Dynamic Partitioning and Additional Branch Coverage for Test Case Selection

Dynamic Partitioning and Additional Branch Coverage for Test Case Selection

Arnaldo Marulitua Sinaga , Arie Satia Dharma, Oscar Hutajulu, Anita Ginting and Gustina Simanjuntak

Faculty of Informatics and Electrical Engineering, Institut Teknologi Del, Toba Samosir 22381, Indonesia

Abstract. Good test cases are those test case with high failure detecting capability. The effectiveness of testing is relied on the test case selection process. The earlier good test cases being selected, the higher the effectiveness of testing. Partitioning is one of the method to improve the effectiveness of test case selection. Dynamic Partitioning is method that has been proved empirically can improve the effectiveness of random selection. This method is proposed based on the intuition that a successful test case has high probability in detecting failure in the next execution. On the other hand, Addition Branch Coverage method has been proposed to improve the effectiveness of testing. It selects test case if it can add the covered branch during test execution. The empirical results show that the Additional Branch Coverage is the best method compare with othe coverage-based method. In this research, the combination of those two methods, Dynamic Partitioning and Additional Branch coverage is proposed with the hypothesis that this combination will combine the advantages of those two methods, and hence produce better method. The effectiveness of the proposed method has been investigated empirically by comparing it with the Random Testing, Dynamic Partitioning and Additional Branch Coverage. The Random Testing usually used as the benchmark in software testing research. The results show that the new method performed more effective that all other studied methods. The combination of Dynamic Partitioning and Additional Branch Coverage improve the effectiveness of test case selection in term of the number of executed test cases to detect all existing failures.

1. Introduction
Software testing is the process of verification of the quality of the software product [1]. This makes software testing has a very important role in software development lifecycle. Software testing is aimed to detect all failures in a software product, and hence they can be fixed sooner before deployed [2,3,4]. However, software testing can be very 5657costly if the test case selection is ineffective and inefficient. It is important to select good test cases earlier to increase the fault detection capability of testing process. Good test case is test cases that has high probability in detecting failure [2,8,11]. The earlier a failure detected the sooner fault can be fixed, thus the cost of testing can be reduced. Therefore, it is important to find effective test case selection strategy. Cai et al [7] introduced Dynamic Partitioning strategy for test case selection. This strategy selects test cases by incorporating online feedback information from testing process. The empirical research show that the Dynamic Partitioning outperformed Random Testing significantly [7,8]. Dynamic partitioning strategy is based on the intuition that the test cases that successfully detected failure previously have higher probability to

detect failure in the next execution [7]. The good test cases are partitioned separately with the poor test cases. The test cases' membership updated dynamically based on the last execution result.

On the other research Rothermel et al. [9] conducted investigation on the test case prioritization by employing coverage information of test cases. They found that the strategy with branch coverage information performed better than all other coverage information. They than proposed new strategy by applying Additional method by using branch coverage information named Additional Branch Coverage. This method performed the best among the studied methods [9]. In Additional Branch Coverage, the selected test case contributes to the completenes of branch coverage. A test case will only be selected if it touch one or more branch when being executed.

Hypothetically, the combination of these two methods, Dynamic Partitioning and Additional Branch Coverage, produce a method with higher fault-detection capability by combining the advantages of those two methods. This research investigates this hypotisis empirically. The effectiveness of the proposed method is measured in term of the number of test cases executed to detect all of existing failure

