Indian Journal of Science and Technology, Vol 9(30), DOI: 10.17485/ijst/2016/v9i30/86588, August 2016
ISSN (Print) : 0974-6846
ISSN (Online) : 0974-5645

Experimental Analysis of m-ACO Technique for Regression Testing

Kamna Solanki*, YudhVir Singh and Sandeep Dalal

M. D. University, Rohtak - 124001, Haryana, India

Abstract

Objectives: Experimental evaluation of "m-ACO" (Modified Ant Colony Optimization) technique for test case prioritization has been performed on two well known software testing problems namely "Triangle Classification Problem" and "Quadratic Equation Problem". Apart from these two problems, m-ACO has been experimentally evaluated using open source software JFreeChart. **Methods:** m-ACO finds the optimized solution to test suite prioritization by modifying the phenomenon used by natural ants to reach to its food source and select the food. This paper attempts to experimentally and comparatively evaluate the proposed m-ACO technique for test case prioritization against some contemporary meta-heuristic techniques using two well known software testing problems and open source problem. Performance evaluation has been measured using two metrics namely APFD (Average Percentage of Faults Detected) and PTR (Percentage of Test Suite Required for Complete Fault Coverage). **Findings:** The proposed technique m-ACO proves its efficiency on both the parameters. m-ACO achieves higher fault detection rate with minimized test suite as comparative to other meta-heuristic techniques for test case prioritization. **Improvements:** The proposed technique m-ACO basically works by modifying the food source searching and selection pattern of the real ants. Real ants grab every type food source it comes across; while modified ants evaluate the food fitness and uniqueness before selection. This phenomenon enhances the quality and diversity of deposited food source.

Keywords: Fault Coverage, Genetic Algorithm, Regression Testing, Software Testing, Test Suite Prioritization

1. Introduction

Our dependency on software is increasing everyday as almost all gadgets, services and computer applications we are using today are pre-occupied with software. Parallel to this, the complexity of the software is also increasing at the same rate. Therefore, software development has become an utmost difficult task requiring highest level of skills and expertise. Moreover, testing of the software has also become equally difficult and time consuming with growing software size and complexity as the test suite size keeps on increasing as the size and complexity of the software increases. As quality of the software highly depends on effectiveness of software testing, so the challenge for software developers is to produce quality software through usage of effective testing techniques. This ever increasing software size and complexity demands the efficient management of resources and application of effective techniques especially during software testing. Test Suites are preserved for potential re-utilization in later phases by software developers[1,2]. This re-utilization of Test suite is termed as regression testing. Test Suite Prioritization is an effective way of managing the Test Suite by re-scheduling the execution order of test cases. Execution of test cases based on some prioritization criteria helps in achieving complete fault coverage using reduced Test Suite. Test Suite prioritization is an important regression testing technique which reorders the execution sequence of test cases to detect the faults early with minimum efforts[3,4]. Numerous researchers have applied nature inspired meta-heuristic techniques for development of Test Suite prioritization techniques[5–8]. In this regard, a novel technique m-ACO (modified Ant Colony Optimization Algorithm) for test case prioritization has been proposed[9]. m-ACO is a modified form of the ACO (Ant Colony Optimization) algorithm which alters the food source selection behaviour of natural ants to increase the diversity of food source deposited in a given time. This

*Author for correspondence

paper presents experimental evaluation of the proposed m-ACO technique for test case prioritization on two well known software testing problems namely "Triangle Classifier Problem" , "Quadratic Equation Problem" and an open source software Jfreechart on two parameters namely APFD (Average Percentage of Faults Detected) and PTR (Percentage of Test Suite Required for Complete Fault Coverage).

2. Proposed Model for m-ACO

While solving the problem of test case prioritization using proposed m-ACO, the first and foremost step is to build a modified Ant Colony (m-ACO) system model so that the modified ant system can build solutions using proposed model as shown in Figure 1. If there are n test cases in the original test suite 'T', which reveals m faults. We assume that number of modified ants is equal to number of test cases for solution generation. Ti indicates the i^{th} test case in test suite 'T' and S_k indicates probable test case sequence after a modified ant has selected (k-1) test cases. Initially, we place n modified ants in alternative test sequence list S1. Each modified ants randomly selects test case nodes in next sequence, until every modified ant finds all faults. After completion of one- iteration, a modified ant builds a test case sequence. Next iteration is decided based on two criteria namely pheromone updating rule and food uniqueness as described in pseudo-code. It probably requires several rounds of iteration until the prioritization goal is satisfied based on rate on fault detection rate at each node.

