TEST CASE SELECTION & PRIORITIZATION USING ANT COLONY OPTIMIZATION

Bharti Suri
Computer Science Department
Assistant Professor, USIT, GGSIPU
New Delhi, India
bhartisuri@gmail.com

Shweta Singhal
Information Technology Department
Research Student, USIT, GGSIPU
New Delhi, India
shweta_niec@yahoo.co.in

***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 we use test case selection and prioritization techniques. Ant Colony Optimization is a meta-heuristic approach that has been applied for time-constraint test case selection and prioritization. We performed an experiment to evaluate the effectiveness of the proposed algorithm and compared it with other techniques using APFD metrics. Our results highlight close proximity to optimal solution and show the time reduction achieved by this technique.**

***Keywords* - *R*egression Testing, Test case selection, Test case prioritization, Ant colony optimization.**

I. INTRODUCTION

To cope up with the ever changing and demanding IT environment, we need to modify our software in its maintenance phase. After modifications are made, 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, 6]. 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, time and space complexity of the proposed algorithm [7] has been computed which provided further motivation to implement the technique. The work of implementation of the proposed algorithm and its analysis for forty runs of the tool on two sample programs has been shown. The analysis demonstrates the usefulness and effectiveness of using ACO technique for test suite selection and prioritization. Analysis of the technique highlights assurance of 100% fault coverage whether ordering is optimal or not. A comparison has also be done against the no order, random order, reverse order and optimal order prioritization shown using APFD metric. This also proves the closeness of selection and prioritization using ACO with the optimal ordering..

II. ANT COLONY OPTIMIZATION

It is an astonishing discovery of entomologists that the real power of ants resides in their colony brain [9]. Ants are blind and they communicate within the colony by the use of a chemical substance called pheromone. As ants move from their nest to food source and vice-versa, they deposit the pheromone trail on that path. This pheromone trail is then smelled by other ants which tend to follow the path with maximum pheromone trail. Point of our interest is that, an ant reaches the food source and comes back to its nest. Thus, the ant on the shortest path is also the earliest to return, depositing more amount of

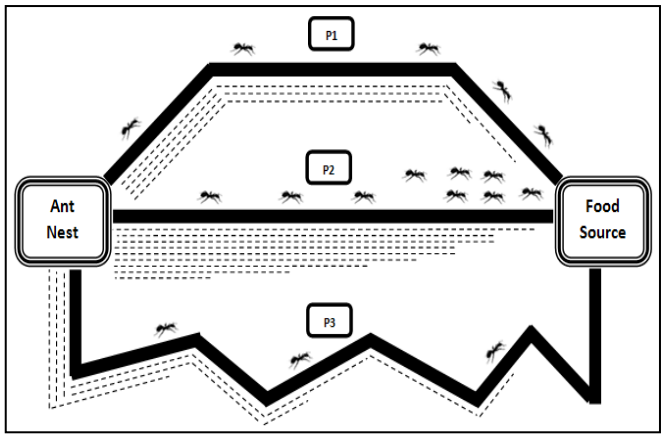

Figure 1. ACO Example. Above the path shown are the ants that follow the path and their pheromone trail is shown below the path

pheromone on that path (while going and returning to the nest). This increases the probability of other ants following this path [8]. The ants following this path will again lay more pheromone on the path. Thus even more ants tend to follow this path and continuing the process, ultimately all ants converge to the shortest or the best path.

This process of ants converging to the shortest path is illustrated in the example shown in fig. 1. There are three possible paths (P1, P2, P3) from ant nest to the food source. Initially an ant can choose any of the three paths. After collecting the food from source ants also come back to their nests. Thus the ant on the shortest path is also the first one to return to the nest, which implies that more amount of pheromone is deposited on this path. New ants starting from the nest would tend to follow this path and deposit more pheromone. Finally, it is observed that all the ants converge to the best path (P2 in the fig. 1) and this path has the maximum pheromone trail deposited. This phenomenon of finding the best shortest path by a colony of ants is known as Ant Colony Optimization.

