Adaptive Random Testing By Static Partitioning

Korosh Koochekian Sabor,
Department of Electrical and Computer Engineering
Concordia University
Montréal, Canada
k_kooche@encs.concordia.ca

Stuart Thiel
Department of Computer Science and Software Engineering
Concordia University
Montreal, Canada
stuart.thiel@concordia.ca

***Abstract*— Despite the importance of the random testing approach, random testing is not used in isolation, but plays a core role in many testing methods. On the basis that evenly distributed test cases are more likely to reveal non-point pattern failure regions, various Adaptive Random Testing (ART) methods have been proposed. Many of these methods use a variety of distance calculations, with corresponding computational overhead; newly proposed methods like ART by bisection, random partitioning or dynamic iterative partitioning try to decrease computational overhead while maintaining the performance. In this article we have proposed a new ART method that has similar performance to existing ART methods while having less computational overhead.**

***Index Terms*— random testing, adaptive random testing, static, static partitioning.**

I. INTRODUCTION

In the literature, a general consensus exists that exhaustive testing (testing a program with all possible inputs) is not feasible [1]. Lots of research has been conducted in this field and diverse methods to improve failure detection capability of test cases have been proposed ([2,3,4,5,6]). A standard approach for automatic selection of test cases is random testing [7,8]. The intuition behind random testing is to select test cases randomly until a stopping condition such as detecting a failure, executing predefined number of test cases or ending of a time limitation is met. Despite the fact that random testing has substantial practical advantages, including that it simple to perform and can be used to calculate reliability estimates [9], it does not gain the benefit of any available information from the program under test [10].

As described by Chan et al in [11] failure patterns can be categorized into three typical categories. These categories are: 1) block pattern; 2) strip pattern; or 3) point pattern. As shown in Fig. 1, the block pattern is where failures are clustered in one or more continuous areas, and is the most common case. The strip pattern shown in Fig. 2 involves a continuous area that is elongated in one dimension while quite narrow in the other. The point pattern shown in Fig. 3 involves failures that are distributed in small groups in the input domain.

A simple but effective improvement in random testing is presented in [12,13]. Given the idea that software failures can be categorized according to failure pattern categories which have been mentioned in Fig. 1,2,3 the intuition of the new method is to use information about the location of executed test cases to improve efficiency. It is obvious that the chance of revealing a failure will increase by distributing test cases more evenly when the failure pattern is of block or strip type. Based on this simple idea, numerous adaptive random testing (ART) methods have been proposed. In ART methods, initially test cases are selected randomly with subsequent selections being restricted so as to distribute the selection of test cases more evenly.

Some ART methods, such as Distance-Based ART (D-ART) or Restricted Random Testing (RRT), have high computational overhead. To decrease this overhead, numerous ART methods, such as ART by bisection or ART by random partitioning have been proposed [14].

In this paper we propose a method which has similar performance to that of other already proposed ART methods, while having less computational overhead.

In this article, elements of the input domain are known as failure-causing inputs if testing the program given those inputs produces incorrect outputs. The failure rate is calculated as the division of the number of failure causing inputs by the total number of inputs in the input domain. F-measure, which is used as an effectiveness metric in this article, is defined as the number of test cases required for detecting the first failure. F-Measure is more prominent for software engineers and more specifically testers, as in most cases when a failure is detected, testing will stop and debugging will start.

This article is organized as follows: In section 2, various ART methods are presented. In section 3, the algorithm of the proposed method, named adaptive random testing by static partitioning, is presented. In Section 4, we present our simulation results. Our conclusion is presented in Section 5.

