Test Case Prioritization: An Approach Based on Modified Ant Colony Optimization (m-ACO)

Kamna Solanki[1]
M.D. University, Rohtak, India
kamna.mdurohtak@gmail.com[1]

Yudhvir Singh[2]
M.D. University, Rohtak, India
kamna.mdurohtak@gmail.com[1]

Sandeep Dalal[3]
M.D. University, Rohtak, India
kamna.mdurohtak@gmail.com[1]

***Abstract*—Intense and widespread usage of software in every field of life has attracted the researchers to focus their attention on developing the methods to improve the efficiency of software testing; which is the most crucial and cost intensive phase of software development. Software testing aims to uncover the potential faults in Application Under Test by running the test cases on software code. Software code keeps on changing as the uncovered faults during testing are fixed by the developers. Regression testing is concerned with verifying the modified software code to ensure that changes in software code does not induce any undesired effect on rest of the code. Test Case Prioritization is a regression testing technique which re-schedule the execution sequence of test cases to improve the fault detection rate and enhance the performance of regression test suite. This paper focuses on proposing a novel method "m-ACO" for test case prioritization and the performance evaluation of the proposed method using Average Percentage of faults Detected.**

Keywords—Software Testing; Regression Testing; Test Case Prioritization; APFD

I. INTRODUCTION

Software testing aims to validate and verify any software program against the software requirement specifications [1][2][3]. Regression basically defines the process of moving to an older state. Regression Testing comes into picture during that part of test cycle where a software program S' undergoes functional testing to make sure that along with the recently amended or modified code; the code carried over un-amended from the older version of software program S also behave as expected. So, regression testing is beneficial and required whenever a latest version of any software program is available by modifying an older version. Sometimes, regression testing is also referred as "Program Re-validation" [4].

Consider the software maintenance process for any software program S whose version A has been released. After the release of the software program S, if there is a need of any kind of corrective, adaptive or perfective maintenance, then the software code will be modified by the developers leading to a newer version of software program S'(release version B). Afterwards, the modified software program S' must be rigorously tested again to ensure the correct functioning of any newly added functionality. But during the modifications in the software code S, the developers might have mistakenly changed that portion of code which leads to improper functioning of software Program S. Hence, regression testing verifies that any kind of malfunctioning of the modified software program S' is detected and rectified prior to the release of S' as version B [5].

Regression testing is an essential step in almost all commercial software development environments. The need for regression testing arises as corrective and progressive changes occur frequently in a development environment. Regression testing improves confidence in that existing and unchanged code will continue to behave as expected after the changes have been incorporated. Regression testing ensures that a changed software code against any kind of unintended amendments. Since several well-known software failures can be blamed on not testing changes and amendments in a software system thoroughly and properly, hence quality of the software highly depends on the efficiency and effectiveness of regression testing [4].

Software testing and re-testing is a continuous process during the software development life cycle for early defect identification and localization [5]. The size of test suites keeps on growing during regression testing for verification and validation of old and new functionalities. Regression testing can be stated as a necessary evil to ensure the validity of the changed software. The original test suite is always available for re-use in regression testing; but entire original test suite cannot be executed due to time and budget constraints. Hence it becomes necessary to reduce the size of the original test suite and choose a subset of test cases from the original test suite which covers the modifications and will be executed in least time and has the capability to cover all the faults. A regression test filtration can be helpful in selecting a suitable subset of original test suite. It is worthy to use a filtration process if the total cost of the filtering process and total costs of executing and auditing the selected test cases using filtration process is less than the cost of executing and auditing

all of the tests in the original test suite. Therefore, selection of test cases for regression testing is an important issue [5] [6].

A. Selecting Test Cases for Regression Testing

One of the critical tasks of software testing is selection of appropriate test cases for execution of regression test suite. One possible strategy can be to Re-run entire Test Suite (Retest All), but it is impractical to re-run the entire test suite after every code change due to time and resources constraints. Another strategy can be to Re-run a subset of existing Test Suite (Selective Retesting); which can be divided into following three methods [6].