Derived from this real behavior of ants, Dorigo proposed Ant Colony Optimization (ACO) in the year 2006[8]. This technique has already been used in solving various combinatorial optimization problems such as knapsack problem, travelling salesman problem, distributed networks, data mining, telecommunication networks, vehicle routing, test data generation [8, 10, 11, 12, 13, 14, 15, 16] etc. As proposed by Singh et al [7], ACO can be applied to time constraint test suite selection and prioritization.

III. Test Suite Selection & Prioritization using ACO

A. *Description of the Technique*

Suppose 'T' = $\{t_1, t_2 \ldots\ldots t_n\}$ is the set of all 'n' number of test cases, and 'F' = $\{ f_1, f_2 \ldots\ldots f_x \}$ be the set of all faults seeded in the test program. Some or all the faults from 'F' are covered by each test case $\{t_1, t_2 \ldots\ldots t_n\}$ in the original test suite. 'z_i' , is the time elapsed while ants cover new nodes in the graph. For evaluating a complete path, the total time constraint, TC, is taken to be MAX. The same number of artificial ants are generated as the number of test cases. 'S', which consists of 'm' test cases ($m<n$, $S \subset T$), is a subset of the original test suite and consists of the selected test cases for each ant. There are 'n' subsets for 'n' ants. Pheromone deposited is represented by 'w_i,' the weight of the ith edge in the graph. The deposition rate of pheromone is +1 or 100% for each edge crossed on the optimal path. The evaporation rate of pheromone is taken to be 10%, which is reduced from the weight of each edge, after iteration is complete [8]. The proposed algorithm is shown in Fig.2, taken from [7].

B. *Complexity of the Algorithms*

The whole effort of building the algorithm is wasted if the algorithm itself takes more time to run than to rerun all the test cases in the regression test suite. To prove the efficiency of ACO in test case selection and prioritization with respect to execution time, we computed the complexity of the algorithm as explained.

The algorithm, as shown in fig. 2 has a running time bounded by the time required to generate the artificial ants a1 to an in Step 1, plus the number of iterations for the three nested loops in Step 2. Let 'T' be the regression test suite, i.e. a set of test cases with $|T|$ number of total test cases in it. And let 'TC' be the time constraint input by the user.

```
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,....,a_n }
    S_1,.....,S_n = ∅
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 ∀ 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 ≥0
          S_K = ∅ , ∀ k ∈ 1 to n
    While (TC ≥ currTime);
Select_test_case(a_k,tmpT)
{
 If there is no pheromone deposited on the edge starting from node tmpT
   and ending at node ∉ S_k and is starting from tmpT
   Then choose random edge.
 Else if (number of edges starting from node tmpT and ending at node ∉ 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 ∉ S_k having maximum pheromone.
 }
```

Figure 2. ACO Algorithm for test suite selection and prioritization proposed by Singh et al. [7]

Generation of artificial ants in Step 1 is an O(|T|) operation [17]. The innermost loop (Do-while) calls select_test_cases() till all the faults are covered. This can repeat for maximum |T| times as there are at most |T| 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, |T| calls will be made to select_test_case(). The second nested loop (for loop) clearly repeats for |T| 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.|T|2) for worst case, which is equivalent to O(|T|2).

Select_test_case() itself takes constant time to execute and is independent of the size of the input test suite (T) 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(Tc|T|) ≈ O(|T|), as the function select_test_case() is called only once to cover all the faults. This complexity is acceptable as compared to the NP-complete problem of time aware prioritization [4].