II. RELATED WORKS

Distance-based ART (D-ART) randomly selects a set of candidate test cases from the input domain, but only selects one test case from the candidate set to be executed based on the criterion of maximizing the minimum distances between the candidate test cases and all of the previously executed test cases [15]. Consider the size of candidate test cases as e and the number of test cases required for detecting the first failure as n. Given that each time the next test case is chosen, the distance between every element in the candidate test case set and all of the previously executed test cases should be calculated, the number of distance calculations required for choosing the ith test case could be calculated as $e*(i-1)$. As a result, the order of time complexity of Distance-Based ART

978-1-4673-7022-6/15 $31.00 © 2015 Crown Copyright
DOI 10.1109/AST.2015.13

could be calculated by Eq. 1, [16]. The time complexity of this algorithm is quadratic.

Fig. 1. Block pattern

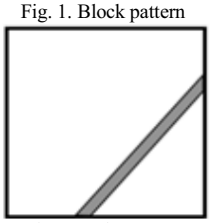

Fig. 2. Strip pattern

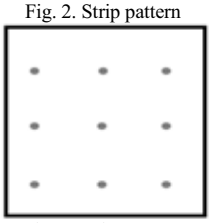

Fig. 3. Point pattern

$$\sum_{i=1}^{n} e \times (i-1) = e \times \sum_{i=1}^{n-1} i \in O(n^2) \qquad (1)$$

In restricted random testing (RRT) a restriction region is created around each executed test case. Test cases will be selected randomly from the entire input domain, but will be rejected if they fall within a restriction region, and the next test case will be selected randomly [17]. Given this method, choosing the *i*th test case requires the selection of a test case that is outside of all the *i-1* previously restricted regions. If we identify as *c* the number of test cases which should be generated to find a test case outside of all the restricted regions, then the order of time complexity of RRT could be calculated by Eq. 2, [16]. As is obvious from Eq. 2 the time complexity of this algorithm is quadratic.

$$\sum_{i=1}^{n} c \times (i-1) = c \times \sum_{i=1}^{n-1} i \in O(n^2) \qquad (2)$$

Mirror adaptive random testing (MART) [18] that aims to decrease the distance calculation overhead have been proposed. In MART, firstly the entire input domain is partitioned into disjoint subdomains, after that the ART method would only be applied in one subdomain and mirror functions would be used to map the test cases into other subdomains. As such, if we partition the input domain into *M* disjoint subdomains the distance calculation overhead will be decreased by $\frac{1}{M^2}$

Given the aim of reducing calculation overhead, two methods named ART by random partitioning and ART by bisection are proposed in [14]. In order to eliminate computation overhead, these methods use various partitioning techniques. In these methods, instead of choosing a test case from a candidate set, a region which the next test case will be selected from will be chosen and the test case that is selected from this region will be executed next. The main difference between the ART by random partitioning and ART by bisection is the partitioning scheme that is used. The intuition in ART by random partitioning is to partition the input domain according to the previously executed test case. Given the new partitioning, selecting from the biggest partition will have the highest chance of being far from the previously executed test cases. In ART by bisection, the input domain would iteratively be divided into equal sized partitions and a partition would be selected as the next test case generation region only if there are no executed test cases within that region. The time complexity of both of the above mentioned approaches are linear with respect to the number of executed test cases [19]. Although integrating a localization concept improved the performance of both methods, it created edge preference problem. To eliminate the edge preference problem in ART by random portioning, K.K.Sabor et al used an enlarged input method domain in [20, 21].

Although these two partitioning methods decrease the computational overhead, they do not perform as well as Distance-Based ART or RRT. To address this lower performance, ART through iterative partitioning has been proposed, with the aim of decreasing the computational cost while maintaining the performance [16].

