Article

Generating and prioritizing optimal paths using ant colony optimization

Mukesh Mann, **Om Prakash Sangwan**
School of ICT, Gautam Buddha University, Greater Noida, 201312, India
E-mail: Mukesh.gbu@gmail.com, Sangwan_op@yahoo.co.in

Received 14 September 2014; *Accepted 20 October 2014*; *Published online 1 March 2015*

Abstract
The assurance of software reliability partially depends on testing. Numbers of approaches for software testing are available with their proclaimed advantages and limitations, but accessibility of any one of them is a subject dependent. Time is a critical factor in deciding cost of any project. A deep insight has shown that executing test cases are time consuming and tedious activity. Thus stress has been given to develop algorithms which can suggest better pathways for testing. One such algorithm called Path Prioritization –Ant Colony Optimization (PP-ACO) has been suggested in this paper which is inspired by real Ant's foraging behavior to generate optimal paths sequence of a decision to decision (DD) path of a graph. The algorithm does full path coverage and suggests the best optimal sequences of path in path testing and prioritizes them according to path strength.

Keywords Control Flow Graph (CFG); ant colony optimization (ACO); Ant System (AS); Decision to Decision (DD) graph.

Computational Ecology and Software
ISSN 2220-721X
URL: http://www.iaees.org/publications/journals/ces/online-version.asp
RSS: http://www.iaees.org/publications/journals/ces/rss.xml
E-mail: ces@iaees.org
Editor-in-Chief: WenJun Zhang
Publisher: International Academy of Ecology and Environmental Sciences

1 Introduction
Testing is an important aspect of the software development life cycle. It focuses on the process of testing the newly developed / under development software system, prior to its use. The program is executed with desired input(s) and the output(s) is / are observed accordingly. Observed/Actual output(s) is compared with the expected output(s), if they are same then the program under test is said to be correct as per its specification(s), otherwise there is something wrong somewhere in the program. Testing is the process of executing a program with the intent of finding faults (Mayers, 1977). The growing importance of software Systems for small and big organization results in more complex software systems. Thus it is one of the logic that more stress has been given on quality of developed software (s). History has shown that software faults not only caused a loss of money but also precious human lives. Thus if we don't improve the quality while system is under development, these losses may only get bigger (Beizer, 1990).

In this paper we have proposed a novel algorithm for automatic generation and prioritization of optimal path in decision to decision Control Flow Graph (CFG).

One of the promising areas of research in Artificial Intelligence (AI) is software testing (Briand, 2002; Pedrycz et al., 1998; Phil, 2004; Harman, 2007). One such metahurestic technique of AI is Ant colony optimization (Dorigo et al., 2005 and Ayari et al., 2007) inspired by real Ant colonies that forage for food and construct solutions for discrete optimization problems (Dorigoa et al., 2005). During last few many years Ant Colony Optimization approach has been used by many authors to automatic test sequence generation for state-based software testing (Li et al., 2005). Mohan V and others (Mohan et al., 2008) has proposed a graph based algorithm for deciding the shortest path between starting and frontier nodes. All nodes present in execution state sequence graph are covered by this approach but all paths were not covered thus leaving the algorithm in weak state.

This paper aims to design an algorithm which is inspired by real Ant's foaging behavior to generate optimal paths sequence of a control flow graph CFG (Singh, 2012) and prioritize them according to combined level of pheromone and heuristic value over the path. A CFG is a graphical representation of the source code of a program. The statements of a program are represented by nodes and flow of control by edges in the program graph (Singh, 2012). In our work we have concentrated on decision to decision (DD) paths of a CFG, which is a directed graph in which nodes are sequence of statements and edges are control flow amongst the nodes.

2 Related Work

The basic aim of various proposed techniques in software testing is to find maximum faults in minimum time. During the last few years many techniques, approaches and methodologies have been suggested that not only reduce time and cost during software test life cycle (STLC), but also results in optimized generation and prioritization of test cases (Brottier et al., 2006; Korel et al., 1991; Leon et al., 2003; Binkley, 1992; Do et al., 2006; Leung et al., 1991; Do et al., 2006). Numerous approaches which are based on empirical studies for test case prioritization have been studied and proposed which reduces time and cost during STLC (Chilenski et al., 1994; Rothermel et al., 2002; Kim et al., 2000; Walcott et al., 2006; Leon et al., 2003). Major area of research is in structural testing that focus on (i) the code, (ii) code's Structure, (iii) internal design of code and (iv) how the code is actually coded. The general techniques that include structural testing are (Boujarwah et al., 1997) (i) control flow/coverage testing, (ii) basic path testing (iii) loop testing, and (iv) data flow testing (Srinivas et al., 2012).

The coverage testing further includes statement coverage, branch coverage, decision coverage, function coverage and conditional coverage. In a statement coverage every node in CFG is covered at least once (Liu et al., 2008; Frankl et al., 2000). In branch coverage we test every branch of the program i.e. each edge in CFG is traversed at least once (Mitra et al., 2011). Hence we try to test every true and false condition of the program. In Decision Coverage, for each decision at decision point we measure the percentage of the total number of paths traversed in the test. If every possible path in decision point has been traversed, it is said to achieve full coverage (Mitra et al., 2011; Liu et al., 2008; Frankl et al., 2000). Here condition is to make sure that for each branch all condition expression will be tested.

