Test Case Prioritization for Regression Testing Using Antcolony Optimization

Mr. Mehul B Patel
M.Tech in
Computer Engineering

Abstract— Regression testing is a crucial and often costly software maintenance activity. In order to regain confidence in correctness of the system whenever modifications are made, we retest the software using existing test suite. But regression test suites are often too large to re-execute in the given time and cost constraints and thus, it becomes essential to prioritize the tests in order to cover maximum faults in minimum time. In this paper, ant colony optimization is used, which is a new way to solve time constraint prioritization problem. This paper presents the regression test prioritization technique to reorder test suites in time constraint environment along with an algorithm that implements the technique. We performed an experiment to evaluate the effectiveness of the proposed algorithm and compared it with other techniques using APFD metrics.

Keyword: - Regression Testing, Prioritization, Ant Colony Optimization

I. INTRODUCTION

After modifications are made in any software, we need to retest the software using existing test suite so that we regain the confidence in correctness of our system. This is called Regression testing. Regression test suites being too large to re-execute in the given time and cost constraints are reduced or re-ordered. This can be achieved by using one or more of the three techniques, Test case selection, minimization or prioritization. Test Case Prioritization is the reordering of the test suite according to an appropriate criterion like code, branch, condition and fault coverage etc. [1]. We can also select a subset of the original test suite on the basis of some criteria, often called as Regression Test Selection [2]. Or using Test case minimization we can identify and remove the redundant test cases [3]. Considering the cost of executing a test case, many cost-aware prioritization techniques have been proposed [4, 5,

Taking time as cost, Time-Aware test suite prioritization was proposed by Walcott [4]. It uses execution time of the test cases as a parameter for test case prioritization in addition to Fault Executing Potential (FEP) criteria. Execution time acts as the cost of executing the test case. Prioritization of test cases is then done according to maximum FEP and minimum cost of execution. Time constrained test case prioritization problem has been reduced to zero/one knapsack problem which is NP-complete [4]. Thus, techniques that solve combinatorial optimization problems can be applied to time constrained prioritization of test cases.

Many techniques have been used for selecting and prioritization according to one or more of the chosen criteria(s). Ant Colony Optimization (ACO) is a technique that was used by Singh et al. [7] for solving Time-Constrained Test Case Selection and Prioritization problem using Fault Exposing Potential (FEP) criteria. ACO is a nature inspired technique proposed by Dorigo et al [8] for solving combinatorial optimization problems. Recently many nature inspired algorithms are being applied to solve optimization problems. ACO is an approach based on the real life of ants, precisely on their food source searching process as described in later sections of the paper.

In this paper, The results shown in the paper provides motivation for implementing the algorithm and automating the technique. This algorithm has been used as the basis of this paper. In this paper we present a tool called TCPACO for the same and show results for the execution of the tool on the same example as used by Singh [3]. The outcome of the execution provides near optimum results and further motivates to test the tool on various larger examples to confirm the generality of its achievements.

II. RELATED WORK

ACO is a meta heuristic approach introduced in [4]. It has been successfully used to solve many NP hard optimization problems. Artificial ants have now been successfully applied on a considerable number of applications leading to world class performances for problems like vehicle routing, quadratic assignment, scheduling, sequential ordering, routing in Internet-like networks and more [22, 26, 46, 47,67, 85]

Rothermel [11] has addressed the issues related to prioritization. Prioritization for large software development environments was described by Rothermel. Prioritization of test cases based on historical execution of test data has been proposed by Kim [13]. Also empirical study was performed by Li [14] using various greedy algorithms. Time-aware regression test prioritization has also been proposed [2] where testing is performed within a fixed period of time.

III. ANT COLONY OPTIMIZATION

Ant colony optimization technique is a set of instructions based on search algorithms of artificial intelligence for optimal solutions; here the iconic member is ANT System, as proposed by Colorni, Dorigo and Maniezzo [15, 16, 17]. Ants are blind and small in size and still are able to find the shortest route to their food source. They make the use of antennas and pheromone liquid to be in touch with each other. ACO inspired from the behavior of live ants, are capable of synchronization with searching solutions for local problem by maintaining array list to maintaining previous information gathered by each ant.

Moreover, [18] ACO deals with two important processes, namely: Pheromone deposition and trail pheromone evaporation. Pheromone deposition is the phenomenon of ants adding the pheromone on all paths they follow. Pheromone trail evaporation means decreasing the amount of pheromone deposited on every path with respect to time. Updating the trail is performed when ants either complete their search or get the shortest path to reach the food source. Each combinatorial problem defines its own updating criteria depending on its own local search and global search respectively.

