Adaptive Random Testing with Filtering: An Overhead Reduction Technique

Kwok Ping Chan  
Department of Computer Science,  
University of Hong Kong, Pokfulam,  
Hong Kong SAR, China  
kpchan@cs.hku.hk

T. Y. Chen  
Faculty of Information and Communication Technologies,  
Swinburne University of Technology, Hawthorn 3122, Australia  
tychen@it.swin.edu.au

Dave Towey  
Department of Computer Science,  
University of Hong Kong, Pokfulam,  
Hong Kong SAR, China  
dptowe@cs.hku.hk

Abstract

Adaptive Random Testing (ART) is an approach to testing software based on Random Testing (RT), but incorporating additional mechanisms to ensure a more widespread and even distribution of test cases over the input domain. It has been found that ART, under certain conditions, can significantly outperform RT, in terms of number of test cases required to detect a failure (a measure referred to as the F-measure). One implementation of ART, based on the use of exclusion zones and restriction of test case selection to outside of these zones, is Restricted Random Testing (RRT). In this paper, we present an overview of the basic RRT method, using circular and spherical exclusion regions, and then introduce an alternative exclusion shape, motivated by the promise of lower computational costs. Investigation into this alternative shape (square) exclusion method lead to a hybrid implementation of RRT, called filtering. Filtering enables the combination of the computationally cheaper square exclusion shape and the faster (for failure finding) original, circular exclusion shape. Simulation and experimental evidence are also presented supporting the methods.

1. Introduction

Software Testing has often been categorized into either White Box or Black Box methods, the primary distinction being that White Box testing methods make use of structural information of the Software Under Test (SUT), while Black Box methods have no such information, and only have access to the SUT’s input and output information.

One particularly simple implementation of Black Box testing is Random Testing (RT). Random Testing selects test cases (combinations of inputs) representing a single use of the software at random from the input domain [12]. Its usefulness as a testing strategy [11, 13] has been debated for many years, but its simplicity and ease-of-use make it an attractive option for many situations, from early stages of software development [14] to safety critical applications [10].

Adaptive Random Testing (ART) techniques [9] are Black Box testing methods based on RT. One implementation of ART is Restricted Random Testing (RRT) [7, 8], also known as Restricted ART. Tests and simulations involving ART have shown it to significantly outperform ordinary Random Testing, by an average of up to 40%, in terms of the number of test cases required to find a failure. It has been suggested that whenever RT has been selected as the testing method, it may be worthwhile replacing it with ART [9].

One source of overhead in the RRT methods is the necessity of calculating the distances between various test cases, to ensure they lie outside the exclusion regions. Because the methods use a circular or spherical restriction shape, the use of alternative shapes may lead to a reduction in computations. This, in part, motivated our investigation into RRT with different exclusion shapes.

A second motivation behind the investigation is the known difficulty of approximating the Maximum Target Exclusion Ratio (Max R) [7, 8]. The Max R is the value for the main control parameter at which we can achieve best performance; the difficulty associated with its estimation is related to the shape of the exclusion regions. Since a simpler exclusion region shape should make estimation of Max R easier, this represented another excellent reason to investigate alternative exclusion shapes.

Investigations into alternative exclusion shapes led to an overhead reduction method based on a combination of the desirable features of different exclusion region shapes. This method, called filtering, reduces much of the computational overheads, but maintains the failure finding efficiency rates.

The rest of this paper is laid out as follows: in Section 2, the background information, and original motivation of the ART methods is outlined. One version of ART, Restricted Random Testing (RRT), is then explained in some detail. In Section 3, a variation of RRT, using square exclusion regions, is explained and investigated. Some simulation and experimental data is presented for the method. In Section 4, the new overhead reduction technique of filtering is explained, and in Section 5, we offer some conclusions and discussion.
2. Background

2.1. Failure Patterns and the F-measure

![Failure Patterns Diagram]

Failure patterns are those patterns of test cases in an input domain which, when applied to the SUT, result in a failure, or reveal an error. These patterns have been categorized into three major types [3]: point, strip, and block. Figure 1 gives examples of these failure pattern types in two dimensions (2D). In the figure, the shaded areas represent the failure-causing regions, and the borders represent the outer boundaries of the input domain. For point-type patterns, the failure-causing test cases (points) are individual or in small groups. The strip-type patterns are characterised by a narrow strip of failure-causing inputs. And in the block-type patterns, the failure-causing inputs are concentrated in contiguous regions.

