Adaptive Random Testing Through Dynamic Partitioning

T.Y. Chen
School of Information Technology
Swinburne University of Technology
John Street,
Hawthorn Victoria 3122 Australia
tychen@it.swin.edu.au

G. Eddy
Department of Information Systems
The University of Melbourne
Parkville 3052, Australia
garyre@unimelb.edu.au

R. Merkel
Swinburne University of Technology
John Street,
Hawthorn Victoria 3122 Australia
rmerkel@it.swin.edu.au

P.K. Wong
Hong Kong Institute of Vocational Education (Sha Tin)
21 Yuen Wo Road
Sha Tin, New Territories, Hong Kong
spkwong@vtc.edu.hk

Abstract

Adaptive Random Testing (ART) describes a family of algorithms for generating random test cases that have been experimentally demonstrated to have greater fault-detection capacity than simple random testing. We outline and demonstrate two new ART algorithms, and demonstrate experimentally that they offer similar performance advantages, with considerably lower overhead than other ART algorithms.

Keywords: *random testing, adaptive random testing, partition testing, proportional sampling strategy.*

1. Introduction

Random testing is a standard, well-studied approach to automatic test case selection ([6], [7]). In essence, test cases are chosen randomly until a stopping condition is met. This might be the detection of a failure, the completion of a predefined number of tests, or the expiration of a set time limit. Some random testing strategies select tests using a uniform distribution, while others use a non-uniform distribution, such as a distribution matching the program's expected input profile. Random testing has the substantial practical advantage that it is often very simple to perform, and can be used to calculate reliability estimates [5].

On the other hand, random testing can be ineffective since it does not make use of any information about the likely characteristics of the program being tested [9]. A straightforward improvement on such simple random testing would be to make use of generic information about typical failure patterns seen in many programs. These typical failure patterns, as described by Chan et al. [1], include the *block* pattern, the *strip* pattern, and the *point* pattern. Figure 1 shows these typical failure patterns in a two-dimensional input domain. The block pattern (Figure 1(a)), describes the most common case, where failures are clustered in one or a few contiguous areas. Strip patterns (Figure 1(b)) involve a contiguous area, but elongated greatly in one dimension while quite narrow in the other. Finally, point-pattern failures (Figure 1(c)) are spread throughout the input domain in very small groups (or even individually).

Example 1 (based on the example in [1]) shows a pseudocode example of a program snippet which contains an error leading to a block failure pattern:

Example 1

```
INTEGER X, Y, Z
INPUT X, Y
IF(X >= 10 AND X <= 12)
   AND (Y >= 8 AND Y <= 11)
 THEN
      Z = X / 2 * Y;
           // correct statement
           // is  Z = X / 7 * Y
 ELSE
      Z = X * Y
OUTPUT Z
```

A type of program fault called a *domain error*[11] often leads to strip failure patterns, as shown in Example 2:

Example 2

```
INTEGER X, Y, Z
```

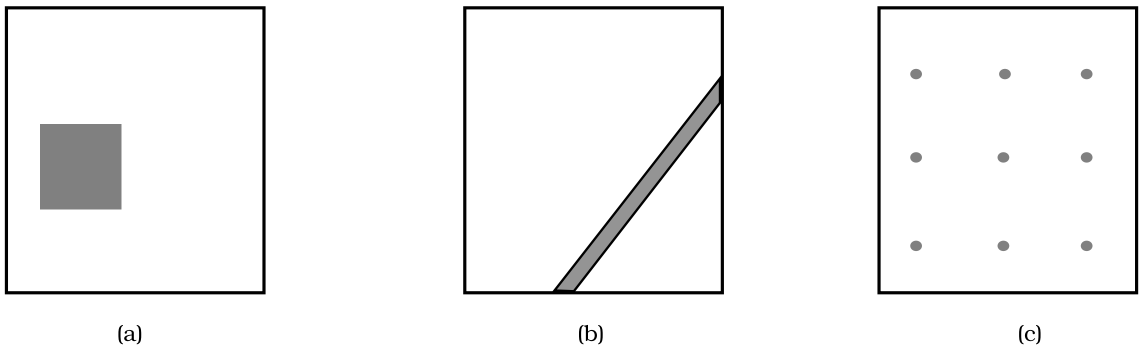

Figure 1. Block, Strip, and Point failure patterns in a two-dimensional input domain. Each box represents an input domain, and shaded areas represent failure-causing inputs.

```
INPUT X, Y
IF ( 2 * X - Y > 10)
             // should be
             // IF (2 * X - Y > 18)
 THEN
     Z = X / 2 * Y
 ELSE
     Z = X * Y
OUTPUT Z
```

