Applying Feedback Information for Random Partition Testing

Arnaldo Marulitua Sinaga
Faculty of Informatics and Electrical Engineering
Institut Teknologi Del, Indonesia
aldo@del.ac.id

***Abstract*—Random testing is a basic strategy in software testing. Partition testing is another basic testing strategy that separates input domain into disjoint sub domains. However, the effectiveness of random testing and partition testing should be improved by using feedback information. Feedback-controlled testing has been introduced in adaptive testing strategy. Computational overhead become a major issue in adaptive testing strategy. The algorithms for parameter estimation have high computational overhead. In this paper, a simplified parameter estimation is introduced by applying feedback mechanism. Experimental results show that the studied method can improve the cost-effectiveness of random testing with lower computational overhead than the adaptive testing strategy.**

***Index Terms*—random testing, partition testing, feedback mechanism, adaptive testing, software testing.**

I. INTRODUCTION

Software is now widely used in society. The importance of good quality software is emerging. Software testing is a method to verify and validate software quality. However, the cost of software testing can be very expensive. It can be reach up to 50% of the total cost of software cost. Software testing has to be conducted properly and effectively to reduce the cost. Input in testing process is required to be sufficient.

The challenges in software testing become formidable due to the software maintenance. Once software is modified, it has to be re-tested to ensure correctness of that software. Validation of the new version of the software and identification of faults that have been injected into the program code are revealed by the regression testing [1] - [4]. It is necessary to conduct regression testing into software, which has been changed during maintenance even though it is such a high-priced testing. Therefore, more efficient test case selection strategies are needed to improve the cost-effectiveness of software production. By cost effective it is meaning that the number of test cases used in detecting and removing all defects on software under test is minimum. Empirical approaches are essential to evaluate the cost-effectiveness of test case selection.

Random testing is a fundamental technique in software testing. The simplicity and ease of use of random testing makes it the most used strategies in software testing [5]. In fact, random testing uses many test cases to detect failures. The effectiveness of random testing is low because it does not use any information to guide the test case selection or generation [6]. Another basic software testing is partition testing. Separating the program input domain into disjoint sub domains is the characteristic of partition testing [7], [8]. When T be the input of the program under test, a partition of T separates it into disjoint sub domains T_1, T_2, ..., T_n. The purpose of partitioning is to make the division of test cases in such a way that test cases are selected based on the sub domains [8]. The partitioning could have hefty influences on the effectiveness of test case selection in software testing [9].

In partition strategies, the information of the program such as data flow, functional properties, information on commonly occurring errors and combinations are used to determine the partition [7]. This information is obtained during test cases generation and/or partition before testing process execution. Off-line partition consumes time for analysis the entire test cases.

In partition testing, test case chosen and executed randomly [5], [7], [10]. The effectiveness of partition with random testing can be improved in recently developed dynamic partitioning [9]. Cai et al. [9] introduced an algorithm that uses on-line dynamic partition. Differ from the conventional partition strategies, in dynamic partition, test cases are selected dynamically during the testing process, rather than statically before testing. The test cases selection is conducted by employing feedback information from program execution.

In dynamic partitioning, the membership of a test case in a sub domain is dynamically adjusted by using feedback information of test execution from testing process. Experimental results show that the dynamic partitioning approach is much more cost-effective than conventional approaches. The improvement and enhancement of those strategies is needed.

Differ from dynamic partitioning, in this paper, we investigate the effectiveness of partitioning by incorporating the feedback information from the execution of each partition. Cai [11] proposed adaptive testing that uses testing history data to define action to execute test cases from the selected partition. Cai et al. [12] implemented this method by utilizing Genetic Algorithm to define the actions for test case selection. Their experiments show that the proposed adaptive testing can improve the effectiveness of random testing. However, the overhead of computational strategy incurred in selecting the actions in test case selection made this testing costly [13], [14]. Applying the intuition of adaptive testing with lower complexity will contribute to provide better testing strategy. This research is intended to propose a strategy that applying the feedback mechanism as utilized in adaptive testing strategy, with simplified in parameter estimation to reduce computational complexity.

The remaining of this paper is organized as follows: Section II reviews related work. In Section III, the studied method is presented. Section IV reports the experimental setting. The results of the experiments are presented in Section V. Section VI presents conclusion and future work.