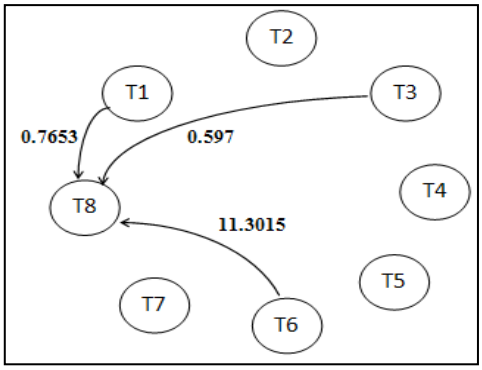

Figure 3. Directed graph for CollAdmission

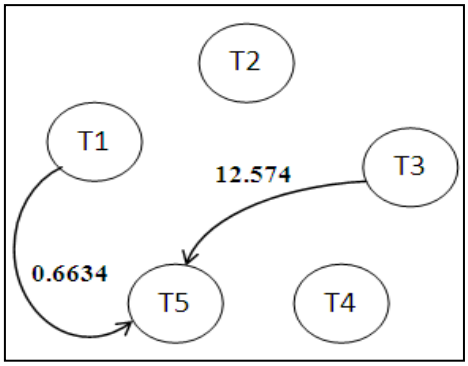

Figure 4. Directed graph for Hotel_Mgmnt

C. Graph Theory

T To implement the ACO algorithm we need to represent graphs in some mathematical form. This can be done in two standard ways: adjacency list and adjacency matrix [18]. It is preferred to use adjacency matrix representation for dense graphs [18]. Initially, there is equal probability of any set of edges being the best path, and due to the random nature of ACO, we create a fully connected graph G[V, E]; where V is the set of vertices representing the test suite 'T', and E is the set of edges, weight on which determines the amount of pheromone on that path. Initially weight on each edge is 0 as no pheromone is deposited. We chose adjacency matrix representation for our implementation, as ours is a fully connected directed graph with initial weight 0 on all the edges. Though we are interested only in the edges which would have their weight > 0 at the end of the algorithm, but due to the random nature of ACO, it could be any edge(s).

In our case, adjacency matrix is of the size N x N, where N is the number of vertices in the graph (or the number of test cases in the original regression test suite, |T|). The final graphs constructed for sample program CollAdmission's sample run 3 and program HotelMgmnt's sample run 6 are shown in fig 3 & 4 respectively. Edges with 0 weights are not shown for the sake of simplicity; otherwise it is a fully connected graph.

IV. Experimental Design

A. Programs

For our analysis we used two C++ programs. Using fault seeding technique [19, 20] five modified versions and black box test cases for two programs were generated. First program CollAdmission is a 281 LOC college program for admission of courses. Second program HotelMgmnt is a 666 LOC menu driven program where hotel database has been tried to implement using classes. Both of the programs are live and available at [21].

B. Variables

The independent variables manipulated by the experiment are:

1) Subject programs with five faults each.
2) Selection and prioritization techniques (no order, random order, reverse order, optimal, ACO)

.

For each of the programs, on each run we measured:

1) The ratio of the total execution time of the test suite to the reduced ACO execution time.
2) Whether the optimal path has been found or not.

For each program run, random nature of ACO led to different paths using which three dependent variables were computed:

1) Percentage reduction in test suite size,
2) Percentage reduction in execution time, and

3) Percentage of the number of times the best path is found.

C. Design

To build confidence in the technique ACO, we repeated forty runs for each of the programs with same test suites. Each run yielded the path found whether or not optimal. The time constraint chosen as the stopping criteria of the algorithm was taken to be 300 for all 80 runs. Time reduction, optimality of technique and the fault detection effectiveness using APFD metrics were obtained from the output data.

V. Data Analysis

For our experiment, out of the complete test pool, we randomly chose a test suite of nine and five test cases for CollAdmission & HotelMgmnt respectively. Execution time for each of the test cases was then calculated and used as an input to the ACO algorithm. The algorithm was executed 40 times each for the test programs with a constant time constraint TC=300. The results obtained are summarized in Fig.5- Fig.10.

Fig.5 & Fig.6 represent the distribution of optimal and other paths found for 40 test runs on both the programs. It shows that 82% times the optimal path was found by the ACO technique for both the programs. Also, it is depicted in the figures that the cases for which optimal path have not been found, 2nd, 3rd or 4th optimal path has been found. Though 18% times the optimal path was not found, the path found is near optimal. It leads to total fault coverage and less execution time than randomly choosing the test cases.