Example 3 shows an example of a point failure pattern, which are believed to be less common than the block and strip types:

Example 3

```
INTEGER X, Y, Z
INPUT X, Y;
IF (X mod 4 = 0 AND Y mod 6 = 0)
 THEN
   Z = X / 2 * Y
          // correct statement
          // is  "Z = X / 7 * y"
 ELSE
   Z = X * Y
OUTPUT Z
```

When the failure patterns are of block or a reasonably "thick" strip types, it is intuitively ineffective to choose test cases which are close to each other. Rather, the chances of hitting these failure patterns are increased by ensuring test cases are somehow "spread widely" throughout the input domain. The method of *Adaptive Random Testing* (ART) was indeed based on such an intuition [3]. Simply speaking, in adaptive random testing, test cases are randomly selected, but subject to some constraint or mediation process to ensure that they meet the "wide spread" criterion mentioned above.

There are various ways to implement adaptive random testing. One of the more straightforward methods is outlined here [3]. In this method, two sets of inputs are maintained, the executed set and the candidate set. The executed set contains all previously executed test cases. A set of candidate inputs is then generated. The member of the candidate set which is furthest from the elements of the executed set is selected, the test is executed, and if the test does not reveal a failure it is added to the executed set. The candidate set is then discarded and a new one selected. The process is repeated until a termination condition, such as detection of a failure, execution of a specified number of test cases, or reaching a time limit, is achieved.

It has been experimentally demonstrated that ART can significantly outperform random testing in terms of the number of test required to detect the first failure, particularly when failure patterns are of the block type. Mak [8] compared this form of ART and pure random testing by comparing the number of test cases required to detect the first failure in a group of programs seeded with typical errors. The improvement was generally in the order of 30%, and occasionally up to 50%.

Another category of test case selection techniques, *partition testing*, has also received substantial study. In essence, partition testing involves dividing the input domain up into a fixed number of disjoint partitions, and choosing test cases from within each partition. A typical example is the domain testing strategy proposed by White and Cohen [11].

Partition testing performs at its best if such partitions are *homogeneous*, that is either all elements of the partition will cause a failure or none of them will. In practice, the only partitioning strategy which guarantees this treats every single possible input to the program as an individual partition, trivially degrading the partition testing strategy to that of exhaustive testing. Therefore, we must usually settle for strategies which approximate this ideal. For instance, some strategies partition the input domain according to program execution paths. In these strategies, termed *path-coverage partitioning* inputs from each different partition trigger different "paths" through the program's source code. These partitions, however, are not necessarily homogeneous.

Partition testing has powerful intuitive appeal, and analytical results show that even simple partitioning schemes may be more effective at fault detection than random testing. It does however, have several drawbacks that may counterbalance its advantages over random testing. Chief among these is the initial overhead of partitioning. Complex partitioning schemes such as the path-coverage partitioning model require a significant amount of preprocessing before testing can begin. Even simpler schemes require some overhead, all of which is expended at the very start of the testing procedure - which can be wasteful, particularly in the preliminary phases of testing where faults may be detected after only a few test cases. Therefore, it is important to develop partitioning schemes that have acceptable overheads.

Experimental analysis has shown that ART uses significantly fewer test cases than random testing to detect the first failure. However, the overhead of generating candidate sets and determining the next test case from the candidate set may be quite substantial. It is possible that the extra time taken to choose tests can outweigh the advantage of performing fewer tests.

In this context, therefore, we have developed and investigated two new strategies inspired by both ART and to some extent by partition testing, which incrementally divide the input domain into "implicit partitions" to ensure the wide spread of test cases characteristic of ART, while retaining the randomness of random testing.

2. Background

2.1. Overview of Algorithms

As discussed in the previous section, Adaptive Random Testing is essentially a random testing method, but with a mechanism which attempts to widely spread test cases over the input domain. There are obviously different methods available to enforce the spreading of test cases.

In this paper, we propose two new ways to implement the notion of spreading. One way is to use the selected test case itself to partition the input domain. This method is termed *ART by Random Partitioning*. This method is outlined in section 3.