Artificial ants leave a virtual trail accumulated on the path segment they follow. The path for each ant is selected on the basis of the amount of "pheromone trail" present on the possible paths starting from the current node of the ant. In case of equal or no pheromone on adjacent paths, ants

The basic steps for the ACO technique applied to test case selection & prioritization are shown in the form of flow chart in Fig.2. Initially ants start from the same test case randomly choose the path. Pheromone trail on a path increases the probability of the path being followed. Ant then reaches the next node and again does the path selection process as described above. This process continues till the ant reaches the starting node. This finished tour gives the solution for shortest or best path which can then be analyzed for optimality.

IV. TEST SUITE SELECTION & PRIORITIZATION USING ACO

The proposed test case prioritization technique using Ant Colony Optimization within a time restricted framework [3] is implemented and evaluated. The technique uses the fault detection and execution time information of the regression test suite as an input. In the proposed algorithm, execution time acts as cost of executing the test case. Prioritization is done in order to achieve total fault detection and minimum cost of execution. We abbreviate the technique as TCPACO.

The basic block diagram for the TCPACO (Test Case Prioritization using Ant Colony Optimization) system is shown in Fig.1. The inputs to the system include details of the test suite i.e., the test cases along with the faults covered by them and their execution time. These inputs are generally tabulated and are to be entered by the tester. The User of the TCPACO tool needs only to enter the time constraint details at the run time. The output then produced has path details for each iteration, pheromone details, best path details and the final selected & prioritized test suite

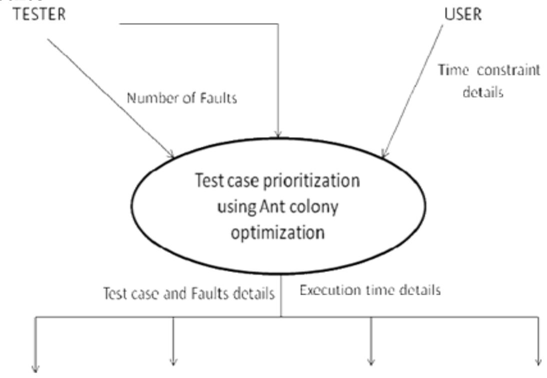

TESTER
USER
Number of Faults
Time constraint details
Test case prioritization using Ant colony optimization
Test case and Faults details
Execution time details

Fig. 1: Block diagram for TCPACO System

number as the number of the ant and the current test case is stored in a set 'S'. Now in the next step we determine probabilistically (based on the amount of pheromone and randomly) which test case is the next to be visited by the ant. Moving on to the selected test case and add it in the set

'S'. Since, the aim is to cover all the faults, thus it is checked here whether or not all faults have been covered. If not, then again determine the next node to be visited in a similar manner. If yes, then record the execution time for the complete path of each ant and clear the set 'S'. Now determine the best path in this iteration and update the pheromone on this path. Since the next aim is to achieve prioritization within the time constraint, thus it is checked whether 'TC' has been reached or not? If yes, then stop the execution and end, else, repeat the same algorithm for next iteration.

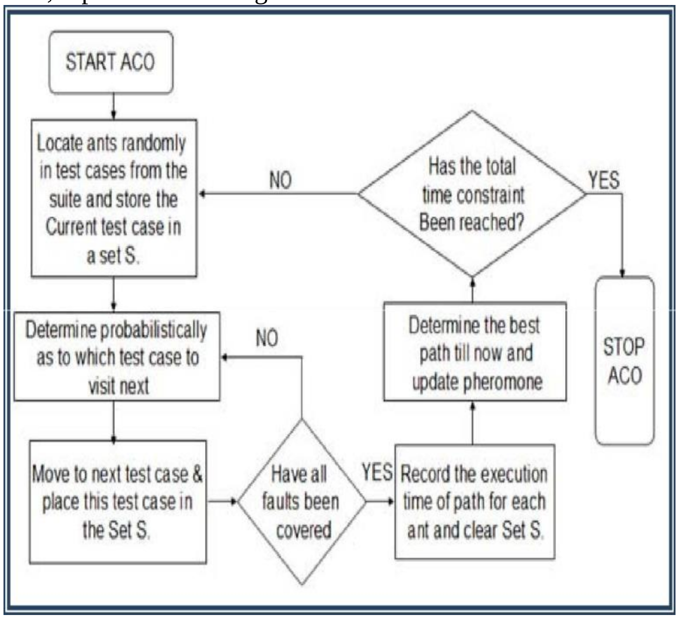

START ACO
Locate ants randomly in test cases from the suite and store the Current test case in a set S.
NO
Has the total time constraint Been reached?
YES
Determine probabilistically as to which test case to visit next
NO
Determine the best path till now and update pheromone
STOP ACO
Move to next test case & place this test case in the Set S.
Have all faults been covered
YES
Record the execution time of path for each ant and clear Set S.