Fig.7 and Fig.8 depict the percentage reduction in execution time using ACO selection and prioritization as compared to no prioritization. The results are shown for 40 test runs on both programs with same set of input data. The average reduction in execution time is 82.5% and 58.17% for CollAdmission and HotelMgmnt respectively. The difference is due to the difference in size and type of the chosen programs. This leads to the result that ACO might not always lead to optimal solution. But the solution is still very fruitful as no test run gave >10% of the best possible percentage time reduction using ACO

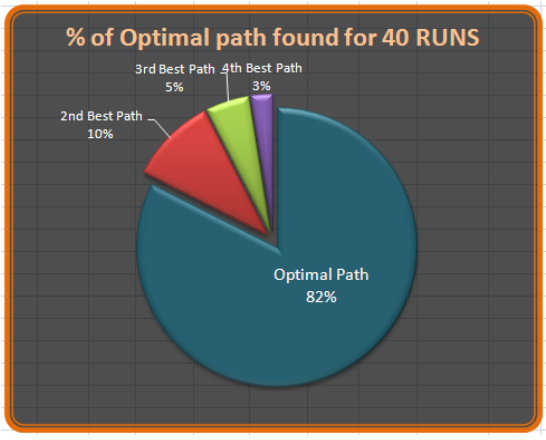

Figure 5. CollAdmission, TC=300

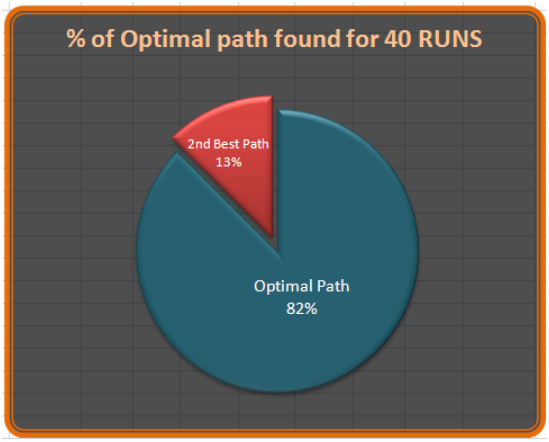

Figure 6. HotelMgmnt, TC=300

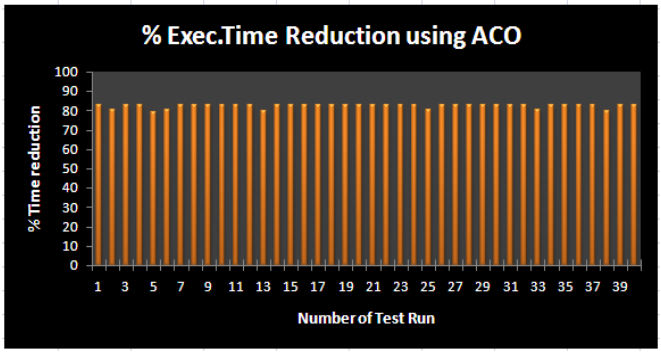

Figure 7. Execution time reduction for CollAdmission

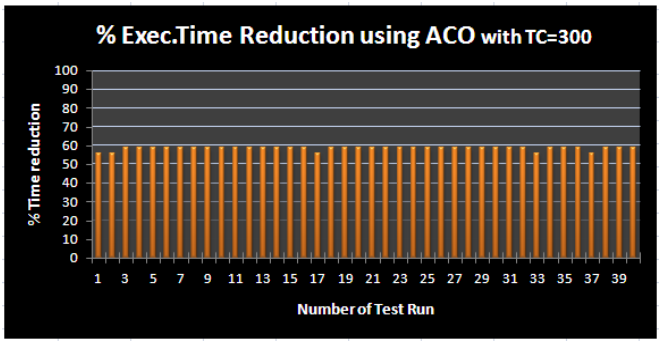

Figure 8. Execution time reduction for HotelMgmnt

VI. Comparison

The test programs mentioned in above section were compared with the following ordering: No order, Random order, Reverse order, Optimal order and ACO order of the test cases. The orderings with of these approaches are listed in Table 1 & 2 for both programs respectively. Comparison has been done by computing the Average Percentage of Faults Detected (APFD) [22, 23, 24]. The results are depicted in fig.7 & fig.8. We can infer from here that the selection and prioritization achieved using ACO lead to results that are near to the optimum ordering. The ACO technique is more

effective than other approaches in terms of the percentage of fault coverage attained.