Previous investigations have suggested that point-type failure patterns are far less common than the other two types [3]. It has also been found that, when the failure pattern is not point-type, the failure-finding efficiency of Random Testing (RT) can be improved by slightly modifying the basic test case selection pattern [9]. The Adaptive Random Testing (ART) methods were motivated by this insight.

The F-measure is an increasingly popular measure of how effective a testing strategy is [7, 8, 9]. It represents the expected number of test cases that will be required to find a first failure in the software. Obviously, the lower the F-measure, the faster the testing strategy has identified a failure or error.

2.2. Restricted Random Testing

Restricted Random Testing (RRT) [7, 8], also known as Restricted ART (R-ART), is one implementation of ART. It is based on the use of exclusion regions, and restriction of test case selection to outside of the excluded areas. When testing according to the RRT method, the input domain from which test cases may be selected is restricted to only those regions not close to previously executed test cases. In simulations, RRT has been shown to outperform RT by an average of about 40%, in terms of the F-measure.

2.3. Exclusion Region

To apply RRT, an Exclusion Ratio (R), correlated to the entire input domain size, is determined. For example, a target of 95% exclusion of the input domain may be set. According to the dimensionality of the input domain, and the number of executed test cases, an exclusion region size for each executed test case is calculated. For example, in a 2D input domain, with total area of 100, if R = 95%, and if there are 10 previously executed test cases, RRT will impose 10 exclusion regions, each of area 9.5, centered on each executed test case. After the next test case, when there are 11 executed cases, RRT will then impose 11 exclusion regions, each of approximate area 8.6, around the executed test cases. In other words, the size of the exclusion zones is kept the same for each test case, but decreases for each successive execution.

As explained previously [4], due to the circular shape of the exclusion region, and other features of the RRT algorithm, the Actual Exclusion Ratio may be less than the Target Exclusion Ratio. It has also been established that the performance of the RRT algorithm is best, in terms of faster finding of failure, with higher values of R. This trend continues to a certain value, beyond which the algorithm ceases to function. The cause for this is linked to the Actual Exclusion ratio, which is assumed to be so close to 100% that test cases can no longer be generated. The value of the Exclusion Ratio beyond which the Actual Exclusion ratio is 100% is called the Maximum Exclusion Ratio, or Max R.

Because empirical evidence suggests that best failure-finding rates are obtained when the Max R value is used, it is obviously desirable to select this value. Problems arise though when the input domain dimensions are not proportional to each other, as the Max R in these cases is less well known. This was one of the motivations behind the Normalized version of RRT [7], which, by normalizing the input domain, enabled the selection of Max R with some confidence.

2.4. Ordinary and Normalized RRT

The Ordinary RRT (ORRT) method imposes a uniform exclusion zone centered on each executed test case: circular in 2D; spherical in 3D; etc. For programs whose input domains are less homogeneous (e.g., not square in 2D), this use of a fixed exclusion shape may result in an unexpected bias of the exclusion region. To alleviate this potential difficulty, a normalizing feature was incorporated into the RRT algorithm to produce the Normalized RRT [7] method, which uses a virtual input domain to implement the test case selection and exclusion. This virtual input domain is a unit square, cube, or hypercube, according to the dimensions of the original input domain. Using the NRRT method, test
cases are initially selected from the virtual input domain, and are then scaled to the actual input domain of the program, and executed. When an executed test case does not cause a failure, an exclusion zone around the initial point (in the virtual input domain) is defined, and subsequent test cases are drawn from outside this zone. Because the virtual input domain is homogeneous, the value of Max R can be estimated in advance, and hence the algorithm can be applied with the optimal Exclusion Ratio.

As explained previously [4], there is a potential distortion of the failure pattern (and hence influence on the failure-finding efficiency) with NRRT. The distortion can have a favourable or unfavourable effect, depending on the relative shape and position of the failure pattern within the input domain. Because this information is not usually available in advance of testing, the NRRT method may not be the best choice of testing strategy. However, because of the very positive results obtained with NRRT, and due to the fact that we do have the requisite information about failure patterns for our simulations and experiments, the NRRT method has been included in this investigation.

2.5 Overheads

Because RRT uses restriction to ensure a widespread distribution of test cases over the input domain, when evaluating the suitability of a potential test case it is necessary to determine whether or not it lies inside any restricted area. This requires calculation of the distances between the potential test case and all previously executed test cases, and a comparison with the exclusion radius (the length of which determines the distance from a previously executed test case that the exclusion region extends).

