Implementing Ant Colony Optimization for Test Case Selection and Prioritization

Bharti Suri

Assistant Professor, Computer Science Department
USIT, GGSIPU
Delhi, India

Shweta Singhal

Student M.Tech (IT)
USIT, GGSIPU
Delhi, India

Abstract— Regression Testing is an inevitable and a very costly activity to be performed, often in a time and resource constrained environment. Thus we use techniques like Test Case Selection and Prioritization, to select and prioritize a subset from the complete test suite, fulfilling some chosen criteria. Ant Colony Optimization (ACO) is a technique based on the real life behavior of ants. This paper presents an implementation of an already introduced Ant Colony Optimization Algorithm for Test Case Selection and Prioritization. Graph representation and example runs explained in the paper show how the random nature of ACO helps to explore the possible paths and choose the optimal from them. Results show that ACO leads to solutions that are in close proximity with optimal solutions.

Keywords - Regression Testing, Ant Colony Optimization, Implementation, Test Case Selection, Test Case Prioritization

I. INTRODUCTION

The maintenance phase of a software product needs to go through the inevitable regression testing process. It is required to retest the existing test suite whenever any addition, deletion or modifications are made to the software. Re-running the test cases from the existing test suite to build confidence in the correctness of the modified software is referred to as regression testing. Quite often software developers encounter time and budget constraints, which makes it almost impossible to rerun all the test cases within the specified constraints. Thus we use test case minimization, selection and prioritization techniques for regression testing.

Regression test selection is a process of reducing the test suite by selecting a subset from the original test suite. Although this is a very cost effective method for regression testing but it can leave out certain important test cases from the selected subset of test cases. Regression test prioritization means scheduling the test cases in an increasing or decreasing order to meet some performance goal [1]. Various prioritization criteria may be applied to the regression test suite with the objective of meeting those criteria. Test cases can be prioritized in terms of random, optimal, statement coverage total or additional, branch coverage total or additional, failure rates, or total fault exposing potential (FEP) [1] of the test cases.

We often perform regression test prioritization in a time constrained environment where testing is done for a fixed period of time. Walcott et al [2] in 1996 gave one such technique for time-aware test case prioritization. Time-aware prioritization intelligently schedules the test suite in terms of both the execution time and potential fault detection information. Walcott et al used Genetic Algorithms to solve this problem. In the year 2010, Singh et al. [3] also proposed a time-constrained prioritization technique using Ant Colony Optimization (ACO). 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 ACO_TCSP 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 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 CASE SELECTIONAND 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 ACO_TCSP.
The basic block diagram for the ACO_TCSP (Ant Colony Optimization for Test Case Selection & Prioritization) 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 ACO_TCSP 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.

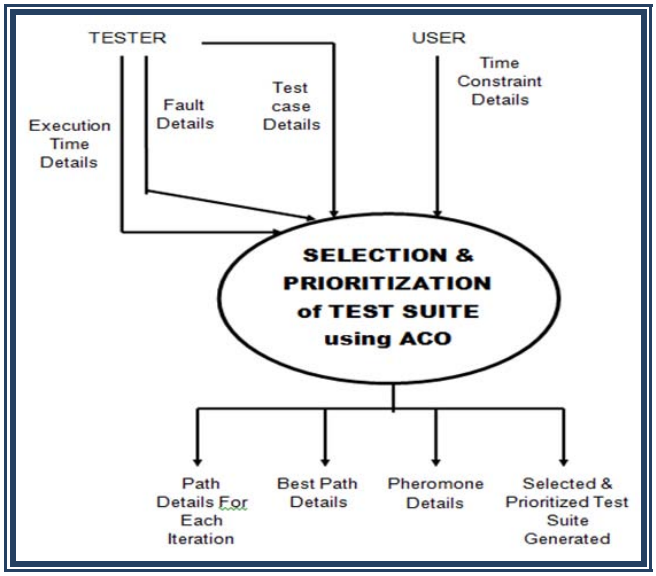

Figure 1. Block diagram for ACO_TCSP System

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 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.