Fig. 2: Flowchart for System

A. Problem of Selection and Prioritization of Test Cases

The For a given test suite, the problem of selection and prioritization of test cases can be stated as follows:

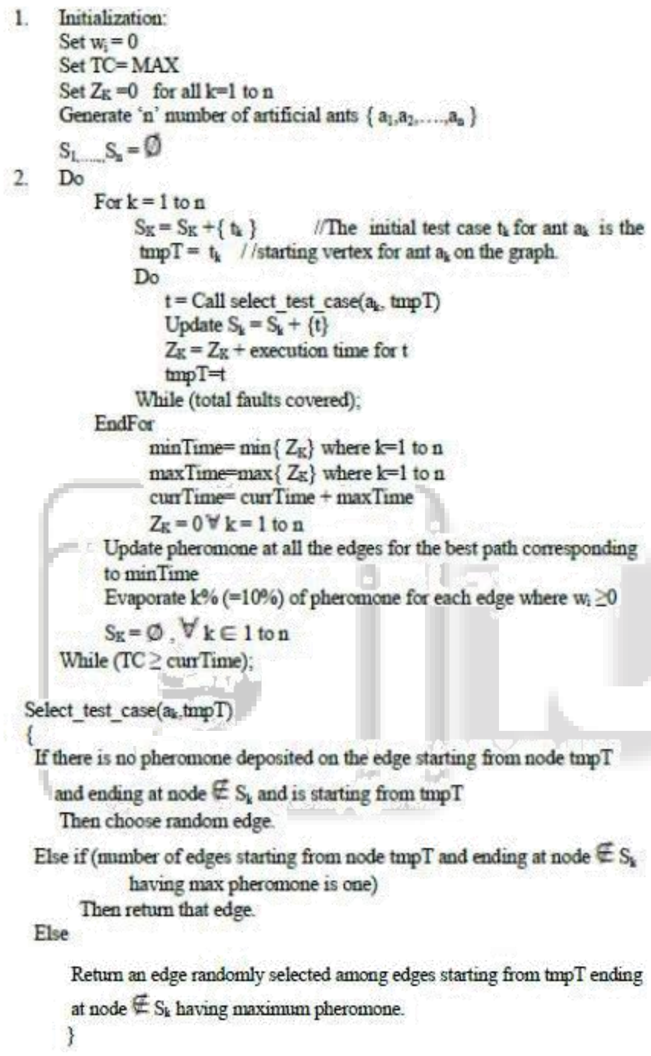
Given the original set 'T' (Test Suite) of 'n' test cases, find subset 'S', which consists of 'm' test cases (m<n, S T), such that the test cases are selected and prioritized on the basis of maximum fault coverage capacity of the test cases.

B. Assumption

1. Original test suite, $T=\{t_1,t_2\ldots\ldots t_n\}$
2. Set of all faults, $F=\{ f_1,f_2\ldots\ldots f_x \}$
3. Each test case $\{t_1,t_2\ldots\ldots t_n\}$ in the original test suite covers some or all the faults from 'F'. '
4. ' i' , is the time elapsed for each ant as it moves from one node (test case) to other. TC, is the total time constraint put on the selection and prioritization problem i.e., total time constraint for evaluating complete path of an ant. It is set to constant MAX.
5. Number of artificial ants to search through the test case space is n (number of test cases).
6. For each ant 'j', a list that contains the selected test cases (and thus faults covered on the current path) is represented as $S_j = \{ s_1,s_2,\ldots..,s_m \}$.
7. w_i, is the weight of each edge 'i', which is assumed to be the amount of pheromone deposited on the edge.
8. Assume pheromone deposition rate to be +1 or 100% for each ant that has crossed the edge on a best path.
9. Assume pheromone evaporation rate to be k% of w_i(current weight of i^th^ edge) to be reduced for each edge after each iteration of the loop. (k = 10% for our algorithm).

The problem can be represented in the form of an undirected graph G(V,E) where V is the set of vertices and E is the set of edges in the graph. test cases are represented by the vertices in the graph. ‘w_i’ is the weight of ‘i^{th}’ edge in the graph. It represents the pheromone trail associated with the edge e. which reflects the amount of fault coverage ‘f_i ‘ on the chosen path within time constraint, ‘TC’. Initially it is set to zero for all edges.

