Restricted Random Testing: Adaptive Random Testing by Exclusion.

Article *in* International Journal of Software Engineering and Knowledge Engineering · August 2006

Source: DBLP

3 authors, including:

Dave Towey
University of Nottingham Ningbo China
175 PUBLICATIONS **2,045** CITATIONS

SEE PROFILE

CITE AS:

Kwok Ping Chan, Tsong Yueh Chen, and Dave Towey (2006),
"Restricted Random Testing: Adaptive Random Testing by Exclusion",
International Journal of Software Engineering and Knowledge Engineering (IJSEKE),
Vol. 16, No. 4, pp.553-584.

Restricted Random Testing: Adaptive Random Testing by Exclusion

Kwok Ping Chan

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

Tsong Yueh 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, The University of Hong Kong,
Pokfulam, Hong Kong SAR, China
dptowey@cs.hku.hk

Restricted Random Testing (*RRT*) is a new method of testing software that improves upon traditional Random Testing (*RT*) techniques. Research has indicated that failure patterns (portions of an input domain which, when executed, cause the program to fail or reveal an error) can influence the effectiveness of testing strategies. For certain types of failure patterns, it has been found that a widespread and even distribution of test cases in the input domain can be significantly more effective at detecting failure compared with ordinary *RT*. Testing methods based on *RT*, but which aim to achieve even and widespread distributions, have been called Adaptive Random Testing (*ART*) strategies. One implementation of *ART* is *RRT*. *RRT* uses exclusion zones around executed, but non-failure-causing, test cases to restrict the regions of the input domain from which subsequent test cases may be drawn. In this paper, we introduce the motivation behind *RRT*, explain the algorithm and detail some empirical analyses carried out to examine the effectiveness of the method. Two versions of *RRT* are presented: Ordinary *RRT* (*ORRT*) and Normalized *RRT* (*NRRT*). The two versions share the same fundamental algorithm, but differ in their treatment of non-homogeneous input domains. Investigations into the use of alternative exclusion shapes are outlined, and a simple technique for reducing the computational overheads of *RRT*, prompted by the alternative exclusion shape investigations, is also explained. The performance of *RRT* is compared with *RT* and another *ART* method based on maximized minimum test case separation (*DART*), showing excellent improvement over *RT* and a very favorable comparison with *DART*.

Keywords: Software Testing; Random Testing; Adaptive Random Testing; Restricted Random Testing.

*All correspondence should be addressed to: Dave Towey, Department of Computer Science, The University of Hong Kong, Pokfulam, Hong Kong SAR, China. e-mail: dptowey@cs.hku.hk Tel: +852 2857 8266 Fax: +852 2915 5702

1. Introduction

Software Testing, the branch of Software Engineering associated with most software quality assurance, is an essential part of the software development process. Since exhaustive testing, the checking of all possible inputs, is usually prohibitively difficult or expensive, it is essential for testers to make best use of their limited testing resources.

A simple way of testing software is to draw test cases at random from the input domain and apply them to the Software Under Test (SUT), a method referred to as Random Testing (RT). There are many reasons why Random Testing is a popular choice: in addition to its simplicity, and the efficiency of test case generation [10], reliability estimates and statistical analyses are also easily performed. In spite of some controversy over its effectiveness, many real-life applications do make use of Random Testing [12, 13, 16, 17].

Previous research has indicated that, under certain conditions, the failure-finding efficiency of RT can be improved by ensuring that the selection of test cases be more evenly spread and well distributed over the input domain [5, 6, 11]. Methods based on RT, but which incorporate additional mechanisms to achieve an even distribution of test cases, have been called Adaptive Random Testing (ART) methods.

This goal of achieving a well-distributed test case selection can be achieved in a number of ways. An implementation proposed by Mak [11] (see also [7]), attempts to obtain an even spread of test cases by selecting that test case, from several potential cases, which has the largest minimum distance from previously executed cases. This Distance-based implementation of ART ($DART$) has been found to outperform RT by as much as 50%, in terms of the number of test cases required to find a failure region. Further to comparisons given previously ([4, 5, 6]), in this paper we present more data from a complete rerun of $DART$ on a full set of fault-seeded programs, and compare these results with results obtained when the different RRT methods were applied to this full set of fault-seeded programs.

An alternative approach to implement ART, based on the use of exclusion, is the Restricted Random Testing (RRT) method. By excluding regions surrounding previously executed test cases, and restricting subsequent cases to be drawn from other areas of the input domain, RRT ensures an even distribution, and guarantees a minimum distance amongst all cases. In experiments, the RRT method has outperformed RT by up to 80% on some occassions, again, in terms of the number of test cases required to find a failure region.

In the next section, the background and motivation for this research is explained. Some preliminary information and notation are also discussed. Section 3 explains the details of the method, and introduces its two variations: Ordinary RRT ($ORRT$) and Normalized RRT ($NRRT$). Although aspects of $ORRT$ and $NRRT$ have been presented previously [5, 6], further investigation and research have yielded deeper insights into the methods, and enabled a unified view of RRT. In this section, the control parameters of the algorithm, as well as factors influencing them, are

explained with some mathematical justification. Simulations and experiments involving some fault-seeded programs were carried out to examine the effectiveness of the *RRT* method; the details and results are given in Section 4. Also in Section 4, an investigation into the use of an alternative to the normal circular exclusion shape is explained. This square exclusion shape version of *RRT* led to a simple technique for reducing some of the computational costs of *RRT*, filtering. Filtering is also explained in this section. The initial investigation into alternative exclusion shapes and filtering [4] was expanded to include a fuller analysis and more comprehensive comparison over the complete set of fault-seeded programs. Discussion of the results, their implications, and suggestions for when and how to use the method are given at the end of Section 4. Finally, in Section 5, we offer a summary of this paper, and make some concluding remarks.

2. Preliminaries

2.1. *Background*

In Software Testing, test cases are points located in the input domain that are used to test the software for failures. A test case that reveals a failure is referred to as a failure-causing input [8]. Random Testing (*RT*) is the name given to approaches to testing software where the test cases are selected randomly from the input domain.

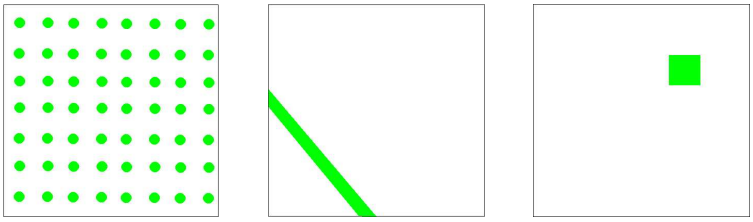
Random test case selection methods are usually implemented either with or without replacement of executed test cases [7]. Although selection with replacement has been criticized, particularly because test cases should not be duplicated, the assumption of replacement leads to simpler mathematical models and hence simpler analyses.