II. RELATED WORK

Adaptive testing strategy is proposed by Cai [11] that combines adaptive theory and software testing. This strategy utilizes the testing history data that collected online to improve the effectiveness of random testing. In adaptive testing, software under test is treated as a control object that modeled as a Control Markov Chain, whereas the testing process is treated as a control problem and testing strategy serves as the controller. Figure I present a pictorial view of adaptive testing strategy. There are four components of adaptive testing model: software under test, database (history data), testing strategy and parameter estimation. The database provides history testing data to be used for parameter estimation and controller. The value of parameters for controller are adjusted by parameter estimation component. These parameters together with the historical data are used by controller to define the actions. The controlled object (software under test) executes the actions to generate the test results. The test results update the database (historical data).

The main issue in adaptive testing is how to perform optimal parameter estimation. The key role in the strategy is the parameter θ_i (the probability of finding a defect at action i-th). The parameter estimation utilizes estimation algorithms to estimate the parameters. Cai et al. [12] utilize Genetic Algorithm for parameter estimation. The process of Genetic Algorithm includes initiation of population, fitness calculation, best gene selection, crossover, mutation and produce next generation. The computational complexity of Genetic Algorithm is in *$O(m^3)$* [13].

In their experimental work, Cai et al. [12] proposed an adaptive testing strategy by using a fixed-memory feedback mechanism. They used only a fixed number of historical data for parameter estimation to avoid aging data. They choose two different amounts of data: l =100 and l = 200. The reason to this option was arbitrary. Their experimental results show that the proposed methods performed better with l =100. They also define the number of equivalence classes (partitioning) into four partitions. The reason of partitioning the input domain of the subject program into four (equivalence) classes was arbitrary. It is found that the adaptive testing strategy outperformed the random testing strategy.

III. STUDIED METHOD

As has been mentioned in Section I, the proposed method follows the idea of adaptive testing strategy, as the counterpart of adaptive control in software testing [11], [12]. The strategy is adjusted online by incorporating the history data during testing.

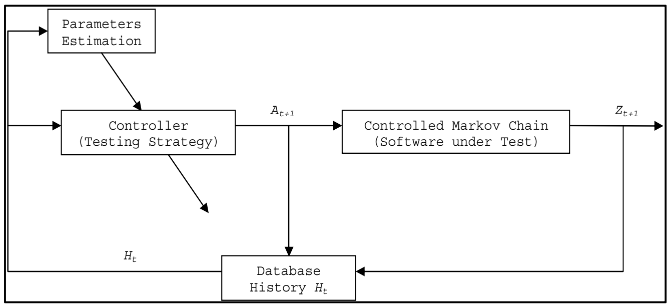
Parameters Estimation
Controller (Testing Strategy)
A_{t+1}
Controlled Markov Chain (Software under Test)
Z_{t+1}
H_t
Database History H_t

Fig. 1. Adaptive Testing Model [12]

In order to reduce the computational overhead of adaptive testing strategy, despite applying Genetic Algorithm for parameter estimation, the studied method applies a simplified method to estimate the actions for test case selection. This strategy is based on the intuition that if a partition is successful in the previous action, it has higher possibility to be successful again in the next execution.

The followings are some parameters that has to be defined in this strategy:

- The number of partition (t)
- The number of history data to be used (l)
- The number of Faults existing in the software under test (N)
- The probability of finding a defect at action i-th (θ_i)

Following Cai et al. [12], the number of partition in this studied method is set to 4 and the number of memory data is set to 100 (t = 4 and l = 100). This number is defined arbitrarily. As also assumed in the adaptive testing strategy [12], the software under test consists of N faults at the beginning and an action can detect at most one fault.

The Equation 1 is used to calculate the probability of each partition to be selected (detect a failure).

$$\theta_i = \theta_{i-1} + (S_{p_i} * \frac{t-1}{N}) - (\frac{l-S_{p_i}}{l}) \qquad (1)$$

where,

- p is the partition index
- i is the current action index.
- Sp_i is the number of action/s that detected failure. Equation 2 is the formula to calculate Sp_i.

$$S_p = \sum_{n=1}^{l} a_{p_n} \qquad (2)$$

where a represents the history testing data: $a = \{a_1, a_2, \ldots a_{100}\}$, the value of a_i is set to 1 if i-th action detects a failure otherwise a_i is set to 0.