C. *Proposed Algorithm*



```
1.  Initialization:
    Set $w_i = 0$
    Set TC= MAX
    Set $Z_K = 0$  for all k=1 to n
    Generate ‘n’ number of artificial ants { $a_1, a_2, \ldots, a_n$ }
    $S_1 \ldots S_n = \emptyset$
2.  Do
        For k = 1 to n
            $S_K = S_K + \{ t_k \}$        //The initial test case $t_k$ for ant $a_k$ is the
            tmpT = $t_k$    //starting vertex for ant $a_k$ on the graph.
            Do
                t = Call select_test_case($a_k$, tmpT)
                Update $S_k = S_k + \{t\}$
                $Z_K = Z_K$ + execution time for t
                tmpT=t
            While (total faults covered);
        EndFor
            minTime= min{ $Z_K$} where k=1 to n
            maxTime=max{ $Z_K$} where k=1 to n
            currTime= currTime + maxTime
            $Z_K = 0 \ \forall\ k = 1$ to n
        Update pheromone at all the edges for the best path corresponding
        to minTime
        Evaporate k% (=10%) of pheromone for each edge where $w_i \geq 0$
        $S_K = \emptyset, \forall\ k \in 1$ to n
    While (TC $\geq$ currTime);

Select_test_case($a_k$,tmpT)
{
 If there is no pheromone deposited on the edge starting from node tmpT
   and ending at node $\notin S_k$ and is starting from tmpT
   Then choose random edge.
 Else if (number of edges starting from node tmpT and ending at node $\notin S_k$
            having max pheromone is one)
     Then return that edge.
 Else
     Return an edge randomly selected among edges starting from tmpT ending
     at node $\notin S_k$ having maximum pheromone.
 }
```

Fig. 3: TCPACO Algorithm for test suite selection and prioritization

D. *Complexity of the TCPACO Algorithm*

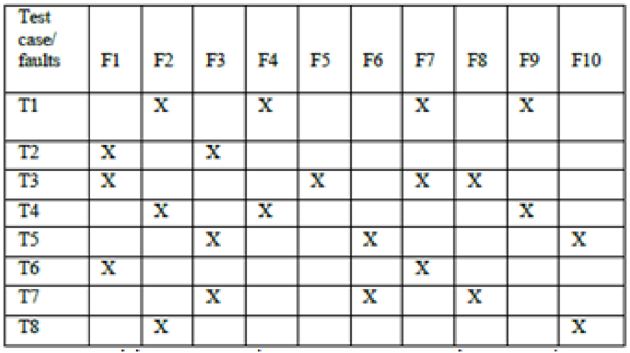
Generation of artificial ants in Step 1 is an O(N) operation. The innermost loop (Do-while) calls select_test_cases() till all the faults are covered. This can repeat for maximum N times as there are at most N test cases in the test suite T. Thus, in the best case only 1 call to the select_test_case() needs to be made, while in the worst case, N calls will be made to select_test_case(). The second nested loop (for loop) clearly repeats for N iterations. The outermost Do-while loop repeats until the execution time of selected test cases for every iteration increases beyond the user defined constraint Tc, which is constant and independent of the size of the test suite. Hence, the overall complexity of the ACO algorithm is $O(TC.N^2)$ for worst case, which is equivalent to $O(N^2)$.

Select_test_case() itself takes constant time to execute and is independent of the size of the input test suite (N) or input time constraint (Tc). Thus, the Best case complexity for the ACO algorithm for test suite selection and prioritization comes out to be O(N), as the function select_test_case() is called only once to cover all the faults.

V. EXAMPLE

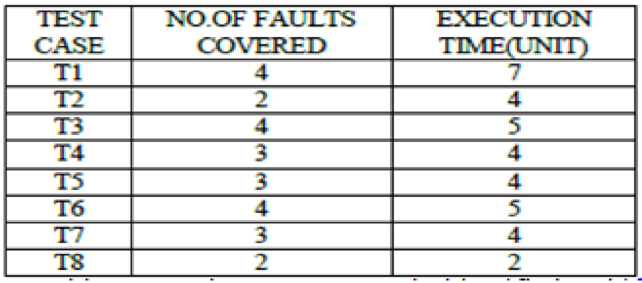
Consider a test suite with 8 test cases in it, covering a total of 10 faults.Test prioritization schemes typically create a single reordering of the test suite that can be executed after many subsequent changes to the program under test. Reordering of test suite can be more effective at finding faults if testing must be terminated earlier [8]. In this section we present the execution of our algorithm in time based prioritization.