One metric used to evaluate the effectiveness of a testing method is the number of test cases required to find a (first) failure, called the *F-measure* [7, 11]. In addition to the obvious desire to find any failure-causing inputs as quickly as possible (i.e., with as few test cases as possible), since testing often stops when a failure is found, the *F-measure* is not only intuitively appealing, but is also a very realistic and useful metric. As elsewhere [4, 5, 6] the *F-measure* is used to evaluate the testing strategies presented in this paper.

Following the notation of Chen and Yu [8], for an input domain D, we let d denote the domain size, and m the size of the failure-causing input regions. The failure rate, θ, is then defined as m/d. For *RT* (with replacement), the expected number of test cases required to find the first failure (the *F-measure*) is equal to $1/\theta$, or equivalently, d/m. So, when using Random Testing on an input domain with a failure rate (θ) of 0.1%, we expect the *F-measure* to be 1000.

2.2. *Failure Patterns*

A failure pattern for a program is defined as the collection of all points within the input domain which, if executed, would reveal a fault or cause the Software Under



(a) Point Pattern (b) Strip Pattern (c) Block Pattern

Fig. 1. Examples of Failure Pattern Types.

```
int X, Y, Z;
cin >> X >> Y;
if ( ((X MOD 4) == 0)
     AND
     ((Y MOD 4) == 0) )
{
    Z = function_1(X, Y);
    /* CORRECT CODE IS
        Z = function_3(X, Y);
    */
} else {
    Z = function_2(X, Y);
}
cout << Z;
```

(a) Point Pattern

```
int X, Y, Z;
cin >> X >> Y;
if ( X + Y > 12)
/* CORRECT CODE IS
    if ( X + Y > 10)
*/
{
    Z = function_1(X, Y);
} else {
    Z = function_2(X, Y);
}
cout << Z;
```

(b) Strip Pattern

```
int X, Y, Z;
cin >> X >> Y;
if ( (X > 10) AND (X < 20) )
   AND
   ( (Y > 30) AND (Y < 40) )
{
    Z = function_1(X, Y);
    /* CORRECT CODE IS
        Z = function_3(X, Y);
    */
} else {
    Z = function_2(X, Y);
}
cout << Z;
```

(c) Block Pattern

Fig. 2. Code fragments producing examples of the different Failure Pattern Types.

Test to fail. Chan et al. [3] report on how different failure patterns can influence the performance of some testing strategies. They identify three major categories of failure patterns: *point*, characterised by individual or small groups of failure-causing input regions; *strip*, characterised by a narrow strip of failure-causing inputs; and *block*, characterised by the failure-causing inputs being concentrated in either one or a few regions. Examples of each of these are given in the schematic diagrams in Figs. 1a-1c, where the outer boundaries represent the borders of the input domain, and the shaded areas represent the failure-causing regions. Some fragements of (C++) code producing each of these types of failure pattern are given in Figs. 2a-2c.

Programs in which the failure-causing region is a large proportion of the entire input domain usually represent little difficulty in testing when compared with programs having a small failure region. In fact, it has been pointed out that detecting failure regions for programs where the proportion is high is a relatively trivial task, and can be performed by any reasonable testing strategy [5]. Programs where the failure-causing regions are comparatively small represent a far more interesting challenge to Software Testing methods in general, and are the focus of this paper.

It has been suggested that for non-point patterns, by slightly modifying the basic Random Testing technique, and requiring that the testing candidates be more evenly distributed, and far separated from each other, the failure detection capability can be significantly improved [7]. This suggestion motivates the Adaptive Random Testing (ART) methods.

2.3. *Adaptive Random Testing*

Adaptive Random Testing (ART) refers to those approaches to Software Testing which are based on Random Testing (RT), but which incorporate some additional mechanism to encourage a more widespread and even distribution of test cases over the input domain.

Mak proposed an ART strategy based on maximizing minimum distances between test cases [11]. This Distance-based version of ART ($DART$) makes use of two sets of test cases: the executed set, a set of test cases which have been executed but without causing failure; and the candidate set, a set of cases selected randomly from the input domain. Each time a test case is required for execution, the element in the candidate set that is "farthest away" from all the executed test cases is selected. When examining the elements to determine which is "farthest away," the Euclidean distance is calculated, in this way, test cases are selected such that those with the maximum minimum distance from the already executed test cases are chosen.

One version of this method is the Fixed Size Candidate Set ($FSCS$) version, in which the candidate set is maintained with a constant number of elements. Each time an element is selected and executed, the element is added to the executed set, and the candidate set is completely reconstructed. Mak reported that the failure finding efficiency of $FSCS$ improves as the size of the candidate set, k, increases, but that for $k \geq 10$, there is little difference in the total number of test cases required to find the first failure (*F-measure*) [11].

3. Method

An implementation of ART involving the use of exclusion areas, and the restriction of test case selection to outside of these areas, is called the Restricted Random Testing (RRT) method. When using a Random Testing (RT) method, aside from avoiding repetition, information related to previously executed test cases is not used. Test cases are repeatedly drawn from the input domain until a failure region is detected or a maximum number of test cases have been tried. By incorporating details of the executed test cases into the generation of subsequent cases, the RRT method is able to encourage a more evenly spread distribution of test cases in the input domain, which should improve the failure-finding efficiency for non-point failure patterns [7].

When testing according to the RRT method, given a test case that has not revealed failure, rather than simply select another test case randomly, the area of

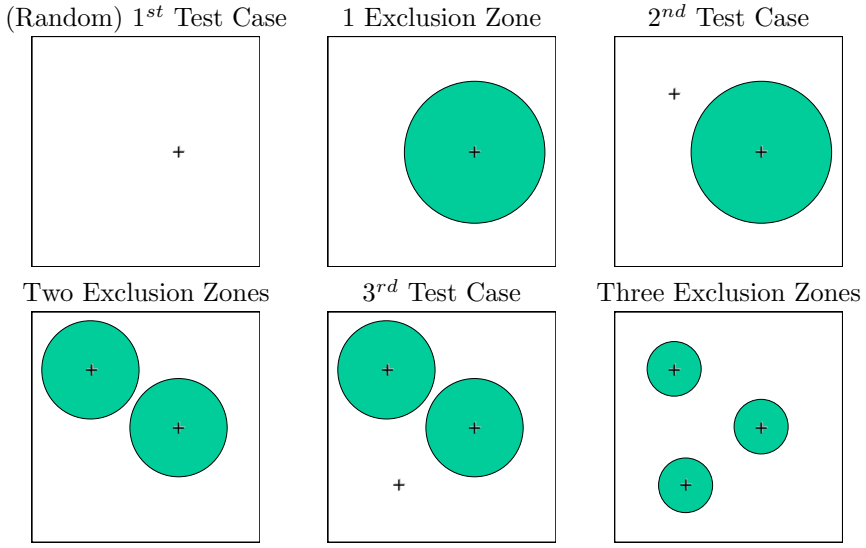


Fig. 3. Example of RRT Test Case generation process for first few test cases.

the input domain from which subsequent test cases may be drawn is restricted. In two dimensions, a circular exclusion zone around each non-failure-causing input is created, and subsequent test cases are restricted to coming from outside of these regions. By employing a circular zone, a minimum distance (the radius of the exclusion zone) between all test cases is ensured.