- *Test Case Minimization (TCM)/Test Suite Reduction (TSR):* These techniques remove the redundant test cases permanently to minimize the number of test cases in a test suite.
- *Test Case Selection (TCS)/ Regression Test Selection (RTS):* These techniques select some of the test cases and focus on the ones that test the changed part of the software. Contrary to the TSR, RTS does not eliminate the test cases, but selects the test cases that are related to the changed portion of the source code.
- *Test Case Prioritization (TCP):* These techniques identify the efficient ordering of the test cases to maximize certain properties such as rate of fault detection or coverage rate.

Identification of optimized prioritization order for Fault Coverage is still an open research problem. In this regard, this paper presents a novel technique for test case prioritization with simultaneous empirical evaluation of its ability for fault detection rate [8]. Despite all the efforts of researchers in improving the quality of the software testing, efficiency and effectiveness of software testing is still an open challenge and remains below the marginal line of customer's expectations. Even after being a widespread and mainstream validation approach in software industry, testing is still largely ad hoc, expensive, and unpredictably effective. Although in last decade, many researchers have tried to focus their efforts to enhance the quality of software testing process to a reasonably good level. But still, more research needs to be done in this direction and needs to go a long way as results are very inconclusive and contradictory. Literature study concludes that only a limited amount of research work has been done based on a hybrid approach of usage of meta-heuristic algorithms in regression testing. Therefore, this research gap must be covered by conducting more research. In this regard, a number of researchers have applied nature inspired techniques and algorithms in regression testing during last decade and got extremely good results [9].

II. LITERATURE REVIEW

A number of techniques have been proposed in the area of test case prioritization [3] [6] [8]. Numerous among these are akin to “Total fault-detection technique” where the main goal is to reveal all the defects or faults inserted in specific code modules [8]. The primary approaches used to improve upon the fault detection rate include: “Greedy algorithms”, where a greedy approach is adopted in the selection of the test cases i.e. prioritization of the most optimum as first [9]; “Evolutionary algorithms” where a progressive evolution among the test sub-ensemble combination is allowed so as to eventually formulate a prioritized test suite [10]; “Non-evolutionary algorithms” wherein the objective based prioritization is done [11], [12]. “Need specific algorithms” have also been used for prioritization of required types or needs of test suites [13]. These are different from general test suite prioritization methods that are applicable on any regression test suite prioritization. One more way to solve prioritization conundrum is “Variable analysis algorithms” wherein the analysis of the relationship between variables that are altered and their usage in other areas of code are done [13] [14].

Many research studies have been conducted regarding the usage of meta-heuristic techniques in software testing during last one decade. The following section discuss some of the research papers which make use of nature inspired techniques in software testing to improve the efficiency of test suite on some criteria.

P. R. Srivastava discussed the application areas of artificial intelligence in software testing optimization [15]. Srivastava et al. proposed the use of Ant Colony Optimization algorithms for test path generation by designing a prototype tool named as PPTACO. Although the tool was able to provide higher code coverage, but has higher level of repetition of states which have multiple paths [16]. P.R. Srivastava and K.M. Baby also proposed the use of Ant Colony Optimization algorithms for automatic and full coverage of testing of all state transitions [17]. Srivastava et al. proposed the use of Cuckoo Search Algorithm for test case prioritization and applied Tabu Search algorithm for automated test data generation and prioritization, but the proposed technique had a run time issue as it needed CFG as input every time the algorithm was run [18].

D. Jeyamala and V. Mohan proposed a new, non-pheromone-based test case optimization approach which was based on the behavior of biological bases. Their proposed approach was based on ABC (Artificial Bee Colony Optimization) and works by joining the local search methods used by employed bees with global search methods showed by onlookers and scouts. They evaluated the performance of proposed ABC approach against GA(Genetic Algorithm)based approach and concluded that ABC technique was better than GA technique on three parameters as it was more fast, scalable and near to global optimal solution. The research limitation was that the effectiveness of proposed approach was evaluated only against GA based Approach [19].D. Jeyamala and V. Mohan also proposed usage of Hybrid Genetic algorithm approach for Test Suite Optimization and Test Sequence generation which combine the features of GA and Local Search techniques. The performance of the proposed approach has been evaluated against GA and BA (Bacteriologic Algorithm) [20].