The partition with the highest θ is selected to for the next action. At the beginning, all the partitions have the same probability to be selected ($\theta = \frac{1}{t} = 0.25$), hence will be selected randomly.

The complete algorithm for the studied method is as follows:

1. Initialize the value of θ of each partition to 0.25. Set N to the number of seeded faults into software under test.
2. Select a partition randomly (p).
3. Select a test case from the selected partition randomly.
4. Execute the selected test case into software under test.
5. If the action detects a failure set N to $N-1$, a_{pi} to 1 and fix the correspondent fault, otherwise set a_{pi} to 0.
6. If all the seeded faults have been detected and fixed then stop, otherwise go to Step 7.
7. Calculate the value of θ_p based on the new historical data.
8. Select partition with the highest θ
9. Go to Step 3.

IV. EXPERIMENTS

A series of experiments are conducted to investigate the effectiveness of the studied method. The experiment setup is explained in this section including the software under test, effectiveness measurements and the design of experiments.

A. Software Under Test

In this experiment, the software under test is SPACE. SPACE is a widely used real program consists of 9564 lines of C code, and functions as an interpreter for an array definition language (ADL) [15], [16]. The program reads a file that contains several ADL statements, and checks the contents of the file for adherence to the ADL grammar and to specific consistency rules. If the ADL file is correct, space outputs an array data file containing a list of array elements, positions, and excitations. Along with this program, 13,585 test cases are included.

B. Effectiveness Measurement

The effectiveness of studied method is measured by using F-measure, the number of test cases executed to detect the first failure in the software under test. The lower F-measure the more cost-effective the corresponding method.

C. Experiment Design

As has been explained in the previous section, the studied method applied partitioning into test suite. The test suite is partitioned into four disjoint classes. In order to investigate whether the way of partitioning influence the effectiveness of studied method, in this experiment, two types of partitioning are conducted. The first way of partitioning is by applying random partition. Each of test case is classified into one of partition randomly. The second way of partitioning is by applying coverage information into the process. Branch coverage of each test cases is used to classified them. Test cases with similar coverage are grouped into the same partition.

The first partitioning, random partitioning, resulted four partitions as follows:

- Partition 1 consists of 3417 test cases
- Partition 2 consists of 3346 test cases
- Partition 3 consists of 3469 test cases
- Partition 4 consists of 3293 test cases

The second partitioning, branch coverage partitioning, resulted four partitions as follows:

- Partition 1 consists of 4898 test cases
- Partition 2 consists of 2683 test cases
- Partition 3 consists of 2273 test cases
- Partition 4 consists of 3581 test cases

Both partitions are then executed to the studied methods and respectively RP-FT and BR-FT. As the basic and fundamental technique of the testing method, RT is implemented as a benchmark. Therefore, there are three methods that have been implemented:

1. RT (Random Testing): purely random test case selection without partitioning.
2. RP-FT (Random Partitioning – Feedback-based Testing): random partitioning that applied into proposed feedback based testing.
3. BR-FT (Branch Coverage-Based Partitioning – Feedback-based Testing): branch coverage partitioning that applied into proposed feedback based testing.

The experiments use the mutants (faulty versions) provided in the SPACE package. Following the experiments in Zhou et al. [15], [16], form 38 of mutants provided, we only used 33 mutants. Some of the reduced mutants cannot be detected by the provided test suite and some of them are performed instable (can be detected by different test cases in different execution). All the faults are seeded into the original/correct version program of SPACE. Based on the assumption of the studied method, each of action can only detect one fault. When a failure is detected (the actual output is different from the expected one), one of the touched fault is deleted and replaced by the correct/original one. This is executed until all the faults are fixed.

Since all studied methods contains randomness, we execute each of method for 100 trials. This is to reduce the threat to validity of the results.