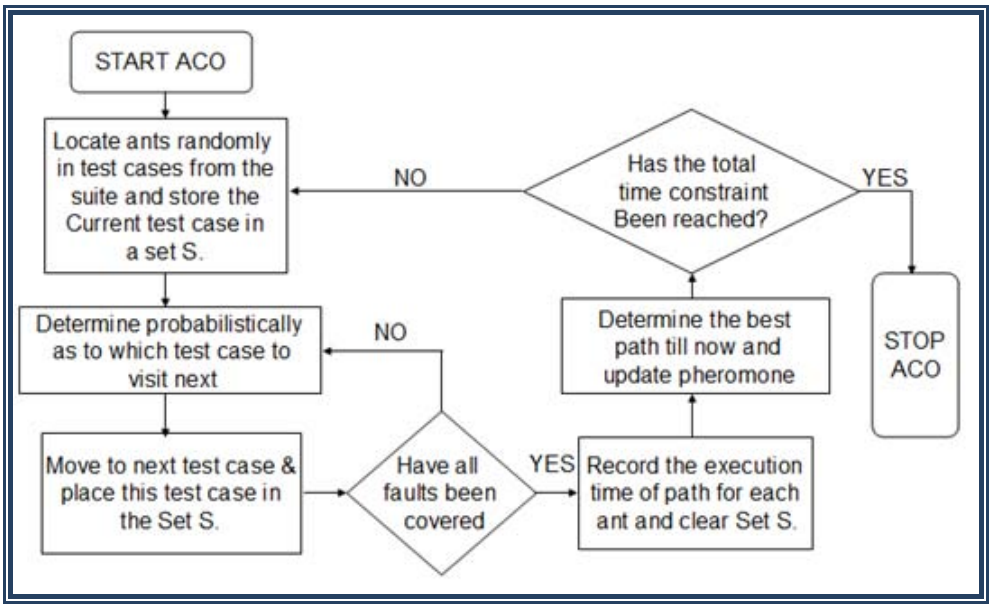

Figure 2. Flow chart for ACO_TCSP System

V. IMPLEMENTING THE TECHNIQUE

The algorithm has been coded as "ACO_TCSP" which is a C++ code complied using TurboC++ compiler, 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 modules from the implementation are explained below:

1) Init_ant() : void

It is an initialization module for the class 'ant'. It is called from the main aco() module before the start of the ACO algorithm.

2) *Enter_ts() : void*

The module has been defined so as to input the details of the test suite on which ACO algorithm for test case selection and prioritization has to be run.

3) Newnode(int) : int

To find whether the node discovered by the ant is an already covered or a new node.

4) Aco() : void

This is the module that actually runs the ACO algorithm's one iteration for all the ants. This module is called from the main() module and is called in a loop till the total time constraint (TC) is reached. The output for this module is the path for each ant and the best path for this particular iteration. The best path of this iteration is chosen according to total fault coverage and minimum execution time. (+1) pheromone is then added on all the edges covered by the best path of current iteration. Also (-10%) pheromone is reduced from all the edges on the graph, corresponding to the real life pheromone evaporation phenomenon in ant colonies.

5) Main() : void

Execution of the implemented code starts from the Main module. Other modules are called from this module and the final results are displayed.

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 ACO_TCSP tool for test case selection and prioritization within a time constrained environment, the tool was applied on the same example as taken by Singh [3] to propose the algorithm. The ACO_TCSP was run four times on the example with constant time constraint, TC=85 time units. The input to the ACO_TCSP assumes a priori knowledge of the faults detected and the execution time of all test cases. The same is tabulated in Table 1. The result of the simulation of table 1 and TC as input for 4 sample runs are given in Table 2. 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 run is shown. It can be inferred from the Table 2 that 3 out of 4 times, the optimal path was found by ACO_TCSP. Though different paths were explored by artificial ants in all he runs, still they could converge to the optimal path.

