
38. N. Tracey, J. Clark, J. McDermid, and K. Mander, "A Search-Based Automated Test-Data Generation Framework for Safety-Critical Systems," in Systems Engineering for Business Process Change: New Directions: Collected Papers from the EPSRC Research Programme, P. Henderson, Ed. London: Springer London, 2002, pp. 174–213.
39. S. Varshney and M. Mehrotra, "Search-Based Test Data Generator for Data-Flow Dependencies Using Dominance Concepts, Branch Distance and Elitism," Arab. J. Sci. Eng., vol. 41, no. 3, pp. 853–881, 2016.

AUTHORS PROFILE

Sanjiv Sharma is an assistant professor at KIET Group of Institutions, Ghaziabad. He received his B.Tech. degree from MMMEC, Gorakhpur affiliated to AKTU, Lucknow, India in 2008, M.Tech degree from Shobhit University Meerut, India in 2014, and pursuing Ph.D. from Jamia Milllia University, New Delhi, India. His research interests include software testing and nature inspired algorithm. One can connect Sanjiv Sharma on martin.mmmec@gmail.com.

S.A.M. Rizvi is a professor and former HoD at Jamia Millia University, New Delhi India. He received his Ph.D from Dr. R. M. L. Avadh University, India, in 1996. His research interests are Knowledge Engineering, MIS, Automation, Algorithms and Bioinformatics. One can connect SAM Rizvi on samsam_rizvi@yahoo.com.

Vineet Kumar Sharma is a professor and HoD at KIET Group of Institutions, Ghaziabad, India. He received his Ph.D. from Jamia Millia Islamia University, New Delhi, India in 2012. His research interest are algorithms, Software Engineering. One can connect Vineet Kumar Sharma on vineet.sharma@kiet.edu.