So, it is evident that although much research has been conducted in the field of enhancing the effectiveness and

efficiency of test suite through application of nature inspired techniques in software regression testing; still there are many open research challenges. Some of the papers discussed above either not evaluated the performance of their proposed algorithm/technique against any other techniques or compared the performance only against one or two of existing techniques with limited test data which question marks the claimed benefits of the proposed techniques. Some other papers have repetition of test data and run time limitations. Most of the papers have focused on enhancing the code coverage or fault coverage and no due attention has been paid towards maximizing fault diversity, which is an equally important factor for improving the effectiveness of the test suite. None of the papers which use Ant Colony Optimization Algorithm considered the main factor of enhancing fault diversity by checking the quality of the food source selected by an Ant; which can significantly enhance the diversity of faults covered by a test suite. In this regard, this paper proposes certain modifications in the ACO algorithm which determines the quality of the food source before selecting it for enhanced the fault diversity of the test suite and evaluates the performance on the basis of calculation of APFD matrices.

III. Proposed m-ACO Technique

It has been observed that most suitable and practical use of artificial intelligence techniques in software engineering is in the area of software testing. One such technique of artificial intelligence, which finds most suitability for software testing, is ACO technique, which uses a probabilistic approach for solving computational problems. The ACO follows the behavior of ants in natural world. The ACO technique basically is based on the intelligent technique used by the ants to find the food source.

ACO is an optimization algorithm for identification of optimal balance between various variables so as to narrow down the search space in an eventual deduction of the final solution [7]. It is a Meta heuristic swarm intelligence based tool that mimics the behavior of ants searching for food. In the initial phase, the ants move around randomly. However, whenever an ant stumbles on a food source, it goes back to the starting coordinates with simultaneous marking the return path with pheromones. Other ants that come across the pheromone trails follow the path. While doing so; they strengthen the pheromone strength and thus attracting even more ants. This process leads to self-reinforcement of a signal because of ants. Since the ant-colony is applicable on a dynamic system, the ant colony algorithm performs very well in problems of graph with variable topology.

ACO basically follows the food source searching pattern of real ant colonies. It is a meta-heuristic approach which is used to solve many optimization problems in engineering. Ants have the capability to find the optimized path to any particular food source from their home by indirect interaction with all other members of their colony i.e. sending a message to all other members of their colony through environment. This phenomenon is known as *stigmergy*. Decision process of ants is influenced by level of pheromone trails deposited by other ants on the ground because ants move on the path which has higher level of pheromone as compared to other paths. Based on the intelligent and smart way used by ants for food searching, Dorigo et al. in 1990's formulated the usage of Ant Colony Optimization (ACO) algorithm [31]. A number of well known combinatorial optimization problems have been solved using ACO like test data generation, vehicle routing, knapsack problem, travelling salesman problem, telecommunication networks, distributed networks and data mining etc. [31], [32], [33], [34].

This paper proposes a modified Ant Colony Optimization Algorithm (m-ACO) which is a modified form the ACO (Ant Colony Optimization) in the sense that as soon as real 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. The following pseudo code describes the behavior of modified ants in m-ACO algorithm.

Pseudo code for the modified Ant Colony Optimization (m-ACO):

- Create an ensemble set of code modules with variable number and type of faults.
- Randomly assign direction and starting point location.
    a. Position (P) = Random X; Random Y (within the max-X and max-Y)
- Assign working loop do
- Identify the putative solutions for free ants (find the nearest food source -- based on distance)
- Assign the food source a quality indicators index
    a. U = Food uniqueness (Unique faults covered by the module)
- Compare this index with a random number to decide whether to return to the base or to move further on for search of better food source
    a. Random number = N*random number index;
- Assign a pheromone level (proportional to the food quality)
    a. Food Quality = NU – RT
    b. Where, N = number of ants that have reached a specific food source
    c. U = Food uniqueness (Unique faults covered by the module)
    d. R = Rate of Pheromone evaporation
    e. T = Time duration required for the journey from food source to the module
- Perform a sub-global search for better source
- Make a decision for onlooker ants (either to begin/continue the search or strengthen the pheromone trail of some other ant).
- Begin search (If food source is exhausted)

a. Decision (D) =Strengthen the trail (If food source is not exhausted)

- Regularly update pheromone values
- Stop when all food sources (code faults / modules) have been exploited
- end do

As evident from the above pseudo code that ants in the m-ACO go to different directions to search for food source every time; which increases the probability of higher coverage of code. Real 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 select only selects quality food source i.e. increased variety of faults covered.

