Volume 4, Issue 5, May 2014 ISSN: 2277 128X

International Journal of Advanced Research in Computer Science and Software Engineering

Research Paper

Available online at: www.ijarcsse.com

Test Case Prioritization for Multiple Processing Queues using Genetic Algorithm

Ritika Jain, Neha Agarwal

Department of Computer Science and Engineering,
Amity School of Engineering and Technology, Amity University,
Noida, India

Abstract-Regression testing is very important as well as expensive task during maintenance activity. There are exhaustive number of new and old test cases to execute while performing regression testing. One effective approach, test case prioritization technique, arrange test cases in an order that increases their ability to meet some testing goals. However, all existing prioritization techniques prioritize test cases in a single set with the assumption that there is only one processor which selects test cases to run. But there may be more than one processor that performs testing. Present work addresses the solution of such problems by 'division and prioritization method' and 'prioritization and division method'. Both of these methods are used to sort test cases into multiple prioritized subsets using genetic algorithm (GA). $APFD_p$ value obtained using these two methods is above 98% with any number of processors and generations. For comparative analysis a random method is also used, wherein test cases are allocated randomly to number of processors to execute in parallel. Comparing the results of random method with above two proposed methods, it can be concluded that, proposed methods gives better prioritized subsets of test cases having $APFD_p$ value more than random method.

Keywords: test case prioritization, multiple processing queue, genetic algorithm, $APFD_p$.

1. Introduction

Regression testing is a quality control measure which ensures that the newly added source code or any modifications to the previously working software compiles comprehensively according to its specified requirements and that unmodified source code has not been affected by the maintenance activity [1]. Test case prioritization is most effective regression testing technique which neither select nor omit any test case from regression test suite and thus reduces the risk of leaving out any important test case from being executed [2-5]. Test case prioritization arranges test cases in such a way that highest priority test case will be executed first than a second highest priority test case will run and so on, in order to achieve required performance goal [6-8].

Literature witnesses various prioritization techniques which commonly include coverage based prioritization techniques [3, 6-7]. All these techniques needed source code to compute the coverage information of test cases in order to assign priority to them. Researcher [9] proposed black box test cases prioritization technique.

In order to prioritize test cases in different way researchers [10-12] consider historical execution data from latest regression testing about test case cost, faults detected by each test case and severity of each fault in order to determine their priority. Researchers [13-14] applied other techniques such as greedy method and genetic algorithm (GA) to prioritize test cases respectively.

All these existing test case prioritization techniques prioritize test cases in a single set with the assumption that there is only one processor which selects test cases to run. But there may be more than one processor that performs testing. Therefore, researcher [15] proposed test case prioritization technique for multiple processing queues working in parallel. They also defined test case prioritization problem in parallel scenarios and gave a new metric, $APFD_p$, to measure the efficiency of test suite prioritized for multiple parallel processors. They prioritize test suite using existing test case prioritization techniques and then partition them into multiple subsets to facilitate testing in parallel environment.

However, in this paper test cases in regression test suite are randomly distributed evenly to all processors assuming that all test cases have same cost and all processors have same ability. Now each processor is going to run genetic algorithm independently on its test subset and obtain its own prioritized suite.

In existing prioritization techniques, prioritization algorithms are executed on single machine and the test suite is exhaustive, that means genetic algorithm requires large number of computations to obtain best prioritized suite.

In present work each processing queue has small subset of test cases to prioritize that means GA requires small number of computations to obtain the best prioritized test sequence. Since total number of regression test cases are partitioned into multiple processing queues, regression testing time also get reduced and ensures that more severe faults gets revealed quickly.

Organization of paper is as follows, section 2 describes prioritization problem for parallel working processors and the $APFD_p$ metric to measure effectiveness of prioritized test suite. Methodology of present work is explained in section 3. Section 4 described the benchmark case considered for the present analysis and finally results and discussions are demonstrated in section 5 followed by conclusions of the study (section 6).

2. Prioritization in Parallel Scenarios

2.1 Prioritization problem

Existing prioritization problem aims at finding a best prioritized suite that maximize a given objective function, but it neither includes more than one processor (working in parallel) in problem formulation nor does it output prioritized suite assigned to multiple processors.