All exclusion zones are of equal size, and this size decreases with successive test case executions. The size of each zone is related to both the size of the entire input domain, and the number of previously executed test cases. For example, in two dimensions, with a target exclusion region area of A, if there are n points around which we wish to generate exclusion zones, then each exclusion zone area will be A/n, and each exclusion zone radius will be $\sqrt{A/(n\pi)}$.

A graphical representation of the generation of the first few test cases, using the *RRT* method, is shown in Fig. 3. As shown, after each test case is generated (and presumably applied to the program without revealing failure), exclusion zones are created around all non-failure-causing test cases and the next test case is selected from outside these excluded regions. Unlike *DART*, where test case selection is based on choosing the best test case from a candidate set, *RRT* selects the first candidate test case satisfying the exclusion criterion, that is, the first potential test case not lying within any excluded region.

3.1. *Ordinary and Normalized* RRT

The Ordinary *RRT* (*ORRT*) method imposes a uniform exclusion zone around each executed test case: circular in 2 dimensions; spherical in 3D; etc. For programs

whose input domains are not homogeneous (e.g., not square in 2D), this use of a fixed exclusion shape may result in a surprising bias of the exclusion region. To alleviate this potential difficulty, a normalizing feature was incorporated into the *RRT* algorithm to produce the Normalized *RRT* (*NRRT*) method, which uses an intermediate input domain to implement the test case selection and exclusion. Using the *NRRT* method, test cases are initially selected from a unit square, cube, or hypercube, with the dimensionality of the input domain set to be the same as that of the program being tested. Each selected test case is then mapped 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 normalized input domain) is defined, and subsequent test cases are drawn from outside this zone.

3.2. *Target Exclusion versus Actual Exclusion*

Although we attempt to maintain the area of the excluded zone as constant, due to overlapping of the zones, and portions of zones falling outside the input domain, the actual exclusion ratio may be less than the target exclusion. This phenomenon becomes more apparent as the target exclusion is increased, and is even more acutely so in less homogeneous (square) input domains. Table 1 shows some comparisons for target and actual exclusions within a square input domain, for different numbers of exclusion regions (10, 100, 500, 1000, and 5000), averaged over 1,000 trials.

Even though the actual exclusion differs from the target exclusion, the radii calculated remain the same. Since the actual exclusion varies depending on the magnitude of the target exclusion, the shape of the input domain, and the distribution of executed test cases, but the radius of the exclusion zone depends only on the target exclusion (R), in this paper we refer to the target exclusion when discussing our methods and results.

3.3. *Mathematical Support*

In the real world, when testing begins for a program, information about the failure regions is not available. In simulations and controlled experiments however, this information may be known. Having advance knowledge of what portion of the input domain is failure-causing (θ) enables us to calculate the expected *F-measure* by *RT* with replacement ($1/\theta$), which can be used as an upper bound (worst case) on judging the performance of other methods.

By making some assumptions about the *RRT* process, it is also possible to define an *Ideal RRT* method. Under normal execution of the basic *RRT* method, exclusion regions may frequently overlap, and portions of exclusion regions may also fall outside of the Input Domain boundaries (hence the difference between Target and Actual exclusion ratios). Additionally, because the exact location of the failure-causing regions is not known in advance, it is also possible that, at certain times, some portions of the failure-causing regions are inadvertently excluded from test

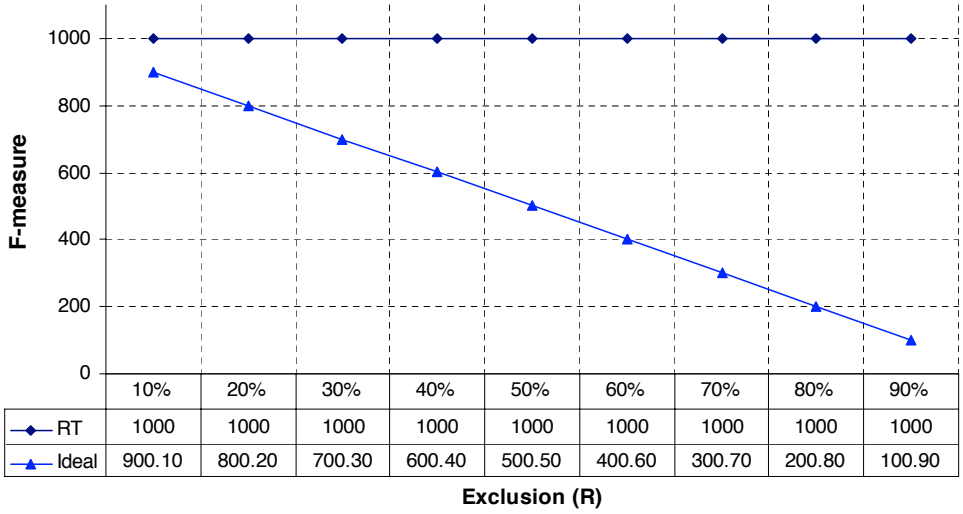
Table 1. Comparison of target and actual exclusion rates for *RRT*. Target Exclusion is the total area of the input domain which we attempt to exclude from random point generation. Actual Exclusion is average percentage of the input domain which is effectively excluded by the exclusion zones.

Target Exclusion	**Actual Exclusion**				
	Total Number of Exclusion Zones				
	10	**100**	**500**	**1000**	**5000**
1%	0.98%	0.99%	1.00%	1.00%	1.00%
10%	9.43%	9.72%	9.81%	9.83%	9.85%
20%	18.21%	19.04%	19.28%	19.34%	19.41%
30%	26.45%	27.98%	28.40%	28.50%	28.64%
40%	34.16%	36.51%	37.14%	37.29%	37.50%
50%	41.32%	44.56%	45.43%	45.65%	45.93%
60%	48.12%	52.13%	53.25%	53.52%	53.89%
70%	54.27%	59.22%	60.54%	60.86%	61.30%
80%	59.97%	65.73%	67.20%	67.59%	68.09%
90%	65.27%	71.59%	73.22%	73.64%	74.19%
100%	69.96%	76.71%	78.48%	78.93%	79.50%
110%	74.23%	81.20%	82.99%	83.41%	83.96%
120%	78.22%	85.02%	86.59%	86.98%	87.47%
130%	81.57%	87.92%	89.30%	89.58%	89.94%
140%	84.37%	89.99%	90.99%	91.15%	91.33%
150%	86.73%	91.44%	91.69%	91.66%	91.59%

case selection. The *Ideal* method assumes a constant area for the exclusion zones (i.e., assumes that the target exclusion and actual exclusion are identical) and also assumes that no "eclipsing" occurs (i.e., at no time is any portion of the failure-causing region excluded). Under these conditions, it is found that the expected *F-measure* for the *Ideal RRT* is $(d - dR + mR)/m$, where d is the size of the input domain, m is the total size of the failure-causing input region(s), and R is the proportion (expressed as a fraction) of the entire input domain that is excluded from the test case generation area. (The derivation is given in Appendix 1.)