IV. Experimental Evaluation

To conduct the experiment, three examples have been considered. These three examples have been executed using the Perl program based on m-ACO technique for test suite prioritization. The priority calculated on the basis of time dependent rate calculation was used as validation parameter. The m-ACO technique for test suite prioritization (encoded in Perl program), in parallel, calculated the suitability of each node for on the basis of optimal values of the code covered, faults detected and time used for execution. The artificial ants in a pseudo-random manner approached the fault carrying modules and determined the suitability. A pheromone factor was scored to attract other ant like processes. Only those pheromone trails that belonged to the most suitable code module became stronger in later iterations by progressive visitations of more ants like processes. The pheromone trail of less visited modules becomes weaker by a constant rate. Ants may still visit these progressively weakening trails but with lower probability.

The main criterion used for prioritizing test cases in this proposed m-ACO technique is based on achieving complete or maximum fault coverage. Then the results were analyzed using APFD (Average Percentage of Fault Detected) metric. Elbaum et al. has developed APFD metric which has been used to quantify the objective of maximizing fault detection rate using a combination of test suites [21]. APFD metric is used to estimate the average fault detection rate per percentage of execution of test suite. Notion for APFD calculations are:

$$APFD = 1 - \frac{TF_1 + TF_2 \ldots . TF_m}{mn} + \frac{1}{2n} \qquad (1)$$

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 [17][18].

A. Problems and Results: Case Study1

The problem taken is a "College program for admission in courses". One of its modules has a test suite having ten test cases which covers ten faults. The input condition for the test suite has total 10 test cases having an initial prioritization order defined as {N1-> N2-> N3-> N4-> N5-> N6-> N7-> N8-> N9-> N10} and 10 faults detected {F1-F10}. The table I described below depicts the faults detected (FD) and the execution time (ET) of every test case.

TABLE I

FAULTS DETECTED BY TEST SUITE FOR CASE STUDY1

Test Cases / Faults	N 1	N 2	N 3	N 4	N 5	N 6	N 7	N 8	N 9	N 10
F1	*					*				
F2				*			*	*	*	
F3		*			*	*				*
F4							*			
F5		*						*	*	
F6				*						
F7				*	*					
F8		*	*							
F9						*				
F10	*									*
FD	2	3	1	3	2	3	2	2	2	2
ET	5	7	11	4	10	12	6	15	8	9

The above mentioned test suite was executed 500 times on the Perl program based on proposed m-ACO prioritization technique .The best prioritization order for test suite in example 2 is N4-> N2-> N1-> N7-> N6 having execution time of 34 units, which appeared 86 percent of the times. VNi which is rate of fault detection is calculated by dividing number of faults revealed by execution time.

VN1=2/5=0.4, VN2=3/7=0.42, VN3=1/11=0.09, VN4=3/4=0.75, VN5=2/10=0.2, VN6=3/12=0.25, VN7=2/6=0.33 VN8=2/15=0.133, VN9=2/8=0.25, VN10=2/9=0.22

Priority is set in decreasing order of VNi values because priority of the test case is directly proportional to the rate of fault detection of a test case. Hence the prioritized test suite is: {N4->N2->N1->N7->N6->N9->N10->N5->N8->N3}.

APFD for prioritized test suite after putting the values in the above described formula comes out to be 0.81.

APFD=1.05-0.24=0.81

The APFD of the non-prioritized order is 0.72.

APFD=1.05-0.33=0.72

However, when the same test suite for above mentioned example is executed using the proposed m-ACO technique,

yields APFD value equals to 0.87**.** The results indicated that execution of 50% test cases in prioritized order covers 100% faults for case study1.

