ISSN (Print) : 0974-6846
Indian Journal of Science and Technology, *Vol 9(19), DOI: 10.17485/ijst/2016/v9i19/89458, May 2016* ISSN (Online) : 0974-5645

An Ant Colony Algorithm to Prioritize the Regression Test Cases of Object-Oriented Programs

Mohapatra Sudhir Kumar[1] and Prasad Srinivas[2]

[1]SOA University, Near PNB Bank Jagmohan Nagar, Khandagiri, Bhubaneswar - 751030, Odisha, India; sudhirmohapatra@hotmail.com
[2]Department of Computer Science & Engineering, GMRIT, GMR Nagar, Srikakulam District, Razam - 532127, Andhra Pradesh, India; srinivas_prasad@hotmail.com

Abstract

Background/Objectives: The limited resources force to choose an effective prioritization technique, which makes an ordering of the test case so that the most suitable test case will execute first. **Methods/Statistical Analysis:** Regression test case prioritization using Ant colony optimization for the object-oriented program is proposed in this paper. Using the previous version test pool a complete graph is generated. The node of the graph represents test case and their fault exposing potential is used by the ant to select a prioritization sequence. **Findings:** The effectiveness of the test case ordering is measured using APFD (Average percentage of Faults Detected). Experiments on three object oriented subject programs are performed to judge the said approach. The empirical results indicate that the algorithm implemented using ant colony optimization gives higher APFD value than the random techniques. **Applications/Improvements:** This technique may be used by the quality assurance team for prioritizing test case as it space and time complexity is less as compared to random ordering.

Keywords: Test Case Prioritization, Regression Testing, Ant Colony Optimization, APFD, Software Testing

1. Introduction

The quality software system can't be maintained to be high without rigorously developing testing methodologies[1,2]. The importance of regression testing is speedily growing as the size and complexness of recent software system grows. In software system development area, in view of the character of software system quality and rising stress on software system product, test suit prioritization is of much importance and significant to the business.

A major percentage of development cost is invested in software. It's tough for the tester to create the product 100 percent bugs free in limited given time[3]. As 365 days the product is usually used, the tester is required to make functionalities of the product bug[4,5]. To achieve 100% correctness, execution of all the test case is not practically possible. It is so as we have a limited resource constraint like time. To overcome the separtial test cases are selected which may find the most number of faults within the given version of the software system[6]. It is also required to prioritized the test case for early detection of a maximum fault and the criteria for prioritization may be code coverage, execution time. Test case prioritization[7-9] is all about making an order of test cases, so the test cases with the highest priority, consistent with some fitness metric, are executed first. Rothermel[8] outline the test case prioritization problem and describe many problems relevant to its solution.

Rothermel[8] define the test case prioritization problem and describe several issues relevant to its solution. The test case prioritization problem is defined (by Rothermel et al.) as follows:

** Author for correspondence*

The Test Case Prioritization (TCP) Problem: Given: T, a test suite; PT, the set of permutations of T; f, a function from PT to the real numbers.

Problem: Find T' ∈ PT such that (∀ T" (T" ∈ PT) (T" ≠ T') [f (T") ≥ (T")])

Here, PT represents the set of all possible prioritization (orderings) of T and f is a function that, applied to any such ordering, yields an award value for that ordering.

Meta heuristic search techniques[10] are high-level frameworks that utilize the automatic detection of heuristics so as to search out solutions to combinatorial problems at an inexpensive procedure price. Test case prioritization which is a type of scheduling problem can be solved by Genetic Algorithm effectively[11].

In this paper, focuses is on fault exposing potential of the test case along with time constraint to prioritize test cases. With totally different objective functions, techniques can have totally different complexity and search-space characteristics. Given a function that assesses the fault exposing the potential of the test case, a cost effective solution to the test case prioritization problem would provide a cost-effective answer to the problem of the knapsack, that is understood to be NP-hard[12].