In ART through iterative partitioning, in each iteration the input domain is partitioned with a $p \times p$ partitioning scheme. As a result, the input domain will have $p \times p$ cells. The first test case is selected randomly. If this test case does not reveal a failure, the cell which this test case is located in will be marked as occupied; its 8 neighbors are marked as adjacent cells. The remaining unmarked cells are considered as the candidate cells. While any candidate cell is available, the next test case will be selected randomly from the candidate cells. If no such cell exists, the $p \times p$ partitioning scheme will be discarded and a new $(p+1) \times (p+1)$ partitioning scheme will be created. Every time that a partition scheme is discarded and a new one is constructed, the method needs to map all of the previously executed test cases according to the new partitioning scheme, marking them and their neighbors as occupied and adjacent cells respectively. These mappings will cause computational overhead. As expressed in [16] if we call the number of dimensions of input domain *k* and the number of test cases required for revealing the first failure *n* then the time complexity of this algorithm will be $O(3^{k+1}.n^{1+1/k})$. As is obvious, when *k* (the dimensionality of the input domain) becomes large then 3^{k+1} will have a huge value and the computational overhead will increase.

III. PROPOSED METHOD

In this article we have proposed a new adaptive random testing approach that partitions the input domain statically. In this approach, the even distribution of test cases is accomplished by coloring the cells with white, red, green or yellow colors. The red cells are those that already contain an executed test case, so no more test cases should be selected from them. White cells are those that do not contain any

executed test case and are not adjacent to any red cells, so they have the highest priority when selecting the next test case generation region. Green cells are those which do not contain any executed test case but have one adjacent red cell, thus their priority for being selected is less than that of white cells. Yellow cells are those that have two or more adjacent red cells, having the least priority. If all of the cells are colored red and no failure has been detected, all of the cells will be colored white and the algorithm will be started again.

The rationale behind coloring the input domain cells is to classify the cells based on the probability that they contain failure test cases. For block and strip type failure patterns, failures in one cell will more likely spill over to some adjacent cells. Conversely, if a cell has no detected failure, then it is less likely that its neighbors will have failures. As such, white cells are the most likely candidates to contain failure test cases, as nothing is known about them, with green cells being less likely, but still good candidates as we only know that one of their neighbors has a passing test case. Yellow cells have at least two neighbors with passing test cases and so are much less likely to be part of a failure pattern. Lastly, red cells have a passed test case, so they are the least likely to contain such failure test cases.

Adaptive random testing by static partitioning algorithm

The proposed method can be applied to any module with any number of input parameters. Each input parameter will be an axis. For example, if a module has 3 parameters as input, then the input space domain will be 3-dimensional. Regardless of dimension, a test case will be chosen from a single cell, which either reveals failure or does not. For simplicity of discussion, here it is assumed that there are only two input parameters for the program under test. Entries of the matrix named *GridCell* are used to show the color of each cell using the integers 0, 1, 2 and 3, representing white, green, yellow and red respectively.

1. Create a $p \times p$ matrix with integer entries, *GridCells*, initialize all the entries in the *GridCells* by number 0 (white color), and add all of the cells as white cells to the *white_cells* list. Set *Green_Cells*, *Yellow_Cells*, and *Red_Cells* lists empty. Set *TotalCellsCount* $= p \times p$.

2. While (testing resources are not exhausted)

2.1) If (*white_cells* items count is not 0) randomly select a white cell.

2.2) Else if (*Green_cells* items count is not 0) randomly select a green cell.

2.3) Else if (*Yellow_cells* items count is not 0) randomly select a yellow cell.

2.4) Randomly select a test case within the selected cell.

2.5) If the selected test case is a failure causing input then break the loop and end the algorithm. Else set the appropriate *GridCells* entry to 3 (color the cell red) and add it to the *Red_Cells* list.

2.6) For each of the selected cell adjacent cells

I. If its *GridCells* entry is 0 set it to 1, delete the cell from the *white_cells* list and add it to the *Green_Cells* list.

II. Else if its proportional *GridCells* entry is 1 set it 2, delete the cell from the *Green_cells* list and add it to the *Yellow_Cells* list.

III. Else if its proportional *GridCells* entry is 2 do not change it (it will be remained in the *Yellow_Cells* list).

2.7) If the number of items in *Red_Cells* is equal to *TotalCellsCount* set all the *GridCells* entries to 0 (sets all the cells color white) and empty the *Red_Cells* list and add all of the Cells to the *white_cells* list.