B. Problems and Results: Case Study2

The problem taken is "Library Management System". One of its modules has a test suite with five test cases which covers five faults. Input condition for the test suite has total 5 test cases having an initial prioritization order defined as {N1-> N2-> N3-> N4-> N5} and 5 faults detected {F1-F5}. The table 2 described below depicts the faults detected (FD) and the execution time (ET) of every test case.

TABLE II
FAULTS DETECTED BY TEST SUITE FOR CASE STUDY2

Test Cases / Faults	N1	N2	N3	N4	N5
F1	*		*		
F2	*				*
F3					*
F4	*	*	*		*
F5			*	*	*
FD	3	1	3	1	4
ET	12.2	10	10.67	7	9.97

mes on tl tization technique .The best prioritization order for test suite in example 3 is N5-> N3 having execution time of 20.67 units, which appeared 87 percent of the times.

The VNi calculations are:

VN1=3/12.2=0.25, VN2=1/10=0.1,

VN3=3/10.67=0.28,VN4=1/7=0.14

VN5=4/9.97=0.40

Hence the prioritized test suite is: {N5-> N3-> N1-> N4-> N2}. APFD for prioritized test suite after putting the values in the above described formula comes out to be 0.86.The APFD of the non-prioritized order is 0.66. However, when the same test suite for above mentioned example is executed using the proposed m-ACO technique, yields APFD value equals to 0.88. The results indicated that execution of 20% test cases in prioritized order covers 100% faults for case study2.

C. Problems and Results: Case Study3

The problem taken is "Hotel Reservation". The table 3 described below depicts the faults detected (FD) and the execution time (ET) of every test case.

TABLE III
FAULTS DETECTED BY TEST SUITE FOR CASE STUDY3

Test Cases / Faults	N1	N2	N3	N4	N5	N6	N7	N8	N9
F1	*	*	*	*	*	*	*	*	
F2	*	*						*	
F3	*		*			*			*
F4				*				*	
F5	*		*	*		*			
FD	4	2	3	3	1	3	1	3	1
ET	11.5	11.5	12.33	10.66	15	8.33	15	10	11

the Perl program based on proposed m-ACO prioritization technique. The optimal test suite prioritization order for example 1 was similar to the best solution (i.e. N8, N6) having execution time of 18.33 units, which appeared 83 percent of the times. It is evident from the formula of APFD that value of APFD can be only computed when advance information about execution time of faults is available. So, APFD metrics are only used for evaluation purpose.

VN1=4/11.5=0.34, VN2=2/11.5=0.17, VN3=3/12.33=0.24, VN4=3/10.66=0.28, VN5=1/15=0.067, VN6=3/8.33=0.36, VN7=1/15=0.067, VN8=3/10=0.3, VN9=1/11=0.09

Hence the Prioritized test suite is: {N6->N1->N8->N4->N3->N2->N9->N5->N7}. APFD for Prioritized test suite comes out to be 0.86.The APFD of the Non-prioritized order is 0.88.However, when the same test suite for above mentioned example is executed using the proposed m-ACO technique; yields APFD value equals to 0.87.This is a special case as the first test case covers 80% of faults, consequently the performance of non-random, non-prioritized (linear arrangement of Test cases) test cases deliver a high performance. However such cases are only fluke. The results indicated that execution of 30% test cases in prioritized order covers 100% faults for case study3.

V. CONCLUSION AND FUTURE WORK

Regression testing is the authentication of previously functional software as to whether it remains so after a change. Regression testing is time and money intensive process. Test Suite Prioritization is very productive and realistic regression testing technique to minimize the quantity of test cases in a test suite by re-scheduling test cases in the priority order which effectively increases the probability of code coverage, rate of fault detection through APFD metric. We have presented a new algorithm m-ACO which is modified version of Ant colony optimization for the test case prioritization. m-ACO works by altering the food source selection criteria of natural ants. The testing of the algorithm on various problems clearly demonstrates its power. The future work on the paper will focus on comparison of the proposed technique against some latest nature inspired technique for performance evaluation.

REFERENCES

[1.] K. Onoma, W.T. Tsai, M. Poonawala and H. Suganuma. "Regression testing in an Industrial Environment". *Communications. Of ACM*, Vol.41, No. 5, pp 81–86, 1988.