2. Dynamic Partitioning

As explained in Section 1, Dynamic Partitioning partitioned test cases based on the previous execution results. If a test case previously detects a failure that it is categorized as good test cases, otherwise it is categorized as poor test case. This approach partitions all test cases into three classes. First category is Poor that consists of all test cases that did not detect failure previously. Second category is Fair that consists of all test cases that have not been executed yet. Third category is Good that consists of test cases that detect failure previously. In the testing process, test cases from Good partition has the highest prioritization to be selected, followed by Fair and the Poor comes as the last. The lower prioritized test cases will only be selected if the higher prioritized partition has been empty. An advanced research by using this approach has been conducted by Zhou et al. [8], that is by proposing five versions of Dynamic Partitioning: Dynamic Partitioning with Fixed Membership (DPFM), Dynamic Partitioning with One-step Varying Membership (DP1S), Dynamic Partitioning with Two-step Varying Membership (DP2S), Dynamic Partitioning with No Upgrade (DPNU), and Dynamic Partitioning that Returns to Poor (DPRP). The empirical results show that DP1S is the best method. Therefore, in this research we only consider DP1S as the Dynamic Partitioning method. The membership of test case with DP1S is defined by employing the information from testing process. If a test case from Good partition did not detect a failure in the last execution, then its membership is downgraded one step lower to Fair, on the other hand if a test case from Fair detected a failure in the last execution, then its membership upgraded one level to Good [8].

3. Additional Branch Coverage

Branch is link inter-node which is associated with the branching of true and false of a decision node [10]. Branch Coverage method is proposed to ensure that all brances in the program code of software under test have been executed in the testing process. Branch Coverage is one of test coverage criteria with each of the branch has been touched by the execution of test cases at least once. This coverage method is also known as *Decision Coverage* [4]. Rothermel [9] introduced and investigated several prioritization including Statement Coverage and Branch Coverage. The results show that Branch Coverage performed better than Statement Coverage [9]. Rothermel et al. [9] also proposed Additional method to those prioritization technique. In the Additional method, a test case is prioritized higher when it touches more coverage elements. As an illustration, a program under test P has 5 branches in its program code: B_1, B_2, B_3, B_4, and B_5. The previous execution indicates that B_1, B_2, and B_3 have been executed. Test case T_a touches B_1 and B_3, whereas test case T_b touches B_3 and B_4. Hence test case T_b is prioritized higher than T_a since T_b contributes one branch coverage whereas T_a with no contribution.

4. Dynamic Partitioning with Additional Branch Coverage

As has been explained in Section 1, this research proposes a new method that combine Dynamic Partitioning and Additional Branch Coverage named Dynamic Partitioning with Additional Branch Coverage. This combination is aimed to obtained more effective test case selection method by combining the advantages of the two combined methods. The Additional Branch Coverage is applied to defined the partition of test suite. Each partition contains at least a set of test cases that resulted from one round of Additional Branch Coverage Method. The algorithm of this partition technique is as follow:

1. Select a test case from test suite-set randomly, remove it from test suite-set and put it into active partition
2. Execute it into program under test and recorded the branch coverage into CoverageLog
3. If all branches in the CoverageLog have been covered and all test suite-set empty (all test cases have been partitioned) then stop; If all branches in the CoverageLog have been covered but test suite-set is not empty then change the active partition and go to Step 1; otherwise go to Step 1.

In this research, the number of partition is set to 4. This is defined arbitrarily by following previous research in partition [11].

The partitioned test suite then goes to the testing process that apply Dynamic Partitioning method. As explained in the previous section, the Dynamic Partitioning algorithm used in this research is DP1S. Like in DP1S, each partition is divided into three partitions named Good, Fair, and Poor and each of them has Used and Unused part. All test cases that have been executed are stored in the Used part whereas the test cases that have not been executed are stored in the Unused part. At the beginning all test cases in each partition are stored in Unused part of Fair of each correspondent partition. The testing process with this algorithm is as follows:

1. Select a partition randomly
2. Select a test case randomly from the the Unused part of Good set of the selected partition (If Good set is empty then from Fair set; if Fair set is also empty then from Poor set).
3. Execute the selected test case and move it to Used part of its originated set.
4. If a failure is detected, a correspondent fault is removed from program under test (new version). All test cases in the Used part are moved to their Unused part accordingly (sampling without replacement). If there are still remaining faults in program under test then go to Step 1.
5. If all failures have been detected and all faults have been removed, the stop; otherwise adjust the membership of test case as follows:
    a. If the executed test case is from Good set, it will be moved back to Good set only if it detects a failure in the last execution, otherwise it will be degraded one step to Fair set.
    b. If the executed test case is from Fair set, it will be upgraded one step to Good set only if it detects a failure in the last execution, otherwise it will be degraded one step to Poor set.
    c. If the executed test case is from Poor set, it will be upgraded one step to Fair set only if it detects a failure in the last execution, otherwise it will be remained Poor set.