This means that the method may incur potentially significant overheads in the generation of the (m+1)th test case. At this instant there are already m exclusion regions around m executed test cases, and the (m+1)th test case is restricted to coming from outside these regions. A simple implementation of the exclusion region is to ensure that the candidate test case is a greater distance from the executed test case than the radius of the exclusion region. For two points, P and Q ((p_1, p_2, ..., p_n) and (q_1, q_2, ..., q_n)), the Euclidean distance between the points can be calculated from the following expression:

$$\sqrt{\sum_{i=1}^{N} (p_i - q_i)^2}$$  \hspace{1cm} (1)

Ignoring possible optimizations, in a best case scenario, where the 1st candidate test case is outside all exclusion regions, there are m distance calculations required to confirm the (m+1)th test case as acceptable. In reality, it is possible that several attempts at generating an acceptable test case will be required. For each unacceptable candidate, there will have been x number of comparisons (and hence x distance calculations) prior to that comparison revealing the test case to be within an exclusion region. The value of x will be between 1 and m, the worst case being that the candidate is found to lie within the final exclusion region checked. Normally, a constraint on the maximum number of attempts (Max) to generate a single, acceptable test case is imposed. As with other methods, we are interested in ways by which the computational overhead may be reduced. One such way is through the use of alternative, simpler, exclusion shapes.

3. Restricted Random Testing with Square Exclusion Regions

Motivated by some shortcomings of the basic RRT method, in particular the computational overheads and the difficult determination of the Max R, some alternative strategies have been investigated. One obvious alternative strategy involves a change in the exclusion region shape, from circular to square. Square exclusion regions (cube, or hyper-cube in higher dimensions) should enable a more easy prediction of the Max R. It should also permit a simpler check for whether a test case lies within its bounds, thereby reducing the computational costs associated with determining the suitability of a potential test case.

We implemented RRT with square exclusion regions (RRT SQ), and investigated the failure finding efficiency through some simulations and experiments with several previously studied error-seeded programs [7, 8].

3.1 Simulation

In the simulations, a failure region was simulated by randomly locating a block shape failure pattern in the input domain. We then applied RT, RRT CIR (the ordinary, circular exclusion method of RRT), and RRT SQ. We applied the RRT methods with different Target Exclusion Ratios (R), increasing to the Max R, and repeated the experiments 10,000 times. Figure 2 shows a comparison between the calculated F-measure for RT, RRT CIR and RRT SQ in 2D, where the failure-causing region was approximately 0.1% of the entire Input Domain.
Figure 2. Comparison of F-measures for Random Testing (RT), Restricted Random Testing with Square Exclusion (RRT_SQ), and Restricted Random Testing with Circular Exclusion (RRT_CIR)

The figure clearly shows a significant improvement over RT with the RRT methods. The characteristic curve, showing a fall in F-measure as the Target Exclusion Ratio (R) increases, is also visible. Although both RRT_CIR and RRT_SQ appear to give similar performance, it should be noted that the Max R for the circular exclusion method is higher (150%) and that the best results (obtained when Max R for each method is used) reveal that the square exclusion method does not obtain results as good as those obtained by the circular exclusion method (approximately, F-measure of 640 for RRT_SQ compared with 580 for RRT_CIR).

3.2 Error-Seeded Program Experiment

In addition to the simulations, we applied the RRT_SQ method to seven error-seed programs that have been used in previous investigations [7, 8]. These are published programs [1, 15], all involving numerical calculations, written in C++ and varying in length from 30 to 200 statements. They vary in dimension from 1 to 4, and were seeded with errors using mutation operations [2]. Four types of mutant operators were used to create the faulty programs: arithmetic operator replacement (AOR), relational operator replacement (ROR), scalar variable replacement (SVR) and constant replacement (CR). These mutant operators were chosen since they generate the most commonly occurring errors in numerical programs [2]. For each program, after seeding in the errors, the range of each input variable was then set such that the overall failure rate would not be too large. Table I summarizes the details of the programs.
Table 1. Program name, dimension (D), input domain, seeded error types, total number of errors, and failure rates for each of the error-seeded programs. The error types are: arithmetic operator replacement (AOR); relational operator replacement (ROR); scalar variable replacement (SVR) and constant replacement (CR)