The proposed algorithm m-ACO modifies the food selection criteria of the natural ants. Modified ants do not select every type of food source it comes across. Instead, a modified ant calculates the food source quality and compares it with a random index. If the quality is higher than the random index, then food source is selected, otherwise the modified ant starts its search for another food source (i.e. fault) as defined in pseudo-code of m-ACO in previous section. In ACO, as soon as natural ants find a food source, it comes back with food to its nest; deposit the food in the nest and then goes to the same food source again until the food source finishes. If that path is shortest, then eventually, all ants will start following the same trail until that food source finishes. This decreases the diversity of the deposited food in a given time by natural (real) ants. In m-ACO, the modified ants as proposed moves in random direction and as soon as it finds a food source, it calculates the uniqueness of the food and compares its value to some random index to make a decision regarding continue its local search for better food source or grab the food and come back to the nest as shown in Figure 2.

Natural ants selects every type of food source which is at the shortest distance until it finishes; while in m-ACO; ants are supposed to check the food quality by calculating the food source fitness which increases the probability of capturing variety of food source. Modified ants only selects quality food source at nearest location i.e. increased variety of faults covered within a specified time. The proposed m-ACO algorithm has been implemented and code for the proposed m-ACO algorithm has been written. Altering the food source selection criteria in the proposed m-ACO algorithm increases the diversity of food source deposited i.e. increased diversity of faults detected by a prioritized test suite.

Figure 2 described above depicts how the real ant selects any kind of food source which it come across randomly (food A); while modified ant moves randomly

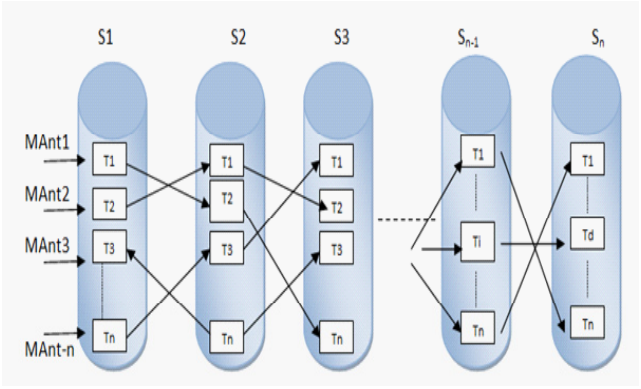

Figure 1. m-ACO model for test case prioritization.

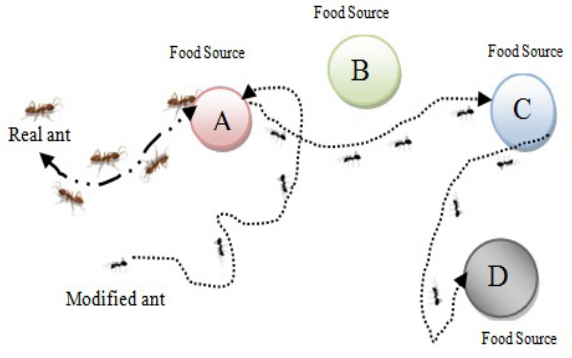

Figure 2. Food source selection behavior of natural (real) and modified ants.

and selects only those food source which is unique (i.e. food D i.e. food which has not been deposited earlier), while it does not choose other food sources which is not unique (i.e. food A and food C, food which has already been deposited/collected).

3. Experimental Evaluation of m-ACO Technique

The proposed m-ACO technique for test case prioritization has been experimentally evaluated using three case studies[9]. This paper presents the experimental evaluation of m-ACO technique for test case prioritization on two well known Software testing problems namely "Triangle Classifier Problem" and "Quadratic Equation Problem" and open-source software JFreeChart. Two parameters have been calculated for experimental evaluation:

- APFD (Average Percentage of Faults Detected).
- PTR (Percentage of Test Suite Required for Complete Fault Coverage).

