International Journal of Engineering & Technology, 7 (4) (2018) 3135-3141

International Journal of Engineering & Technology

Website: www.sciencepubco.com/index.php/IJET
doi: 10.14419/ijet.v7i4.18626
Research paper

Experimental Analysis of "BA-TPF" Technique for Regression Test Optimization

Sandeep Dalal [1], Sudhir [2*], Kamna Solanki [1]

[1] *Assistant Professor, Maharshi Dayanand University, Rohtak, India*
[2] *Research Scholar, Maharshi Dayanand University, Rohtak, India*
**Corresponding author E-mail: sudhir.gehlawat1234@gmail.com*

Abstract

Regression testing focuses on finding those newer faults that may get introduced in the software during the process of fixing the older faults. Therefore, it becomes mandatory to re-test the software system after modifications and amendments to detect these undesired faults. The most important observation in this regard is to understand the fact that it becomes impractical for the software testing team to run the entire set of test cases again during regression testing phase due to cost, time and resource constraints. The ultimate way to deal this complexity of regression testing is to run the optimized set of test cases that can serve the purpose of regression testing without compromising the quality of the test suite. A novel technique "BA-TPF" has been proposed recently in this regard for test suite optimization in regression testing. This paper describes the experimental evaluation of the proposed technique and performance of the proposed technique has been evaluated using TCP, PRT and APPC metrices.

Keywords: *Regression Testing; Software Testing; Systematic Literature Study; Test Case Prioritization.*

1. Introduction

"Regression testing is careful retesting of a system or component to verify that modifications have not caused undesirable effects and that the system or component still complies with its specified requirements" [1]. Therefore, a system needs to be re-tested using an optimized test suite for verifying that the older faults have been fixed and ensuring that intrusion of newer faults did not occur during software modifications [2]. Therefore, effective regression testing is extremely crucial and important for developing a quality software as it consumes 50-60% resources of the software development [3]. Regression testing performance highly depends upon the efficiency and efficacy of the test suite. An optimized test suite during regression testing will serve the purpose of regression in minimum time, effort and resources without compromising the quality of the test suite. The techniques for regression testing are categorized as test suite minimization (eliminating the redundant and least useful test cases), test case selection (filtering the test cases based on some criteria), test case prioritization (re-scheduling the sequence of execution of the test cases) [4][5][6][7][8][9][34][35][36].

2. Proposed "BA-TPF" technique for test suite optimization

Numerous researchers have proposed test suite optimization techniques for regression testing using nature inspired algorithms like ACO [10-16], genetic algorithm and bat algorithm [17 - 21]. Some techniques have been developed using hybridization of BCO, GA and ACO algorithms [21-25]. Other researchers have developed test suite optimization techniques using diverse criteria to achieve higher path coverage, higher requirements coverage or time aware optimization [26-32]. Dalal et. al. recently proposed an approach named "BA-TPF" for test suite optimization which basically works on five layers [33]. First layer represents the user requirements. Second layer represents the user requirements into UML (Unified Modelling Language) state-chart diagram. Afterwards, third layer performs the conversion of UML state-chart diagrams into IG (intermediate Graph). Fourth layer applies the Bat Algorithm (BA) on the generated graph for generation of test sequences. Fifth layer deals with measurement and calculations of TPF (Test Path Fitness) values for test suite optimization. Higher the value of TPF, higher is the priority of the test case in a prioritized test sequence. As shown in figure 1, the steps for calculations of TPF are described below:

The test sequences generated after application of Bat algorithm are taken as input and value of ASCP (All State Coverage Percentage) Metric and ATCP (All Transition Coverage Percentage) metric is calculated for each generated test sequence or test path.

All-State Coverage Percentage (ASCP) Metric: All-state coverage metric deals with measurement of the coverage of states. One can apply ASCP to the test-model and can achieve full coverage in case of at least one traversal of each node of UML state chart diagram.

ASCP= (No. of States Visited/Total No. of States) *100

All-Transition Coverage Percentage (ATCP) Metric: All-transition coverage metric deals with measurement of coverage of transition in a test model. Full coverage of ATCP can be in case of at least one traversal of every transition of UML state chart diagram.

ATCP= (No. of Transitions Visited/Total No. of Transitions) *100

Copyright © 2018 Sandeep Dalal et. al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

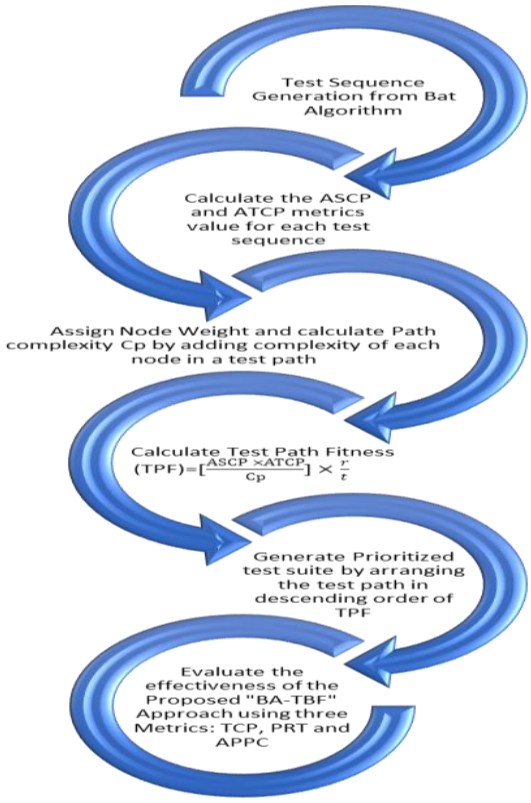


Fig. 1: Layout of Steps for Proposed "BA-TPF" Technique.