The second method we present in section 4 is to repeatedly bisect the input partition into smaller partitions, selecting test cases from those partitions which do not contain previously-performed tests. We call this method *ART by Bisection*.

2.2. Notation

We basically follow the notation used by Chen et al. [3]. Elements of an input domain are known as failure-causing inputs, if they produce incorrect outputs. For an input domain D, we use $d, m,$ and n to denote the domain size, number of failure-causing inputs and number of test cases, respectively. The sampling rate σ and failure rate θ are defined as $\frac{n}{d}$ and $\frac{m}{d}$, respectively.

In this paper, we use the number of test cases required to detect the first failure (referred to as the F-measure) as the effectiveness metric. For random selection of test cases with replacement, the F-measure, denoted by F, is equal to $\frac{1}{\theta}$, or equivalently $\frac{d}{m}$. The lower the F-measure the more effective the testing strategy because fewer test cases are required to reveal the first failure. This is an appropriate effectiveness metric because in practice, when a failure is detected, testing is normally stopped and debugging starts. The testing phase would normally be resumed only after rectification of the fault.

We have conducted a simulation study to evaluate our two proposed methods. In our simulation, we assume the input domain is of two dimensions, and use the notation $\{(a, b)(c, d)\}$ to describe a rectangle with vertices at (a, b), (a, d), (c, b), and (c, d).

3. ART by Random Partitioning

3.1. Algorithm description

This method uses the most recently performed test to subdivide the input domain, and selects test cases randomly from within the largest remaining subdomain. As the partition process is performed on the basis of a randomly selected test case, we call this method *ART by random partitioning*.

This method is based upon the intuition that, given that the boundaries of the subdivisions are located according to the already executed test cases, choosing a test case randomly from within the largest subdivision is more likely to provide a test case distant from existing test cases. By putting all the regions thus generated into a global queue, and selecting the biggest region for the next test, eventually all regions are covered.

As previously mentioned, the algorithm is presented here in a form suitable for testing a program with two real inputs, i, j, where $i_{\min} \leq i \leq i_{\max}, j_{\min} \leq j \leq j_{\max}$. It is trivially extensible to any program with a fixed number of bounded real or integer inputs.

Pseudo-code for the algorithm is as follows:

Algorithm 1 Adaptive random testing by random partitioning

```
1. Set test region candidate list, $C$, to be the set $\{\{(i_{\min}, j_{\min})(i_{\max}, j_{\max})\}\}$.
2. Select a new test region, $T\{(i,j)(s,t)\}$, to be the element in $C$ with the largest area and remove it from $C$. If there is more than one test region with the largest area, choose one randomly.
3. Randomly select a test point, $(x, y)$, from within T; that is, $i \leq x \leq s$ and $j \leq y \leq t$.
4. If the point $(x, y)$ is a failure-causing input, report fault detection and terminate.
5. Otherwise, divide the current test region, $\{(i,j)(s,t)\}$, into four test regions, $\{(i,j)(x,y)\}$, $\{(i,y)(x,t)\}$, $\{(x,j)(s,y)\}$, and $\{(x,y)(s,t)\}$ and add them to $C$.
6. Goto Step 2.
```

The operation of the algorithm as presented above can be seen in Figure 2. Initially, the candidate set contains one domain, the entire input domain. A test case is selected, which does not detect failure. Hence, the domain is divided into four subdomains (regions 1, 2, 3, and 4). In this case, region 1 is the largest region, and so the next case is selected from within it. Again, no failure is detected, so region 1 is partitioned into subregions 1.1, 1.2, 1.3 and 1.4. The algorithm will continue by selecting the largest domain from the 7 candidates.

While calculating the effectiveness of static partitioning schemes has been performed, we believe it is extremely difficult to develop general results for the effectiveness of dynamic partitioning schemes, due to the interaction of the shape of the failure pattern, and the input domain. Therefore we used an experimental approach to measure its effectiveness on simulated failure patterns of the types described in the introduction.