TABLE I. PRIORITIZED ORRDERS FOR COLLADMISSION

No Order	Random Order	Reverse Order	Optimal fault cvg	ACO
T1	T4	T9	T6	T6
T2	T6	T8	T8	T8
T3	T5	T7	T1	T4
T4	T9	T6	T2	T1
T5	T3	T5	T3	T3
T6	T8	T4	T4	T2
T7	T7	T3	T5	T5
T8	T1	T2	T7	T7
T9	T2	T1	T9	T9

TABLE II. PRIORITIZED ORRDERS FOR HOTELMGMNT

No Order	Random Order	Reverse Order	Optimal Fault Coverage	ACO
T1	T4	T5	T3	T3
T2	T3	T4	T5	T5
T3	T1	T3	T1	T1
T4	T5	T2	T2	T4
T5	T2	T1	T4	T2

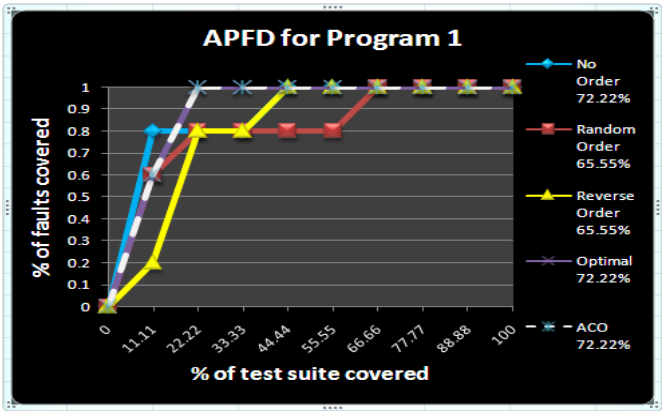

Figure 9. APFD for CollAdmission

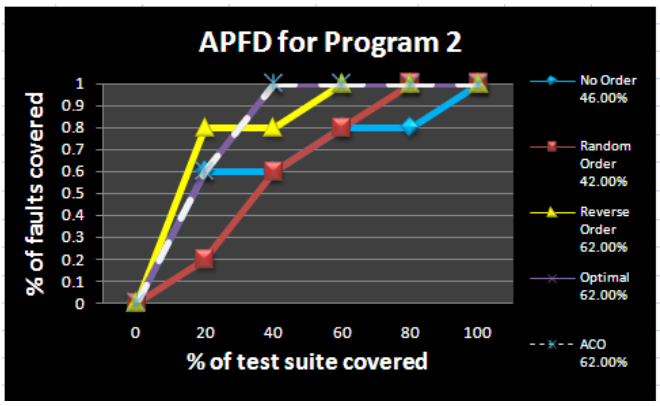

Figure 10. APFD for HotelMgmnt

VII. THREATS TO VALIDITY

Threats to construct validity are the threats related to the whether measurements conform to accuracy or not. The measurement metrics used in this paper is APFD which has already been widely accepted and used [22, 23 24] as a measure for rate of fault detection. Another threat is that the pheromone evaporation rate is taken to be 10%. It does not confirm to real life ants, but it has already been successfully used by [7, 8] and hence we use the same value for the pheromone evaporation rate.

There can be threats to internal validity. These can arise in our system from the fact that real ants also consider the direction in which the smell the food source to decide their path. In this paper only the pheromone amount deposited by other ants is considered to find the path. This approach is valid because almost all the ACO applications [8, 10, 11, 12, 13, 14, 15, 16] have applied it and achieved correct solutions.

The threats to external validity might include the use of seeded faults. As has already been reported, mutation faults can be used to represent real faults in experimental evaluations [25].

VIII. CONCLUSIONS & FUTURE SCOPE

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 complexity and results also encourage the use of ACO in time constraint test case selection and prioritization. The comparison of ACO with other reference techniques using APFD suggests that ordering obtained for ACO are nearly same as the optimal ordering. In future we wish to apply the same 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. 219-249, December 2002.