Since the *Ideal* model represents a best-case scenario, it serves as a lower (best) bound on what performances we can hope to obtain with the *RRT* methods. Fig. 4 shows the expected *F-measure* values for the *Ideal RRT* and for Random Testing (*RT*).

As can be seen from the figure, the expected *Ideal RRT F-measure* values appear to improve significantly over the expected values for ordinary *RT*. Also, the improvement increases as the exclusion ratio (R) increases. However, some of the assumptions underlying the *Ideal* model become less valid as R increases. In particular, the *Ideal* model assumes that the actual exclusion is equal to the target exclusion, but as R increases such that the excluded area approaches the total area of the input domain (less that area consisting of failure-causing inputs), the possibility of maintaining the actual exclusion equal to the target exclusion, and avoiding eclipses, approaches zero. Therefore, although we expect the performance



	10%	20%	30%	40%	50%	60%	70%	80%	90%
RT	1000	1000	1000	1000	1000	1000	1000	1000	1000
Ideal	900.10	800.20	700.30	600.40	500.50	400.60	300.70	200.80	100.90

Fig. 4. Comparison of the expected *F-measures* for Random Testing (RT) and for the *Ideal* Restricted Random Testing (RRT) strategies for a failure-causing proportion (θ) of 0.1% of the entire input domain. The exclusion ratio (R) varies from 10% to 90%.

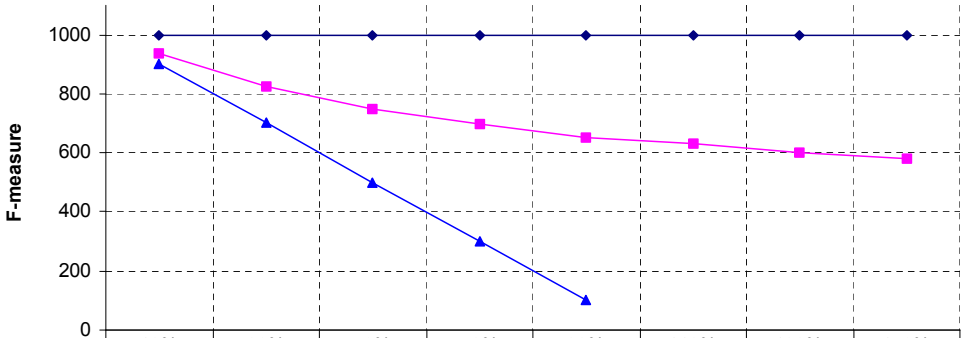
improvement over RT to increase as R increases, it should be less pronounced than the *Ideal* model predicts.

4. Empirical Study

In this section, we present some empirical evidence to support the efficiency of the RRT methods ($ORRT$ and $NRRT$). The first investigation involved the simulation of a failure region within a square input domain. The target exclusion ratio (R) was varied, and an *F-measure* for each R calculated. Results from the simulation indicated that knowledge of a Maximum Target Exclusion ratio ($Max\ R$) would benefit the tester.

Twelve previously studied fault-seeded programs were also used to examine the RRT methods. Seven of these programs were first tested using the $ORRT$ method, giving very favorable results [6]. Motivated by some shortcomings of the $ORRT$ method, in particular the difficulty of obtaining information on the $Max\ R$ in advance for non-homogeneous input domains, the Normalized RRT ($NRRT$) method was developed. $NRRT$ was also applied to the fault-seeded programs and compared with RT and $FSCS$, again with very favorable results [5].

We expanded the investigation to include all twelve fault-seeded programs, and compared $ORRT$ and $NRRT$ with RT and $FSCS$, obtaining very encouraging results, with improvement over RT of up to 80% on some occassions! We also adapted the basic RRT algorithm by changing the exclusion shape from circle to square (thereby reducing some of the overheads). We then applied this alternative method



	10%	30%	50%	70%	90%	110%	130%	150%
RT	1000	1000	1000	1000	1000	1000	1000	1000
RRT	936.85	825.28	747.47	696.60	652.04	630.31	602.80	580.03
Ideal	900.10	700.30	500.50	300.70	100.90			

Target Exclusion (R)

Fig. 5. Expected *F-measures* for Random Testing (RT) and *Ideal RRT* compared with the *F-measure*, averaged over 10,000 trials, for Restricted Random Testing (RRT). The failure region (θ) is 0.1% of the entire input domain. The target exclusion (R) varies from 10% to 150%.

to the simulation and fault-seeded programs. Although the results were (mostly) slightly less good than those achieved with the basic, circular-exclusion, version, insights from the investigation did enable the development of a hybrid approach, filtering. The filtering method incorporates some of the overhead reduction capabilities of the square exclusion method (RRT-SQ), but maintains the failure-finding efficiency of the basic RRT methods.

4.1. *Simulation*

For non-point type failure patterns, it has been suggested that the failure detection capabilities for RT can be improved by ensuring a more even spread of test cases over the input domain [7]. We investigated this by simulating a small failure region in a square input domain, and applying the RRT method, for various target exclusion rates (R) [5, 6]. The failure region was circular in shape, and its size was set at 0.001 of the entire input area ($\theta = 0.1\%$). For a failure region of this size, the expected *F-measure*, by RT, is 1,000 ($1/\theta$).

The target exclusion ratio (R) was varied between 10% and 150%, and the mean *F-measure* was calculated over a sample of 10,000 trials. The results are reproduced in Fig. 5.

As expected, the experimentally measured *F-measure* for RRT lies between the expected values for Random Testing and *Ideal RRT* (the upper and lower bounds for the method's efficiency). Additionally, the failure-finding efficiency appears to improve as the target exclusion (R) increases.

4.2. *Maximum Target Exclusion (*Max R*)*

From the simulation and analyses, it appears that the failure-finding efficiency of the *RRT* method improves as the target exclusion rate (R) increases. It has also been found that the target exclusion and actual exclusion rates vary, especially at higher exclusion ratios (section 3.2), with the higher the exclusion rates, the larger the difference. An obvious cut-off for the *RRT* algorithm is when the actual exclusion is 100%, at which point further test cases cannot be generated. Unfortunately, because determination of the actual exclusion is both difficult and time-consuming, we use the target exclusion in our calculations. Although the target exclusion ratio strongly influences the actual exclusion, the relationship is not exact; other factors, such as the shape of the input domain, the number of exclusion regions, and the location of the exclusion region centers, all affect the actual exclusion.

We define the maximum target exclusion ratio (*Max R*) as the highest target exclusion (R) at which it is possible to generate test cases for a full sample size, n. For example, when repeating a simulation 10,000 times, we continue to increment the target exclusion ratio until we can no longer obtain *F-measures* for the entire 10,000 (n) executions. The highest ratio before this is defined as the *Max R*.