<table>
<thead>
<tr>
<th>Program Name</th>
<th>D</th>
<th>Input Domain</th>
<th>Error Type</th>
<th>Total Errors</th>
<th>Failure Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>From</td>
<td>To</td>
<td>AOR</td>
<td>ROR</td>
</tr>
<tr>
<td>bessj</td>
<td>2</td>
<td>(2.0, -1000.0)</td>
<td>(300.0, 15000.0)</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>bessj0</td>
<td>1</td>
<td>(-300000.0)</td>
<td>(300000.0)</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>cel</td>
<td>4</td>
<td>(0.001, 0.001, 0.001, 0.001)</td>
<td>(1.0, 300.0, 10000.0, 10000.0)</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>erfcc</td>
<td>1</td>
<td>(-300000.0)</td>
<td>(300000.0)</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>gammq</td>
<td>2</td>
<td>(0.0, 0.0)</td>
<td>(1700.0, 40.0)</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>plgndr</td>
<td>3</td>
<td>(10.0, 0.0, 0.0)</td>
<td>(500.0, 11.0, 1.0)</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>sncndn</td>
<td>2</td>
<td>(-5000.0, -5000.0)</td>
<td>(5000.0, 5000.0)</td>
<td>4</td>
<td>1</td>
</tr>
</tbody>
</table>

In previous investigations [2, 3], the original ORRT and NRRT methods (using circular exclusion regions) were applied to the error-seeded programs. For the current investigation, we applied the alternative (square exclusion) versions to these error-seeded programs, varying the target exclusion ratio (R), and averaging the results over a sample size of 5,000. A summary of the maximum target exclusion ratios (Max R), the corresponding F-measure, the best results, and the corresponding target exclusion ratio (R) are given below in Table 2. The corresponding information for the original methods (using circular exclusion regions), taken from Chan et al. [7], is presented in Table 3.

Table 2. Program name, dimension (D), Max R, Improvement (Imp) over RT at Max R, R for best improvement, and best improvement over RT for the error-seeded programs, using ORRT_SQ and NRRT_SQ

<table>
<thead>
<tr>
<th>Program Name</th>
<th>D</th>
<th>Max Imp</th>
<th>Imp at Max R</th>
<th>R for Best Imp</th>
<th>Best Imp</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORRT SQ</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>bessj</td>
<td>2</td>
<td>220%</td>
<td>53.67%</td>
<td>210%</td>
<td>54.06%</td>
</tr>
<tr>
<td>bessj0</td>
<td>1</td>
<td>100%</td>
<td>42.73%</td>
<td>100%</td>
<td>42.73%</td>
</tr>
<tr>
<td>cel</td>
<td>4</td>
<td>51,400%</td>
<td>62.69%</td>
<td>51,300%</td>
<td>62.77%</td>
</tr>
<tr>
<td>erfcc</td>
<td>1</td>
<td>100%</td>
<td>45.67%</td>
<td>100%</td>
<td>45.67%</td>
</tr>
<tr>
<td>gammq</td>
<td>2</td>
<td>210%</td>
<td>19.25%</td>
<td>210%</td>
<td>19.25%</td>
</tr>
<tr>
<td>plgndr</td>
<td>3</td>
<td>430%</td>
<td>50.21%</td>
<td>430%</td>
<td>50.21%</td>
</tr>
<tr>
<td>sncndn</td>
<td>2</td>
<td>100%</td>
<td>-0.24%</td>
<td>30%</td>
<td>1.32%</td>
</tr>
</tbody>
</table>

| NRRT SQ     |   |         |               |                |          |
| Max Imp     |   |         |               |                |          |
| bessj        | 100% | 13.87%  | 90%           | 15.99%         |
| bessj0       | 100% | 42.73%  | 100%          | 42.73%         |
| cel          | 130% | 38.55%  | 130%          | 38.55%         |
| erfcc        | 100% | 45.67%  | 100%          | 45.67%         |
| gammq        | 100% | 15.24%  | 90%           | 15.63%         |
| plgndr       | 100% | 38.17%  | 100%          | 38.17%         |
| sncndn       | 100% | -0.24%  | 30%           | 1.32%          |
Table 3. Program name, dimension (D), Max R, Improvement (Imp) over RT at Max R, R for best improvement, and best improvement over RT for the error-seeded programs, using ORRT_CIR and NRRT_CIR. (Taken from Chan et al. [7])