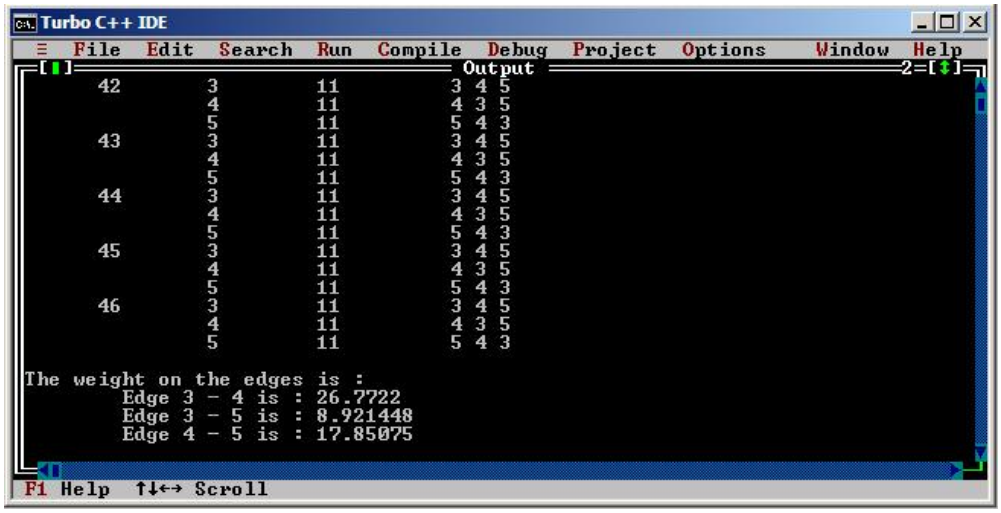

Figure 3. Sample output screen 1 for the tool ACO_TCSP

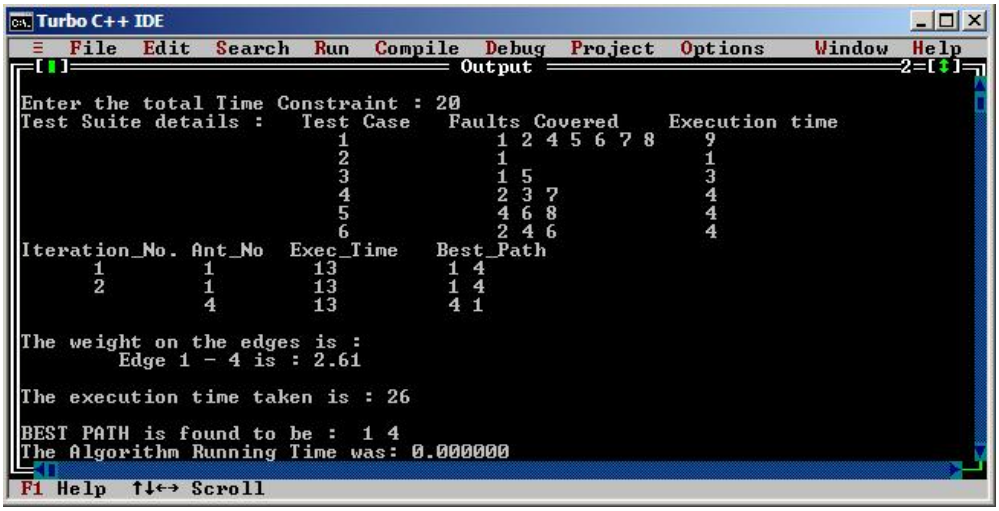

Figure 4. Sample output screen 2 for the tool ACO_TCSP

TABLE I. TEST CASES WITH CORRESPONDING FAULTS COVERED & THE EXECUTION TIME

Test case/ faults	*F1*	*F2*	*F3*	*F4*	*F5*	*F 6*	*F7*	*F8*	*F9*	*F1 0*	*NO.OF FAULTS COVERED*	*EXECUTION TIME (UNIT)*
T1		X		X			X		X		4	7
T2	X		X								2	4
T3	X				X		X	X			4	5
T4		X		X					X		3	4
T5			X			X				X	3	4
T6	X						X				4	5
T7			X			X		X			3	4
T8		X								X	2	2

TABLE II. RESULTS AFTER SAMPLE RUN ON EXAMPLE 1, 4 TIMES.