In this algorithm In the first place the input domain would be partitioned into $p \times p$ cells. Then all of the cells are colored white. As is obvious in step 2, in each iteration cells are selected according to their priority: If no white cells are available then green cells will be selected; If no white and green cells are available then yellow cells will be selected; If only red cells are available then all of the cells are colored white and the algorithm will be started again.

When a cell is selected as the next test case generation region it is colored red and its adjacent cells will be colored. If an adjacent cell is white then it will be colored green, if it is green it will be colored yellow, and if it is yellow it will be remain unchanged. The algorithm will continue until a stopping condition such as ending of testing resources or finding a failure causing input is met.

According to the idea that even distribution will occur when new test cases have the farthest distance from the previously executed ones, every time that a test case should be selected the cells which have an executed test case are discarded. The white cells are those that are not adjacent to any red cells, so they have the highest priority. The green cells are those that are adjacent to one of the red cells, which contain previously executed test cases. Thus a test case selected from a green cell will have more chance of being close to the previously executed test cases than one selected from a white cell. As a result, green cells have less priority than white cells. The yellow cells are those that are adjacent to two or more red cells, so if a test case is selected from them its probability of being close to the previously executed test cases is higher than if it had been selected from green cells. As such, yellow cells have less priority than white or green cells.

An example of the execution of this algorithm is illustrated in Fig. 4 and Fig. 5. In Fig. 4 and 5, green cells are marked by 'G', red cells are marked by 'R', and yellow cells are marked by 'Y'. The remaining unmarked cells are white. Initially, the input domain is partitioned into $p \times p$ cells and all of the cells are colored white. A test case is randomly selected. Since it is not a failure causing input the cell which contains that test case is colored red and its adjacent cells are colored green Fig. 4.

In the second pass because white cells still exist, one is randomly selected and a test case is generated randomly within that cell. Since it is not a failure causing input the cell is colored red and adjacent cells are colored green if they are white and yellow if they are green Fig. 5.

The method continues by selecting white, green and yellow cells until a stopping condition such as detecting a failure or ending of a time limitation is met.

Since we have used 4 colors in this method we can allocate a list of cells for each color. As a result, there will be no need to search the entire input domain for a cell with a specific color. If we call the number of test cases required for detecting the first failure n, since no distance computation with previously executed test cases is required the order of time complexity for the proposed method will be $O(n)$.

IV. Evaluation

In this article a series of simulations in a 2-dimensional input space domain were run, with the aim of measuring failure detection capability of the proposed approach. For each of the failure rates in each test run a failure-causing region of the specified size and pattern was assigned within the input domain. For block pattern, a square has been used as the failure-causing region. For strip pattern, two points on the adjacent borders of the input domain have been randomly selected and have been connected to each other to form a strip with predefined size. For the point pattern, circular regions were randomly located in the input domain without overlapping each other. For each simulation, 5000 test runs were executed.

Fig. 4. ART by static partitioning first pass

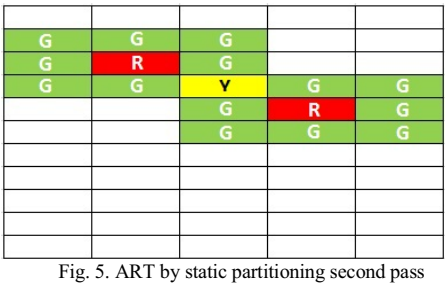

Fig. 5. ART by static partitioning second pass

In the first part of experiments, the performance of the proposed method has been compared with that of random testing when failure pattern is of block type. The results are shown in Table 1. As is obvious, the higher the failure rate is, the better the performance of our method. As a result, the best performance is obtained while the failure rate is 0.01 in which our method would outperform random testing by approximately 29%.