3.2. Experimental evaluation

The algorithm's effectiveness was measured experimentally. A square test domain was used. For each test run, a single, randomly located, failure region of the required size to produce the desired failure rate, and of the desired pattern was created. Block patterns were simulated by randomly designating a square region of the input domain, of size such as to give the desired failure rate as the failure region. For strip failures, two points on adjacent sides were chosen, and a strip joining the two points was designated the failure region. The width of the strip was then chosen to achieve the desired failure region, this width of course varied from run to run. Points close to the corners were not chosen to avoid having overly wide strips. To simulate the point pattern, 50 circular regions were randomly chosen from within the input domain. Regions overlapping already-chosen regains were rejected and another candidate chosen until 50 non-overlapping regions were created. For each type of failure pattern, statistics were collected for several different failure rates. For each combination of failure pattern and rate, 5000 test runs were performed, and the average F-measure for each trial recorded.

From Table 1, we can see that for block patterns, ART by random division has an average F-measure between 20-25% less than random testing. For strip patterns, the F-measure for ART by random division is generally around 5% lower than for random testing, whereas for point patterns the difference is negligible.

4. ART by Bisection

4.1. Algorithm description

This method again relies on partitioning the input domain, but rather than partitioning on the current test case, the process divides the input domain up into equally-sized partitions. As the testing progresses, and a test case from each partition is selected, the partitions are bisected, and test cases are chosen from those subdomains not containing test cases. In this way, we have ensured that test cases are widely spread out as they reside in different partitions.

We call this method *ART by bisection*, as it effectively divides the input domain into equal sized partitions, bisecting each existing partition as more test cases are generated. It ensures that test cases continue to be widely spread by only selecting new cases from partitions which contain no previous test case.

We again present the algorithm in a form suitable testing a program with two real inputs, i, j, where $i_{\min} \leq i \leq i_{\max}, j_{\min} \leq j \leq j_{\max}$, which is trivially extensible to the n-dimensional or integer cases.

Pseudo-code for the algorithm is as follows:

Algorithm 2 Adaptive random testing by bisection

```
1. Set the untested region list,
   $L_{untested} = \{(i_{\min}, j_{\min})(i_{\max}, j_{\max})\}$, in other words a single region encompassing the entire test domain.
2. Set the tested region list, $L_{tested}$, to be null.
```

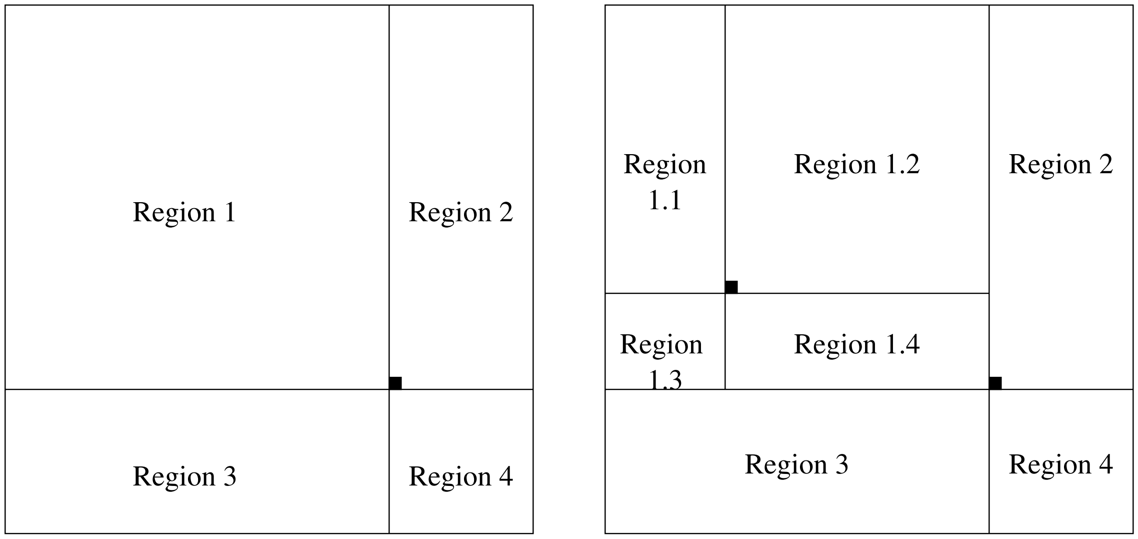

Figure 2. The operation of Algorithm 1 (ART by random partitioning). Each generated test case is used to further partition the region it was generated from.