Because an analytical investigation and estimation of *Max R* would be extremely difficult, if not impossible, we ran simulations to investigate empirically how it varied with the target exclusion (R) and the number of exclusion regions. In the simulations, the number of regions was varied from 100 to 4,000, the sample size (n) was set at 100, and R was incremented by 10% each time. A limit of 100,000 on the number of attempts to generate a valid (outside excluded regions) test case was imposed for each test case. The input domain shape was homogeneous (square in 2D, a cube in 3D, and a hyper- cube in 4D). The results are summarized in Fig. 6.

Theoretically, the lowest possible value for *Max R* is when the entire input domain is excluded with a single exclusion region. For this to occur however, requires that the first test case be located exactly in the centre of the input domain. In a square input domain (2D), when the first test case is in the exact centre of the input domain, if the exclusion radius is equal in length to the distance to the extreme corner (i.e., $r = \pi/2$), then complete exclusion of the input domain is possible. In the 1-dimensional case, because the exclusion shape is the same as the input domain shape (i.e., a line), it is potentially possible to achieve 100% actual exclusion when the first test case is in the centre of the input domain, and the target exclusion rate is 100%.

The results indicate that, when the number of exclusion regions is lower, the maximum target exclusion (*Max R*) is higher. (Although in one dimension, it is theoretically possible to achieve 100% actual exclusion with 100% target exclusion, it is necessary that the first test case be in the exact centre of the input domain. Because this is extremely unlikely, but because it is relatively easy to obtain an actual excluion ratio very close to 100% when the target exclusion ratio is 100%, we consider the Max R for 1D to be 100%). As the number of exclusion zones increases,

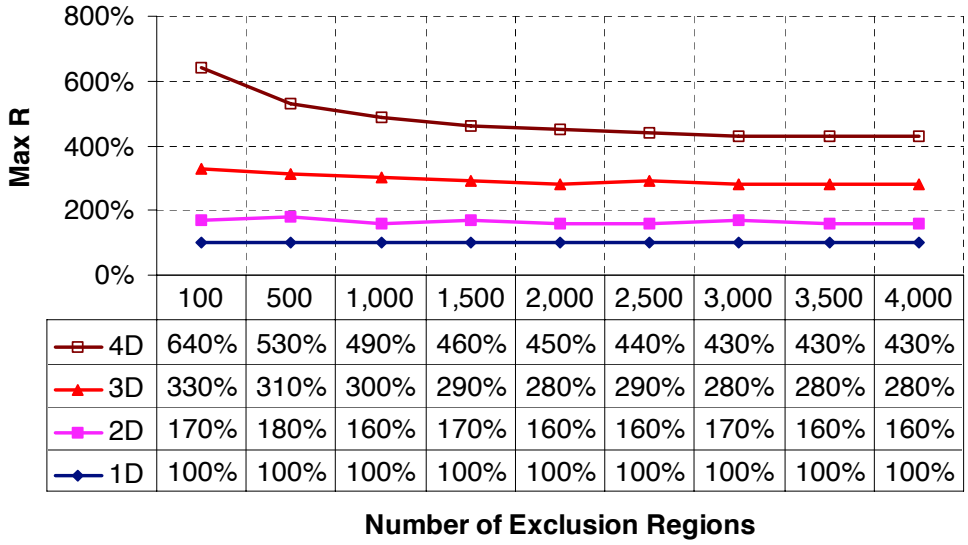


Fig. 6. Results of Maximum Target Exclusion Rate (*Max R*) calculation for a homogeneous input domain. Sample size (n) is 100, and number of exclusion regions varies from 100 to 4,000.

Table 2. Program name, dimension (D), input domain, seeded fault types, and total number of faults for each of the fault-seeded programs. The fault types are: arithmetic operator replacement (AOR); relational operator replacement (ROR); scalar variable replacement (SVR); and constant replacement (CR).

Prog Name	**D**	**Input Domain**		**Fault Type**				**Total Faults**	**Failure Rate**
		From	To	AOR	ROR	SVR	CR		
airy	1	(−5000.0)	(5000.0)				1	4	0.000716
bessj	2	(2.0, −1000.0)	(300.0, 15000.0)	2	1		1	4	0.001298
bessj0	1	(−300000.0)	(300000.0)	2	1	1	1	5	0.001373
cel	4	(0.001, 0.001, 0.001, 0.001)	(1.0, 300.0, 10000.0, 1000.0)	1	1		1	3	0.000332
el2	4	(0.0, 0.0, 0.0, 0.0)	(250.0, 250.0, 250.0, 250.0)	1	3	2	3	9	0.000690
erfcc	1	(−30000.0)	(30000.0)	1	1	1	1	4	0.000574
gammq	2	(0.0, 0.0)	(1700.0, 40.0)		3		1	4	0.000830
golden	3	(−100.0, −100.0, −100.0)	(60.0, 60.0, 60.0)		3	1	1	5	0.000550
plgndr	3	(10.0, 0.0, 0.0)	(500.0, 11.0, 1.0)	1	2		2	5	0.000368
probks	1	(−50000.0)	(50000.0)	1	1	1	1	4	0.000387
sncndn	2	(−5000.0, −5000.0)	(5000.0, 5000.0)			4	1	5	0.001623
tanh	1	(−500.0)	(500.0)	1	1	1	1	4	0.001817

the calculated *Max R* varies less for each dimension.

4.3. *Fault-seeded Programs*

The *RRT* method was applied to twelve fault-seeded programs previously studied by Mak [11]. These published programs, all involving numerical calculations, were taken from two sources [1, 14]. The programs, varying in length from 30 to 200 statements, were originally written in C, FORTRAN, and Pascal, but were converted to C++ for our experiments. Using Mutation Analysis [2], faults in the form of simple mutants were seeded into the different programs. 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 some of the most commonly occurring faults in numerical programs, such as misuse of operators (e.g. m=a*b*c was replaced by m=a+b*c); shifted condition boundaries (e.g. $x > 0$ was replaced by $x > 100$); incorrect path conditions (e.g. $x > 0$ was replaced by $x < 0$); misuse of variables (e.g. m=a*b*c was replaced by m=a*b*d); and incorrect constants (e.g. x=100 was replaced by x=10) [3].

For each program, after seeding in the faults, the range of each input variable was then set such that the overall failure rate would not be too large [3]. The failure rate was calculated using the Monte Carlo Method [15] over a sample of 10,000,000 executions. Table 2 summarizes the details of the fault-seeded programs.