In the second part of our experiments we have compared the performance of ART by static partitioning with that of random testing when the failure pattern is of type strip. The results are shown in Table 2. As is apparent, similar to the block pattern, the strip pattern performance decreases as the failure rate decreases. The reason for the decrease in the performance can be explained as follows: When the failure rate decreases, the probability that a failure-causing test case will be selected as the next test case decreases, so the performance of ART by static partitioning scales down. In the case of strip pattern, the proposed method outperforms random testing by at most 13% in the best case when failure rate is 0.01.

TABLE 1. F-MEASURE FOR ART BY STATIC PARTITIONING COMPARED WITH THAT OF RT WITH BLOCK FAILURE PATTERN

Failure Rate	Expected F-Measure for RT F_{rt}	block pattern	
		F-Msr for Proposed Method	$\frac{F_{art}}{F_{rt}}$
0.01	100	71	0.71
0.005	200	148	0.74
0.002	500	381	0.762
0.001	1000	816	0.816

TABLE 2. F-MEASURE FOR ART BY STATIC PARTITIONING COMPARED WITH THAT OF RT WITH STRIP FAILURE PATTERN

Failure Rate	Expected F-Measure for RT F_{rt}	strip pattern	
		F-Msr for Proposed Method	$\frac{F_{art}}{F_{rt}}$
0.01	100	87	0.87
0.005	200	185	0.925
0.002	500	470	0.94
0.001	1000	975	0.975

TABLE 3. F-MEASURE FOR ART BY STATIC PARTITIONING COMPARED WITH THAT OF RT WITH POINT FAILURE PATTERN

Failure Rate	Expected F-Measure for RT F_{rt}	point pattern	
		F-Msr for Proposed Method	$\frac{F_{art}}{F_{rt}}$
0.01	100	103	1.03
0.005	200	208	1.04
0.002	500	523	1.046
0.001	1000	1050	1.05

In the third part of our experiments we have investigated the performance of the proposed approach while the failure pattern is of point type. Intuitively, when the failure pattern is not a point pattern, more evenly spread test cases have a better chance of hitting the failure patterns [22]. This is consistent with our empirical results. As is apparent from Table 3, in the case of point patterns we needed more test cases to be executed to find the first failure, as compared to the random testing.

Summarizing, the proposed method outperforms random testing by 29% in the best case when the failure pattern is of block type, and outperforms random testing by 13% when the failure pattern is of strip type. When the failure pattern is of point type, the even distribution of test cases does not lead to a

better performance in detecting the first failure, and using the proposed method will not increase the performance.

V. CONCLUSION

In this article we have proposed a new ART approach with the aim of decreasing the distance calculation computational overhead, while distributing test cases evenly. In this method, we have partitioned the input domain into $p \times p$ cells. Subsequently we have used coloring techniques to color cells in order to select test cases from cells which have the farthest distance to the cells which already contain previously executed test cases. The coloring approach has helped us to decrease the order of time complexity to a linear time $O(n)$.

The performance of the proposed method has been compared with that of random testing. The proposed method outperforms random testing when the failure pattern is of block or strip type, But in the case of the point pattern, the proposed method decreases the failure detection capability.

According to the empirical results, since the proposed approach has linear time complexity, we recommend to testers that if adaptive random testing is suitable for their goal, to use ART by static partitioning. Moreover, this approach could be used in combination with other testing methods to increase failure detection capability.

In our future work we would investigate the effect of various partitioning schemes on the failure detection capability of ART by static partitioning. We would also investigate the performance of the proposed method when it is combined with other ART methods.

REFERENCES

[1] B.Beizer, Software Testing Techniques, Second Edition, Van Nostrand Reinhold Company Limited,1990

[2] J. W. Laski , B. Korel, "A data flow oriented program testing strategy", IEEE Transactions on Software Engineering, Vol. 9, No. 3, pp. 347-354, 1983.