However, this definition does not work when there are many processors in parallel. Researcher [15] provide a definition of test case prioritization problem for such situation.

Definition: Test case prioritization problem in parallel scenarios.

Given: T, a test suite; n, the number of processing queues; PT, the set of permutations of T; CT, the set of permutations of {Tn} where {Tn} is the n-division of PT; f, a function from CT to the real numbers.

Problem: Find $\{T'\} \in CT$ such that

$(\forall \{T''\})(\{T"\} \in CT)(\{T"\} \neq \{T'\})[f(\{T'_i\}) \geq f(\{T"_i\})]$

Here, CT represents the set of all possible division of every element in PT, and f is a measure function that evaluates the effectiveness of total performance of multiple prioritized subsets.

2.2 Metric for measuring efficiency

To measure how quickly a prioritized test suite detects faults (the rate of fault detection of the test suite), an appropriate objective function is required. Existing metrics $APFD$ and $APFD_C$ defined in [3] calculates average percentage of fault detected in a single processor environment. However, this metric is not suitably applicable to prioritization problem in parallel scenarios. Thus, a new metric $APFD_P$ has been defined for such situation [15].

A formulaic presentation of $APFD_P$ metric is, let T be a test suite which was divided into k subsets and each subset T_k contains l_k test cases. Let n be the total number of test cases in test suite T. Let F be a set of m faults revealed by T. Let S_j be the first subset which reveals fault i and $T_{Sj}F_i$ be the first test case in subset j which reveals fault i. Now, $APFD_p$ for test suite T' is given by the equation:

$$APFD_p = \left(1 - \frac{\sum_{i=1}^{m}\sum_{j=1}^{k} T_{sj}\ F_i}{mn} + \frac{1}{2n}\right) x100\ \% \qquad (1)$$

3. Methodology

Qu et al. [15] considered that when there is no real time process to run, existing prioritization techniques can be used to prioritize the test suite as a whole and then they partitioned this prioritized suite into multiple subsets.

In present work all test cases in a test suite are evenly partitioned into multiple processors. It is assumed that cost of all test cases and ability of all processors are same. Also all these processors are working independently.

After partition any existing prioritization technique can be used by processors to prioritize test subset allocated to them. In present work each processor will apply genetic algorithm to prioritize test subset. Though genetic algorithm is stochastic search technique and promises to give better solution which is very near to optimal solution, it requires large number of computations and this computation time is proportional to the size of test suites. Since, here each processing queue has small subset of test cases to prioritize; small number of computations are required to get the best prioritized test sequence. Also, as total number of regression test cases are partitioned into multiple processing queues, regression testing time also get reduced and ensure that more severe faults gets revealed very soon. So the test suite division and prioritization algorithm could be written as shown in following algorithm.

Algorithm: Division and Prioritization Algorithm

Input: T={t_n}(test suite), m(number of processing queues)

Output: $\{T'_i\}$(prioritized subsets)

```
Step:   for i=1 to n                    //for every test case
        j=(i mod m) + 1   //calculate the subset to which test case t_i has to be placed
        T_i += t_i                      //save test case to corresponding subset
        end for
        for i = 1 to m                  //for each processor
        T'_i = GA(T_i)                  //apply genetic algorithm to prioritize corresponding subset
        end for
        return {T'_i}
```

4. Benchmark Case

In the proposed work, the selected program to perform experiment is 'eDiary'. Characteristics of selected program used in the experiment are given below (Table 1).

Table 1 Characteristics of selected program – eDiary.

Name	eDiary
Platform language	C++
Number of statements	429 LOC
Test pool size	90
Number of faults	38

In the experiment, out of 90 test cases only 60 are selected for regression testing and these test cases are prioritized by the proposed technique.

5. Results and discussion

5.1 Experiment 1

In present experiment, random sequence of 60 test cases is portioned equally among number of parallel processors as shown in Tables 2-5.