4.4. FSCS *Applied to Fault-seeded Programs*

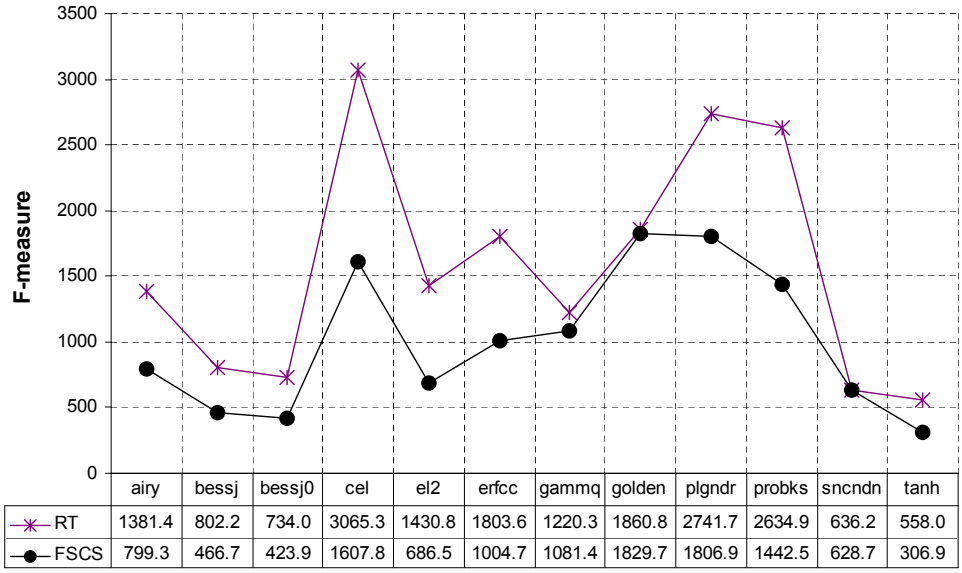
Mak [11] applied the Fixed Size Candidate Set version of *DART* (*FSCS*) to the fault-seeded programs. In this study, he varied the size of the candidate set (k), finding that, in general, the larger the value of k, the smaller the number of test cases required to detect the first failure (*F-measure*). It was also found that for values of k above 10, there was little variation in the failure-finding efficiency, as measured with the *F-measure*. In order to ensure a fair comparison with the *RRT* method, we repeated Mak's experiments in the same machine and operating environment (with the candidate size fixed at 10). The results of the rerun are given in Fig. 7, included in the results are the calculated values for Random Testing (*RT*).

As the figure shows, the Fixed Size Candidate Set version of the Adaptive Random Testing method (*FSCS*), for most programs, appears to improve upon the *RT* results. These results, expressed in terms of their improvement[a] over the *RT* *F-measure* are shown in Fig. 8.

For most of the programs, *FSCS* offers significant improvement over the calculated *F-measure* for *RT*. The *gammq* program, although not as significant as the other fault-seeded programs, does show improvement when tested using *FSCS*

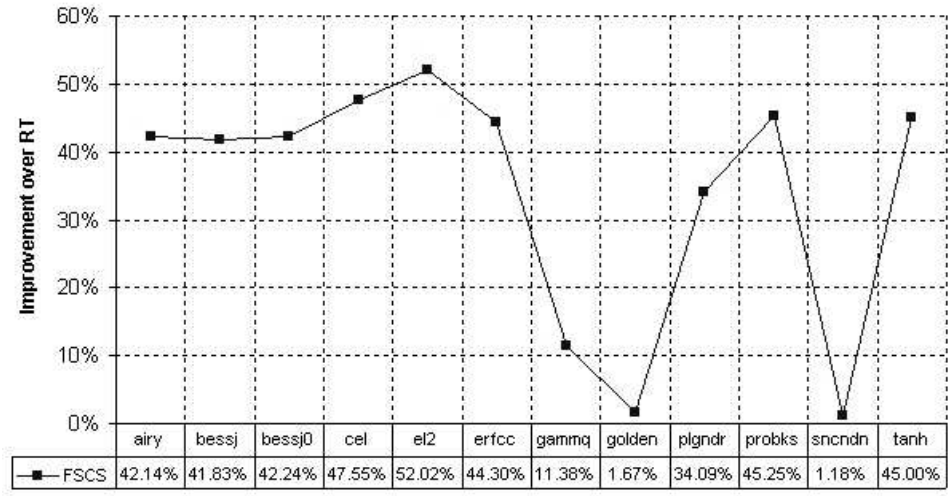
[a]Percentage improvement is calculated using the formula:

$$\frac{F_{RT} - F_{FSCS}}{F_{RT}} \times 100 \quad (1)$$



	airy	bessj	bessj0	cel	el2	erfcc	gammq	golden	plgndr	probks	sncndn	tanh
RT	1381.4	802.2	734.0	3065.3	1430.8	1803.6	1220.3	1860.8	2741.7	2634.9	636.2	558.0
FSCS	799.3	466.7	423.9	1607.8	686.5	1004.7	1081.4	1829.7	1806.9	1442.5	628.7	306.9

Fig. 7. *F-measures* for the Fixed Size Candidate Set version of Adaptive Random Testing (FSCS) and ordinary Random Testing (*RT*) when applied to the fault-seeded programs.



	airy	bessj	bessj0	cel	el2	erfcc	gammq	golden	plgndr	probks	sncndn	tanh
FSCS	42.14%	41.83%	42.24%	47.55%	52.02%	44.30%	11.38%	1.67%	34.09%	45.25%	1.18%	45.00%

Fig. 8. Improvement in *F-measures* for the Fixed Size Candidate Set version of Adaptive Random Testing (FSCS) compared to Random Testing (*RT*), when applied to the fault-seeded programs.

compared with *RT*. Only the 2 programs *golden* and *sncndn* show negligible improvement with *FSCS*, both of only 1-2%.

Table 3. Maximum Target Exclusion rates (*Max R*) for the fault-seeded programs, using Ordinary Restricted Random Testing (*ORRT*). The results are averaged over 5,000 iterations.

Program Name	**Maximum Target Exclusion *Max R***
airy	100%
bessj	220%
bessj0	100%
cel	30,000%
el2	460%
erfcc	100%
gammq	200%
golden	270%
plgndr	460%
probks	100%
sncndn	150%
tanh	100%

4.5. ORRT *Applied to Fault-seeded Programs*

Using the same fault-seeded programs that were tested with the *FSCS* method, we examined the performance of the Ordinary Restricted Random Testing (*ORRT*) method. We incrementally increased the target exclusion ratio (R) up to *Max R* (section 4.2), and obtained an *F-measure* by repeating the experiments 5,000[b] times and calculating the mean value. The value of *Max R* was found to vary for each program, sometimes, as was the case for the *cel* program, quite dramatically (Fig. 10). Further investigation revealed that the *Max R* was strongly influenced by the shape of the input domain. Details of the maximum target exclusion rates (*Max R*) for each of the fault-seeded programs are given in Table 3.

A summary of the results of the comparison between *ORRT* and Random Testing (*RT*) is given in Figs. 9 and 10. The figures show the percentage improvement of *ORRT* over *RT*.

For most of the programs, as was the case with the simulation, there appears to be an increase in the improvement over *RT* corresponding to the increase in target exclusion area. The *gammq* program, although not displaying the relationship between improvement and target exclusion increase as strongly as the other programs,

[b]The sample size was determined partly by ensuring that a statistically reliable mean value of the *F-measure* could be calculated. For the various experiments, using the central limit theorem [9], the sample size required was estimated as data was being generated. The largest value for the different data sets was obtained, and 5,000, being significantly larger than this value, was decided upon as the sample size for all the data.