Failure Rate	Expected F-measure of RT (F_{rt})	Block Pattern Mean F-msr of ART (F_{art})	Block Pattern (F_{art}/F_{rt}) (%)	Strip Pattern Mean F-msr of ART (F_{art})	Strip Pattern (F_{art}/F_{rt}) (%)	Point Pattern Mean F-msr of ART (F_{art})	Point Pattern (F_{art}/F_{rt}) (%)
0.01	100	77.4	77.4%	92.8	92.8 %	99.5	99.5 %
0.005	200	154.9	77.4%	194.4	97.1 %	199.1	99.6 %
0.002	500	390.0	78.0 %	477.5	95.3 %	498.5	99.7 %
0.001	1000	796.5	79.6 %	965.4	96.5 %	980.1	98.0 %

Table 1. F-measures for ART with random partitioning compared with random testing .

3. If $L_{untested}$ is not empty, randomly select a test region $T = \{(i,j)(s,t)\}$ from $L_{untested}$, and delete T from $L_{untested}$. Otherwise goto Step 7.
4. Randomly select a point, (x,y), within T (such that $i \leq x \leq s$ and $j \leq y \leq t$).
5. If the point (x,y) is a failure-causing input, report failure and terminate.
6. Otherwise, append the test region T to L_{tested}, and goto Step 3.
7. Set a temporary list L_{temp} to be null.
8. For each test region in $L_{tested}\{(i,j)(s,t)\}$, if $s - i \geq t - j$, set $p := \frac{s-i}{2}$, and divide the test region into two test regions, $\{(i,j)(p,t)\}$ and $\{(p,j)(s,t)\}$. Otherwise, set $q := \frac{t-j}{2}$, and divide the test region into two test regions, $\{(i,j)(s,q)\}$ and $\{(i,q)(s,t)\}$.
9. For each of the two test regions, if there is a test case in this test region, append the test region to L_{temp}, else append it to $L_{untested}$.
10. Rename L_{temp} to L_{tested}, and goto step 3.

Figure 3 shows Algorithm 2 in operation. Initially, in 3(a), the first test case is selected from the input domain. In Figure 3(b), another test case is selected randomly from the "empty" domain. In 3(c), a further subdivision has taken place, with the third and fourth tests selected from the empty subdomains. The process continues in 3(d).

4.2. Experimental analysis

Algorithm 2 was tested in the same manner as Algorithm 1.

Table 2 shows that for block patterns, ART by bisection outperforms random testing by approximately 25%.