5. The Experiment

In this research, Space is the program under test in the experiments. Space is a C program that frequently used in software testing research [7,8,9,11]. The instruments of Space are downloaded from Software Infrastructure Repository [1,16], that consists of 13,585 test cases adn 6,199 lines of code. Folowing Zhou et al. [8], the number of faults seeded into Space is 34 faults and the way the faults being seeded into the program is similar with Zhou et al. [8]. In order to enhance the validity of this experiments, the execution of all studied methods is conducted for 500 trials. There are 5 methods implemented in this experiments:

1. Random Testing (RT), the most simple testing strategy. All test cases have the uniform probability to be selected. This method usually used as the benchmark in software testing research.
2. Random Partitioning (RP), all test cases are partitioned into four classes randomly.
3. Dynamic Partitioning (DP), test cases are selected by using Dynamic Partitioning method particularly DP1S.
4. Additional Branch Coverage (ABC), all test cases are partitioned by using Additional Branch algorithm. The test cases are selected from the partitions randomly.
5. Dynamic Partitioning with Additional Branch Coverage (DP-ABC), combining the Dynamic Partitioning and Additional Branch Coverage. All test cases are partitioned into four classes by using Additional Branch Coverage. The partition resulted from the execution of this algorithm is as follows:
   - Partition #1 consists of 3418 test cases
   - Partition #2 consists of 3434 test cases
   - Partition #3 consists of 3343 test cases
   - Partition #4 consists of 3300 test cases

The Dynamic Partitioning is implemented into each of the partition for test case selection and membership adjustment.

The effectiveness of each studied method is measured by using F-measure. F-measure indicates the number of test cases applied to detect the first failure in the testing process. The lower F-measure, the more effective the method being investigated [8,11].

6. Experiment Result and Discussion

The results of the experiments with Space as program under test are presented in Table 1. Those results are obtained from the 500 trials of each studied methods. Table 1 provides a simple statistical analysis of the 500 trials results.

Table 1. The Experiment results with Space

			Methods		
	RT	***RP***	***DP***	***ABP***	***DP- ABP***
Avg	1600.44	837.93	756.87	832.00	735.56
Max	5951.00	2937.00	2576.00	2266.00	2266.00
Min	227.00	169.00	124.00	110.00	111.00
Med	1074.50	739.50	655.00	687.00	696.00
Std. Dev	882.73	470.01	416.80	498.75	342.28

The results show all methods with partitioning and coverage information outperformed the RT in all statistical aspects. For the average, DP-ABC comes as the best method. It executes less number of test cases to detect all failures than all other methods. The average F-measure is 735.55, with the saving of 54.04 % compared to RT. DP-ABC also improves the two original methods: DP with the gain of 2.82 % and the ABC with the gain of 11.59 %. For Maximum, DP-ABC and ABC come as the best method, followed by DP, RP and RT respectively. The same result is indicated by Minimum, DP-ABC and ABC gain almost the same result and come as the best. The DP comes as the best with Median followed by ABC and DP-ABC. The results also indicate that DP-ABC is the most stable method among all studied methods. It has the lowest standard deviation that is 342.28 followed by DP, ABC, RP and RT.