V. EXPERIMENTS RESULT AND DISCUSSION

The results of the experiments after 100 trials of execution of each method are presented in Table 1. The results show that the proposed feedback-based method with random partitioning performed best in all statistics. In average, the RP-FT outperformed both RT and BP-FT. The average F-measure of RP-FT is 952.38, less than the F-measure of RT (1191.45) and of BP-FT (1208.17). The RP-FT can improve the cost-effectiveness of RT with the gain of 20.07% $((1191.45 - 952.38)/1191.45)$. The same result is indicated with median and maximum F-measure, the RP-FT outperformed the RT with the gain of 16.47% and 38.92% respectively. The RP-FT is also performed as the best in stability, that is indicated by its standard deviation. The standard deviation of RP-FT is 22.32% lower than that of RT. This result indicates that the proposed feedback-based method can improve the cost-effectiveness of software testing when it can be implemented in lower computational complexity than adaptive testing strategy.

Furthermore, a statistical analysis is conducted. Figure 2 is the Boxplot Diagram of the experimental results. It is indicated that the RP-FT performed as the best method. The p-value from t-test indicates that RP-FT improve the cost-effectiveness of random testing (RT) significantly (p-value = 0.0001).

An interesting finding is also introduced from this result. The Branch Coverage Partitioning (BP-FT) performed worse than the Random Partitioning (RP-FT). The results also show that the BP-FT cannot improve the RT. The average of F-measure of BP-FT is slightly higher than that of RT. BP-FT also performed in similar stability with the RT. The standard deviation of RT and BP-FT are very close (545.89 vs 546.65). The p-value from t-test between BP-FT and RT is more than 0.05 (p-value = 0.829), indicates that the fault-detection capability of BP-FT is not different from that of RT. This results actually in line with the intuition of distance-based technique named Adaptive Random Testing (ART) [17], [18]: the contiguous input domain (similar coverage) tends to have contiguous failure region. Hence, it is better to select the furthest test cases than the closer one. In BP-FT, test cases with similar branch coverage are classified into the same partition. This is against the intuition of ART. The partition with branch coverage information should apply the distance-based method to allocate the test cases. This issue should be investigated further.

In terms of the validity of these experimental results, the result has been verified by observing the testing process and the output carefully. However, there are some issues that have to be considered to improve the findings validity, as follows:

- involving more software under test with various size and platform,
- investigating the effect of the number of partition/equivalence classes,
- investigating the effect of the amount of applied history testing data,
- investigating the method to allocate test cases into partitions.

TABLE I. RESULTS OF THE EXPERIMENTS

Statistics	**Method**		
	RT	***RP-FT***	***BP-FT***
Average	1191.45	952.38	1208.17
Median	1083.50	905.00	1138.00
Maximum	3566.00	2178.00	2680.00
Standard Deviation	545.89	424.05	546.65

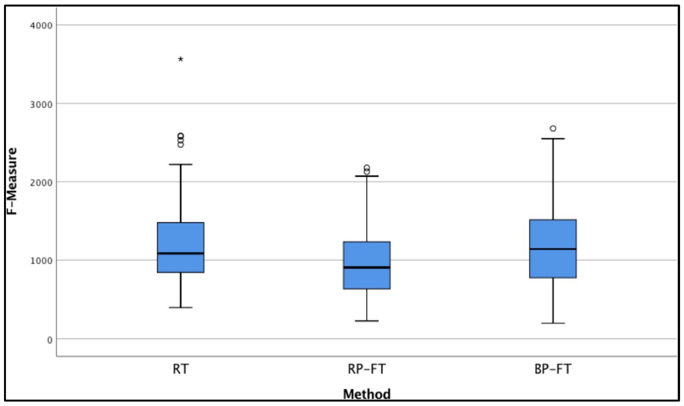

Fig. 2. The Boxplot Diagram

VI. CONCLUSION AND FUTURE WORK

The studied method applied feedback mechanism to control the process of test case selection in partition testing. This method follows the idea of adaptive testing strategy with lower computational overhead. A series of experiments have been conducted to investigate the cost-effectiveness of the proposed method: random partitioning incorporating feedback based testing (RP-FT) and branch coverage based partitioning incorporating feedback based testing (BP-RT). Random testing (RT) is implemented as the benchmark. The experiments results show that the proposed RP-FT outperformed RT significantly. On the other hand, the BP-FT performed similarly with RT. The rationale for this finding is due to the way of branch-coverage partitioning conducted, that is the similar coverage allocated in the same partition. This process resulting worse code-coverage of partitions and hence worse fault-detection capability.

There are some issues that have to be investigated in the future research such as: the number of history data, the number of partition, the allocation method to partition and the various software under test.