2. Related Work

Uma Maheswari[13] proposed a GA and SA based algorithm for test case prioritization and applied the algorithm for object-oriented JAVA program. In[14] Rothermel et al. formally outlined the test suite prioritization drawback and by experiments investigated six prioritization techniques. Four of the techniques were supported the coverage of either statements or branches for a program and 2 of the techniques were supported the calculable ability to reveal faults. Sasikala[15] use realistic load aspect to prioritize test case in regression testing. Mayan[16] use fuzzy logic for automatic test case generation. This technique gives an outline how to use soft computing in software testing problem. In[17], Srivastava and Thiagarajan studied a prioritization technique supported the changes that are created by the program. Their technique orders the given test cases to maximally cover the affected elements of the program in order that defects are likely to be found quickly and inexpensively.

Panigrahi C, Mall R[18,19] proposed S-RTP and H-RTP which determine the affected nodes in ESDG model based on an analysis of control and data dependencies as well as dependencies arising from object relations and then prioritizes regression test cases based on the number of affected nodes exercised by a test case.

The use of greedy algorithms (total and additional) for regression TCP has been widely studied in the literature[8]. However, results obtained from empirical studies carried out by Rothermel et al. indicate that the greedy strategies may not always produce the optimal ordering of test cases. To prioritize regression test cases, Li et al.[20] further proposed other greedy strategies such as the 2-optimal strategy and two meta-heuristic search strategies (Hill-climbing and Genetic algorithm).

Jeffrey and Gupta[21] proposed an approach to prioritize regression test cases based on coverage of a relevant slice of the output of a test case. They defined a relevant slice as the set of statements that influence or can influence the output of a program when running on a regression test case[21]. The main aim of their prioritization technique was to achieve higher rates of fault detection.

Smith A, Geiger J, Kapfhammer GM, Soffa ML[22] use call tree for TCP. Mohapatra, S. K., Prasad, S[23] proposed a genetic algorithm based TCP where code coverage and severity of the test case is taken into consideration.

3. Ant-based Prioritization Technique

The set of test case i.e T={t1,t2,t3,…..,tm} and the set of fault present in the object oriented program F={f1,f2,f3,…….,fn}. The relation between T and F is that each test case ti detects some set of fault let f where f ϵ F. An undirected graph G is constructed in which all vertices represented the test cases. The graph is a complete graph where |V|=|T|. Initially, all ants start from their corresponding vertices. Let the set of ant A={A1,A2,…, An} and |A|≥ |T|.

The solution will start by positioning the ant in a corresponding vertex. Individually each ant searches the best path by adding new edges to its path. For doing this the Choose_adjacent function is used. This function will select the edge with the highest pheromone deposit. If all the all connected edges have same pheromone value then a random edge is selected. The selection of edges Eij when

the current position of the ant is vertex i and next vertex selected is j from all the adjacent vertex of i. this selection is done using the formula 1.

$$Eij = \begin{cases} \text{Random Edge} & \text{If pheromone is same} \\ \text{Edge With highest pheromone} & \text{Otherwise} \end{cases} \quad (1)$$

In single iteration, each ant selects a set of test case which represented by a vertex of the graph. Let the test cases selected by ant Ai is Ti, then Ti ϵ T. Each ant will stop exploring new path once his Ti covers 100% fault. Once each ant selects test case (As test case is represented as a vertex in the graph) which satisfy all requirement, the next job of the algorithm is to select the path with minimum cost i.e. the path whose execution time is minimum. The pheromone on that path is updated and evaporation is done using function Update_pheromone. This function adds 1 to all the edge as pheromone deposit and evaporates 10% of the current pheromone on the best path. This process continues till no update in path possible. That means when in successive iteration the best-selected path is not changed the algorithm stop further execution. The algorithm and its notations are given below.

Notations used in the algorithm

Pi : Selected path of ant Ai
Ti : Test case represented by a vertex of the graph.
Fc : Fault covered.
V: temporarily selected vertex.
Ai : Ant i present in stared his movement from vertex from vertex represented by test case Ti
Timei: Time of execution of ant i
Ev: Adjacent edges of vertex V

Input: Test Case Represented by the complete graph.
Output: A set of prioritized test case.