In condition coverage, a condition is a either a True or False (Boolean value) condition and it cannot be divided further (Sharma et al., 2010). Several Boolean conditions (Mitra et al., 2011) collectively make a single decision. In Function coverage a sample product is mapped into functions and the product is realized/ tested by calling these mapped functions. Test cases are designed to be executed on every function(s) inside the code (Lin et al., 2012; Huang et al., 2012). As described earlier second type of structural testing is basic path testing which further includes flow graph notation, cyclometric complexity, deriving test cases and graph matrix. In graph notation, an edge from one node including predicate node(s) in CFG must be converging at some definite node. The area under edges and nodes is called region of CFG (Zhonglin et al., 2010). McCabe presented the notion of cyclometric complexity. Cyclometric complexity (CC) is software metric that measure the extent to which a program is difficult and it depends on CFG of the program under test (Liu et al., 2009; Lammermann et al., 2008). Authors have drawn CFG from the given design /code and determine the CC of the resultant flow graph V (G). From the CC, linearly independent paths are calculated. Finally the test cases are designed and executed over each independent path (Liu et al., 2009; Mitra et al., 2011; Zhonglin et al., 2010; Lammermann et al., 2008). At the last, flow graphs and basic path sets are derived using Graph matrices (Zhang and Mei, 2010).

3 The Ant Colony: Metahurestic and the PP-ACO Algorithm Details

In next section, we briefly explain how an ant using pheromone evaporation and Heurestic updation forage for food. We proposed an algorithm that we call Path Prioritization –ACO (PP-ACO for short) to solve path prioritization problem.

3.1 Solution construction by moving forward

In PP-ACO is based on Simple ACO (S-ACO for short) which is working in two modes: Forward and backward. In Forward mode, Ant’s moves from source (i.e. Nest) to Destination (Food) and Vice-versa in backward mode. Once an Ant reaches to destination during forward mode, it switches to backward mode and start its journey towards source. In PP-ACO, Forward Ants builds solutions by taking a probabilistic decision about the next move to be taken among all available neighborhood nodes in CFG. This probabilistic decision is governed by pheromone trails and heuristic deposited on the path by other Ants. While in forward mode an Ant does not deposit any pheromone while moving towards destination.

3.2 Backward move and updation

Ant’s by using explicit memory retrace the path they had covered while searching the destination node.

3.3 Solution quality- biased pheromone updates

In PP-ACO algorithm, the Ants keep track of all the nodes visited during the forward move, and the cost associated with the visited edge if the graph is weighted. And hence the solution cost they build can be evaluated and corresponding pheromone and heuristic can be updated on each visited arc.

4 PP-ACO System Inspired by AS

We now present how the real ant foraging behaviour is implemented in PP-ACO algorithm to build the solution for optimal path generation and prioritization problems over CFG graph of a program under test. A variable t_{ij} (artificial pheromone trails) is associated with each arc in CFG, t_{ij} are read and updated by the ants during their journey from source to destination and vice-versa.

4.1 Tour construction

In Ant System (AS), m (artificial) Ants concurrently build a tour of the CFG. Initially, an Ant is put on initial node and by using probabilistic decision rule, to decide which node to visit next. In particular, the probability with which Ant k, currently at node i, chooses to go to node j is (Dorigo et. al., 1996)

$$p_{ij}^{k} = \frac{[\tau_{ij}]^{\alpha}[\eta_{ij}]^{\beta}}{\sum_{l \in \mathcal{N}_i^k}[\tau_{il}]^{\alpha}[\eta_{il}]^{\beta}}; \quad \text{if } j \in \mathcal{N}_i^k; \qquad (1)$$

where, Nij=1/dij is a heuristic value that indicate the path’s visibility for an ant at current node. This information is generally available a priori, two parameters called α and β determines the influence of pheromone and heuristic, and N_i^k is the represents the neighborhood of ant k at node i. P_{ij}^K is the probability of choosing a node. α and β plays major role in above probabilistic rule, If α =0, the visibility factor grows. i.e more close nodes are selected and if β =0, the Pheromone factor grows i.e P_{ij}^K is only pheromone biased. It may result in poor solution building. Thus it is very important to choose good parameter values for the algorithms, Some well know parameter settings are presented in this section are summarized in the table1. At last each ant has a memory M^K that contains all the records about the path visited, the neighborhood while building the solution and the length of the tour T^k it covered.

Experimental study of the various ACO algorithms for the travelling salesman problem (TSP) has identified parameter settings that result in good performance (Dorigo et al., 2005) are given in Table 1.

Table 1 Parameter setting for various aco algorithms for TSP.

ACO algorithm	A	β	Pheromone Evaporation (ρ)	Number of Ants m	Initial Pheromone t_0
AS	*1*	*2 to 5*	*0.5*	*N*	m/c^{nn}
EAS	*1*	*2 to 5*	*0.5*	*N*	$(e+m)/ \rho\, c^{nn}$
Asrank	*1*	*2 to 5*	*0.1*	*N*	$0.5r(r-1)/ \rho\, c^{nn}$
MMAS	*1*	*2 to 5*	*0.02*	*N*	$1/ \rho\, c^{nn}$
ACS	-	*2 to 5*	*0.1*	*10*	$1/nc^{nn}$

In PP-ACO, we have assumed ρ= 1%, α=1, β=1 and m=n. The reason for choosing m=n is that the Ant is working in a sequential way , that is when an Ant had reached from initial position to the food source, She is assumed to be in dead state after depositing a constant amount of pheromone over the path and updating heuristic value over the traversed path. The next Ant will start its journey only after the Ant prior to it had finished its journey.