RUN	Iteration	ANT	Best Path	Exec Time	RESULT
1	1	A1	1,5,8,3	18	Weight on edges after all the iterations 1,5 : 0.531441 3,4 : 6.12459 3,8 : 1.121931 4,5 : 6.12459 5,8 : 0.531441 Rest all edges have 0 weight. Exec. Time for all iterations is 85 Best Path is found to be : 3,4,5 OPTIMUM PATH FOUND IN THIS RUN
	2	A8	8,3,4,5	15	
	3	A5	3,4,5	13	
	4	A3	3,4,5	13	
		A5	5,4,3	13	
	5	A3	3,4,5	13	
		A5	5.4,3	13	
	6	A3	3,4,5	13	
		A5	5,4,3	13	
2	1	A5	5,1,3	16	
	2	A3	3,1,5	16	

		A5	5,1,3	16	
	3	A3	3,1,5	16	Weight on edges after all the iterations
		A5	5,1,3	16	
	4	A3	3,1,5	16	1,3 : 7.902621
		A5	5,1,3	16	1,5 : 7.902621
	5	A3	3,1,5	16	
		A5	5,1,3	16	Exec. Time for all iterations is 96
	6	A3	3,1,5	16	Best Path is found to be : 3,1,5
		A5	5,1,3	16	
3	1	A8	8,1,3,5	18	
	2	A1	1,3,5	16	
		A5	5,3,1	16	Weight on edges after all the iterations
	3	A4	4,5,3	13	
	4	A3	3,5,4	13	1,3 : 1.712421
		A4	4,5,3	13	1,8 : 0.531441
	5	A3	3,5,4	13	3,5 : 7.246521
		A4	4,5,3	13	4,5 : 5.5341
	6	A3	3,5,4	13	Exec. Time for all iterations is 86
		A4	4,5,3	13	Best Path is found to be : 3,5,4
					OPTIMUM PATH FOUND IN THIS RUN
4	1	A1	1,3,5	16	
	2	A1	1,3,5	16	
		A5	5,3,1	16	Weight on edges after all the iterations
	3	A1	1,3,5	16	
		A5	5,3,1	16	1,3 : 3.024621
	4	A8	8,4,5,3	15	3,5 : 7.173621
	5	A3	3,5,4	13	4,5 : 4.149
		A4	4,5,3	13	4,8 : 0.729
	6	A3	3,5,4	13	
		A4	4,5,3	13	Exec. Time for all iterations is 89

VI. GRAPH REPRESENATION

This section represents the final outcomes for the 4 sample runs of ACO_TCSP in graphical form. The final graphs for all the 4 sample runs on the chosen example [3] as given in Table 2 are represented in fig 5 - 8. Test cases are figured as the nodes in the graph and only those edges are shown which have some positive weight (pheromone). Red lines are used to outline the final path found after running the ACO algorithm and the selected test cases (ACO_ordering) have been underlined in red color.