ACO-Pri (Test Case Represented by Complete Graph)
{
While No updation in the best path do
For i = 1 to n
$P_i = P_i \cup \{T_i\}$
$V=T_i$
$Time_i$ =Time to execute (T_i)
While fc=|F| do
t = Choose_Adjacent (A_i, V)
$P_i = P_i \cup \{t\}$
$Time_i = Time_i$ + Time to execute (t)
$V=T_i$
End
End
T_{min} = Min{P1,P2,P3,….,Pn according to { $Time_1$, $Time_2$, $Time_3$, ……,$Time_n$}
Update_pheromone(T_{min})
For i = 1 to n
$P_i = \phi$
End
Add the rest of test case in any order
}
Choose_Adjacent(A_i, V)
{
Ev={Set of all edges connected to adjacent of V)
If (Equal Pheromone at Ev)
Select Random edge
Else
Select edge with max pheromone
}
Update_pheromone(T_{min})
{
Update pheromone in the all edge associated with T_{min}
Evaporate 10% of pheromone
}

4. Theoretical Example

Table 1 contains some test case and corresponding fault detected by the test case. This information has collected a priory in the first implementation of the object-oriented programs. From the table the complete graph let G is created which is display in figure 1 in the graph |V|=|T|. From each vertex of the graph a single or more ants will start exploring the optimal path. In this example, the no of ant will be equal to a number of vertex i.e. |A|=|V|.

Table 1. Test case and the fault detected by them with execution time

Test case \ Faults	Time	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8
T_1	6		▪		▪			▪	
T_2	4	▪		▪					
T_3	8	▪				▪		▪	▪
T_4	9		▪		▪				
T_5	3			▪			▪		
T_6	2	▪						▪	

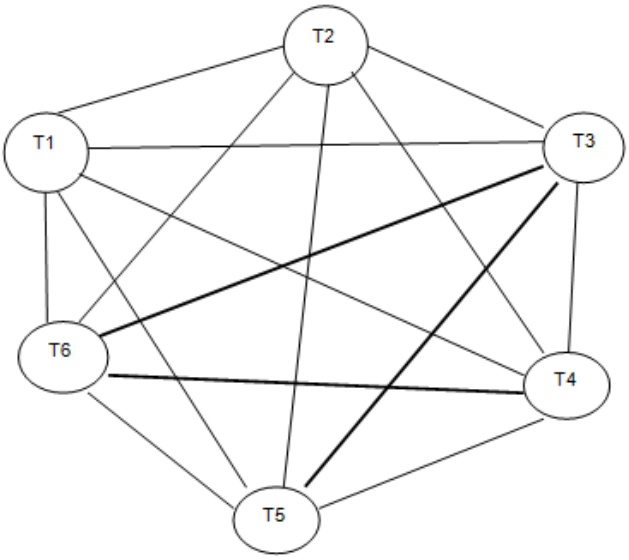


Figure 1. The complete graph generated from Table 1.

Here six ant start from the six vertexes of the complete graph. After the first iteration, each ant selects an optimal path represented in table 2. Out of the entire six paths selected, the path selected by ant A4 is optimal. The time of execution of all the test case selected is 22 which is minimum among all the six ants.

Table 2. Detail of ant position after 1st iteration

A N T							% Fault Covered
	Test case selected by ant after 1st iteration						Total Time
A_1	Test case	T_1	T_3	T6	T2	T5	100
	Time	6	8	2	4	3	23
A_2	Test case	T2	T5	T4	T6	T3	100
	Time	4	3	9	2	8	26
A_3	Test case	T3	T1	T5	T6	T4	100
	Time	8	6	3	2	9	28
A_4	Test case	T4	T6	T3	T5		100
	Time	9	2	8	3		22
A_5	Test case	T5	T6	T1	T3	T4	100
	Time	3	2	6	8	9	28
A_6	Test case	T6	T4	T1	T5	T3	100
	Time	2	9	6	3	8	27

The pheromone along the path T4-T6-T3-T5 is updated. Initially, the cost of all the edge is "0", so 1 is added to the best path and 10% of the pheromone will be lost due to evaporation. The final graph after initial iteration is denoted in Figure 2.