4.2 Update of pheromone trails in sequential way

After an Ant has constructed its tour, t_{ij} are updated by first lowering the constant amount of pheromone followed by addition of pheromone on the arc visited by ants during their tour. The evaporation of pheromone is given as

$$t_{ij} \leftarrow (1-\rho)\, t_{ij}$$

Where ρ is pheromone evaporation rate such that $0 < \rho < 1$. The parameter ρ avoids excess addition of pheromone over the visited path and thus it allow the algorithm to take care about the wrong decisions taken previously. After evaporation, the deposition of pheromone take place on the arc visited in the tour as

$$t_{ij} \leftarrow t_{ij} + \Delta t^k_{ij} \qquad (2)$$

where Δt^k_{ij} is the pheromone's amount deposited by the k^{th} ant on the arcs (i,j) and Δt^k_{ij} is calculated as

$$\Delta t^k_{ij} = \begin{cases} 1/ L_i^{\,k} & \text{; if arc(i,j) belongs to length of Tour } (T^k) \\ 0 & \text{; else} \end{cases} \qquad (3)$$

where $L_i^{\,k}$ (the length of the tour (T^k) constructed by the Ant k in Sequential way.) is calculated by taking the the summation of the lengths of the arcs visited by ant.

In the Similar manner the Heuristic value is updated as

$$\Delta n_{ij} \leftarrow n_o / C_i^k \tag{4}$$

We have used triangle classification program (Singh, 2012) as given in program1 to generate its DD graph using decision node coverage.

/* Program to classify whether a triangle is acute, obtuse or right angle, given the sides of triangle: **Program 1**

```
// header files
#include<stdio.h>
#include<math.h>
1.  void main ()// main begin
2.  {
3.  double a.b.c;
4.  double a1,a2,a3;
5.  int valid=0;
6.  clrscr();
7.  printf(“enter first side of triangle”);          /* enter the side of triangle */
8.  scanf(“%lf”,&a);
9.  printf(“enter    second side of triangle”);
10. scanf(“%lf”,&b);
11. printf(“enter    third side of triangle”);
12. scanf(“%lf”,&c);
/* check whether a triangle is valid or not */
13. if (a>0 && a<100 && b>0 && b<=100 && c>0 &&c<100){
14. if ((a+b)>c && (b+c)>a && (c+a)>b){
15. valid=1;
16. }
17. else{
18. valid= -1;
19. }
20. }
21. if ( valid==1) {
22. a1=(a*a +b*b)/(c*c);
23. a2=(b*b + c*c)/(a*a);
24. a3=(c*c +a*a)/(b*b);
25. if (a1< =1 || a2<=1 ||a3<=1){
26. printf(“obtuse angled triangle”);
27. }
28. else if (a1==1 || a2==1 ||a3==1){
29. printf(“right angled triangle”);
30. }
31. else {
32. printf(“acute angled triangle”);
33. }
34. }
35. else if ( valid == -1){
```

```
printf("\n Invalid triangle ");
}
else {
printf("\n input values are out of range ");
}
getch();
}      //main ends
```

The corresponding DD path graph (Singh, 2012) for the program 1 is shown in Fig. 1.

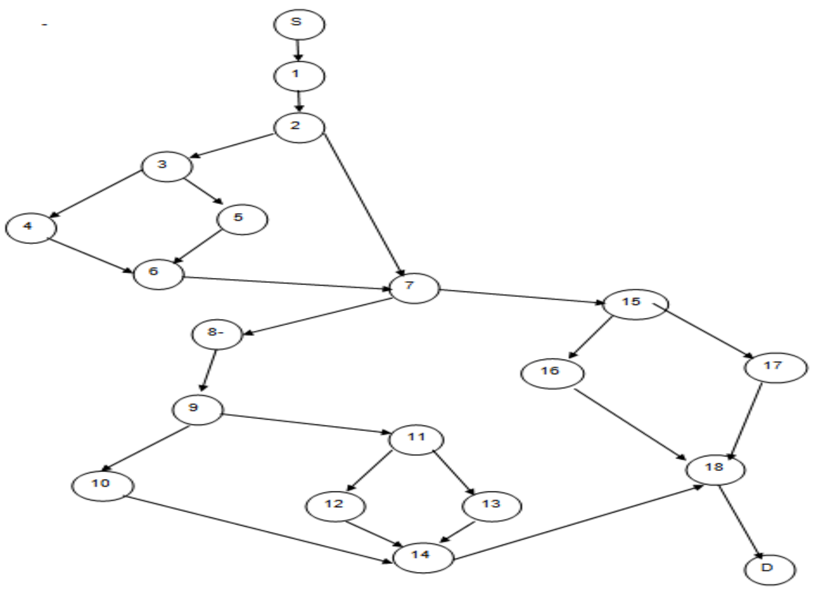

Fig. 1 DD path graph for program 1.

Notice that the cyclometric complexity for this graph is 7. Thus the program has 7 numbers of independent paths as mentioned in Table 2.

Table 2 Independent DD path for the triangle classification problem.