Table 2 Test suite distribution among 3 parallel processors

Processor number	Test subset
1	1-45-12-16-49-35-58-4-29-24-54-14-25-26-8-18-17-60-51-20
2	3-47-30-36-5-10-56-41-21-32-53-42-38-31-28-55-19-34-23-48
3	15-59-57-7-44-43-2-40-11-33-6-27-37-13-52-39-22-50-9-46

Table 3 Test suite distribution among 4 parallel processors

Processor number	Test subset
1	1-47-57-49-10-2-29-32-6-25-31-52-17-34-9
2	3-59-16-5-43-4-21-33-14-38-13-18-19-50-20
3	15-12-36-44-58-41-11-54-42-37-8-55-22-51-48
4	45-30-7-35-56-40-24-53-27-26-28-39-60-23-46

Table 4 Test suite distribution among 5 parallel processors

Processor number	Test subset
1	1-59-36-35-2-21-54-27-31-18-22-23
2	3-12-7-10-4-11-53-25-13-55-60-9
3	15-30-49-43-41-24-6-38-8-39-34-20
4	45-57-5-58-40-32-14-37-28-17-50-48
5	47-16-44-56-29-33-42-26-52-19-51-46

Table 5 Test suite distribution among 6 parallel processors

Processor number	Test subset
1	1-12-49-58-29-54-25-8-17-51
2	3-30-5-56-21-53-38-28-19-23
3	15-57-44-2-11-6-37-52-22-9
4	45-16-35-4-24-14-26-18-60-20
5	47-36-10-41-32-42-31-55-34-48
6	59-7-43-40-33-27-13-39-50-46

Each processor works independently from each other and prioritize assigned test subset by using genetic algorithm. The crossover probability (CP), mutation probability (MP) and number of generations for GA are assigned to be 100%, 100% and 30 respectively. The fitness function used in GA is additional faults detected by each test case [8]. The final prioritized subset obtained by each processor using GA is given in Tables 6-9.

Table 6 Prioritized test subset of 3 parallel processors using GA

Processor number	Prioritized test subset	$APFD_p$
1	45-25-60-1-51-16-29-35-14-24-49-4-17-26-58-8-18-54-12-20	98.77%
2	19-41-55-38-48-23-56-47-32-31-42-21-30-53-10-5-34-3-36-28	

3	39-2-43-22-46-59-57-9-6-44-15-52-37-13-7-40-33-11-27-50

Table 7 Prioritized test subset of 4 parallel processors using GA

Processor number	Prioritized test subset	$APFD_p$
1	25-1-31-29-34-52-57-32-47-10-9-2-49-17-6	98.99%
2	19-16-43-3-59-38-21-13-20-5-14-18-33-4-50	
3	41-22-48-58-55-51-42-44-36-12-37-8-54-15-11	
4	45-60-39-24-46-35-56-40-28-26-30-7-53-23-27	

Table 8 Prioritized test subset of 5 parallel processors using GA

Processor number	Prioritized test subset	$APFD_p$
1	2-35-54-36-59-27-31-23-21-18-1-22	98.81%
2	25-53-4-55-60-13-7-3-12-10-11-9	
3	39-49-41-6-38-43-20-24-8-34-15-30	
4	45-40-57-37-5-58-48-50-32-14-17-28	
5	19-16-52-29-46-51-42-26-33-44-47-56	

Table 9 Prioritized test subset of 6 parallel processors using GA

Processor number	Prioritized test subset	$APFD_p$
1	25-1-29-17-51-12-58-54-8-49	98.99%
2	19-23-38-30-3-5-21-28-56-53	
3	2-52-22-6-37-15-44-57-9-11	
4	45-20-24-26-4-35-18-14-16-60	
5	31-32-55-41-48-10-42-36-34-47	
6	39-40-43-46-50-13-7-33-59-27	

$APFD_p$ value gives the measurement of effectiveness of test subsets prioritized by parallel processors. From above results, it can be observed that $APFD_p$ value is more than 98% in all the cases.

The number of generations for GA in present study is varied from 10 to 40 while keeping CP and MP 100%. Table 10 depicts the influence of number of generations and number of processors on the value of $APFD_p$.