Elbaum et al. has developed APFD metric (Average Percentage of Faults Detected) which can be utilized to quantify the objective of maximizing fault detection rate using a combination of test suites[10,11]. APFD metric is used to estimate the average fault detection rate per percentage of execution of test suite. Higher value of APFD means higher percentage of faults detected. A comparatively high APFD means a better prioritization technique.

Notion for APFD calculations are:

$$APFD = 1 - \frac{TF_1 + TF_2 + TF_3 + \cdots \ldots \ldots \ldots TF_m}{mn} + \frac{1}{2n}$$

Where T is the test suite to be evaluated, m depicts the number of faults in an application under test, n is total test cases in a test suite and TF_j describes the location of the first test case in Test Suit T that reveals fault j. Percentage of Test Cases Required for Complete Fault Coverage (PTR) is a metric which can be utilized to measure the effectiveness of the test suite prioritization technique[12]. An effective test suite prioritization will position the test cases which are most likely to find faults at the starting of the prioritized test sequence. So, it would be helpful to calculate the percentage of those test cases which must run before all faults of the application are revealed. A comparatively low value of PTR means a better prioritization technique.

Notion for PTR calculations are:

$$PTR = \frac{\text{No. Of Test Cases Required for Complete Fault Coverage}}{\text{Total Number of Test Case}} \times 100$$

3.1 Triangle Classifier Problem

The proposed m-ACO technique was also applied to most widely used software testing problem namely "Triangle Classifier Problem". Triangle Classifier Problem is one of the most famous software testing problems, which takes three sides of a triangle as input and classifies a triangle as equilateral or isosceles or scalene triangle or "not a triangle" based on the input.

A Simple Test Suite Triangle Classifier problem takes 17 test cases in sequence {N1->N2->N3->N4->N5->N6->N7->N8->N9->N10->N11->N12->N13->N14->N15->N16->N17} and 6 Faults {F1-F6}. Table 1 described below describes the test suite for triangle problem with Execution Time (ET).

When the above mentioned test suite was executed using the proposed m-ACO program in perl language, the putative solutions generated converge towards the test sequence:

Table 1. Faults detected by test suite for "Triangle Classifier Problem"

	F1	F2	F3	F4	F5	F6	ET
N1	*		*			*	5
N2	*	*	*			*	2
N3	*	*	*			*	3
N4	*	*	*	*			4
N5	*	*	*	*	*		5
N6	*	*	*			*	3
N7	*	*	*			*	6
N8	*		*			*	4
N9	*	*	*			*	5
N10	*	*	*			*	3
N11	*	*	*	*	*		8
N12	*	*	*			*	4
N13	*	*	*			*	6
N14	*		*			*	3
N15	*	*	*			*	2
N16	*	*	*			*	3
N17	*		*			*	5

N5->N2->N1->N3->N4->N6->N7->N17->N8->N9->N12->N15->N16->N10->N13->N14->N11 with an APFD of 0.97.

$$APFD = 1 - \frac{1+1+1+1+1+1}{17*6} + \frac{1}{2*17}$$

$$APFD = 1.03 - 0.059 = 0.97$$

APFD value of the un-prioritized sequence {N1->N2->N3->N4->N5->N6->N7->N8->N9->N10->N11->N12->N13->N14->N15->N16->N17} is 0.89. APFD value of the randomly prioritized sequence {N1->N3->N2->N7->N8->N6->N10->N5->N11->N14->N4->N17->N15->N16->N9->N12->N13} is 0.82. So, it is evident that proposed m-ACO technique performs better than the un-prioritized and randomly prioritized test sequences for triangle classifier problem. PTR values for m-ACO based prioritized order, un-prioritized order and random prioritized order are 12, 30 and 41 percent respectively as shown in Figure 3.

3.2 Quadratic Equation Problem

Quadratic Equation Problem is also one of the famous software testing problems which takes the three variables of the equation as input and gives the output as equal roots, real roots, imaginary roots and not a quadratic equation according to the input. A Simple Test Suite for Quadratic Equation Problem takes 19 test cases in sequence {N1->N2->N3->N4->N5->N6->N7->N8->N9->N10->N11->N12->N13->N14->N15->N16->N17->N18->N19} and 9 Faults {F1-F9}. and 9 Faults {F1-F6}. Table 2 describes the test suite for Quadratic Equation Problem with Execution Time (ET).