S.no	Independent paths
1	S,N1,N2,N7,N15,N17,N18,D
2	S,N1,N2,N7,N15,N16,N18,D
3	S,N1,N2,N7,N8,N9,N11,N11,N13,N14,N18,D
4	S,N1,N2,N7,N8,N9,N11,N11,N12,N14,N18,D
5	S,N1,N2,N7,N8,N9,N10,N14,N18,D
6	S,N1,N2,N3,N5,N6,N7,N8,N9,N10,N14,N18,D
7	S,N1,N2,N3,N4,N6,N7,N8,N9,N10,N14,N18,D

The path prioritization problem for a directed graph V (G) as shown in (Fig. 1) is to decide which path among the several available paths must undergo for testing first and why. The various available paths are shown in Table 3.

Table 3 Various available paths in triangle classification DD path graph.

S.no	All available paths
1	S,N1,N2,N7,N15,N16,N18,D
2	S,N1,N2,N7,N15,N17,N18,D
3	S,N1,N2,N7,N8,N9,N10,N14,N18,D
4	S,N1,N2,N7,N8,N9,N11,N12,N14,N18,D
5	S,N1,N2,N7,N8,N9,N11,N13,N14,N18,D
6	S,N1,N2,N3,N4,N6,N7,N15,N16,N18,D
7	S,N1,N2,N3,N4,N6,N7,N15,N17,N18,D
8	S,N1,N2,N3,N4,N6,N7,N8,N9,N10,N14,N18,D
9	S,N1,N2,N3,N4,N6,N7,N8,N9,N11,N12,N14,N18,D
10	S,N1,N2,N3,N4,N6,N7,N8,N9,N11,N13,N14,N18,D
11	S,N1,N2,N3,N5,N6,N7,N15,N16,N18,D
12	S,N1,N2,N3,N5,N6,N7,N15,N17,N18,D
13	S,N1,N2,N3,N5,N6,N7,N8,N9,N10,N14,N18,D
14	S,N1,N2,N3,N5,N6,N7,N8,N9,N11,N12,N14,N18,D
15	S,N1,N2,N3,N5,N6,N7,N8,N9,N11,N13,N14,N18,D

4.3 The PP-ACO algorithm details

- α and β are two parameters which determine the relative influence of the pheromone trail and the heuristic information on the path traversed by Ants.
- τ_0 and n_o are initial Pheromone and heuristic value deposited on each arc in CFG respectively. Initially τ_0 is set equal to the depth of CFG and n_o is set equal to the no of decision nodes in CFG.
- Visit: this variable specify the visit status of node by the Ant k, initially set to zero.
- N_i^k is the feasible neighborhood of Ant k when being at node i, initially set to zero.
- Key =End Node in CFG
- Gpath=[Total no of nodes in the CFG]. G path for above example is written as Gpath =[1,2,3,4,5,6,7,8,9,10,11,12,Endnode].
- NC= Total Node sequence covered so far.
- ρ= 1%, α=1, β=1; C_i^k =0;

  1. Calculate Depth of CFG using algorithm PP-ACO_ DEPTH as per 2.15.
     a. initialize τ_0 accordingly

```
        b.  Determine number of decision nodes in CFG and set the initial value of $n_o$
2.  i= Start ;visitcount=0;
    2.1. if (vk[i] ==0);
    2.2. Set vk[i] $\leftarrow$ 1;
    2.3. Vistcount $\leftarrow$ Vistcount +1;
    2.4. /*find the feasible neighborhood of Ant k in state i as $N_{ij}^k$
    2.5.
        $P_{ij}^k = \frac{|\tau_{ij}^k|^\alpha \text{ x } |N^k_{ij}|^\beta}{\Sigma_{l\sim Ni}^k [|\tau_{ij}^k|^\alpha \text{ x } |N^k_{ij}|^\beta]}$        $\forall j \in N_i^k$

        Move to the next node as follow
    2.5.1 Select Next Node having $\max(P_{il})^k$  where $l \in N_i^k$
    2.5.2     if  $P_{ij} == P_{il}$  where $l,j \in N_i^k$ & l=j,
              Select next node as per 2.5.2.1
        2.5.2.1  $\forall$  $l\sim N^k$ ,Find V[l]
              if (V[l]==key)
              next node=key;
              elseif (V[l]== V[j]==0) where $l,j \in N_i^k$ & l=j
              next node=j or l. i.e Cjoose next node randomly.
              else
              V[l]==0;
              next node= V[l];

    2.6 Update Pheromone as
        First Lower the Pheromone Trail as

              $\tau_{ij} \leftarrow (1-\rho)\, \tau_{ij}$      $\forall$  $j \in N_i^k$

    2.7  Find Tour Covered so far by Ant K  starting from node i
              $T_i^k$ = i+ Next Node

    2.8       Find the length of Tour $T_i^k$  as
        a.    $C_i^k = C_i^k$ + Length($T_i^k$)  /* Summation Of all Path Cost i.e Distance between
                                              each Node.

        b.    Find total  Nodes Covered So far
              NC=NC U $N_i^k$

    2.9       Start=Next Node
                If(Start!==Key)
                Go to step 2
                  Else if ( v[Start]==0)
                V[Start]=1
                VisitCount[start] $\leftarrow$ VisitCount[start]+1
```

```
            $T_i^k$ = i + Start;
            $C_i^k = C_i^k$ + Length($T_i^k$);
            Else(v[Start]==0)
            Discard The path as it has already visited otherwise add    new path

2.10        Find $\Delta \tau_{ij}$ to be deposited on each arc (i,l)belong to $N_i^k$
```

$$\Delta \tau_{ij} = \begin{cases} 1/C_i^k ; & \text{if arc (i,j) belongs to } T_i^k \\ 0 & ; \text{Otherwise} \end{cases}$$

```
            $\tau_{ij} \leftarrow \tau_{ij} + \Delta \tau_{ij}$
            $\Delta n_i^j \leftarrow n_{ij} / C_i^k$

2.11        While(NC!=Gpath)
              Go to step 2
```