Path Complexity (C_p) for each test sequence is calculated using adding the node complexity of each node in a path using the formulae:

Node Complexity=Node Weight + (Fan-In *Fan-Out)

The Test Path Fitness (TPF) for each test path is calculated using the following:

$$\text{Test Path Fitness (TPF)} = [\frac{\text{ASCP} \times \text{ATCP}}{\text{Cp}}] \times \frac{r}{t} \quad (1)$$

Where

ASCP= All-State Coverage Percentage (ASCP) Metric Value

ATCP= All-Transition Coverage Percentage (ATCP) Metric

Cp =Path Complexity

r=Unique Transition Coverage Constant

t=Repeated Transition Coverage Constant

r is the unique transition coverage constant and its value is decided based upon the number of unique paths (transitions) covered by the test case. Default value of r is set as 1. If a test case covers at least one transition (path) uniquely (i.e. the path is not covered by any other test case), then value of r is taken as 2. t is the repeated transition coverage constant and its value is decided based upon the number of repeated paths (transitions) covered by the test case. Default value of t is set as 1. If a test case covers only those transitions that has already been covered by the prior test cases, then value of t is taken as 2.
A prioritized test sequence is generated by arranging the test sequences in descending order of the TPF values. The test sequence possessing the highest value is assigned the highest priority. The performance of the Prioritized test sequence is evaluated using the three metrices namely: TCP (Test Suite Percentage for Complete Path Coverage), PRT (Percentage of Repeated Transitions), APPC (Average Percentage of Path Covered).

3. Experimental evaluation of the proposed technique

This section describes the basic workflow of the proposed "BA-TBF" approach using case study of ATM (Automatic Teller Machine). The changing values of the parameters considered for implementation of the proposed approach are measured and calculations during each iteration of the algorithm are depicted. The fitness function applied during generation of test sequences by bat algorithm focus on achieving the optimal state and transition coverage so as to cover optimal path first.

3.1. Case Study1- ATM System

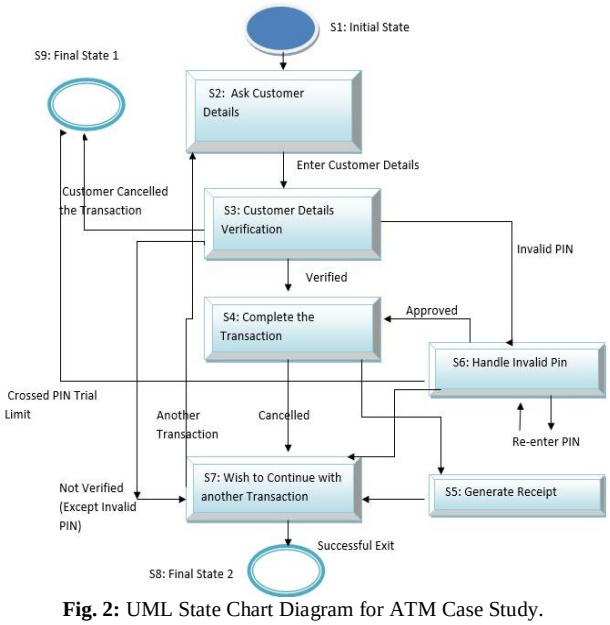


Fig. 2: UML State Chart Diagram for ATM Case Study.

As shown in figure 2, the UML state chart diagram of the present case study of ATM, the bank initially verifies the end-user transaction details. A read receipt is generated, in case the transaction details are approved by the bank. Otherwise, the end-user is asked to again re-submit the correct login details. The invalid pin entered by the user is also dealt with by the system.

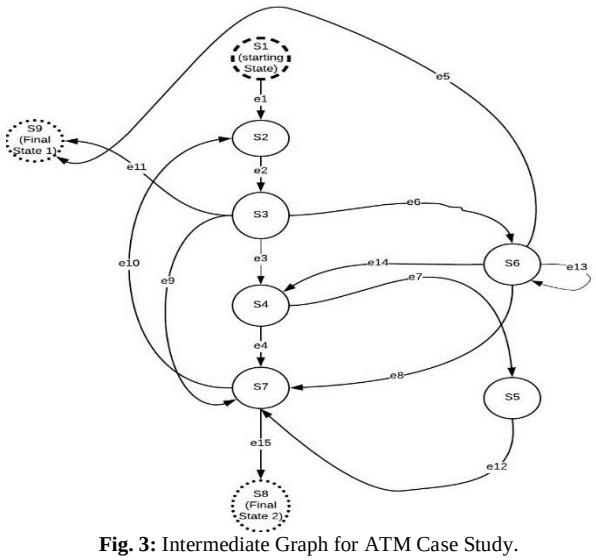


Fig. 3: Intermediate Graph for ATM Case Study.