<table>
<thead>
<tr>
<th>Program Name</th>
<th>D</th>
<th>Max R</th>
<th>Imp at Max R</th>
<th>R for Best Imp</th>
<th>Best Imp</th>
</tr>
</thead>
<tbody>
<tr>
<td>bessj</td>
<td>2</td>
<td>220%</td>
<td>56.74%</td>
<td>220%</td>
<td>56.74%</td>
</tr>
<tr>
<td>bessj0</td>
<td>1</td>
<td>100%</td>
<td>43.03%</td>
<td>100%</td>
<td>43.03%</td>
</tr>
<tr>
<td>cel</td>
<td>4</td>
<td>32,000%</td>
<td>64.31%</td>
<td>32,000%</td>
<td>64.31%</td>
</tr>
<tr>
<td>erfcc</td>
<td>1</td>
<td>100%</td>
<td>46.24%</td>
<td>100%</td>
<td>46.24%</td>
</tr>
<tr>
<td>gammq</td>
<td>2</td>
<td>200%</td>
<td>12.76%</td>
<td>170%</td>
<td>13.94%</td>
</tr>
<tr>
<td>plgndr</td>
<td>3</td>
<td>460%</td>
<td>46.84%</td>
<td>450%</td>
<td>46.94%</td>
</tr>
<tr>
<td>sncndn</td>
<td>2</td>
<td>150%</td>
<td>1.05%</td>
<td>120%</td>
<td>3.01%</td>
</tr>
</tbody>
</table>

Figure 2. Improvement in F-measures for the Restricted Random Testing methods, compared with the Random Testing F-measure. The four sets of data refer to the square (_SQ) and original (_CIR) exclusion region implementations of ORRT and NRRT. All figures refer to the improvement over the calculated RT.

Figure 2 shows a comparison of the best performance of the RRT methods relative to the calculated F-measure result for Random Testing. The figure shows the percentage improvement for each method over the RT F-measure.

\[
\text{Percentage improvement} = \frac{(F\text{-measure}_{\text{RT}} - F\text{-measure}_{\text{method}})}{F\text{-measure}_{\text{RT}}} \times 100
\]

1 Percentage improvement is calculated as follows:
In the experiment, because the algorithm in 1 dimension is the same for both circular and square exclusion methods (i.e. both methods implement exclusion with a line), those programs with only 1-dimensional input domains (bessj0 and esfes) were expected to have identical results. As can be seen from Figure 2, this is the case. As has been noted before [7, 8], because the snenrd program has a point-type failure pattern, it is not expected that the RRT methods would improve much over RT. The figure shows that all RRT results for snenrd are similar and do not represent much improvement over RT.

From the simulation results, it might be expected that the error-seeded program experiments would reveal a clear result, showing the original methods (CIR) to consistently outperform the square exclusion methods (SQ). As can be seen from Figure 2, this is not the case: even though the best results are obtained with the original circular exclusion shapes, the results for the square exclusions are very comparable. The distinct difference in performance noted between the ORRT and NRRT versions of the original method [7, 8] can also be seen for the bessj and cel programs for the square exclusion, although for cel, it appears that the ORRT version gives best performance for the square exclusion regions, whereas the NRRT version was better for the original circular exclusion method. For the gammy and plghdr programs, the distinction is not as clear, but again it appears opposite to the original findings, with the ORRT method seemingly yielding better results than NRRT. The difference, however, is not large. As previously noted [6, 7, 8], the failure region in the bessj program, an incomplete strip or narrow block type, appears to be more easily found with the ORRT version of the algorithm. The reason for this has been identified as being due to the distortion of the failure pattern by the NRRT method [4].

Although the RRT SQ method does indeed reduce the computational overheads of RRT, and thereby increase the speed with which it can execute, it appears to have a (slightly) poorer performance in terms of number of test cases required to find a failure. This was seen in both the simulations and experiments with error-seeded programs. In addition, one of the more attractive aspects of the basic RRT method is that the exclusion regions, being circular, ensure a constant minimum distance amongst executed test cases: all executed test cases are at least the length of the exclusion radius apart. The difference between the minimum and maximum exclusion distances for the square is not large, however, and as the number of test cases increases, this difference fades.

4. Filtering

Although the square exclusion region implementation of RRT (RRT SQ) has a slightly poorer failure-finding performance, it does still have some very attractive features, including the much faster and cheaper computation of inequalities compared with the distance calculations in the basic RRT method.