2.12 Find Strength for all independent Paths as per 2.13.and arrange it in descending Order of path's strength.

2.13 Strength [T^k] = $\Sigma_{i,j \in Tk}$ $\tau_{ij} + n_i$ $\forall$ $i,j \in T^k$
such that i=j and i=Start node~T^k, j=next node after i in T^k.

```
2.13    For each $i \in$Gpath
            Print visitcount[i]
              Stop
```

2.14 **PP-ACO_ DEPTH** pseudo code -To calculate the depth of CFG tree max_Depth()

```
1. If tree is empty then return 0
2. Else perform the following steps
    (a) Recursively, get the maximum depth of left subtree
            i.e.,
        call max_Dept ( tree->left_subtree)
    (a) Recursively, get the max depth of right subtree i.e.,
        call max_Depth( tree->right_subtree)
    (c) Get the max of max depths of left and right
        subtrees and add 1 to it for the current node.
        max_depth = max(max dept of left _subtree, max depth of right _subtree) + 1
    (d) Return max_depth
```

In the proposed algorithm Ant traverse all nodes available in DD path graph of a CFG and hence provide full path coverage. It has the ability to collect knowledge about the number of times a node is traversed. Algorithms will stops when Ant has covered all nodes at least once.

5 Experimental Results

We have used Triangle classification problem to demonstrate the algorithm.The algorithm traverse each node and prioritize the path according to the path strength. Paths having the highest pheromone strength are given the highest priority for testing followed by next lower pheromone strength.

5.1 Example illustration/validation

Consider the triangle classification program given in the program 1. We first draw its DD path shown in the Fig1. Our task is to generate the various independent paths an Ant can move upon and then prioritize these paths (table 2) for testing. We apply the purposed algorithm as follow.

1. Initially an Ant starts from staring node S and find its neighborhood N_s^k= {1}. 1% pheromone is evaporated from the edge covered so far i.e. edge s-1, initially edge has 0.085 pheromone value which after evaporation is left to 0.082. The tour covered after finding each next node to be moved upon is calculated and thus its length i.e. for (s-1), the tour is {s-1} whose length is equal to 1. The next node to be moved upon is node 1.
2. As the next node to be moved upon is not equal to key node (i.e. end node) an Ant's starting node is now node 1and the above steps are again followed.
3. At Node 2, N_2^k = {3, 7}. Apply random proportional rule to decide the next move of Ant. As the probabilities of moving an Ant from node 2 to node 3 & node 7 are same and no node among node 3, 7 is equal to key node (i.e. end node), we choose the next move randomly. Suppose the Ant choose node 3 as its next move. The process is repeated till the Ant reaches to destination node d.
4. As NC(node covered so far)={s,1} which is not equal to GPATH={S,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,D).The Ant again start from the start node S.
5. As NC (node covered so far)={s,1,2,3,4,6,7,8,9,10,14,18,D). The Ant had now covered one full path starting from node 1 to node D. The amount of pheromone deposited on each edge in the traversed path so far is calculated as 1/12=0.083 (as the path length covered so far is equal to 12). The net pheromone is updated on each edge in the path equal to amount left after evaporation of pheromone + amount deposited after path traversed, i.e. 0.082+0.083=0.165.
6. The change in heuristic is calculated i.e. $\Delta n_i^j = n_{ij} / C_i^k$ which is equal to 6/12=0.5.
7. As NC (node covered so far)={s,1,2,3,4,6,7,8,9,10,14,18,D) which is not equal to GPATH={S,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,D). The Ant again starts from the start node S.
8. The process is continued till all nodes are covered.
9. When NC=GPATH, we calculate the strength of each path. For example for the path{s,1,2,3, 4,6,7,8,9,10,14,18,D), the strength is calculated as
   a. S-1= (final pheromone value)*(final heuristic value) i.e.$1.071*1.58*10^{-8}=1.692*10^{-8}$. Similarly for the others edges in the path we calculate edge strengths. Finally adding all edge's strength in the path to give the final value for the path strength. i.e. (S-1+1-2+2-3+3-4+4-6+6-7+7-8+8-9+9-10+10-14+14-18+18-D)= 0.04132.
   b. Find path strength for all independent Paths in the graph.
10. Finally finding the total number of times the Ant had visited a particular node.

The final paths strength for all independent paths as per purposed algorithm has been shown in Table 5. The Ant prioritizes all the available paths in total of nine moves. The result for all moves starting from the start node up to the key node is shown in Table 4.

Table 4 Result **s**ummary for different moves of an Ant.