This experimental result has also been analysed more detail using further statistical approaches. The Analysis of Variance (ANOVA) is conducted to examine whether or not the studied methods are all equal [13]. The p-value returned from ANOVA test is less than 0.001, which indicated that the compared methods are significantly different. Further analysis is conducted to compare the performance of the proposed DP-ABC with all other studied methods by using t-test. The t-test results show that DP-ABC outperformed ABC, RP and RT significantly with the p-value returned from the test is smaller than 0.01. However, the p-value of the t-test of DP-ABC and DP is 0.38, indicates they are not significantly different. This result indicates that the advantage of DP contributes more than ABC to the effectiveness of DP-ABC. This finding need to be investigated furthermore, since in this experiment the influence of particular method is not investigated. Further investigation also needed in order to increase the validity of this experiment, that is by applying more program under test with larger in size and complexity and also with larger test suite.

7. Conclusion

This research proposes a test case selection method that combine the Dynamic Partitioning and the Additional Branch Coverage with the intention of improving the effectiveness of software testing by combining the advantages of the two methods. The experimental results show that the new method performed better than the two originated methods. The new method named as Dynamic Partitioning with Additional Branch Coverage (DP-ABC). DP-ABC applied less test cases to detect all existing faults in the program under test. The statistical analysis indicate that DP-ABC has significantly outperformed the other methods except Dynamic Partitioning. The DP-ABC can reach up to more than 50% saving compare to Random Testing. In term of stability, the DP-ABC comes as the most stable methods. This finding agree with the hypotesis in this research.

However, a further investigation is needed to find out how the new methods improve the originated methods Dynamic Partitioning and Additional Branch Coverage. Future research with more program under test and larger test suites is needed.

Acknowledgments

This work was supported in part of IT Del Research Grant

References

[1] Naik, Kshirasagar and Tripathy, Priyadarshi. Software Testing and Quality Assurance-Thory and Practice. New Jersey: Wiley. 2008
[2] Bentley, John E., Bank, Wachovia, and Charlotte, NC. “Software Testing Fundamentals-Concepts, Roles, and Terminology” Kajian, 2003.
[3] Ahamed, Riaz S. S., “Studying the Feasibility and Importance of Software Testing: An Analysis”, International Journal of Engineering Science and Technology, 2009, vol. 1, no. 3, pp. 119-128.
[4] Myers, G. (2012). The Art of Software Testing 3rd. New Jersey: John Wiley & Sons, Inc.
[5] Cem Kaner, “What is a Good Test?” Kajian, Department of Computer Science, Florida Institute of Technology, Florida, 2003.
[6] Pressman, Roger S. Software Engineering-A Proactitioner’s Approach. New York: McGraw-Hill. 2001.
[7] K. -Y. Cai, T. Jing, and C. G. Bai, “Partition testing with dynamic partitioning,” in Proceedings of the 29th Annual International Computer Software and Applications Conference (COMPSAC 2005), vol. 2. IEEE Computer Society Press, July 2005, pp. 113-116.
[8] Z. Q. Zhou, A. Sinaga, L. Zhao, W. Susilo, and K. -Y. Cai, “Improving software testing cost-effectiveness through dynamic partitioning,” in Preceeding of the 9th International Conference on Quality Software (QSIC’09). IEEE Computer Society Press, 2009, pp. 249-258.
[9] G. Rothermel, R. H. Untch, C. Chu, M. J. Harrold, “Prioritizing Test cases For Regression Testing”. IEEE Transaction on Software Engineering, vol. 27, no. 10, October, 2001, pages 929-948.
[10] Wong, W. Eric. ”Controlflow-based Coverage Criteria”, Department of Computer Science, The University of Texas, Dallas, Texas.
[11] A. Sinaga, “On Feedback-Based Software Testing,” Doctor of Philosophy Thesis, Computer Science Department, Univerity of Wollongong, Wollongong, Australia, May 2013.
[12] Software-artifact Infrastructure Repository. http://sir.unl.edu. accessed at 25th November 2015.

[13] Sawyer, S. F. (2009). Analysis of variance: the fundamental concepts. Journal of Manual & Manipulative Therapy, 17(2), 27E-38E.