The regression test suite T contains eight test cases with the initial ordering {T1, T2, T3, T4, T5. T6, T7, T8} as described in Table 1. This example assumes a priori knowledge of the faults detected by T in the program P. As shown in Table 2, test case T1 can find four faults, {f2, f4, f7, f9} in seven minutes, T2, finds two fault, {f1,f3} in four minutes, and T3 finds four faults, { f1, f5, f7, f8} in five minutes. Test cases T4 and T5 find three faults in four minutes, { f2, f4, f9} and { f3,f6, f10}. Test cases T6 and T8 find two faults in five and two minutes, {f1, f7} and {f2, f10}. Test case T7 finds three faults in four minutes, {f3, f6, f8} respectively. Each test case covers the faults as shown in the table below:



Test case/ faults	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10
T1		X		X			X		X	
T2	X		X							
T3	X				X		X	X		
T4		X		X					X	
T5			X			X				X
T6	X						X			
T7			X			X		X		
T8		X								X

Table 1: Sample Test cases, Faults covered



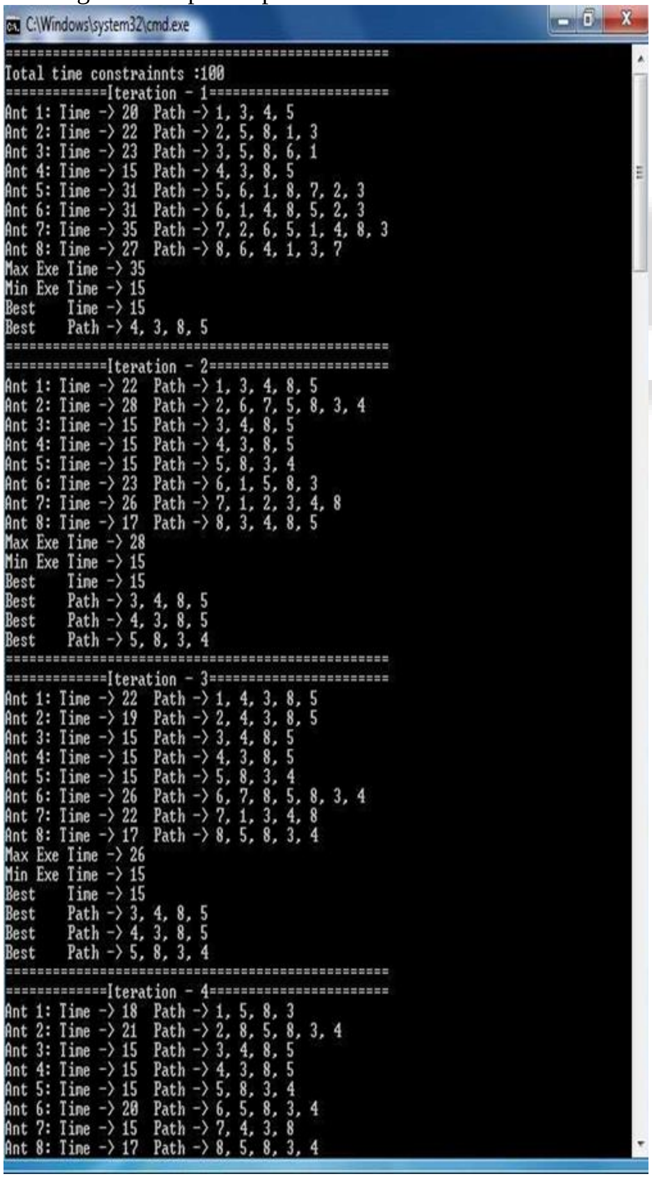
TEST CASE	NO.OF FAULTS COVERED	EXECUTION TIME(UNIT)
T1	4	7
T2	2	4
T3	4	5
T4	3	4
T5	3	4
T6	4	5
T7	3	4
T8	2	2

Table 2: Sample Test cases, Faults identified and its Execution time.

Total time limit assumed to be 100 time units. This is the stopping criteria for execution of algorithm. The paths traversed by various.