[3] J. Offutt and S. Liu, "Generating test data from SOFL specifications", The Journal of Systems and Software, Vol. 49, No. 1, pp. 49-62, 1999.

[4] J. Offutt, S. Liu, A. Abdurazik and P. Ammann, "Generating test data from state-based specifications", Software Testing, Verification & Reliability, Vol. 13, No.1, 2003, pp. 25-53.

[5] P. Stocks and D. Carrington, "A framework for specification-based testing", IEEE Transactions on Software Engineering, Vol. 22, No. 11, pp.777-793, 1996

[6] L. J. White and E. I. Cohen, "A domain strategy for computer program testing", *IEEE* Transactions on Software Engineering, Vol. 6, No. 3, pp. 247-257, 1980.

[7] R. Hamlet. "Random testing". In Encyclopedia of Software Engineering, pp 970–978. Wiley, New York, 1994.

[8] P. S. Loo and W. K. Tsai. "Random testing revisited". Information and Software Technology, Vol.30. No 7, pp. 402-417, 1988.

[9] E. J.Weyuker and B. Jeng. "Analyzing partition testing strategies". IEEE Transactions on Software Engineering, Vol. 17, No. 7, pp.703-711, July 1991.

[10] G. Myers, The Art of Software Testing, NewYork: John Wiley & Sons, 1979

[11] F. T. Chan, T. Y. Chen, I. K. Mak, and Y. T. Yu. "Proportional sampling strategy: guidelines for software testing practitioners". Information and Software Technology, Vol. 38, No 12, pp775– 782, 1996.

[12] T. Y. Chen, T. H. Tse and Y. T. Yu, "Proportional sampling strategy: a compendium and some insights". The Journal of Systems and Software, 58, pp. 65-81, 2001.

[13] I. K. Mak, on the effectiveness of random testing, Master's thesis, The University of Melbourne, 1997.

[14] T. Y. Chen, G. Eddy, R. Merkel and P. K. Wong, "Adaptive random testing through dynamic partitioning", Proceedings of the 4th International Conference on Quality Software (QSIC2004), pp79 – 86, 2004.

[15] T. Y. Chen, H. Leung, and I. K. Mak, "Adaptive random testing" in Proceedings of the 9th Asian Computing Science Conference, LNCS 3321, pp. 320-329, 2004.

[16] T. Y. Chen, D.H.HUANG AND Z.Q. ZHOU, "On Adaptive Random Testing Through Iterative Partitioning", journal of information science and engineering 27, pp-1449-1472, 2011.

[17] K. P. Chan, T. Y. Chen and D. Towey, "Normalized restricted random testing" , 8th Ada-Europe International Conference on Reliable Software Technologies, Toulouse, France, June 16-20, 2003, Proceedings. LNCS 2655, pp368-381, Springer-Verlag Berlin Heidelberg 2003.

[18] T.Y.Chen, F.C.Kuo, R.G.Merkel, S.P.Ng , "Mirror adaptive random testing", Elsevier,2004

[19] T.Y.Chen, D.H.Huang, "Adaptive Random Testing by Localization", in Asia-Pacific software Engineering Conference, p 1530-1362, 2004.

[20] K. K. Sabor and M. Mohsenzadeh "Adaptive random testing through dynamic partitioning by localization with restriction and enlarged input domain". In Proceeding Of the 2012 international conference on information technology and software engineering, Lecture Notes in Electrical Engineering Volume 212, pp 147-155, 2013.

[21] K. K. Sabor and M. Mohsenzadeh. "Adaptive random testing through dynamic partitioning by localization with distance and enlarged input domain". International journal of innovative technology and exploring engineering. November 2012, pp 1-5.

[22] T.Y.Chen, H.leung, I.K.Mak, "Adaptive Random testing" , *9th* Asian Computing Science Conference on Computer Science, LNC 3321, pp320-329, Springer-Verlag Berlin Heidelberg, 2004