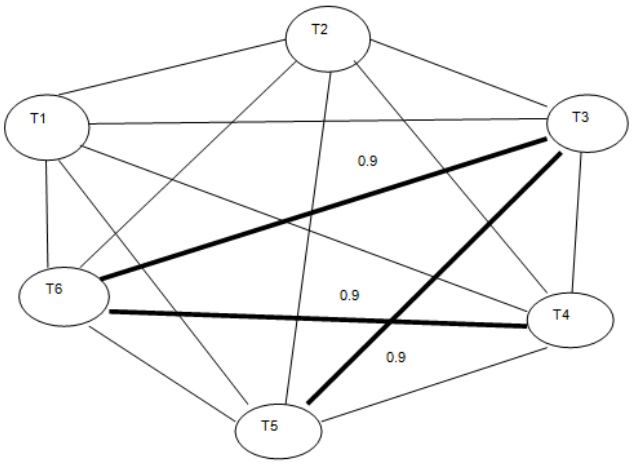


Figure 2. The complete graph after 1st iteration with pheromone update.

In the next iteration again all the ant search an optimal path in the graph with available pheromone deposit. Out of the entire selected path, the path selected by ant A3 is optimal. It selects T3-T5-T4 vertex whose execution time will be 20. The pheromone along the path is updated. When the ant start execution from vertex T3 the pheromone at adjacent edges are e36=0.9, e35=0.9, e34=0, e32=0, e31=0. From all this adjacent edge ant will select edge with the highest pheromone deposit. The edge e36=0.9 and e35=0.9 are with the same pheromone so randomly it select e35=0.9.

When the pheromone on this selected path is updated for edge e35 the update value is e35 =0.9 + 1=1.9. The evaporation will be 10% so at last pheromone deposit is 1.71. Accordingly rests of the edges are updated. The graph and table is represented in Table 3 and Figure 3.

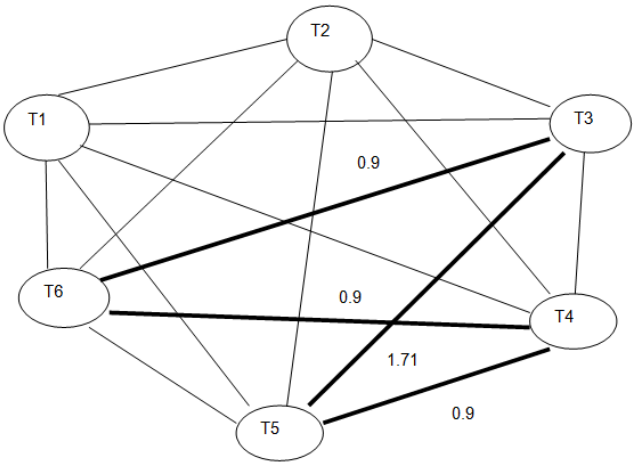


Figure 3. The complete graph after 2nd iteration with pheromone update.

Table 3. Detail of ant position after 2nd iteration

A N T							%Fault Covered
	Test case selected by ant after1st iteration						Total Time
A1	Test case	T1	T3	T6	T4	T5	100
	Time	6	8	2	9	3	28
A2	Test case	T2	T1	T6	T5	T3	100
	Time	4	6	2	3	8	22
A3	Test case	T3	T5	T4			100
	Time	8	3	9			20
A4	Test case	T4	T6	T3	T5		100
	Time	9	2	8	3		22
A5	Test case	T5	T2	T3	T6	T4	100
	Time	3	4	8	2	9	25
A6	Test case	T6	T4	T2	T3	T5	100
	Time	2	9	4	8	2	25

The 3rd iteration of the algorithm not able to produce a new optimal path as given in Table 4. The path selected in this phase is T3-T5-T4 which is same as the previous iteration, so the algorithm stops here and the optimal path produce is T3-T5-T4 with execution time 20.