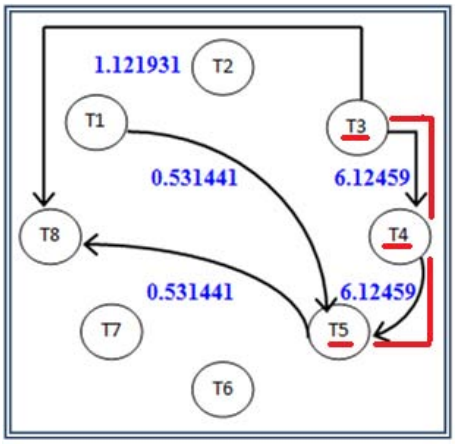

Figure 5. Graph representing final outcome for run 1

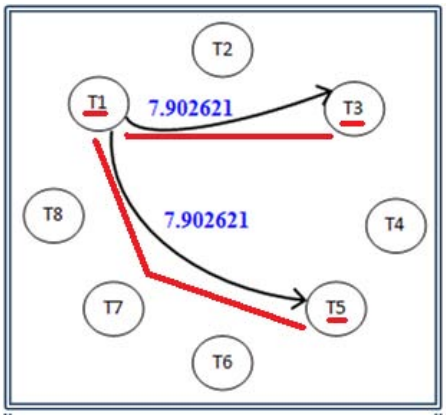

Figure 6. Graph representing final outcome for run 2

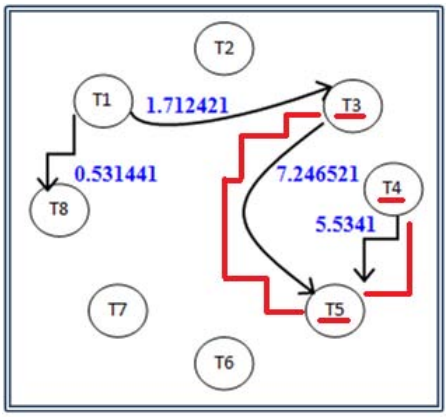

Figure 7. Graph representing final outcome for run 3

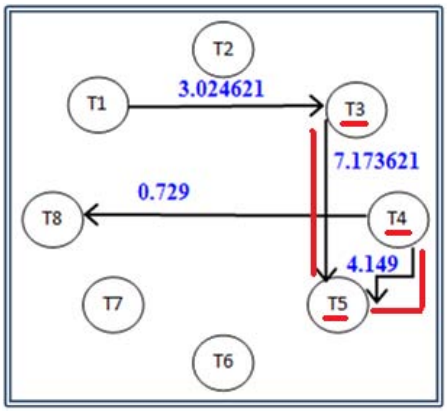

Figure 8. Graph representing final outcome for run 4

It can be observed from the graphical representation that the nodes connected via the edges with maximum weight (pheromone) are selected as the final solution or the best path. Thus we can also prioritize the remaining test suite on the basis of pheromone on the edges in reducing order. Also it can be observed that reduction in the test suite using ACO_TCSP is approximately 62.5%. Another major observation is that all the paths need not be traversed to cover all faults in the specified time.

VII. CONCLUSION & FUTURE WORK

Ant colony Optimization is a promising technique for solving test case selection and prioritization problem. In this study a tool ACO_TCSP for the same has been developed and applied on an example. Though in these tests the best solution was not found for all cases still the results obtained are in close proximity to the optimal results. The reduction in test suite size is achieved to be 62.5% in all the 4 test runs. This encourages the use of the developed tool by testers. In future we plan to apply the tool on many more examples to prove the usability and effectiveness of the proposed technique.

REFERENCES

[1] G. Rothermel, R. H. Untch, C. Chu, and M. J. Harrold, "Test Case Prioritization: An Empirical Study", Proceedings of the International Conference on Software Maintenance, pages 179-188, September 1999.

[2] K. R. Walcott, M. L. So a, G. M. Kapfhammer, and R. S. Roos." Time aware test suite prioritization", In Proceedings of ISSTA, pages 1-11, 2006.

[3] Y.Singh, A.Kaur, B.Suri, "Test case prioritization using ant colony optimization", ACM SIGSOFT Software Engineering Notes, Vol.35 No.4, pages 1-7, July 2010.

[4] Dorigo, M., Maniezzo, V., und Colorni, A.: The Ant System: Optimization by a colony of cooperating agents. IEEE Transactions on Systems, Man, and Cybernetics Part B: Cybernetics. 26(1):29–41. 1996.

[5] M.L. den Besten, T. Stützle and M. Dorigo (2000) Ant colony optimization for the total weighted tardiness problem. In: M. Schoenauer, K. Deb, G. Rudolph, X. Yao, E. Lutton, J.J. Merelo and H.-P. Schwefel (eds.), Proceedings of PPSN-VI, Sixth International Conference on Parallel Problem Solving from Nature, volume 1917 of Lecture Notes in Computer Science. Springer-Verlag, Berlin, Germany, pp. 611–620.