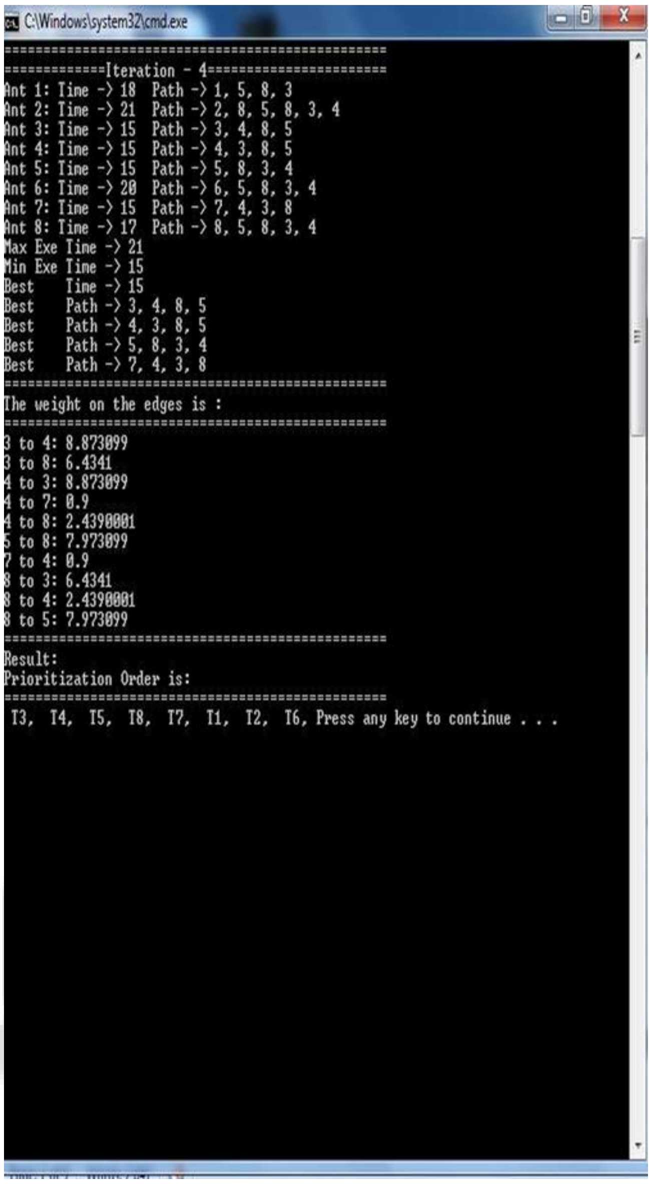
VI. IMPLEMENTING THE TECHNIQUE

The algorithm has been coded as “TCPACO” which is a JAVA code, implemented on a Pentium Core 2 Duo PC at 2.66GHz (1 Gb RAM). The tool is made up of 10 modules having 5 global functions, 13 global variables, 2 structures and one user defined class. Some of the screenshots for output screens of the tool are shown in Fig 3 and 4. In order to evaluate the efficacy of the TCPACO tool for test case selection and prioritization within a time constrained environment, the tool was applied on the same example as taken in table 1. The TCPACO was run 4 iteration on the example with constant time constraint, TC=100 time units. The input tothe TCPACO assumes a priori knowledge of the faults detected and the execution time of all test cases. The same is tabulated in Table 1. In this table, for each run the best path and its execution time of all iterations are reported. Also the final weight on the edges and the path found in that iteration is shown. The optimal path was found by TCPACO is {T3, T4, T5, T8, T7. T1, T2, T6}. Though different paths were explored by artificial ants in all he iteration, still they could converge to the optimal path



```
C:\Windows\system32\cmd.exe
================================================
Total time constrainnts :100
============Iteration - 1======================
Ant 1: Time -> 20  Path -> 1, 3, 4, 5
Ant 2: Time -> 22  Path -> 2, 5, 8, 1, 3
Ant 3: Time -> 23  Path -> 3, 5, 8, 6, 1
Ant 4: Time -> 15  Path -> 4, 3, 8, 5
Ant 5: Time -> 31  Path -> 5, 6, 1, 8, 7, 2, 3
Ant 6: Time -> 31  Path -> 6, 1, 4, 8, 5, 2, 3
Ant 7: Time -> 35  Path -> 7, 2, 6, 5, 1, 4, 8, 3
Ant 8: Time -> 27  Path -> 8, 6, 4, 1, 3, 7
Max Exe Time -> 35
Min Exe Time -> 15
Best    Time -> 15
Best    Path -> 4, 3, 8, 5
================================================
============Iteration - 2======================
Ant 1: Time -> 22  Path -> 1, 3, 4, 8, 5
Ant 2: Time -> 28  Path -> 2, 6, 7, 5, 8, 3, 4
Ant 3: Time -> 15  Path -> 3, 4, 8, 5
Ant 4: Time -> 15  Path -> 4, 3, 8, 5
Ant 5: Time -> 15  Path -> 5, 8, 3, 4
Ant 6: Time -> 23  Path -> 6, 1, 5, 8, 3
Ant 7: Time -> 26  Path -> 7, 1, 2, 3, 4, 8
Ant 8: Time -> 17  Path -> 8, 3, 4, 8, 5
Max Exe Time -> 28
Min Exe Time -> 15
Best    Time -> 15
Best    Path -> 3, 4, 8, 5
Best    Path -> 4, 3, 8, 5
Best    Path -> 5, 8, 3, 4
================================================
============Iteration - 3======================
Ant 1: Time -> 22  Path -> 1, 4, 3, 8, 5
Ant 2: Time -> 19  Path -> 2, 4, 3, 8, 5
Ant 3: Time -> 15  Path -> 3, 4, 8, 5
Ant 4: Time -> 15  Path -> 4, 3, 8, 5
Ant 5: Time -> 15  Path -> 5, 8, 3, 4
Ant 6: Time -> 26  Path -> 6, 7, 8, 5, 8, 3, 4
Ant 7: Time -> 22  Path -> 7, 1, 3, 4, 8
Ant 8: Time -> 17  Path -> 8, 5, 8, 3, 4
Max Exe Time -> 26
Min Exe Time -> 15
Best    Time -> 15
Best    Path -> 3, 4, 8, 5
Best    Path -> 4, 3, 8, 5
Best    Path -> 5, 8, 3, 4
================================================
============Iteration - 4======================
Ant 1: Time -> 18  Path -> 1, 5, 8, 3
Ant 2: Time -> 21  Path -> 2, 8, 5, 8, 3, 4
Ant 3: Time -> 15  Path -> 3, 4, 8, 5
Ant 4: Time -> 15  Path -> 4, 3, 8, 5
Ant 5: Time -> 15  Path -> 5, 8, 3, 4
Ant 6: Time -> 20  Path -> 6, 5, 8, 3, 4
Ant 7: Time -> 15  Path -> 7, 4, 3, 8
Ant 8: Time -> 17  Path -> 8, 5, 8, 3, 4
```