Each transition is an edge from one state to other state as shown in figure 3. Each transition has one pre-vertex state and one post vertex state in intermediate graph. There exists one complete cycle e2>e3>e4>e10: S2>S3>S4>S7>S2 (Complete loop) in the intermediate graph.

The intermediate graph helps to better understand the transitions and states of UML state chart diagram in a convenient way. UML state chart diagram helps to understand and calculate All-state Coverage metrics and Intermediate graph helps to calculate the All-transitions coverage metrics. Table 1 depicts all the transitions and the respective pre-vertex and post-vertex node of each transition.

Consider the iterations of bat cycle shown in table 2. The current state and first pre-vertex for e1 transition is S1. Now, S2 can be reached only through S1, therefore S2 is considered as next optimal state after S1. Now S2 serves as the current state. Progressing the same way, S3 is also covered. Four bats are generated from state S3 as there are four outbound transitions from S3 namely, e3, e6, e9 and e11 as shown in figure 3. Fitness function for each possible next state is calculated using the formulae:

$$\text{Fitness Function } \phi = [1/(1\text{-IF}) * (v_j/f_j)] * \mu \quad (2)$$

Where $\mu = CF \times IF$

CF (Chance Factor) = (1/no. of final states)

IF (Inclination Factor) = (fan-in×.75) + (fan-out×.25) in range [0-1]

$$F_j = f_{lowest} + (f_{highest} - f_{lowest}) * \mu \quad (3)$$

Fj denotes the bat frequency to search its pray.

$$V_j^t = V_j^{t-1+} (p_j^t - p^*) f \quad (4)$$

where V_j^t is velocity of bat at time t and it increases as the bat moves near the pray and p* is the local optimal solution.

$$P_{new} = (€ \times Lt) + P_{old} \quad (5)$$

Where pnew is the new position and € is random integer between 0 and 1.

Lt is the loudness of the bat which decreases as the bat moves near its pray

$L_j^{t+1} = Lj^t \times ĉ$ where ĉ is a constant (taken as 0.85)

$f_{min} = 1$, $f_{max} = 10$.

Among various probable next states, the fitness function(ϕ) value is observed for each state. State S4 has minimum value of ϕ among all, therefore S4 becomes the next current state. In this way, value of fitness function is calculated for every node and the node possessing minimum value of fitness function serves as the next current state. At the end, one complete test path is produced.

Table 2 shows the full sequence of one test path starting from primary (start) to ending states. The test path described in this case study shows self-loops transitions (e13) and transition cycles (e2>e3>e4>e10) to depict the worth and usefulness of the proposed algorithm. The presence of a self-loop transition on a state makes the priority of the node as highest, otherwise the self-loop may either enhance un-necessary repetitions or it may leave out as un-traversed. If there are multiple outbound transitions from a state, then the transition which have already been traversed must be blocked to sidestep un-necessary repetitions in upcoming iterations.

e1>e2>e3>e4>e10>e2>e6>e13>e8>e15 is the final transition of test sequence produced. Moving ahead in the same direction, all other test paths generated are described as: (e1>e2>e3>e7>e12>e15), (e1>e2>e9>e15), (e1>e2>e11), (e1>e2>e6>e5), (e1>e2>e3>e4>e15), (e1>e2>e6>e14>e4>e15), (e1>e2>e6>e14>e7>e12>e15). Now, it's clear that the proposed approach produces the test sequences with minimized repetitions and better path coverage.

After test case sequence generation using "BA" (Bat Algorithm) is complete, next step is to measure the "Test Path Fitness" (TPF) using the formulae stated in equation 1. To measure and calculate TPF value, the variables used in the formulae of TPF used are calculated. Table 3 depicts the calculation of ASCP and ATCP metrices. Table 4 depicts the weight assignment to nodes for calculation of node complexity. Node complexity is calculated in table 5 and test path complexity calculations by adding the node complexities is depicted in table 6.

Table 7 actually shows the calculations made to measure the value of TPF. Value of r (unique transition coverage constant) has been taken as 2 in TC1, TC3, TC4 and TC5 as there exists at least one unique transition in each test case. Value of r is taken as 2 in TC1, TC3, TC4 and TC5 as the transition e8, e10 and e13 are covered uniquely in TC1, transition e9 is covered uniquely in TC3, transition e11 is covered uniquely in TC4 and transition e5 is covered uniquely in TC5.Now, the prioritized test sequence is generated by arranging the test cases in descending order of Test Path Fitness (TPF) value as shown in table 8. Value of t is taken as 2 in TC6 and TC8 as these test case covers only those transitions that has already been covered by the prior test cases.