[6] G. Di Caro and M. Dorigo (1998) AntNet: Distributed stigmergetic control for communications networks. Journal of Artificial Intelligence Research, **9**, 317–365.

[7] L.M. Gambardella and M. Dorigo (2000) Ant Colony System hybridized with a new local search for the sequential ordering problem. INFORMS Journal on Computing, **12**(3), 237–255.

[8] L.M. Gambardella, È D. Taillard and G. Agazzi (1999) MACS-VRPTW: A multiple ant colony system for vehicle routing problems with time windows. In: D. Corne, M. Dorigo and F. Glover (eds.), New Ideas in Optimization. McGraw Hill, London, UK, pp. 63–76.

[9] D. Merkle, M. Middendorf and H. Schmeck (2000) Ant colony optimization for resource-constrained project scheduling. In: Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2000), Morgan Kaufmann Publishers, San Francisco, CA, pp. 893–900.

[10] T. Stützle and M. Dorigo (1999) ACO algorithms for the quadratic assignment problem. In: D. Corne, M. Dorigo and F. Glover (eds.), New Ideas in Optimization McGraw Hill, London, UK, pp. 33–50.

[11] G. Rothermel, R.H. Untch, C. Chu and M.J. Harold, "Test Case Prioritization," IEEE Transactions on Software Engineering, Vol.27, No.10, pp.928-948, Oct., 2001.

[12] A. Srivastava and J. Thiagarajan, "Effectively Prioritizing Tests in Development Environment," Proceedings of the International Symposium of Software Testing and Analysis, Rome, 22-24 pp.97-106, July, 2002.

[13] Kim, J. M., and A. Porter, "A history-based test prioritization technique for regression testing in resource constrained environments," In Proceedings of the 24th International Conference on Software Engineering, pp.119-129, 2002.

[14] Z. Li, M. Harman, and R. M. Hierons "Search algorithms for regression test case prioritization," IEEE Trans. On Software Engineering, Vol.33, No.4, April, 2007.

[15] A. Colorni, M. Dorigo, and V. Maniezzo, “Distributed optimization by ant colonies”, Proceedings of ECAL'91, European Conference on Artificial Life, Elsevier Publishing, Amsterdam, 1991.
[16] M. Dorigo, “Optimization, learning and natural algorithms”, Ph.D. Thesis, Politecnico diMilano, Milano, 1992.
[17] M. Dorigo, V. Maniezzo, and A. Colorni, “The ant system: an autocatalytic optimizing process”, Technical Report TR91-016, Politecnico di Milano (1991).
[18] Sara Alspaughy, Kristen R. Walcotty, Michael Belanichz, Gregory M. Kapfhammerz and Mary Lou Soffay, “Efficient Time-Aware Prioritization with Knapsack Solvers”, University of Virginia, Allegheny College.

AUTHORS PROFILE

Bharti Suri is a Asst. Professor at the University School of Information Technology, Guru Gobind Singh Indraprastha University, Kashmere Gate, India. She holds master’s degrees in computer science and information technology. She is pursuing Ph.D in the area of software testing. Her areas of interest are software engineering, software testing, software project management, software quality, and software metrics. Suri is a lifetime member of CSI. She has completed University Grants Commission (UGC) funded major research project as co-investigator in the area of software testing. She has many publications in national and international journals and conferences to her credit. Suri can be contacted by e-mail at bhartisuri@gmail.com .

Shweta Singhal is a research student at the University School of Information Technology, Guru Gobind Singh Indraprasha University, Kashmere Gate, India. She holds graduate’s degree in Electronics and Communication Engineering. She is pursuing M.Tech in Information Technology. Her areas of interest include software engineering, software testing, and network security. Singhal can be reached by e-mail at miss.shweta.singhal@gmail.com.