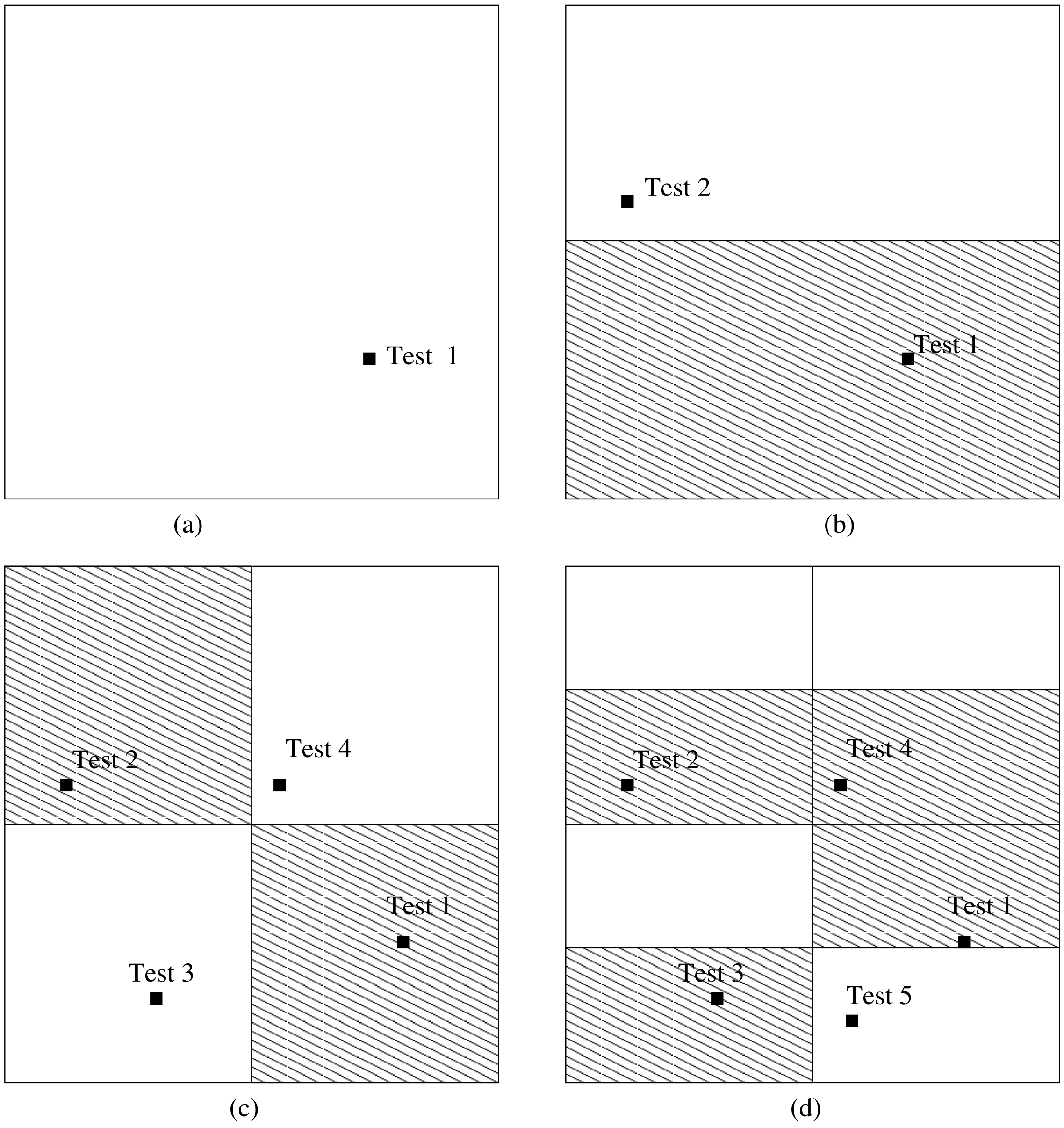

Figure 3. The operation of Algorithm 2 (ART by bisection)

For strip patterns, there is 5-8% improvement on random testing. The method is marginally more effective than random testing for point patterns.

5. Discussion

In all cases, the experimental results show that both new methods significantly outperform pure random testing on block patterns, make a slight but useful improvement over random testing for strip patterns, and are about the same as random testing on point patterns. This is consistent with the performance of other ART-based methods [2], but the methods are significantly lower in overhead than other ART methods proposed so far.

Since partition testing is generally considered to impose greater processing overhead than random testing, much research has been aimed at identifying conditions under which partition testing will perform better than random testing. Two conditions that have been identified are the Equal-Size-Equal-Number strategy (Weyuker and Jeng [10]), and the Proportional Sampling (PS) strategy (Chen and Yu [4]). The equal-size-equal-number strategy divides the input domain into partitions of equal size and selects an equal number of test cases from each. The proportional sampling strat-

Failure Rate	Expected F-measure of RT (F_{rt})	Block Pattern: Mean F-msr of ART (F_{art})	Block Pattern: (F_{art}/F_{rt}) (%)	Strip Pattern: Mean F-msr of ART (F_{art})	Strip Pattern: (F_{art}/F_{rt}) (%)	Point Pattern: Mean F-msr of ART (F_{art})	Point Pattern: (F_{art}/F_{rt}) (%)
0.01	100	72.1	72.1 %	91.9	91.9 %	97.7	97.7 %
0.005	200	148.8	74.3 %	191.0	95.5 %	178.7	98.1 %
0.002	500	368.0	73.6 %	475.3	95.1 %	466.4	98.7 %
0.001	1000	741.0	74.1 %	966.2	96.6 %	967.5	96.8 %

Table 2. F-measures for ART with bisection compared with random testing .

egy requires a uniform sampling rate of test cases in all partitions (that is, the number of test cases selected from a partition is proportional to its size). Both of these strategies have been shown to have an equal or greater probability of detecting at least one failure than random testing.

The new algorithm, ART by bisection, exhibits properties of both the Equal-Size-Equal-Number and Proportional Sampling strategies. At any time, all partitions are of the same size and at most one test case is drawn from any partition. In other words, in addition to using fewer test cases to detect the first failure than random testing, ART by bisection also has an equal or greater probability of detecting at least one failure than random testing. Such an additional advantage has not yet been guaranteed by other ART methods.