So, the prioritized test sequence generated is TC1>TC3>TC5>TC4>TC2>TC7>TC8>TC6 using the descending order of TPF value i.e. from the most-fit path to the least-fit path as shown in table 8. Higher value of TPF implies higher fitness of test case in the prioritized test sequence.

Table 1: Transitions and Respective Nodes Covered

Transition	States	Transition	States
e1	e1: S1 >S2	e8	e8: S6> S7
e2	e2: S2> S3	e9	e9: S3> S7
e3	e3: S3>S4	e10	e10: S7> S2
e4	e4: S4>S7	e11	e11: S3>S9
e5	e5: S6> S9	e12	e12: S5>S7
e6	e6: S3> S6	e13	e13: S6>S6
e7	e7: S4>S5	e14	e14: S6>S4
		e15	e15: S7>S8

Table 2: Bat Traversal Iterations for One Cycle

Iteration	Current state	No. of Bats	Fan-out-Next State	Fitness function (ϕ)	Optimal State	Transition Covered
1	S1	1	1-S2	2.786	S2	e1
2	S2	1	1-S3	9.342	S3	e2
3	S3	4	4- S4, S6, S7, S9	14.37, 19.78, 23.98, 16.56	S4	e3
4	S4	2	2-S5, S7	27.67, 19.76	S7	e4
5	S7	2	2-S2, S8	17.98, 28.79	S2	e10
6	S2	1	1-S3	67.56	S3	e2
7	S3	3	3-S6, S7, S9	98.98, 107.87, 102.76	S6	e6
8	S6	1	1-S6	116.43	S6	e13
9	S6	3	3-S4, S7, S9	68.96, 35.87, 56.89	S7	e8

10	S7	1	1-S8	99.38	S8	e15

Table 3: Metrics Calculations for Test Sequence

Test Case	Path Generated	ASCP Metric	ATCP Metric
TC1	e1>e2>e3>e4>e10>e2>e6>e13>e8>e15	(7/9)*100=77.8	(10/15)*100=66.7
TC2	e1>e2>e3>e7>e12>e15	(7/9)*100=77.8	(6/15)*100=40
TC3	e1>e2>e9>e15	(5/9)*100=55.6	(4/15)*100=26.6
TC4	e1>e2>e11	(4/9)*100=44.4	(3/15)*100=20
TC5	e1>e2>e6>e5	(5/9)*100=55.6	(4/15)*100=26.6
TC6	e1>e2>e3>e4>e15	(6/9)*100=66.7	(5/15)*100=33.3
TC7	e1>e2>e6>e14>e4>e15	(7/9) *100=77.8	(6/15) *100=40
TC8	e1>e2>e6>e14>e7>e12>e15	(8/9) *100=88.8	(7/15) *100=46.6

Table 4: Node Weight Assignment Table

Node Covered	K	Size S	Node Weight
S9	9	10	1
S8	8	9	2
S7	7	8	3
S7 or S6 or S9	6	7	4
S7 or S5	5	6	5
S4	4	5	6
S4 or S7 or S6 or S9	3	4	7
S3	2	3	8
S2	1	2	9
S1	0	1	10

Table 5: Calculation of Node Complexity

Node Covered	Node Weight	Fan-In*Fan-Out	Node Complexity= Node Weight + (Fan-In *Fan-Out)
S1	10	0	10+0=10
S2	9	2	9+2=11
S3	8	4	8+4=12
S4	7+6=13	2	13+2=15
S5	5	1	5+1=6
S6	7+4=11	4	11+4=15
S7	3+4+5+7=19	4	19+4=23
S8	2	0	2+0=2
S9	1+4+7=12	0	12+0=12

Table 6: Calculation of Test Path Complexity

Test Case	Test Path Generated	Test Path Complexity
TC1	e1>e2>e3>e4>e10>e2>e6>e13>e8>e15	10+11+12+15+15+23+2=88
TC2	e1>e2>e3>e7>e12>e15	10+11+12+15+6+23+2=79
TC3	e1>e2>e9>e15	10+11+12+23+2=58
TC4	e1>e2>e11	10+11+12+12=45
TC5	e1>e2>e6>e5	10+11+12+15+12=60
TC6	e1>e2>e3>e4>e15	10+11+12+15+23+2=73
TC7	e1>e2>e6>e14>e4>e15	10+11+12+15+15+23+2=88
TC8	e1>e2>e6>e14>e7>e12>e15	10+11+12+15+15+6+23+2=94