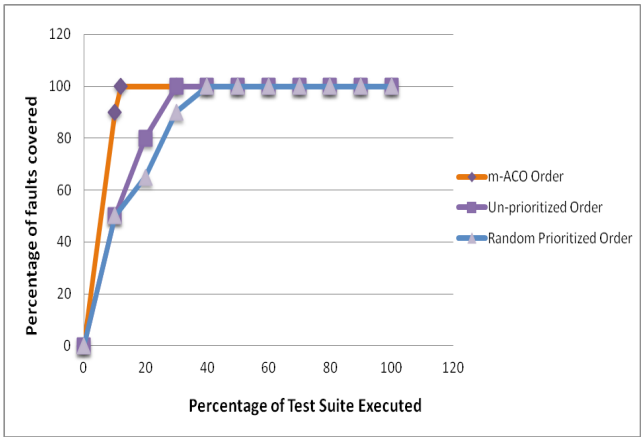

Figure 3. Comparison of PTR values of triangle problem.

Table 2. Faults detected by test suite for "Quadratic Equation Problem"

	F1	F2	F3	F4	F5	F6	F7	F8	F9	ET
N1	*	*								3
N2			*					*	*	5
N3	*			*			*		*	2
N4	*			*	*				*	6
N5	*			*		*		*		3
N6	*		*			*			*	4
N7	*	*								2
N8										6
N9	*			*	*	*	*			4
N10	*		*		*	*		*		7
N11	*	*			*	*			*	3
N12	*			*	*	*		*		2
N13		*			*					7
N14		*	*							3
N15		*			*		*		*	5
N16	*			*	*		*		*	6
N17	*			*	*	*			*	3
N18	*		*		*	*		*		4
N19		*								1

When the above mentioned test suite was executed using m-ACO, the putative solutions generated converge towards the test sequence:

N9->N7->N2->N1->N5>N4->N10->N3->N6->N15->N12->N11->N14->N16->N19->N13->N17->N18->N8 with an APFD of 0.93.

$$APFD = 1 - \frac{1+2+3+1+1+1+1+3+3}{19*9} + \frac{1}{2*19}$$

$$APFD = 1.026 - 0.094 = 0.93$$

APFD value of the un-prioritized sequence N1->N2->N3->N4->N5->N6->N7->N8->N9->N10->N11->N12->N13->N14->N15->N16->N17->N18->N19 is 0.89. APFD value of the randomly prioritized sequence N11->N13->N2->N9->N8->N6->N12->N10->N1->N3->N5->N7->N15->N16->N14->N17->N19->N18->N4 is 0.89. So, it is evident that proposed m-ACO technique performs better than the un-prioritized and randomly prioritized test sequences for Quadratic Equation Problem.

PTR values for m-ACO based prioritized order, un-prioritized order and random prioritized order are 16, 25 and 21 percent respectively as shown in Figure 4.

3.3 JFreeChart: An Open Source Software

Empirical studies in software testing research may not be comparable, reproducible or characteristic of practice. One reason is that real bugs are too infrequently used in software testing research. Extracting and reproducing real bugs is challenging and as a result hand-seeded faults or mutants are commonly used as a substitute. Defects4J, is a database and extensible framework providing real bugs to enable reproducible studies in software testing research[13].

The initial version of Defects4J contains 357 real bugs from 5 real-world open source programs namely JFreeChart, Closure Compiler, Apache Commons Lang, Apache Commons Math and Joda-Time. Each real bug is accompanied by a comprehensive test suite that can expose (demonstrate) that bug. Defects4J features a framework to easily access faulty and fixed program versions and corresponding test suites. This framework also provides a high-level interface to common tasks in software testing research, making it easy to conduct and reproduce empirical studies. JFreeChart is a free Java chart library that is useful for developers to display professional quality charts. The JFreeChart is the most widely used chart library for Java with more than 2.2 million downloads till date[13]. As per the record on Defects4J, JFreeChart has 26 reported bugs with the final corresponding test suite of 9 test cases which cover these faults as shown in following Table 3.