Move no.	NC Node Covered	t_0 Initial Pheromone	$t_e \leftarrow (1-\rho)^* t_0$ Evaporation	C_i^k Length of tour	$\Delta\ t_{ij} = 1/ C_i^k$	$t_{new} \leftarrow t_e + \Delta\ t_{ij}$	n_o	$\Delta\ n_{ij} \leftarrow n_o / C_i^k$ Left Heurestic
FIRST	S-1	0.083	0.082	12	0.083	0.165	6	0.5
	1-2	0.083	0.082	12	0.083	0.165	6	0.5
	2-3	0.083	0.082	12	0.083	0.165	6	0.5
	3-4	0.083	0.082	12	0.083	0.165	6	0.5
	4-6	0.083	0.082	12	0.083	0.165	6	0.5
	6-7	0.083	0.082	12	0.083	0.165	6	0.5
	7-8	0.083	0.082	12	0.083	0.165	6	0.5
	8-9	0.083	0.082	12	0.083	0.165	6	0.5
	9-10	0.083	0.082	12	0.083	0.165	6	0.5
	10-14	0.083	0.082	12	0.083	0.165	6	0.5
	14-18	0.083	0.082	12	0.083	0.165	6	0.5
	18-D	0.083	0.082	12	0.083	0.165	6	0.5
Second	S-1	0.165	0.163	7	0.142	0.3058	0.5	0.0714
	1-2	0.165	0.163	7	0.142	0.3058	0.5	0.0714
	2-7	0.083	0.082	7	0.142	0.2248	6	0.857
	7-15	0.083	0.082	7	0.142	0.2248	6	0.857
	15-16	0.083	0.082	7	0.142	0.2248	6	0.857
	16-18	0.083	0.082	7	0.142	0.2248	6	0.857
	18-D	0.083	0.082	7	0.142	0.2248	6	0.857
Third	S-1	0.3058	0.3027	7	0.142	0.4447	0.0714	0.0102
	1-2	0.3058	0.3027	7	0.142	0.4447	0.0714	0.0102
	2-7	0.2248	0.2225	7	0.142	0.3645	0.857	0.1224
	7-15	0.2248	0.2225	7	0.142	0.3645	0.857	0.1224
	15-17	0.083	0.0821	7	0.142	0.2241	6	0.857
	17-18	0.083	0.0821	7	0.142	0.2241	6	0.857
	18-D	0.2248	0.2225	7	0.142	0.3645	0.857	0.1224
Fourth	S-1	0.4447	0.4402	13	0.0769	0.5171	0.0102	$7.84*10^{-4}$
	1-2	0.4447	0.4402	13	0.0769	0.5171	0.0102	$7.84*10^{-4}$
	2-3	0.165	0.1633	13	0.0769	0.2402	0.5	0.03846
	3-5	0.083	0.0821	13	0.0769	0.1590	6	0.4615
	5-6	0.083	0.0821	13	0.0769	0.1590	6	0.4615
	6-7	0.165	0.1633	13	0.0769	0.2402	0.5	0.03846
	7-8	0.165	0.1633	13	0.0769	0.2402	0.5	0.03846
	8-9	0.165	0.1633	13	0.0769	0.2402	0.5	0.03846
	9-11	0.083	0.0821	13	0.0769	0.1590	6	0.4615
	11-12	0.083	0.0821	13	0.0769	0.1590	6	0.4615
	12-14	0.083	0.0821	13	0.0769	0.1590	6	0.4615
	14-18	0.165	0.1633	13	0.0769	0.2402	0.5	0.03846
	18-D	0.3645	0.3608	13	0.0769	0.4377	0.1224	$9.415*10^{-3}$
FIFTH	S-1	0.5171	0.5119	7	0.142	0.6539	$7.84*10^{-4}$	$1.12*10^{-4}$
	1-2	0.5171	0.5119	7	0.142	0.6539	$7.84*10^{-4}$	$1.12*10^{-4}$
	2-7	0.3645	0.3608	7	0.142	0.5028	0.1224	0.01748
	7-15	0.3645	0.3608	7	0.142	0.5028	0.1224	0.01748
	15-16	0.2248	0.2225	7	0.142	0.3645	0.857	0.1224
	16-18	0.2248	0.2225	7	0.142	0.3645	0.857	0.1224