Table 7: Test Path Fitness Calculation for Each Path

Test Case	Path Generated	ASCP Metric	ATCP Metric	Path Complexity (C_p)	Test Path Fitness
TC1	e1>e2> e3>e4> e10>e2> e6>e13>e8>e15	77.8	66.7	88	[(77.8×66.7)/88] ×(2/1) = 117.92
TC2	e1>e2> e3>e7> e12>e15	77.8	40	79	[(77.8×40)/79] ×(1/1) =39.3
TC3	e1>e2> e9>e15	55.6	26.6	58	[(55.6×26.6)/58] ×(2/1)=50.98
TC4	e1>e2> e11	44.4	20	45	[(44.4×20)/45] ×(2/1)= 39.46
TC5	e1>e2> e6>e5	55.6	26.6	60	[(55.6×26.6)/60] ×(2/1)= 49.2
TC6	e1>e2> e3>e4> e15	66.7	33.3	73	[(66.7×33.3)/73] ×(1/2)=15.21
TC7	e1>e2> e6>e14> e4>e15	77.8	40	88	[(77.8×40)/88] ×(1/1) =35.36
TC8	e1>e2> e6>e14> e7>e12> e15	88.8	46.6	94	(4138.08/94) × (1/2) = 22.01

Table 8: Prioritized Test Sequences

Test Case	Path Generated	Test Path Fitness (TPF) Value	Position in Prioritized Sequence
TC1	e1>e2>e3>e4>e10>e2>e6>e13>e8>e15	117.92	1
TC3	e1>e2>e9>e15	50.98	2
TC5	e1>e2>e6>e5	49.2	3
TC4	e1>e2>e11	39.46	4
TC2	e1>e2>e3>e7>e12>e15	39.3	5
TC7	e1>e2>e6>e14>e4>e15	35.36	6
TC8	e1>e2>e6>e14>e7>e12>e15	22.01	7
TC6	e1>e2>e3>e4>e15	15.21	8

3.2. Experimental Evaluation of proposed "BA-TPF" Approach for ATM case study

Experimental Analysis of the proposed approach has been conducted to measure the TCP (Test Suite Percentage for Complete Path Coverage) Metric. A test sequence with lower value of TCP metric is considered as best and cost-effective because it needs lesser test cases to be executed without compromising the path coverage capability. Minimum value of TCP implies achieving complete path coverage with minimum time and minimum efforts as minimum test cases execution is required.

TCP = (No. of test cases required for complete path coverage)/ (total no. of test cases)

TC1>TC3>TC5>TC4>TC2>TC7>TC8>TC6 sequence in prioritized order requires the execution of only first six test cases out of total eight test cases to achieve complete path coverage as the test cases TC8 and TC6 only covers those transitions (or paths) that have been already covered by the prior test cases. So, TCP for the prioritized sequence can be calculated as:

TCP = (6/8) *100= 75 %

Therefore, execution of 75% test cases in prioritized order can achieve complete path coverage.

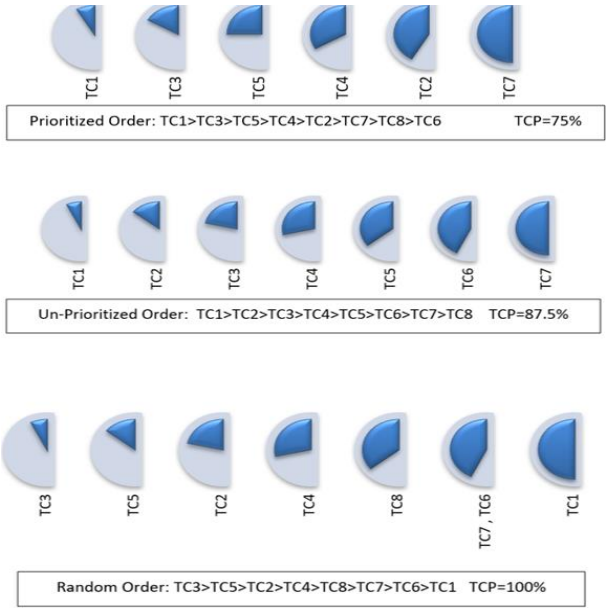


Fig. 4: Test Cases Required for Complete Path Coverage.

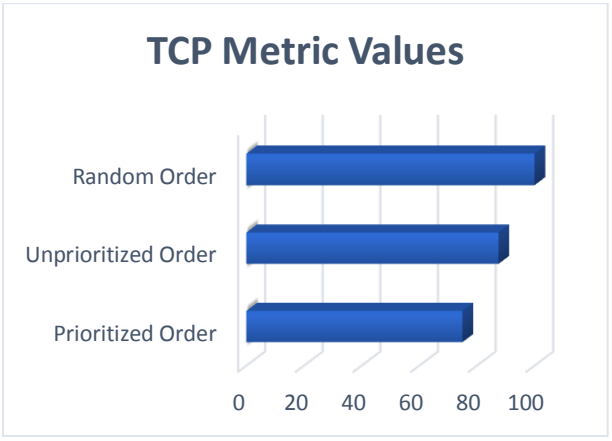
TC1>TC2>TC3>TC4>TC5>TC6>TC7>TC8 sequence of unprioritized order requires the execution of first seven test cases out of total eight test cases to achieve complete path coverage. So, TCP for the un-prioritized sequence can be calculated as:

TCP = (7/8) *100= 87.5 %

It is clear that execution of 87.5% test cases in unprioritized order can achieve complete path coverage. The random prioritized TC3>TC5>TC2>TC4>TC8>TC7>TC6>TC1 test sequence requires execution of all the test cases to achieve complete path coverage. So, the TCP value for the random order sequence can be calculated as:

TCP = (8/8) *100=100 %

Execution of 100% test cases in randomly prioritized order can achieve complete path coverage as shown in figure 4. It is evident from the value of TCP that the prioritized test sequence using the proposed technique is capable of achieving the complete path coverage with minimum number of test cases.