When the above mentioned test suite for "JFreeChart" was executed using the proposed m-ACO technique,

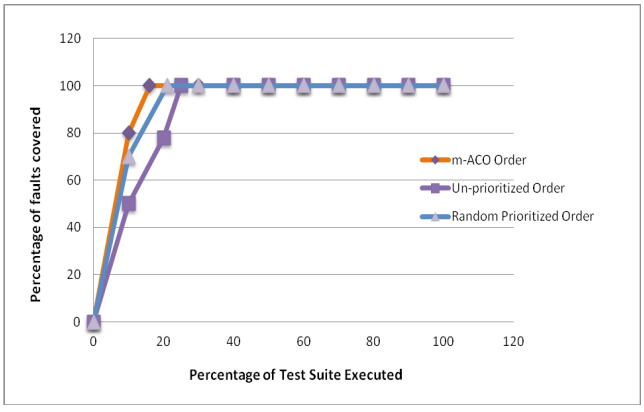

Figure 4. Comparison of PTR values of Quadratic Equation Problem.

Table 3. Faults detected by test suite for "JFreeChart"

	N1	N2	N3	N4	N5	N6	N7	N8	N9
F1	*				*				
F2							*	*	
F3		*				*			*
F4	*						*		
F5		*							
F6			*			*			
F7		*						*	
F8	*		*						*
F9					*				
F10		*							
F11						*			*
F12			*				*		
F13			*	*					
F14		*							
F15		*	*					*	
F16	*								*
F17	*			*	*				
F18				*					
F19		*							*
F20			*				*		
F21				*	*				
F22			*					*	
F23	*			*					
F24	*			*	*				
F25								*	*
F26			*					*	*
ET (Execution Time)	6	9	5	4	6	3	8	11	5

the achieved prioritized test suite N3->N9->N2->N1->N4>N8->N5->N7->N6 yields an APFD value of 0.76. The APFD for un-prioritized test suite N1->N2->N3->N4->N5>N6->N7->N8->N9 is 0.74. The APFD for randomly prioritized test suite N2->N4->N8->N1->N9>N7->N6->N3->N5 is 0.74. PTR values for m-ACO based prioritized order, un-prioritized order and random prioritized order are 78, 78 and 100 percent respectively as shown in Figure 5.

So, it is evident that the proposed m-ACO technique performs quite well in case of real bugs from the open source software JFreeChart by attaining higher APFD values. Figure 6 depicted below makes a comparative evaluation of three problems namely Triangle Classifier Problem, Quadratic Equation Problem and JFreeChart

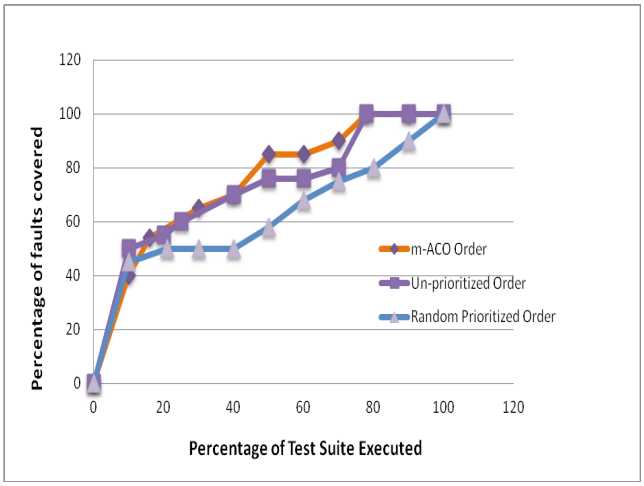

Figure 5. Comparison of PTR values of JFreeChart software problem.

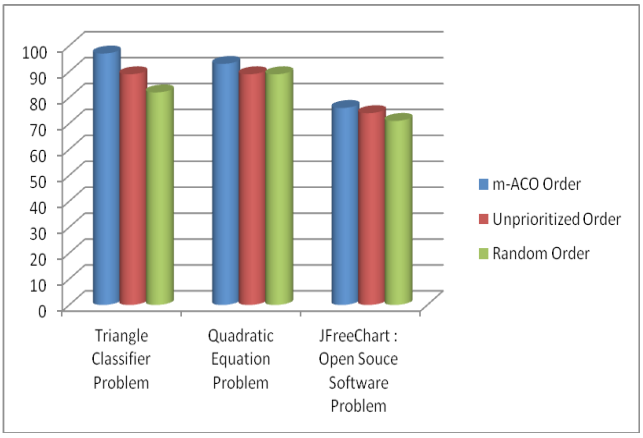

Figure 6. Comparison of APFD values of triangle problem, Quadratic Equation Problem, JFreeChart Software problem.