Table 10 Results of division and prioritization method

No. of generations	$APFD_p$.			
	No. of processors = 3	No. of processors = 4	No. of processors = 5	No. of processors = 6
10	98.55%	98.81%	98.81%	99.03%
20	98.59%	98.55%	98.99%	99.03%
30	98.77%	98.99%	98.81%	98.99%
40	98.55%	98.81%	98.77%	99.03%

From above Table 10, it can be seen that the $APFD_p$ value is more that 98% with any number of processors and generations.

5.2 Experiment 2

In experiment 2, test sequence of 60 test cases is first prioritized using GA with 100% value of CP and MP. The fitness function used in GA is additional fault detected by each test case similar to experiment 1. The prioritized test sequence obtained by applying GA with different number of generations is shown in Table 11.

Table 11 Prioritized test sequence

No. of generations	Test suite	$APFD$
10	45-3-52-16-60-22-39-2-19-25-23-31-35-1-4-26-41-43-8-17-20-21-37-38-40-44-47-49-53-55-56-5-6-7-14-18-28-36-46-34-12-29-32-33-48-51-54-59-27-57-9-58-10-11-15-24-42-50-30-13	96.10
20	45-39-2-52-25-23-31-35-1-4-51-22-41-43-8-17-20-21-37-38-40-44-47-3-19-26-60-5-12-55-34-16-53-13-9-48-28-27-56-46-6-54-15-36-29-7-30-57-11-49-50-10-14-33-18-58-32-59-24-42	96.40
30	45-39-2-23-53-21-6-31-19-41-58-25-35-7-28-1-4-36-22-43-24-17-20-16-37-38-40-5-47-3-52-26-60-44-12-55-34-13-30-48-51-56-8-9-27-46-54-15-29-18-57-11-49-50-10-	96.79

	14-33-32-59-42	
40	45-39-2-23-53-21-6-31-19-41-58-25-35-7-28-1-4-36-22-43-24-17-20-16-37-38-40-5-47-3-52-26-60-44-12-55-34-13-9-49-51-30-48-56-8-27-46-54-15-29-42-57-11-50-10-14-33-32-59-18	96.79

$APFD$ is defined to represent the weighted "Average of the percentage of faults detected" during the execution of the test suite. $APFD$ values range from 0 to 100; higher values imply faster (better) fault detection rates. It is emphasized to note that the value of $APFD$ increases as the number of generations increases upto a certain limit after that it remains constant with increase in number of generation.

This prioritized sequence (Table 11) is then partitioned among number of processors preserving their priority obtained from GA. The $APFD_p$ value of this experiment is depicted in Table 12.

Table 12 Results of prioritization and division method

No. of generations	$APFD_p$.			
	No. of processors = 3	No. of processors = 4	No. of processors = 5	No. of processors = 6
10	98.50%	98.81%	98.85%	98.94%
20	98.72%	98.85%	98.94%	99.03%
30	98.77%	99.03%	99.07%	99.07%
40	98.77%	99.03%	99.07%	99.07%

From Table 12, it can be noticed that the $APFD_p$ value is more that 98% with any number of processors and generations.

5.3 Experiment 3

In this section, equal numbers of test cases are assigned randomly to each processor and then their $APFD_p$ value is observed as depicted in Table 13.

Table 13 Results of random method

No. of processors	$APFD_p$
3	96.44%
4	97.19%
5	97.67%
6	97.58%

From this experiment, it can be seen that the value of $APFD_p$ is less as compared to those obtained from experiment 1 and 2 for any number of processors. Based on these data, it can be concluded that the division and prioritization method (experiment 1) as well as prioritization and division method (experiment 2), both are of benefit for improving fault detection rate when testing in parallel scenario.

6. Conclusions

The test case prioritization problem addresses on searching for a best permutation of test cases to execute. All existing prioritization techniques prioritize test cases for a single machine. These techniques define prioritization problem and metric to measure the efficiency of prioritized test suite for the same. However, these techniques are not applicable to prioritization scenario where multiple processors are involved. So for such scenario, prioritization problem and metric for efficiency measurement ($APFD_p$) is presented in this communication.