Fig. 3: Output screen 1 for the Example

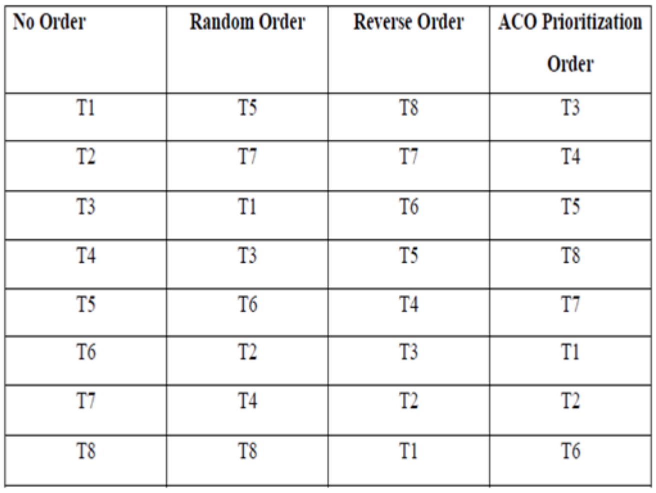


```
C:\Windows\system32\cmd.exe
================================================
============Iteration - 4======================
Ant 1: Time -> 18  Path -> 1, 5, 8, 3
Ant 2: Time -> 21  Path -> 2, 8, 5, 8, 3, 4
Ant 3: Time -> 15  Path -> 3, 4, 8, 5
Ant 4: Time -> 15  Path -> 4, 3, 8, 5
Ant 5: Time -> 15  Path -> 5, 8, 3, 4
Ant 6: Time -> 20  Path -> 6, 5, 8, 3, 4
Ant 7: Time -> 15  Path -> 7, 4, 3, 8
Ant 8: Time -> 17  Path -> 8, 5, 8, 3, 4
Max Exe Time -> 21
Min Exe Time -> 15
Best    Time -> 15
Best    Path -> 3, 4, 8, 5
Best    Path -> 4, 3, 8, 5
Best    Path -> 5, 8, 3, 4
Best    Path -> 7, 4, 3, 8
================================================
The weight on the edges is :
================================================
3 to 4: 8.873099
3 to 8: 6.4341
4 to 3: 8.873099
4 to 7: 0.9
4 to 8: 2.4390001
5 to 8: 7.973099
7 to 4: 0.9
8 to 3: 6.4341
8 to 4: 2.4390001
8 to 5: 7.973099
================================================
Result:
Prioritization Order is:
================================================
 T3,  T4,  T5,  T8,  T7,  T1,  T2,  T6, Press any key to continue . . .
```

Fig. 4: Output screen 2 for the Example

VII. RESULT ANALYSIS

The orderings with respect to these approaches for exampl are listed in Table 3. These approaches are compared by calculating Average Percentage of Faults Detected (APFD) using the same example



No Order	Random Order	Reverse Order	ACO Prioritization Order
T1	T5	T8	T3
T2	T7	T7	T4
T3	T1	T6	T5
T4	T3	T5	T8
T5	T6	T4	T7
T6	T2	T3	T1
T7	T4	T2	T2
T8	T8	T1	T6

Table 3: Order of test cases for various prioritization approaches based on Example.