Open Source Software Problem. The graph compares the APFD values of the specified problems in context of proposed prioritization technique m-ACO, random prioritization and un-prioritized order.

4. Comparative Evaluation of m-ACO and Other Meta-Heuristic Techniques

Regression testing efforts can be minimized up to a certain level using various techniques[14]. Numerous researchers have solved the problem of test case generation[15] and prioritization using meta-heuristic techniques like GA[16], Simulated Annealing[17], BCO and ACO etc. This section discusses the results obtained when test suite prioritization achieved using proposed m-ACO was comparatively evaluated against GA (Genetic Algorithm), BCO (Bee Colony Optimization) and ACO (Ant Colony Optimization) based techniques for the "Triangle Classifier Problem", "Quadratic Equation Problem" and "JFreeChart". The following Table 4 displays the comparative APFD Values of GA, BCO, ACO and m-ACO. The technique with Higher the value of APFD is better in terms of average percentage of faults detected. Figure 7 graphically represents the comparative APFD values and clearly depicts that m-ACO outperforms its contemporary techniques in terms of Average percentage of faults detected.

Table 5 displays the comparative PTR values of GA, BCO, ACO and m-ACO. The technique with least value of PTR is optimal as it represents the percentage of test cases of the original test suite to be executed to achieve complete fault coverage. Figure 8 graphically represents the comparative PTR values and shows that m-ACO requires the minimum number of test cases to achieve complete coverage.

Table 4. Comparative APFD values

	GA	BCO	ACO	m-ACO
Triangle Problem	0.88	0.95	0.93	0.97
Quadratic Equation Problem	0.86	0.91	0.89	0.93
JFreeChart	0.70	0.76	0.73	0.76

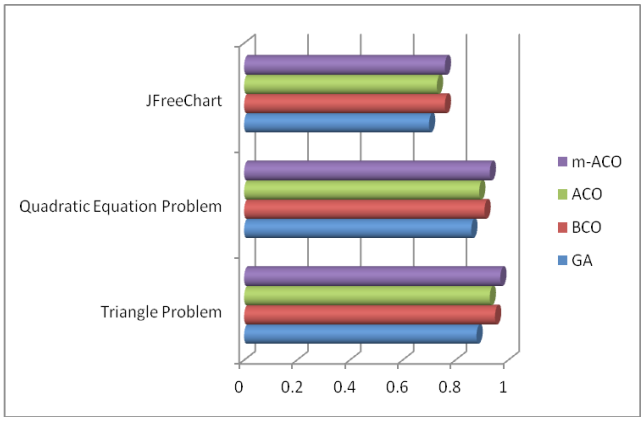

Figure 7. Graphical representation of comparative APFD values.

Table 5. Comparative PTR values

	GA	BCO	ACO	m-ACO
Triangle Problem	18	12	16	12
Quadratic Equation Problem	20	18	18	16
JFreeChart	84	78	80	78

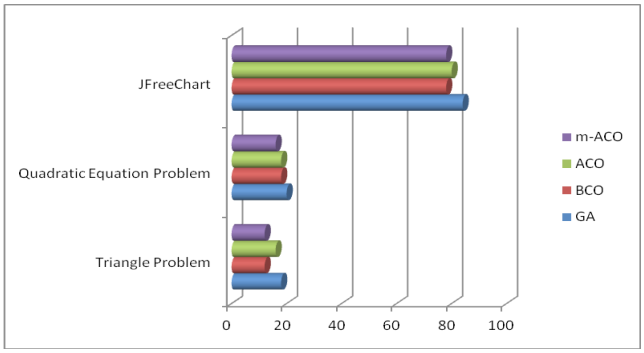

Figure 8. Graphical representation of comparative PTR values.