[2.] B. Beizer. "Software Testing Techniques". Van Nostrand Reinhold, New York, NY, 1990.

[3.] H. Leung and L. White. "Insights into Regression Testing". *Proceedings of IEEE International Conference on Software Maintenance*, pp 60–69, 1989[Online].

[4.] G. Myers. "The Art of Software Testing", NY,USA: John Wiley, 1979

[5.] A. P. Mathur. "Foundations of software testing". China Machine Press, 2008.

[6.] S. Yoo and M. Harman. "Regression Testing Minimisation, Selection and Prioritization : A survey". *Journal of software testing , Verification and Reliability,* Vol. 22, No. 2, pp. 67-120, 2012.

[7.] M. Dorigo, V. Maniezzo and A. Colorni. "The Ant System: Optimization by a colony of cooperating agents". *IEEE Transactions on Systems, Man, and Cybernetics Part B: Cybernetics*. Vol.26, No.1, pp 29–41, 1996.

[8.] G. Rothermel, R. H. Untch and M.J. Harrold. "Test Case Prioritization: An Empirical Study". *Proceedings of the International Conference on Software Maintenance*, Oxford, UK, pp. 179-188, 1999.

[9.] Z. Li, M. Harman and R. M. Hierons. "Search algorithms for regression test case prioritization". *IEEE Transactions on Software Engineering*, San Francisco, CA, USA, pp. 225-237, 2007.

[10.] AL-Salami and M.A. Nada. "Evolutionary Algorithm Definition". *American Journal of Engineering and Applied Science Publications*, Vol. 2, No. 4, pp.789-795, 2010.

[11.] N. Bryson. "A goal programming method for generating priorities vectors". *Journal of Operational Research Society,* England, pp. 641-648, 1995.

[12.] G. Crawford and C. Williams. "A note on the analysis of subjective judgment matrices". *Journal of Mathematical Psychology, The Rand Corporation*, USA, pp. 387-405, 1985.

[13.] Y. Singh, A. Kaur and B. Suri. "Regression Test Selection and Prioritization Using Variables: Analysis And Experimentation", New Age International Publishers, New Delhi, pp. 1-15, 2008.

[14.] W Wong, J. Horgan, S. London and H. Agrawal. "A study of effective regression testing in practice". *Proceedings of IEEE Eighth International Symposium on Software Reliability Engineering.* pp. 264-274, 1997.

[15.] P. R. Srivastava. "Application of Genetic Algorithms in Software Testing," *International Journal of Software Engineering and it's Application(IJSEA) -Science & Engineering Research Support Society, Republic of Korea, ISSN: 1738-9984,* Vol. 3, No. 4, pp. 87-96, 2009

[16.] P. R. Srivastava, K.M. Baby and G. Raghurama. "An approach of optimal path generation using ant colony optimization". *Proceedings of IEEE International Conference TENCON 2009,*pp. 1-6, 2009.

[17.] P. R. Srivastava, K.M. Baby. "Automated Software Testing using Meta-Heuristic Technique based on an Ant Colony Optimization". IEEE International Symposium on Electronic System Design ,pp 235-240, 2010.

[18.] P.R. Srivastava, A. Vijay, B. Barukha, P. S. Sengar, and R. Sharma. "An Optimized technique for Test Case Generation and Prioritization Using Tabu Search and Data Clustering". Source available on DBLP and SCOUPS.

[19.] D. Jeyamala and V. Mohan. "ABC-Artificial Bee Colony Optimization based Test Suite Optimization Technique". *International Journal of Software Engg.,* Vol. 2, No. 2, pp. 1-33, 2009.

[20.] D. Jayamala and V. Mohan. "Quality Improvement and Optimization of Test cases- A hybrid genetic algorithm based approach". *ACM SIGSOFT Software Engg. Notes,* Vol. 35, No. 3, pp. 1-14, 2010

[21.] S. Elbaum, A. Malishevsky and G.Rothermel. "Test case prioritization: A family of empirical studies". *IEEE Transactions on Software Engineering,* Vol. 28, No. 2, pp 159-182, 2002