So, APFD result of deferent prioritization approaches are,

Prioritization Approaches	APFD Value
No Order	49 %
Random Order	40 %
Reverse Order	51 %
ACO Order	74 %

Table 4: APFD Result for Example

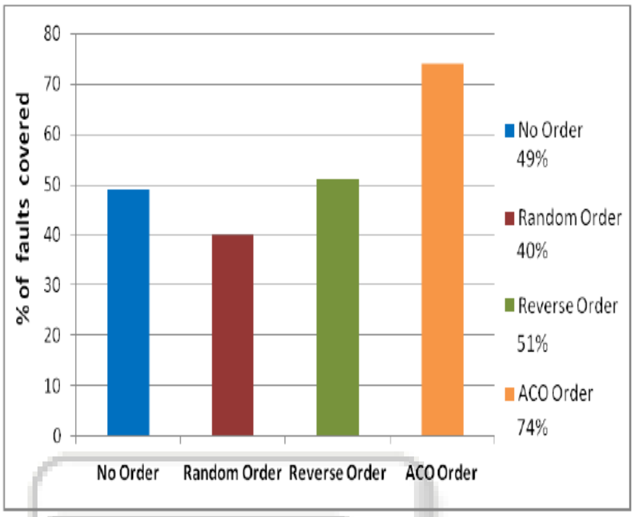


Fig. 5: Average Percentage of Faults Detected (APFD) for Example

The results are shown in Table 4. It shows that the prioritization achieved using ACO that is nearest to optimal solution. The technique is superior to other approaches also as is clearly shown with the percentage of fault coverage achieved.

VIII. Conclusion

In comparison of ACO with other techniques for the example by calculating APFD (Average Percentage of Faults Detected) for each technique. Application of the ACO technique to the problem of test case selection and prioritization leads to solutions which are optimal or near optimal.The obtained results also encourage the use of ACO in time constraint test case selection and prioritization. ACO is strong & robust as it involves positive feedback and parallel computations and hence, it can lead to better solutions in optimum time.

In future the proposed system with Test case prioritization using Ant colony optimization algorithm will be apply for larger and complex systems.

References

[1] G.Rothermel, R.J.Untch, and C.Chu, "Prioritizing test cases for regression testing", IEEE Transaction on Software. Eng., vol. 27(10), pp. 929-948, 2001.

[2] T.L.Graves, M.Harrold, M.J.Kim, A.Porter and G.Rothermel, "An empirical study of regression test selection techniques", ACM Transactions on Software Engineering and Methodology, vol. 10(2), 2001.

[3] G.Rothermel, M.Harrold, J.Ronne, C.Hong, "Empirical studies of test suite reduction", Software Testing, Verification and Reliability, vol. 4(2), pp. 19-249, December 2002.

[4] K. R. Walcott, M. L. Soffa, G. M. Kapfhammer, and R. S. Roos, ACM/SIGSOFT International Symposium on Software Testing & Analysis (ISSTA), Portland Maine, USA, pp. 1–11, 2006Tavel, P. 2007 Modeling and Simulation Design. AK Peters Ltd.

[5] Empirical Validation of Variable based Test Case Prioritization/Selection Technique.Yogesh Singh, Arvinder Kaur, Bharti Suri, JDCTA , 116-123 (2009).

[6] Ant System: Optimization by a colony of cooperating agents.Dorigo, M., Maniezzo,V. and Colorni, A. , IEEE Trannsactions on Systems, Man and Cybernetics,1996.

[7] Regression Test Suite Prioritization using Genetic Algorithms. Krishnamoorthi, R., Sahaaya, S.A. and Mary, A., International Journal of Hybrid Information Tech,2009.

[8] Regression Testing Minimisation, Selection and Prioritisation :A Survey,S.Yoo, M. Harman King's College,London,Centre for Research on Evolution, Search & Testing, Strand, London, WC2R 2LS, UK,2007.

[9] Dorigo,M.http://iridia.ulb.ac.be/mdorigo/ACO/ACO.hml

[10] Gomez,O. and Baren,B. (2005): Omicron ACO. A New Ant Colony Optimization Algorithm. Clei electronic journal, 8(1), paper 5.

[11] Graves, T. L., Harrold, M. J., Kim, M.J., Porter, A.and Rothermel, G. (2001): An empirical study of regression test selection techniques. ACM Trans. On Softw. Eng. and Meth. 10(2).

[12] Li, H. and Peng Lam, C. (2005): Software Test Data Generation Using Ant Colony Optimization. pp: 1.