5. Conclusion

Test suite prioritization is an important and most widely used technique for regression testing in software industry. It is widely used in conjunction with test suite selection technique for minimizing regression test cost and efforts. The proposed m-ACO technique for test suite prioritization is based on modified ant behavior of natural Ant Colony Optimization algorithm to find prioritized sequence of test cases. Modified ants do not select every type of food source it comes across; instead food fitness and uniqueness is evaluated for selection. Food source uniqueness has never been used earlier in Ant Colony Optimization for test suite prioritization. This paper makes an experimental and comparative evaluation of the proposed m-ACO technique for test suite prioritization on well known software testing problems namely "Triangle Classifier Problem", "Quadratic Equation Problem" and open source software "JFreeChart". Performance evaluation has been performed against GA, BCO and ACO based test case prioritization techniques. The experiments conducted have been evaluated using APFD and PTR metric. The results obtained clearly depicts that the proposed m-ACO technique performs quite well in terms of two observed parameters. The future research will try to focus on measuring the effectiveness of the proposed m-ACO technique on some other parameters using diverse case studies.

6. References

1. Beizer B. Software Testing Techniques. 2nd ed. India: Dreamtech Press; 2003.
2. Catal C, Mishra D. Test Case Prioritization: A systematic study. Software Quality Journal. 2013; 21(2):445–78.
3. Chandu PMSS, Sasikala T. Implementation of regression testing of test case prioritization. Indian Journal of Science and Technology. 2015 Apr; 8(S8):2903. DOI: 10.17485/ijst/2015/v8iS8/61922.
4. Leung H, White L. Insights into regression testing. Proceedings of the IEEE International Conference on Software Maintenance; 1989 Oct. p. 60–9.
5. Srivastava PR. Test case prioritization. Journal of Theoritical and Applied Information Technology. 2008; 4(3):178–81.
6. Kaur A, Goyal S. A bee colony optimization algorithm for code coverage test suite prioritization. International Journal of Engineering Science and Technology. 2011; 3(4):2786–95.
7. Singh Y, Kaur A, Suri B, Singhal S. Test case prioritization using Ant Colony Optimization. ACM SIGSOFT Software Engineering Notes. 2012; 35(4):1–7.
8. Li Z, Harman M, Hierons RM. Search algorithms for regression test case prioritization. IEEE Transactions on Software Engineering. 2007; 33(4):225–37.
9. Solanki K, Singh Y, Dalal S. Test case prioritization: An approach based on modified Ant Colony Optimization. Proceedings of IEEE International Conference on Computer, Communication and Control; Indore, India. 2015 Sept. Available at IEEE-xplore Digital Library.
10. Elbaum S, Malishevsky A, Rothermel G. Test case prioritization: A family of empirical studies. IEEE Transactions on Software Engineering. 2002; 28(2):159–82.
11. Elbaum S, Rothermel G, Kanduri S, Malishevsky AG. Selecting a cost-effective test case prioritization technique. Software Quality Journal. 2004; 12(3):185–210.
12. Raju S, Uma GV. Factors oriented test case prioritization technique in regression testing using genetic algorithm. European Journal of Scientific Research. 2012; 74(3):389–402.
13. Just R, Jalali D, Ernst MD. Defects4J: A database of existing faults to enable controlled testing studies for Java programs. ACM International Symposium on Software Testing and Analysis; 2014. p. 437–40.
14. Jacob TP, Ravi. An optimal technique for reducing the effort of regression test. Indian Journal of Science and Technology. 2013 Aug; 6(8):5065–9. DOI: 10.17485/ijst/2013/v6i8/36345.
15. Maheshwari V, Prasanna M. Generation of test case using automation in software systems: A review. Indian Journal of Science and Technology. 2015 Dec; 8(35):1–9. DOI: 10.17485/ijst/2015/v8i35/72881.
16. Musa S, Sultan AB, Ghani AB, Baharom S. Software regression test case prioritization for object-oriented programs using genetic algorithm with reduced-fitness severity. Indian Journal of Science and Technology. 2015 Nov; 8(30):1–9. DOI: 10.17485/ijst/2015/v8i30/86661.
17. Maheswari RU, JeyaMala D. Combined genetic and simulated annealing approach for test case prioritization. Indian Journal of Science and Technology. 2015 Dec; 8(35):1–5. DOI: 10.17485/ijst/2015/v8i35/81102.