Graph. 1: TCP Metric Representation for Prioritized, Prioritized and Random Order Sequences.

Experimental Analysis of the proposed approach has also been conducted to measure another important metric named as Percent Repetition Transitions (PRT) Metric. PRT metric calculates the percentage of the redundant transitions in a test sequence. Notion for Calculation of PRT can be defined as:

PRT = (No. of repeated transition in a test sequence that achieve complete path coverage)/ (Total no. of test cases required for complete path coverage)

A test sequence with lower value of PRT metric is considered as better as it possesses the lesser redundancy in comparison to the other test sequences. Minimum value of PRT implies achieving complete path coverage with minimum repetitions and redundancy. TC1>TC3>TC5>TC4>TC2>TC7>TC8>TC6 test sequence in prioritized order requires the execution of only first six test cases out of total eight test cases to achieve complete path coverage as the test cases TC8 and TC6 only covers those transitions (or paths) that have been already covered by the prior test cases. To calculate the total no. of repeated transitions in a test sequence, one has to calculate the repeated transitions of individual test case as shown in table below. So, PRT for the prioritized sequence can be calculated as:

PRT = (17/6) *100= 28.33

TC1>TC2>TC3>TC4>TC5>TC6>TC7>TC8 test sequence in unprioritized order requires the execution of first seven test cases out of total eight test cases to achieve complete path coverage as

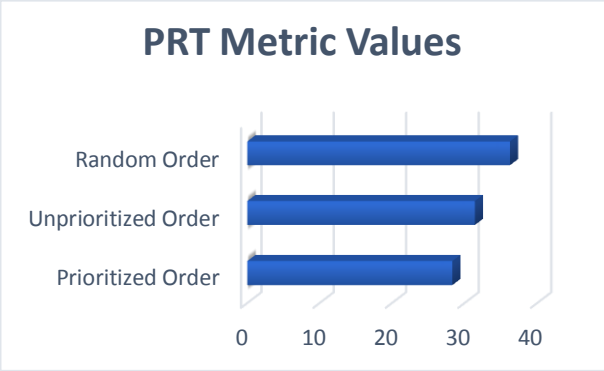
discussed earlier. Repeated transitions for each test case have been discussed in table shown below. So, PRT for the un-prioritized sequence can be calculated as:

PRT = (22/7) *100 = 31.42

TC3>TC5>TC2>TC4>TC8>TC7>TC6>TC1 test sequence in random order requires execution of all the test cases to achieve complete path coverage. So, the PRT value for the random order sequence can be calculated as:

PRT = (29/8) *100 = 36.25

It is evident from the value of TCP that the prioritized test sequence using the proposed technique is capable of achieving the complete path coverage with minimum number of test cases.



Graph. 2: PRT Metric Representation for Prioritized, Unprioritized and Random Order Sequences.

Experimental Analysis of the proposed approach has also been conducted to measure another important metric named as APPC (Average Percentage of Path Covered) Metric

$$APPC = [1 - \frac{TP1+TP2+TP3.....TPm}{mn}] + \frac{1}{2n} \quad (1)$$

Where

n=no. of test cases

m=no. of test paths covered

TP_m= Position of test case that covers the m^{th} test path first

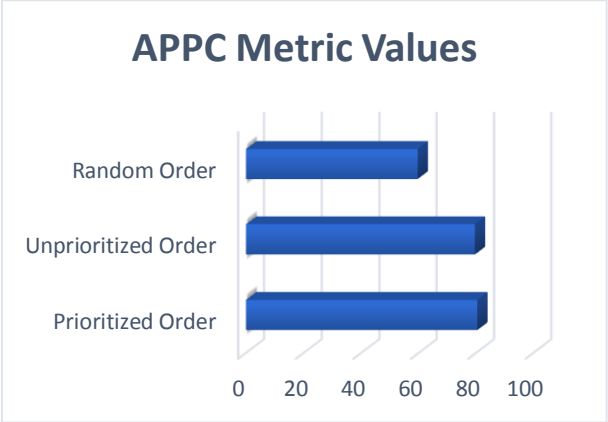
APPC for the unprioritized test suite TC1>TC2>TC3>TC4>TC5>TC6>TC7>TC8 is calculated as:

$$APPC = [1 - \frac{1+1+1+1+5+1+2+1+3+1+4+2+1+7+1}{15\times8}] + \frac{1}{2\times8}$$

$$APPC = [1 - \frac{32}{120}] + \frac{1}{16}$$

APPC= (1-0.267) +0.063= 0.796 = 79.6%

Similarly, Prioritized test sequence TC1>TC3>TC5>TC4>TC2>TC7>TC8>TC6 obtained APPC value as 80.42% and random prioritized sequence TC3>TC5>TC2>TC4>TC8>TC7>TC6>TC1 obtained APPC value 59.6%.