Table 4. Detail of ant position after 3rd iteration

A N T							% Fault Covered
	Test case selected by ant after 1st iteration						Total Time
A_1	Test case	T1	T2	T5	T3		100
	Time	6	4	3	8		21
A_2	Test case	T2	T4	T6	T3	T5	100
	Time	4	9	2	8	3	26
A_3	Test case	T3	T5	T4			100
	Time	8	3	9			20
A_4	Test case	T4	T6	T5	T3		100
	Time	9	2	3	8		22
A_5	Test case	T5	T3	T6	T4		100
	Time	3	8	2	9		22
A_6	Test case	T6	T3	T5	T4		100
	Time	2	8	3	9		22

Once the sequence is generated, the rest of the test case which are not part of the above sequence can be added in any order. In the above example, the priorities sequence is {T3-T5-T4} follow by any order of {T1-T2-T6}.

5. Experimental Result

Several empirical studies have undertaken to check the effectiveness of our new technique. The technique used for comparison is random technique. Following is two research question, based on which we carried out our experiment.

5.1 Research Questions

Q1: Can the ordersequence technique raise the pace of fault detection more significantly than the random technique.

Q2: Can the ordersequence technique is efficient in terms of space and time complexity.

5.2 Subject Program and Test Case

Table 5 shows the details of subject programs. It also shows the collected test case-requirement matrices. In the table, the 1st column lists all the subject programs. The 2nd column shows the number of lines of code (LOC) of each subject program. The third column lists the size of the corresponding subject program's test suite pool. Here T X R denotes the number of all the test cases and a number of test requirements. Total three programs were considered all of whose code range from 1425 to 3095 LOC. The Java program taken for experiments are Automata lift controller (ALC), Traffic signal controlling (TSC), stock index prediction (STOCK), developed by MCA students of SOA University.

Table 5. Programs used in experiment

Program	Source file (LOC)	Test suite pool (T X R)	Mutation fault	Total statement coverage
ALC	1864	169 X 98	520	79.2%
TSC	2987	228 X 96	891	58.1 %
STOCK	3095	397 X 128	993	67.2%

5.3 Acquisition of Useful Information of Test Case

We use Emma, and ant tool to collect the coverage information of an SUT (System Under Test). The information collected are the code coverage, time of execution of the test case. The details information is given in the table.

5.4 Experimental Setup

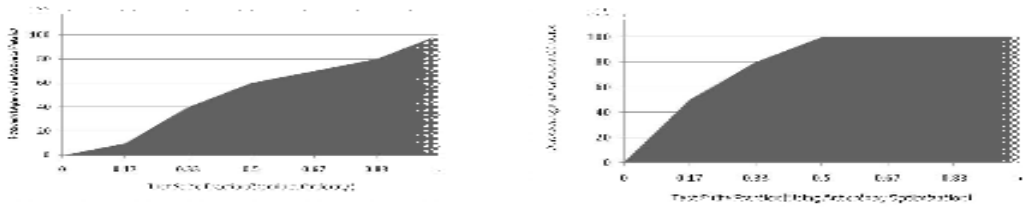
The three techniques are implemented using MATLAB. Once program specific information is gather the result is used by MATLAB to generate optimal ordering. These techniques are implemented on a PC with Intel core i5 2.40 GHz CPU and 4 GB M memory running the Windows 7 operating system.

Q1: In order to compare the rate of fault detection, we used APFD metric. Higher APFD value means faster fault detection rate.