Acknowledgment

This work was supported in part by Institut Teknologi Del, Indonesia.

References

[1] K.-Y. Chai, J. W. Cangussu, R. A. DeCarlo, and A. P. Mathur, "An overview of software cybernetics," in Proc. the 11th International Workshop on Software Technology and Engineering Practice. IEEE Computer Society Press, 2003, pp. 77 – 86.

[2] J.-M. Kim, A. Porter, and G. Rothermel, "An empirical study of regression test application frequency," in ICSE '00: Proceedings of the 22nd international conference on Software engineering. New York, NY, USA: ACM, 2000, pp. 126–135.

[3] Y. Li and N. J. Wahl, "An overview of regression testing," SIGSOFT Softw. Eng. Notes, vol. 24, no. 1, pp. 69–73, 1999.

[4] G. Rothermel, S. Elbaum, A. G. Malishevsky, P. Kallakuri, and X. Qiu, "On test suite composition and cost-effective regression testing," ACM Trans. Softw. Eng. Methodol., vol. 13, no. 3, pp. 277–331, 2004.

[5] J. Mayer and C. Schneckenburger, "An empirical analysis and comparison of random testing techniques," in ISESE'06. ACM, 2006, pp. 105 – 114.

[6] P. M. S. Bueno, M. Jino, and W. E. Wong, Diversity oriented test data generation using metaheuristic search techniques. Information Sciences 2014; 259:490–509.

[7] S. Ntafos, "On random and partition testing," in ISSTA 98, 1998, pp. 42–47.

[8] E. J. Weyuker and B. Jeng, "Analyzing partition testing strategies," IEEE Transactions on Software Engineering, vol. 17, no. 7, pp. 703–711, 1991.

[9] K.-Y. Cai, T. Jing, and C. G. Bai, "Partition testing with dynamic partitioning," in Computer Software and Applications Conference, 2005. COMPSAC 2005. 29th Annual International, vol. 2. IEEE, July 2005, pp. 113 – 116.

[10] T. Y. Chen, R. Merkel, G. Eddy, and P. K. Wong, "Adaptive random testing through dynamic partitioning," in QSIC '04: Proceedings of the Quality Software, Fourth International Conference. Washington, DC, USA: IEEE Computer Society, 2004, pp. 79–86.

[11] K.-Y. Cai. Optimal software testing and adaptive software testing in the context of software cybernetics. Information and Software Technology 2002; 44:841–855.

[12] K.-Y. Cai, B. Gu, H. Hu, Y.C. Li. Adaptive software testing with fixed-memory feedback. Journal of Systems and Software 2007; 80:1328–1348.

[13] P. Xiao, Y. Yin, B. Liu, B. Jiang, and Y. K. Malaiya. Adaptive testing based on moment estimation. IEEE Transactions on Systems, Man, and Cybernetics: Systems, 2017; 99: 1-12.

[14] Lv J, Hu H, Cai K-Y, Chen TY. Adaptive and Random Partition Software Testing. IEEE Transactions on Systems, Man, and Cybernetics: Systems December 2014; 44(12):1649–1664.

[15] Z. Q. Zhou, A. Sinaga, L. Zhao, Susilo W, Cai K-Y. Improving software testing cost-effectiveness through dynamic partitioning. Proceedings of the 9th International Conference on Quality Software (QSIC'09), IEEE Computer Society Press, 2009; 249–258.

[16] Z. Q. Zhou, A. Sinaga, and W. Susilo, L. Zhao, K-Y Cai. A cost-effective software testing strategy employing online feedback information. Information Sciences 2018; 422:318–335.

[17] Z. Q. Zhou, "Using coverage information to guide test case selection in adaptive random testing," in Proceedings of the 34th Annual International Computer Software and Applications Conference (COMPSAC'10), 7th International Workshop on Software Cybernetics. IEEE Computer Society Press, 2010, pp. 208–213.

[18] Z. Q. Zhou, A. Sinaga, and W. Susilo, "On the fault-detection capabilities of adaptive random test case prioritization: Case studies with large test suites," in *Proceedings of the 45th Annual Hawaii International Conference on System Sciences (HICSS'12)*. IEEE Computer Society Press, 2012, pp. 5584–5593.