Graph. 3: APPC Metric Representation for Prioritized, Unprioritized and Random Order Sequences.

4. Conclusion

This paper performs the experimental analysis and performance evaluation of the proposed "BA-TPF" technique for regression test optimization. The proposed technique generates the test sequences using BA and then optimizes and prioritize these generated test sequences by arranging the test sequences in descending order of TPF values, which measure the fitness of the path. The case study of ATM has been elaborated in the paper to depict the working of the proposed technique. Experimental evaluation of the proposed approach against un-prioritized and random order prioritization proves that the proposed "BA-TPF" approach diminishes the redundancy ratio, minimizes the test suite size and enhances the path coverage capacity. Future research in this direction will focus on comparative performance evaluation of the proposed technique against contemporary techniques.

References

[1] Leung, H. K., & White, L. (1989, October). Insights into regression testing (software testing). In *Software Maintenance, 1989. Proceedings. Conference on* (pp. 60-69). IEEE.

[2] Rothermel, G., Untch, R. H., Chu, C., & Harrold, M. J. (1999). Test case prioritization: An empirical study. In *Software Maintenance, 1999. (ICSM'99) Proceedings. IEEE International Conference on* (pp. 179-188). IEEE. https://doi.org/10.1109/ICSM.1999.792604.

[3] Rothermel, G., Untch, R. H., Chu, C., & Harrold, M. J. (2001). Prioritizing test cases for regression testing. *IEEE Transactions on software engineering*, *27*(10), 929-948. https://doi.org/10.1109/32.962562.

[4] Yoo, S., & Harman, M. (2012). Regression testing minimization, selection and prioritization: a survey. *Software Testing, Verification and Reliability*, *22*(2), 67-120. https://doi.org/10.1002/stv.430.

[5] Solanki, K., & Singh, Y. (2014). Importance of Selecting Test Cases for Regression Testing. *IOSR Journal of Computer Engineering (IOSRJCE) e-ISSN*, 2278-0661. https://doi.org/10.9790/0661-16444351.

[6] Elbaum, S., Kallakuri, P., Malishevsky, A., Rothermel, G., & Kanduri, S. (2003). Understanding the effects of changes on the cost-effectiveness of regression testing techniques. *Software testing, verification and reliability*, *13*(2), 65-83. https://doi.org/10.1002/stvr.263.

[7] Elbaum, S., Malishevsky, A. G., & Rothermel, G. (2002). Test case prioritization: A family of empirical studies. *IEEE transactions on software engineering*, *28*(2), 159-182. https://doi.org/10.1109/32.988497.

[8] Raju, S., & Uma, G. V. (2012). Factors oriented test case prioritization technique in regression testing using genetic algorithm. *European Journal of Scientific Research*, *74*(3), 389-402.

[9] Solanki, K., Dalal, S., A Literature study of various regression testing approaches. In Computing for Sustainable Global Development (INDIACom), 2018 fifth International Conference on 2018 Mar 15. IEEE. (In Press).

[10] Solanki, K., Singh, Y., & Dalal, S. (2016). A Comparative Evaluation of "m-ACO" Technique for Test Suite Prioritization. *Indian*

Journal of science and technology, *9*(30). https://doi.org/10.17485/ijst/2016/v9i30/86423.

[11] Solanki, K., Singh, Y., & Dalal, S. (2016). Experimental analysis of m-ACO technique for regression testing. *Indian Journal of Science and Technology*, *9*(30). https://doi.org/10.17485/ijst/2016/v9i30/86588.

[12] Suri, B., & Singhal, S. (2011). Implementing ant colony optimization for test case selection and prioritization. *International journal on computer science and engineering*, *3*(5), 1924-1932.

[13] GAO D, Guo X, Zhao L. (2015). Test case prioritization for regression testing based on ant colony optimization. *In Software Engineering and Service Science (ICSESS), 6th IEEE International Conference on* (pp. 275-279). IEEE. https://doi.org/10.1109/ICSESS.2015.7339054.

[14] Solanki, K., Singh, Y., Dalal, S., & Srivastava, P. R. (2016). Test case prioritization: An approach based on modified ant colony optimization. In *Emerging Research in Computing, Information, Communication and Applications* (pp. 213-223). Springer, Singapore.

[15] Suri, B., & Singhal, S. (2011). Analyzing test case selection & prioritization using ACO. *ACM SIGSOFT Software Engineering Notes*, *36*(6), 1-5. https://doi.org/10.1145/2047414.2047431.

[16] Suri, B., & Singhal, S. (2012, September). Literature survey of ant colony optimization in software testing. In *Software Engineering (CONSEG), 2012 CSI Sixth International Conference on* (pp. 1-7). IEEE.

[17] Kaur, A., & Goyal, S. (2011). A genetic algorithm for fault-based regression test case prioritization. *International Journal of Computer Applications*, *32*(8), 975-8887.