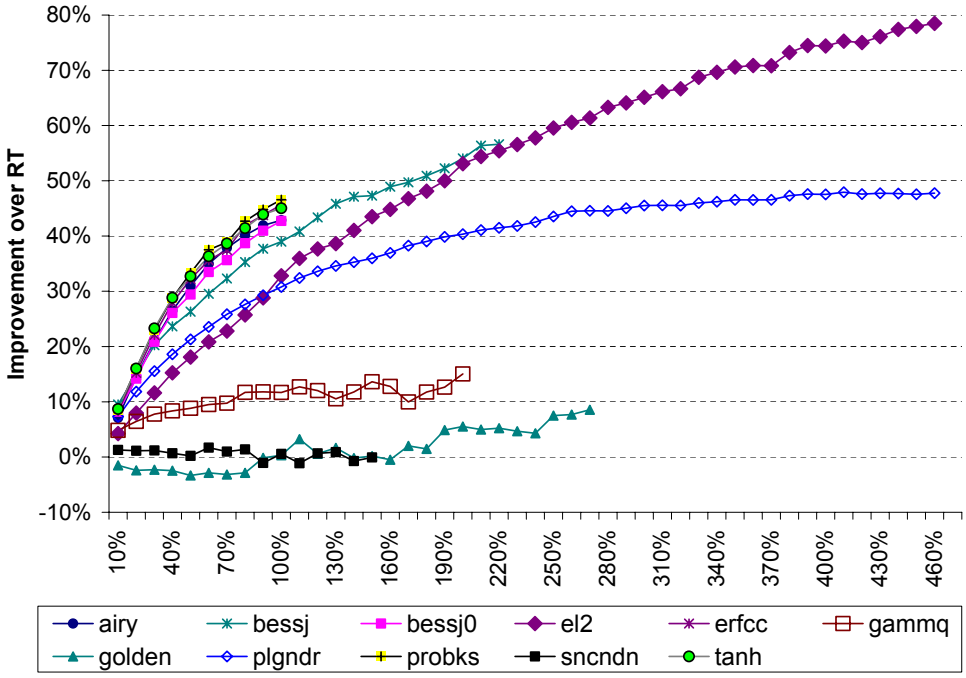


Fig. 9. Comparison of the *F-measures* for Ordinary Restricted Random Testing ($ORRT$) when applied to 11 of the fault-seeded programs. The target exclusion (R) varies from 10% to 460%. The figures show the improvement of $ORRT$ over RT.

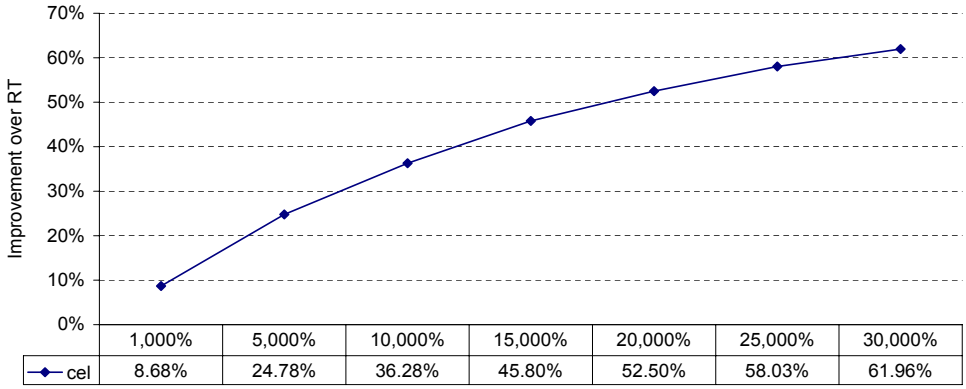


Fig. 10. Improvement in the *F-measures* for Ordinary Restricted Random Testing ($ORRT$) compared with RT, when applied to the *cel* program. The target exclusion (R) varies from 1,000% to 30,000%.

Table 4. Program name, *Max R*, Improvement over *RT* at *Max R*, *R* for Best Improvement, and Best Improvement over *RT* for the fault-seeded programs, using *ORRT*.

Program Name	**Max Target Exclusion (*Max R*)**	**Improvement over *RT* at *Max R***	***R* for Best Improvement over *RT***	**Best Improvement over *RT***
airy	100%	42.86%	100%	42.86%
bessj	220%	56.62%	220%	56.62%
bessj0	100%	42.73%	100%	42.73%
cel	30,000%	61.96%	30,000%	61.96%
el2	460%	78.50%	460%	78.50%
erfcc	100%	45.67%	100%	45.67%
gammq	200%	15.05%	200%	15.05%
golden	270%	8.57%	270%	8.57%
plgndr	460%	47.75%	**410%**	**47.93%**
probks	100%	46.57%	100%	46.57%
sncndn	150%	-0.06%	**60%**	**1.71%**
tanh	100%	45.02%	100%	45.02%

does show overall improvement over *RT*. Likewise, the *golden* program shows only slight improvement over *RT*, with only a shallow improvement slope according to the increase in target exclusion. Only *sncndn*, as was found in the *FSCS* results, does not show any significant improvement.

Although initial trials with the *cel* program appeared to generate very poor results [6], in this study, when we extended the target exclusion range to *Max R* (30,000%), it was found that the program actually did respond favorably, and did display the characteristic improvement related to the increasing target exclusion (*R*) (Fig. 10).

As expected, in almost all cases, the best improvement in failure-finding efficiency is obtained when the maximum target exclusion (*Max R*) is used. Table 4 summarizes the *Max R* and best improvement over *RT* information for *ORRT*. Only for the 2 programs *plgndr* and *sncndn* is the best improvement not at *Max R*. Nevertheless, for *plgndr*, the difference is very small, and for *sncndn*, since there was no significant overall improvement over *RT*, the relationship between *Max R* and best improvement was not expected to hold.

By varying the size and shape of the input domains for the programs and in simulations, it was discovered that these parameters affect the value of the maximum target exclusion (*Max R*). Since both the simulation and *ORRT* results indicate that the best improvements over *RT* are obtained when *Max R* is used, prior knowledge of the *Max R* would be very valuable.

The *ORRT* algorithm was adjusted to incorporate a scaling feature enabling the

exclusion zones to be normalized to the shape of the input domain. This version of *RRT*, Normalized Restricted Random Testing (*NRRT*), normalized the input domain, and made it possible to approximate the *Max R*, depending on the dimensionality of the program being tested. In the next section, results from applying the *NRRT* method to the fault-seeded programs are presented.

4.6. NRRT *Applied to Fault-seeded Programs*

We adapted the *RRT* method so that, instead of a uniform exclusion zone (circle, sphere, etc.) around each non-failure-causing test case, the exclusion zone was scaled to the shape of the input domain. From experimental data (section 4.2) we were able to approximate the *Max R* for various homogeneous input domains; by defining the exclusion zone initially to be within a unit square/cube/hypercube, and then mapping the points to the actual input domain, we were better able to use this *Max R* information to predict the optimum target exclusion ratio (R) in advance. We refer to this version of *RRT* as Normalized Restricted Random Testing (*NRRT*).