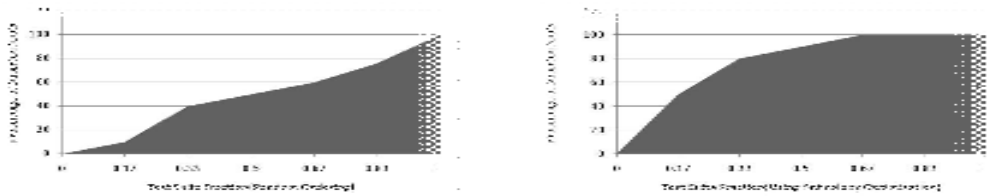
The APFD metric was planned by Rothermel et al.[8] to measure average rate of fault detection. It is used in papers by researcher[21,24] of TCP for calculation of effectiveness of the prioritization ordering. A test suite average rate of fault detection is calculated by taking the weighted average of the quantity of faults detected throughout the execution of the program. A higher APFD value represents quicker fault detection rate, where the APFD value lies between 0 to 1.

The calculation of APFD can be understood by the following illustration. Let T be the initial test suite containing n test cases, and let F be a collection of m faults revealed by T. Let O_0 be an ordering of T. In T_0, let TF_i be the primary test case that reveals a fault i. Then the APFD metric for test suite O_0 will be obtained by using the equation[18]:

$$APFD = 1 - \frac{TF_1 + TF_2 + \ldots\ldots\ldots + TF_n}{n * m} + \frac{1}{2n} \quad (2)$$



APFD calculation of Automata lift controller (ALC) using random and ant colony based prioritization



APFD calculation of Traffic signal controlling (TSC) using random and ant colony based prioritization

APFD calculation of stock index prediction (STOCK) using random and ant colony based prioritization

Figure 4. Result of APFD on the all three subject program.

The above figure 4 shows the graph of the all three subject program. The 1st figure of each subject program represents APFD graph of random ordering which achieves 100% APFD value after execution of 100% of the test case. But when we go for Ant Colony optimization (Present in the second figure of each subject program) after execution of 50% of test suit fraction 100% APDF is achieve. The ant technique detects fault earlier than random ordering.

Q2: In this section, the algorithm's efficiency in terms of space and time complexity is detected.

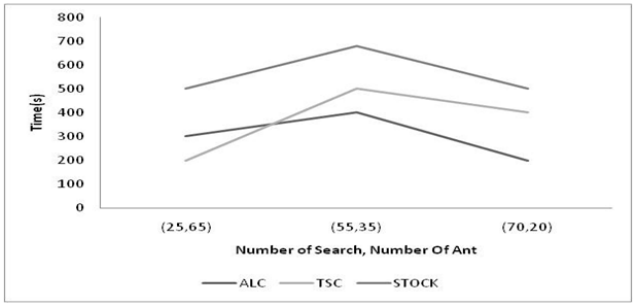


Figure 5. No. of search and no of Ant used per search.

In the above figure 5 x-axis represents (Number of searches, Number of ants), y-axis denotes execution time of the Ant algorithm. The maximum time taken by the algorithms is 11.66 minute for the stock program. The algorithm uses 100KB of the memory shown by the MATLAB implementation of the program.

6. Conclusions

In this paper, an Ant based algorithm is implemented to prioritize test case. The experimental analysis demonstrates that the approach can create an ordering of the test case with better APFD value by consuming a reasonable amount of time and memory. This technique detects bugs in the early stage of execution which can be use full in early termination of regression testing due to resource constraint. Further, research includes the improvement of the performance for real life large object-oriented application. We can also introduce some implementation techniques that are commonly used by other test case prioritization technique and implementing it in parallel.