To address this problem, two methods - 'division and prioritization method'and 'prioritization and division method' have been proposed to sort test cases into multiple prioritized subsets. Both of these methods are useful in partitioning and prioritizing test suite among multiple processing queues. $APFD_p$ value obtained using these two methods is above 98% with any number of processors and generations.

For comparative analysis an experiment with random method is conducted, to obtain $APFD_p$ values when test cases are allocated randomly to number of processors for executing parallel. Comparing the results of random method with above two proposed methods, it can be concluded that, proposed methods gives better prioritized subsets of test cases having $APFD_p$ value more than random method. Hence proposed methods can help the practitioners to sort test cases for improving the efficiency when working in parallel scenarios.

REFERENCES

1. Leung, H.K.N. and White, L. (1989), 'Insights into regression testing'. Proceedings of the International Conference on Software Maintenance', Miami, Florida, U.S.A., pp. 60-69.
2. Elbaum, S., Malishevsky, A.G. and Rothermel, G. (2000), 'Prioritizing test cases for regression testing'. Proceedings of the 2000 ACM SIGSOFT International Symposium on Software Testing and Analysis, Portland, Oregon, U.S.A., pp. 102-112.

3. Elbaum, S., Malishevsky, A. and Rothermel, G. (2001), 'Incorporating varying test costs and fault severities into test case prioritization'. Proceedings of the 23rd International Conference on Software Engineering, Toronto, Ontario, Canada, pp. 329-338.
4. Rothermel, G., Untch, R.H., Chu, C. and Harrold, M.J. (1999), 'Test case prioritization: an empirical study'. Proceedings of the International Conference on Software Maintenance, pp. 179-188.
5. Rothermel, G., Untch, R.H., Chu, C. and Harrold, M.J. (2001) 'Prioritizing test cases For regression testing', *IEEE Transactions on Software Engineering*, Vol. 27 No. 10, pp. 929-948.
6. Askarunisa, A., Shanmugapriya, L. and Ramaraj, N. (2010) 'Cost and coverage metrics for measuring the effectiveness of test case prioritization techniques', *INFOCOMP Journal of Computer Science*, Vol. 9 No. 1, pp. 43-52.
7. Elbaum, S., Malishevsky, A.G. and Rothermel, G. (2002) 'Test case prioritization: a family of empirical studies', *IEEE Transactions on Software Engineering*, Vol. 28 No. 2, pp. 159-182.
8. Jain, R. and Agarwal, N. (2013) 'Additional fault detection test case prioritization', *BVICAM's International Journal of Information Technology*, Vol. 5 No.2, pp. 623-629.
9. Qu, B., Nie, C., Xu, B. and Zhang, X. (2007) 'Test case prioritization for black box testing', In Proceedings of the international computer software and applications conference, pp. 465-474.
10. Huang, Y.C., Peng, K.L. and Huang, C.Y. (2012) 'A history-based cost-cognizant test case prioritization technique in regression testing', *The Journal of Systems and Software*, Vol. 85 No.3, pp. 626– 637.
11. Khalilian, A., Azgomi, M.A. and Fazlalizadeh, Y. (2012) 'An improved method for test case prioritization by incorporating historical test case data', *Science of Computer Programming*, Vol. 78 No. 1, pp. 93-116.
12. Park, H., Ryu, H. and Baik, J. (2008), 'Historical value based approach for cost cognizant test case prioritization to improve the effectiveness of regression testing'. Proceedings of the 2nd International Conference on Secure System Integration and Reliability Improvement, pp. 39-46.
13. Malhotra, R. and Bharadwaj, A. (2012) 'Test case prioritization using genetic algorithm', *International Journal of Computer Science and Informatics*, Vol. 2 No. 3, pp 63-66.
14. Li, Z., Harman, M. and Heirons, R.M. (2007) 'Search algorithms for regression test case prioritization', IEEE Transactions On software engineering, vol. 33, no. 4, pp. 225-237.
15. Qu, B., Nie, C. and Xu, B. (2008) 'Test case prioritization for multiple processing queues', International symposium on information science and engineering, pp. 646-649.