Motivated by the lower computation overheads of the square exclusions, a hybrid approach, called filtering, was developed. In this approach, we use a Bounding Square/Cube/Hypercube, and filter the executed test cases through; calculating the distance only for those executed test cases inside the Bounding Region. The Bounding Region corresponds to a square/cube restriction zone, requiring only the cheaper inequality operation. The number of test cases that will lie inside the Bounding Region is significantly less than the total number in the entire input domain. With normal distribution of test cases, the number expected to fall inside the bounding region is proportional to the size of the bounding region compared to the size of the entire input domain (m is the number of executed test cases).

\[
\text{Expected number of test cases in Bounding Region} = m \times \frac{\text{Size of Bounding Region}}{\text{Size of Input Domain}}
\]

The magnitude of the Bounding Region size is twice that of the exclusion region radius.

\[
\text{Size of Bounding Region} = [2r]^n
\]

The value of the exclusion radius (r) depends on the size of each exclusion region, which in turn depends on the size of the entire input domain, the Exclusion Ratio (R), and the total number of exclusion regions (m).

\[
\text{Size of Exclusion Region} = \frac{R \times \text{Size of Input Domain}}{m}
\]

The formula to calculate the radius changes according to the dimensions of the input domain. Table 4 summarizes the expected number of test case to fall inside a bounding region for 2, 3 and 4 dimensions.

Experiments have verified the filtering method’s speed compared with the ordinary RRT_CIR implementation, while maintaining identical failure-finding results. Indeed, as Table 4 shows, the (maximum) number of test cases on which the more expensive RRT_CIR method will be applied is a small constant, determined by the size of the exclusion regions: e.g., with target exclusion rate (R) of 90%, in 2D, it is expected to be applied to less than 2, all other test cases being filtered.
Table 4. Expected number of test cases falling in Bounding Region, for 2, 3, and 4 Dimensions. \( A \) is the total input domain Area; \( m \) is the number of executed test cases; \( R \) is the Exclusion Ratio; and \( r \) is the radius of the exclusion regions.

<table>
<thead>
<tr>
<th>( N )</th>
<th>Area/Volume/ Hyper Volume (for circle, sphere, etc)</th>
<th>Expected number of test cases inside Bounding Region</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>( \pi \times r^2 = \frac{A \times R}{m} )</td>
<td>( \frac{4R}{\pi} ) E.g. ( R = 150% ) means less than 2 test cases expected</td>
</tr>
<tr>
<td>3</td>
<td>( \frac{4}{3} \pi \times r^3 = \frac{A \times R}{m} )</td>
<td>( \frac{6R}{\pi} ) E.g. ( R = 270% ) means less than 6 test cases expected</td>
</tr>
<tr>
<td>4</td>
<td>( \frac{1}{2} \pi^2 \times r^4 = \frac{A \times R}{m} )</td>
<td>( \frac{32R}{\pi^2} ) E.g. ( R = 430% ) means less than 14 test cases expected</td>
</tr>
</tbody>
</table>

5. Discussion / Conclusion

In this paper we have present results from a recent investigation into the use of an alternative exclusion shape in the Restricted Random Testing method \([7, 8]\). The investigation was prompted by the desire for simpler and computationally cheaper methods of performing exclusion, compared with the traditional use of a circular exclusion shape. The alternative shape selected was a square, and the resulting method (\( RRT_{SQ} \)), in additional to reducing the computation overheads associated with exclusion calculation, performed very well in terms of the number of test cases required to find a failure (the \( F \)-measure).

Incorporating the lower overheads of \( RRT_{SQ} \) and the better failure-finding ability of \( RRT_{CIR} \), a new method was then presented, \( filtering \). This method uses the cheaper \( RRT_{SQ} \) method to filter previously executed test cases from outside a bounding region, and then applies the more efficient (for failure-finding) \( RRT_{CIR} \) method to any remaining test cases within the bounding region. It should be noted that \( filtering \) is applicable to anywhere the basic \( RRT \) method has been applied. This means that overhead reductions may be possible in \( Fuzzy ART \) \([6]\), \( Mirrored ART \) \([4]\), and in the \( Ageing \) methods \([5]\), all of which incorporate some aspects of the exclusion principle.

Acknowledgement

We should like to gratefully acknowledge the support given to this project by the Research Grants Council of the Hong Kong Special Administrative Region, China (Project No: HKU 7007/99E), and a Discovery Project Grant of the Australian Research Council.

References