Recently, Chen et al. [3] have reported interesting similarities between ART and PS. First, “there is still an element of randomness associated with the PS strategy”. Secondly, they state that “intuitively, enforcing a uniform sampling rate somehow attempts, via the process of partitioning, to evenly spread out the test cases”. They also point out that the motivation and intuition behind ART and PS are substantially different. ART was proposed with the goal of performing better for block and strip failure patterns, in terms of the number of test cases to detect the first failure. PS was proposed to perform better than random testing in terms of the probability of detecting at least one failure, in the absence of any information, albeit the failure pattern. ART by bisection clearly provides an example of a testing method which is both an efficient implementation of ART and meets the conditions of a PS strategy. It is of great interest to understand why such striking similarities exist despite the fact that ART and PS were developed from very different intuitions.

Chan et al. [2] have used the principle of exclusion to achieve wide spread of random test cases in their development of alternative ART implementations. ART by exclusion selects test cases only from parts of the input domain which are distant from all previously executed test cases. This is done by defining an exclusion region around each previously executed but non failure-causing test case. In ART by bisection, new test cases are not selected from partitions that contain already executed test cases. Therefore, ART by bisection can be viewed as using the “principle of exclusion” to achieve wide spread.

When compared against other ART methods, the two presented incur considerably lower overheads. Calculations of “distance” are not required, nor are candidates generated and discarded without use - the two most computationally expensive operations in previous ART implementations ([8], [2]). The methods presented clearly demonstrate that the overheads of implementing ART can be significantly reduced, whilst retaining its much improved effectiveness over pure random testing. We believe these methods offer a significant practical improvement over pure random testing.

Though our methods involve the notion of partitioning, they are very different from conventional partition testing strategies where partitioning is normally performed prior to the selection of any test cases. So, while our methods can be viewed as a kind of partition testing, all partitioning is done “on the fly” so that the cost of processing is proportional to the number of test cases that have already been executed. In addition, there is no need for the number of test cases to be determined in advance. This can be a major advantage in practice, since it is common practice to stop testing after the first failure is detected.

Our method does have some limitations, some of which are shared by all random testing methods. Obviously, our method has the same limitations as random testing with regards to the requirement for a testing oracle. A program fault that causes a very low failure rate may not be effectively detected by random testing, and the improvement made by our methods is unlikely to provide much assistance in this case. It is also not straightforward to generate random tests for programs which take complex data structures as input. If some of the program’s input parameters are suitable for dynamic partitioning, but others are not, a hybrid approach may be practical where the non-partitionable parameters are simply randomly generated.

In conclusion, we have proposed two innovative test case generation methods. They have been demonstrated to use significantly fewer test cases to detect the first failure than pure random testing, and have lower selection overheads than previous ART methods. Furthermore, the two methods integrate the concepts of random testing and partition testing. As far as we know, this is the first attempt to integrate the two concepts. Our experimental results suggest to testers that if random testing is suitable for their purpose, it is worthwhile to seriously consider using our two new methods instead.

6. Acknowledgements

This research was partially supported by the Discovery Project Grant of the Australian Research Council (DP0345147).

References

[1] 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*, 38:775–782, 1996.

[2] K. P. Chan, T. Y. Chen, and D. Towey. Restricted random testing. In *Proceedings of the 7th European Conference on Software Quality*, Lecture Notes in Computer Science, 2002.

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

[4] T. Y. Chen and Y. T. Yu. On the relationship between partition and random testing. *IEEE Transactions on Software Engineering*, 20:977–980, December 1994.

[5] R. H. Cobb and H. D. Mills. Engineering software under statistical quality control. *IEEE Software*, 7:44–54, 1990.

[6] R. Hamlet. Random testing. In *Encyclopedia of Software Engineering*, pages 970–978. Wiley, New York, 1994.

[7] P. S. Loo and W. K. Tsai. Random testing revisited. *Information and Software Technology*, 30:402–417, 1988.

[8] I. K. Mak. On the effectiveness of random testing. Master's thesis, Department of Computer Science, The University of Melbourne, 1997.

[9] G. J. Myers. *The Art of Software Testing*. John Wiley and Sons, Inc, 1979. ISBN 0-471-04328-1.

[10] E. J. Weyuker and B. Jeng. Analysing partition testing strategies. *IEEE Transactions on Software Engineering*, 17:703–711, July 1991.

[11] L. White and E. Cohen. A domain strategy for computer program testing. *IEEE Transactions on Software Engineering*, 6:247–257, 1980.