	18-D	0.4377	0.4333	7	0.142	0.5753	$9.415*10^{-3}$	$1.345*10^{-3}$
SIXTH	S-1	0.6539	0.6473	12	0.083	0.7303	$1.12*10^{-4}$	$9.33*10^{-6}$
	1-2	0.6539	0.6473	12	0.083	0.7303	$1.12*10^{-4}$	$9.33*10^{-6}$
	2-3	0.2402	0.2377	12	0.083	0.3207	0.03846	$3.205*10^{-3}$
	3-4	0.165	0.1633	12	0.083	0.2463	0.5	0.0416
	4-6	0.165	0.1633	12	0.083	0.2463	0.5	0.0416
	6-7	0.2402	0.2377	12	0.083	0.3207	0.03846	$3.205*10^{-3}$
	7-8	0.2402	0.2377	12	0.083	0.3207	0.03846	$3.205*10^{-3}$
	8-9	0.2402	0.2377	12	0.083	0.3207	0.03846	$3.205*10^{-3}$
	9-10	0.165	0.1633	12	0.083	0.2463	0.5	0.0416
	10-14	0.165	0.1633	12	0.083	0.2463	0.5	0.0416
	14-18	0.2402	0.2377	12	0.083	0.3207	0.03846	$3.205*10^{-3}$
	18-D	0.5753	0.5695	12	0.083	0.6525	$1.345*10^{-3}$	$1.1208*10^{-4}$
SEVENTH	S-1	0.7303	0.7229	7	0.142	0.8649	$9.33*10^{-6}$	$1.33*10^{-6}$
	1-2	0.7303	0.7229	7	0.142	0.8649	$9.33*10^{-6}$	$1.33*10^{-6}$
	2-7	0.5028	0.4977	7	0.142	0.6397	0.01748	$2.497*10^{-3}$
	7-15	0.5028	0.4977	7	0.142	0.6397	0.01748	$2.497*10^{-3}$
	15-17	0.2241	0.2218	7	0.142	0.3638	0.857	0.1224
	17-18	0.2241	0.2218	7	0.142	0.3638	0.857	0.1224
	18-D	0.6525	0.6459	7	0.142	0.7879	$1.1208*10^{-4}$	$1.601*10^{-5}$
EIGHTH	S-1	0.8649	0.8562	7	0.142	0.998	$1.33*10^{-6}$	$1.9*10^{-7}$
	1-2	0.8649	0.8562	7	0.142	0.998	$1.33*10^{-6}$	$1.9*10^{-7}$
	2-7	0.6397	0.6633	7	0.142	0.7753	$2.497*10^{-3}$	$3.567*10^{-4}$
	7-15	0.6397	0.6633	7	0.142	0.7753	$2.497*10^{-3}$	$3.567*10^{-4}$
	15-16	0.3645	0.3608	7	0.142	0.5028	0.1224	0.01748
	16-18	0.3645	0.3608	7	0.142	0.5028	0.1224	0.01748
	18-D	0.7879	0.7800	7	0.142	0.9220	$1.601*10^{-5}$	$2.287*10^{-6}$
NINETH	S-1	0.998	0.9880	12	0.083	1.071	$1.9*10^{-7}$	$1.58*10^{-8}$
	1-2	0.998	0.9880	12	0.083	1.071	$1.9*10^{-7}$	$1.58*10^{-8}$
	2-3	0.3207	0.317	12	0.083	0.4	$3.205*10^{-3}$	$2.670*10^{-4}$
	3-5	0.1590	0.1574	12	0.083	0.2404	0.4615	0.03845
	5-6	0.1590	0.1574	12	0.083	0.2404	0.4615	0.03845
	6-7	0.3207	0.317	12	0.083	0.4	$3.205*10^{-3}$	$2.670*10^{-4}$
	7-8	0.3207	0.317	12	0.083	0.4	$3.205*10^{-3}$	$2.670*10^{-4}$
	8-9	0.3207	0.317	12	0.083	0.4	$3.205*10^{-3}$	$2.670*10^{-4}$
	9-11	0.1590	0.1574	12	0.083	0.2404	0.4615	0.03845
	11-13	0.083	0.082	12	0.083	0.165	6	0.5
	13-14	0.083	0.082	12	0.083	0.165	6	0.5
	14-18	0.3207	0.317	12	0.083	0.4	$3.205*10^{-3}$	$2.670*10^{-4}$
	18-D	0.9220	0.9127	12	0.083	0.9957	$2.287*10^{-6}$	$1.905*10^{-7}$

The final path according to the total path strength and priority has been shown in Table 5. The path having the maximum combined strength of pheromone and heuristic has been given the highest priority.

Table 5 Independent path's strength vs. priority vs. If- Else statements.

S.no	All Independent paths	Strength	Priority	No. Of if- else Statements covered
1	S,N1,N2,N7,N15,N16,N18,D	0.01813	7	3
2	S,N1,N2,N7,N15,N17,N18,D	0.0896	3	4
3	S,N1,N2,N7,N8,N9,N10,N14,N18,D	0.02108	6	3
4	S,N1,N2,N7,N8,N9,N11,N12,N14,N18,D	0.1565	2	4
5	S,N1,N2,N7,N8,N9,N11,N13,N14,N18,D	0.1748	1	5
6	S,N1,N2,N3,N4,N6,N7,N8,N9,N10,N14,N18,D	0.04132	4	4
7	S,N1,N2,N3,N5,N6,N7,N8,N9,N10,N14,N18,D	0.03951	5	4

The proposed algorithm generates all DD paths in the CFG and prioritizes them according to the path strength. One interesting point of observation with the algorithm is its stoppage criteria. The algorithm will stop only when all nodes have been covered, thus making 100% path coverage.

6 Discussion

This paper presented an Ant colony optimization approach to automatically generate and prioritize the optimal test path in a directed DD graph using the PP-ACO, an Ant can effectively explore the CFG states and automatically prioritize the path for testing. PP-ACO prioritizes the test paths according to Ant's natural foraging behavior and thus helps software testers to greater extent. A number of similar extensions of this algorithm are possible for test case generation and prioritization problems. This algorithm suggests a critical use of metahurestic approach in the field of software testing. A part from ACO, Particle Swarm Optimization (PSO), Artificial Bee Colony Optimization (ABC), Simulated Annealing and Genetic Algorithm are few other metahurestic approaches on which research may be carried out to exploit natural intelligence of species and to solve NP problems in software testing.