[18] Panichella, A., Oliveto, R., Di Penta, M., & De Lucia, A. (2015). Improving multi-objective test case selection by injecting diversity in genetic algorithms. *IEEE Transactions on Software Engineering*, *41*(4), 358-383. https://doi.org/10.1109/TSE.2014.2364175.

[19] Raju, S., & Uma, G. V. (2012). Factors oriented test case prioritization technique in regression testing using genetic algorithm. *European Journal of Scientific Research*, *74*(3), 389-402.

[20] Catal, C. (2012, September). On the application of genetic algorithms for test case prioritization: a systematic literature review. In *Proceedings of the second international workshop on evidential assessment of software technologies* (pp. 9-14). ACM.

[21] Suri, B., Mangal, I., & Srivastava, V. (2011). Regression test suite reduction using a hybrid technique based on BCO and genetic algorithm. *Special Issue of International Journal of Computer Science & Informatics (IJCSI), ISSN (PRINT)*, 2231-5292.

[22] Baudry, B., Fleurey, F., Jézéquel, J. M., & Le Traon, Y. (2005). Automatic test case optimization: A bacteriologic algorithm. *IEEE Software*, *22*(2), 76-82. https://doi.org/10.1109/MS.2005.30.

[23] Dobuneh, M. R. N., Jawawi, D. N., Ghazali, M., & Malakooti, M. V. (2014, September). Development test case prioritization technique in regression testing based on hybrid criteria. In *Software Engineering Conference (MySEC), 2014 8th Malaysian* (pp. 301-305). IEEE.

[24] Silva, D., Rabelo, R., Campanha, M., Neto, P. S., Oliveira, P. A., & Britto, R. (2016, July). A hybrid approach for test case prioritization and selection. In *Evolutionary Computation (CEC), 2016 IEEE Congress on* (pp. 4508-4515). IEEE.

[25] Mayan, J. A., & Ravi, T. (2015). Structural software testing: hybrid algorithm for optimal test sequence selection during regression testing. *International Journal of Engineering and Technology (IJET)*, *7*(1).

[26] Walcott, K. R., Soffa, M. L., Kapfhammer, G. M., & Roos, R. S. (2006, July). Time aware test suite prioritization. In *Proceedings of the 2006 international symposium on Software testing and analysis* (pp. 1-12). ACM.

[27] Hla, K. H. S., Choi, Y., & Park, J. S. (2008, July). Applying particle swarm optimization to prioritizing test cases for embedded real time software retesting. In *Computer and Information Technology Workshops, 2008. CIT Workshops 2008. IEEE 8th International Conference on* (pp. 527-532). IEEE.

[28] Arafeen, M. J., & Do, H. (2013, March). Test case prioritization using requirements-based clustering. In *Software Testing, Verification and Validation (ICST), 2013 IEEE Sixth International Conference on* (pp. 312-321). IEEE.

[29] Krishnamoorthi, R., & Mary, S. S. A. (2009). Factor oriented requirement coverage-based system test case prioritization of new and regression test cases. *Information and Software Technology*, *51*(4), 799-808. https://doi.org/10.1016/j.infsof.2008.08.007.

[30] Kavitha, R., Kavitha, V. R., & Kumar, N. S. (2010, October). Requirement based test case prioritization. In *Communication Control and Computing Technologies (ICCCCT), 2010 IEEE International Conference on* (pp. 826-829). IEEE.

[31] Salehie, M., Li, S., Tahvildari, L., Dara, R., Li, S., & Moore, M. (2011, March). Prioritizing requirements-based regression test cases: A goal-driven practice. In *Software Maintenance and Reengineering (CSMR), 2011 15th European Conference on* (pp. 329-332). IEEE.

[32] Di Nardo, D., Alshahwan, N., Briand, L., & Labiche, Y. (2015). Coverage-based regression test case selection, minimization and prioritization: A case study on an industrial system. *Software Testing, Verification and Reliability*, *25*(4), 371-396. https://doi.org/10.1002/stvr.1572.

[33] Dalal, S. & Sudhir (2018).BA-TPF: A novel approach towards Test suite optimization. *International Journal of Engineering and Technology.In Press.*

[34] Dalal, S., & Solanki, K. (2018). "Performance Analysis of BCO-mGA Technique for Test Case Selection". *Indian Journal of Science and Technology*, *8*(1), pp1-10. https://doi.org/10.17485/ijst/2018/v11i9/110048.

[35] Dalal, S., Chhillar, R. (2013) "Empirical study of root cause analysis of software failure". *ACM SIGSOFT Software Engineering Notes.* 38(4):1–7. https://doi.org/10.1145/2492248.2492263.

[36] Dalal, S., and Chillar, R. (2013). "A Novel Technique for Generation of Test Cases Based on Bee Colony Optimization and Modified Genetic Algorithm". *International Journal of Computer Applications, vol. 68, no. 19, pp. 12-17. https://doi.org/10.5120/11687-7359.*