[4] K. R. Walcott, M. L. Soffa, G. M. Kapfhammer, and R. S. Roos, "Time aware test suite prioritization", Proceedings of ACM/SIGSOFT International Symposium on Software Testing & Analysis (ISSTA), Portland Maine, USA, pp. 1–11, 2006.

[5] X. Qu, M. Cohen, and G. Rothermel, "Configuration-Aware Regression Testing: An Empirical Study of Sampling and Prioritization", Proceedings of the International Synposium on Software Testing and Analysis, pp. 75-86, July 2008.

[6] S.Elbaum, G.Rothermel, S. Kanduri, A. G Malishevsky, "Selecting a Cost-Effective Test Case Prioritization Technique", Software Quality Journal, Kluwer Academic Publishers, vol. 12(3), pp. 185-210, 2004.

[7] Y.Singh, A.Kaur, B.Suri, "Test case prioritization using ant colony optimization", ACM SIGSOFT Software Engineering Notes, vol. 35(4), pp. 1-7, July 2010.

[8] M. Dorigo, K.Socha, “An Introduction to Ant Colony Optimization”, IRIDIA Technical Report Series, TR/IRIDIA/2006-010, 2006.

[9] P.E.Merloti, San Diego, “Optimization Algorithms Inspired by Biological Ants and Swarm Behavior”, State University, Artificial Intelligence Technical Report – CS550, San Diego, 2004.

[10] K.Ayari, S.Bouktif, and G.Antoniol, “Automatic Mutation Test Input Data Generation via Ant Colony”, pp. 1074, 2007.

[11] G.Caro, Di and M.Dorigo, “AntNet: Distributed stigmergetic control for communications networks”, Journal of Artificial Intelligence Research, vol. 9, pp. 317-365, 1998.

[12] M.Dorigo, V.Maniezzo, and A.Colorni, “Ant System: Optimization by a colony of cooperating agents”, IEEE Trannsactions on Systems, Man and Cybernetics, vol. B(26), pp. 29-41, 1996.

[13] H.Li, and C.Peng Lam, “Software Test Data Generation Using Ant Colony Optimization”, pp. 1, 2005.

[14] L.Li, S.Ju, and Y.Zhang, “Improved Ant Colony Optimization for the Travelling Salesman Problem”, International Conference on Intelligent Computation Technology and Automation, pp. 76, 2008.

[15] R.S.Parpinelli, H.S.Lopes, and A.A.Freitas, “Data mining with an ant colony optimization algorithm”, IEEE Transactions on Evolutionary Computation, vol. 6, pp. 321–332, 2002.

[16] P.Zhao, P.Zhao, and X.Zhang, “New Ant Colony Optimization for the Knapsack Problem”, 2006.

[17] T.H.Cormen, C.E.Leiserson, R.L.Rivest and C.Stein, “Introduction to Algorithms”, PHI Publications, 2009 edition.

[18] V.Adamchick, “Graph Theory”, book, a chapter in Concepts of Mathematics, pp. 21-127, 2005,.

[19] T. A. Budd, “Mutation Analysis of Program Test Data”, PhD thesis, Yale University, New Haven, CT, May 1980.

[20] R.A.DeMillo and A.P.Mathur, “On the use of software artifacts to evaluate the effectiveness of mutation analysis for detecting errors in production software”, in Thirteenth Minnowbrook Workshop on Software Engineering, July 1990.

[21] www.sourceforge.com .

[22] B. Pfahring, “Multi-agent search for open scheduling: adapting the Ant-Q formalism,” Technical report TR-96-09, 1996.

[23] C. Gagné, W. L. Price and M. Gravel, “Comparing an ACO algorithm with other heuristics for the single machine scheduling problem with sequence-dependent setup times,” Journal of the Operational Research Society, vol. 53, pp. 895-906, 2002.

[24] P. Toth and D. Vigo, “Models, relaxations and exact approaches for the capacitated vehicle routing problem,” Discrete Applied Mathematics, vol. 123, pp. 487-512, 2002.

[25] J.H.Andrews, L.C. Briand and Y. Labiche, “Is mutation an appropriate tool for testing experiments?”, In International Conference of Software Engineering, pp. 402-411, May 2005.