References

Ayari K, Bouktif S, Antoniol G. 2007. Automatic mutation test input data generation via ant colony. Genetic and Evolutionary Computation Conference. 1074-1081, London, UK

Beizer B. 1990. Software Testing Techniques (2nd ed). Van Nostrand Reinhold Co., New York, NY, USA

Binkley D. 1992. Using semantic differencing to reduce the cost of regression testing. Proceedings of the IEEE Conference on Software Maintenance. 41-50

Boujarwah AS, Saleh K. 1997. Compiler test case generation methods: a survey and assessment. Information and Software Technology, 39(9): 617-625

Brottier E, Fleurey F, Steel J, et al. 2006. Metamodel-based test generation for model transformations: an algorithm and a tool. 17th International Symposium on Software Reliability Engineering.

Chilenski JJ, Miller SP. 1994. Applicability of modified condition/decision coverage to software testing. Software Engineering Journal, 9: 193-200

Do H, Rothermel G, Kinneer A. 2006. Prioritizing Junit test cases: An empirical assessment and cost-benefits analysis. Empirical Software Engineering: An International Journal, 11: 33-70

Do H, Rothermel G. 2006. On the use of mutation faults in empirical assessments of test case prioritization techniques. IEEE Transactions on Software Engineering, 32: 733-752

Dorigo M, Maniezzo V, Colorni A. 1996. Ant system: optimization by a colony of cooperating agents. IEEE Transactions on Systems, Man, and Cybernetics, Part B, 26(1): 29-41

Dorigo M, Stutzle T. 2005. Ant Colony Optimization. MIT press, USA

Frankl PG, Weyuker EJ. 2000. Testing software to detect and reduce risk. Journal of Systems and Software, 53(3): 275-286

Harman M. 2007. The current state and future of search based software engineering. International Conference on Software Engineering. Future of Software Engineering. 342-357, IEEE Computer Society Press, Washington, DC, USA

Huang YC, Peng KL, Huang CY. 2012. A history-based cost-cognizant test case prioritization technique in regression testing. Journal of Systems and Software, 85(3): 626-637

Kim JM, Porter A, Rothermel G. 2000. An empirical study of regression test application frequency. 22nd International Conference on Software Engineering. 126-135

Korel B, Laski J. 1991. Algorithmic software fault localization. Annual Hawaii International Conference on System Sciences. 246-252, Hawaii , USA

Lammermann F, Baresel A, Wegener J. 2008. Evaluating evolutionary testability for structure oriented testing with software measurements. Applied Soft Computing, 8(2): 1018-1028

Leon D, Podgurski A. 2003. A comparison of coverage-based and distribution-based techniques for filtering and prioritizing test cases. 14th Proceedings of International Symposium on Software Reliability Engineering. 442-453, Denver, USA

Leung HKN, White L. 1991. A cost model to compare regression test strategies. Proceedings of the Conference on Software Maintenance. 201-208

Li HZ, Peng LAM. 2005. An ant colony optimization approach to test sequence generation for state based software testing. Proceedings of the Fifth International Conference on Quality Software (QSIC'05). 255-264

Liu H, Kuan Tan HB. 2009. Covering code behavior on input validation in functional testing. Information and Software Technology, 51(2): 546-553

Liu S, Chen Y. 2008. A relation-based method combining functional and structural testing for test case generation. Journal of Systems and Software, 81(2): 234-248

Lin YD, Chou CH, Lai YC, et al. 2012. Test coverage optimization for large code problems. Journal of Systems and Software, 85(1): 16-27

Mayers GJ. 1997. The Art of Software Testing, John Wiley and Sons, USA

McMinn P. 2004. Search-Based Software Test Data Generation: A Survey. Software Testing, Verification and Reliability, 14: 212-223

Mitra P, Chatterjee S, Ali N. 2011. Graphical analysis of MC/DC using automated software testing. Electronics Computer Technology, 3: 145-149

Mohan V, Jeya M. 2008. Intelligent tester-test sequence optimization framework using multi-agents. Journal of Computers, 3(6)

Nidhra S, Dondeti J. 2012. Black box and white box testing techniques –a literature review. International Journal of Embedded Systems and Applications, 2(2): 29-50

Rothermel G, Harrold MJ, von Ronne J, et al. 2002. Empirical studies of test-suite reduction. Journal of Software Testing, Verification and Reliability, 12: 4

Pedrycz W, Peters JF. 1998. Computational Intelligence in Software Engineering, World Scientific, Singapore

Saglietti F, Oster N, Pinte F. 2008. White and grey-box verification and validation approaches for safety- and security-critical software systems. Information Security Technical Report, 13(1): 10-16

Sharma M, Chandra BS. 2010. Automatic generation of test suites from decision table – theory and implementation. Software Engineering Advances (ICSEA)-Fifth International Conference. 459-464

Singh Y. 2012. Software Testing (2012 ed). 132-136, Cambridge University Press, USA

Walcott KR, Soffa ML, Kapfhammer GM, et al. 2006. Time-aware test suite prioritization. Proceedings of the International Symposium on Software Testing and Analysis. 1-12, ACM Press, USA

Zhang ZL, Mei LX. 2010. An improved method of acquiring basis path for software testing. Computer Science and Education, 2010 5th International Conference on. Date of Conference. 1891-1894