7. References

1. Ostrand T, Weyuker E and Bell R. Where the Bugs Are. Boston, MA: Proceedings of the ACM SIGSOFT International Symposium on Software Testing and Analysis. 2004 Jul; p. 86-96.
2. Ahmed A. New York: Auerbach Publications: Software Testing as a Service. 2009.
3. Ramler R, Biffl S and Grunbacher P. Value-Based Management of Software Testing, Book Chapter.
4. Boehm B and Huang L. Value-Based Software Engineering: A Case Study. IEEE Computer. 2003 Mar; 36:33-41.
5. Zhang L, Hou SS, Guo C, Xie T and Mei H. Chicago, Illinois, USA: Time Aware Test-Case Prioritization using Integer Linear Programming, ISSTA'09. 2009 Jul.
6. Ashraf E, Rauf A and Mahmood K. Value based Regression Test Case Prioritization. San Francisco, USA: WCECS 2012, Proceedings of the World Congress on Engineering and Computer Science 2012. 2012 October 24-26; I.
7. Rothermel G, Untch R, Chu C and Harrold MJ. Test Case Prioritization: An Empirical Study. Proc. Int'l Conf. Software Maintenance. 1999 Sept; p. 179-88.
8. Rothermel G, Untch R, Chu C and Harrold MJ. Prioritizing Test Cases for Regression Testing. IEEE Trans. Software Eng. 2001 Oct; 27(10):929-48.
9. Wong WE, Horgan JR, London S and Agrawal H. A Study of Effective Regression Testing in Practice. Proc. Eighth Int'l Symp. Software Reliability Eng. 1997 Nov; p. 230-38.
10. John Wiley & Sons: Modern Heuristic Techniques for Combinatorial Problems, C.R. Reeves, ed. 1993.
11. Husbands P. Genetic Algorithms for Scheduling. Artificial Intelligence and the Simulation of Behaviour (AISB) Quarterly. 1994; (89).
12. Garey M and Johnson DS. Freeman: Computers and Intractability: A Guide to the Theory of NP-Completeness. 1979.
13. Uma Maheswari R, Jeya Mala D. Combined Genetic and Simulated Annealing Approach for Test Case Prioritization. Indian Journal of Science and Technology. 2015; 8(35):1-5.
14. Elbaum S, Malishevsky AG and Rothermel G. Test Case Prioritization: A Family of Empirical Studies. IEEE Trans. Software Eng. 2002 Feb; 28(2):159-82.
15. Chandu PMSS, Sasikala T. Implementation of Regression Testing of Test Case Prioritization. Indian Journal of Science and Technology. 2015 Apr; 8(S8). Doi: 10.17485/ijst/2015/v8iS8/61922.
16. Bandaru Ramya, Albert Mayan J. Novel Approach for Whole Test Suite Generation using Metamorphic Relations. Indian Journal of Science and Technology. 2016; 9(10):1-7.
17. Srivastava and Thiagarajan J. Effectively Prioritizing Tests in Development Environment. Proc. ACM SIGSOFT Int'l Symp. Software Testing and Analysis (ISSTA '02). 2002; p. 97-106.
18. Panigrahi C, Mall R. An approach to prioritize regression

test cases of object-oriented programs. J CSI Trans ICT (Springer). 2013; Doi: 10.1007/s40012-013-0011-7.
19. Panigrahi C, Mall R. A heuristic-based regression test case prioritization approach for object-oriented programs. Innovations Syst Softw Eng. 2014; 10:155-63. Doi: 10.1007/s11334-013-0221-z.
20. Li Z, Harman M, Hierons R. Search algorithms for regression test case prioritization. IEEE Trans Softw Eng 2007; 33(4):225–37
21. Jeffrey D, Gupta N. Experiments with test case prioritization using relevant slices. J Syst Softw 2008; 81:196–221.
22. Smith A, Geiger J, Kapfhammer GM, Soffa ML. Atlanta, Georgia, USA: Test suite reduction and prioritization with call trees. In: Proceedings of ASE'07. 2007.
23. Mohapatra SK, Prasad S. Evolutionary Search Algorithms for Test Case Prioritization. IEEE, International Conference on Machine Intelligence and Research Advancement (ICMIRA). 2013; 115-19. Doi: 10.1109/ICMIRA.2013.29.
24. Elbaum S, Malishevsky A, Rothermel G. Incorporating varying test costs and fault severities into test case prioritization. Ontario: Proceedings of the 23rd International Conference on Software Engineering, 2001; p. 329-38.