We applied the *NRRT* method to the fault-seeded programs, incrementally varying the target exclusion ratio (R), and obtaining the *F-measure* by repeating the experiments 5,000 times and calculating the mean value. Again, we expected the highest failure-finding efficiency to be when the maximum target exclusion (*Max R*), was used; now though, because the initial input domain was normalized, programs with the same dimension were expected to have similar values for *Max R*. For those programs whose imput domains were already normalized (*el2*, *golden*, *sncndn*, and the 1D programs *airy*, *bessj0*, *erfcc*, *probks*, and *tanh*), it was expected that *NRRT* would perform identically to *ORRT* [c].

A summary of the results of the comparison between *NRRT* and Random Testing is shown in Fig. 11. The figure shows the percentage improvement of *NRRT* over *RT*.

For most of the programs, as before, the improvement over Random Testing increases according to the increase in target exclusion ratio (R). Unlike for *ORRT* though, with *NRRT*, *gammq* does show a strong correlation between R and improvement over *RT*. For *bessj*, however, the correlation, was significantly less pronounced with *NRRT* than with *ORRT*. Both *plgndr* and *cel* perform better with *NRRT* than with *ORRT*, with steeper improvement curves (faster rates of improvement), and overall better improvement over *RT*.

As expected, because the input domains for *el2*, *golden*, *sncndn* and the 1D programs (*airy*, *bessj0*, *erfcc*, *probks*, and *tanh*) were already normalized, the results for *NRRT* were identical to those obtained with *ORRT*. Also as expected, for programs with the same dimensionality, the maximum target exclusion ratios (*Max R*) are similar, and the best results are usually obtained when *Max R* is used.

[c]To illustrate this, unlike in previous studies [4, 5, 6], the experimental data presented in this paper for the *RRT* methods was generated by applying the same potential test case sequences to the different algorithms.

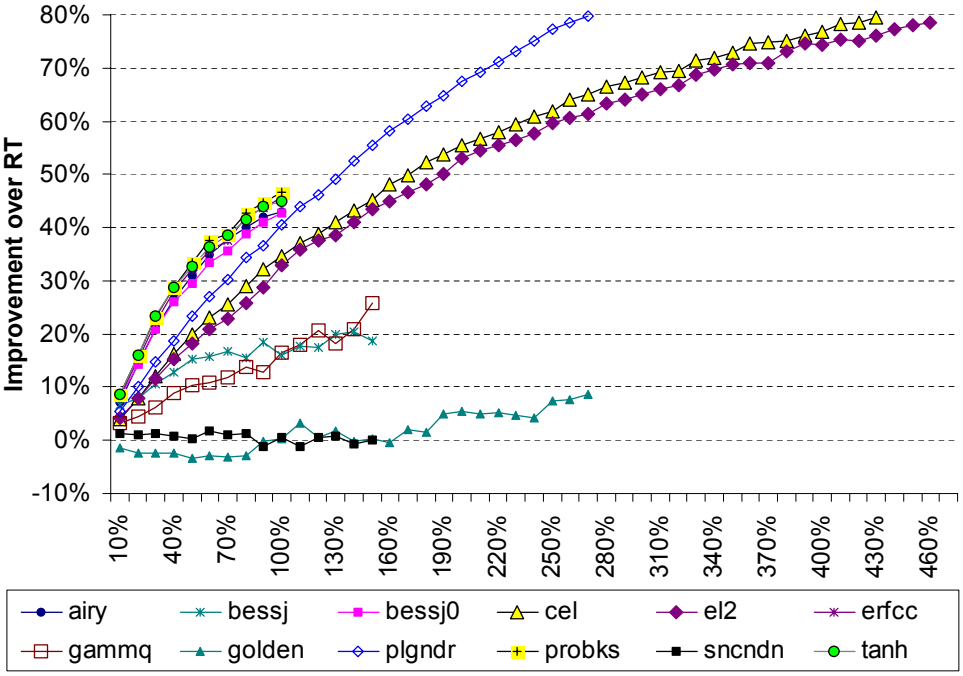


Fig. 11. Comparison of the *F-measures* for Normalized Restricted Random Testing (*NRRT*) when applied to the fault-seeded programs. The target exclusion (R) varies from 10% to 460%. The figure shows the improvement of *NRRT* over *RT*.

These results are summarized in Table 5. Again, two of the programs do not have the best improvement when the *Max R* value was used: for *bessj*, the difference between the best improvement and the improvement for *Max R* is small; and for *sncndn*, as explained for *ORRT*, since there was no overall improvement over *RT*, the relationship between *Max R* and best improvement was not expected to hold.

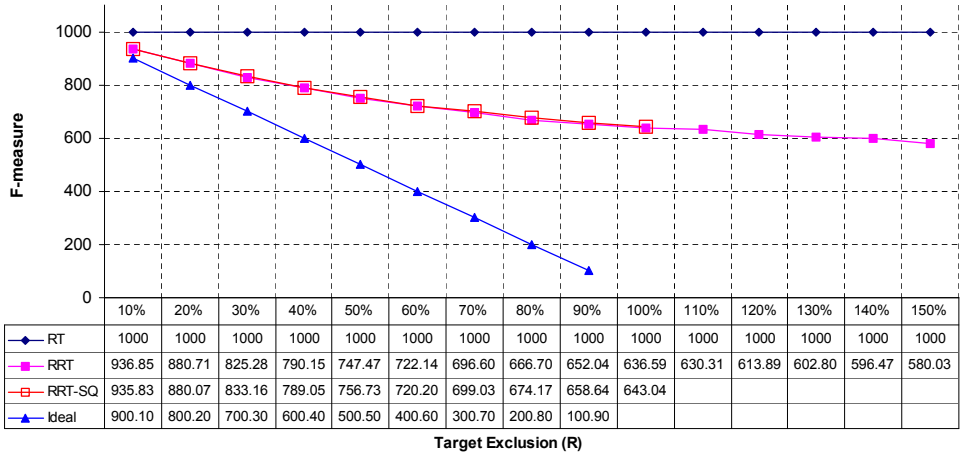
4.7. *RRT with Square Exclusion Shapes*

Because the distance calculations associated with circular exclusion zones can become computationally quite expensive, and because of the known complexity of the relationship between the Target and Actual Exclusion ratios (due, in part, to the circular shape of the exclusion region), we were interested in investigating *RRT* with alternative exclusion shapes [4].

We applied a square exclusion region version of *RRT* (*RRT-SQ*) to the same simulation previously conducted with the circular exclusion version, the results of which are shown in Fig. 12. An advantage of the square exclusion shape is the cheaper inequality operations used to verify if a test case lies within the exclusion region (compared with the distance calculations necessary for a circular exclusion

Table 5. Program name, Dimension (D), *Max R*, Improvement over *RT* at *Max R*, *R* for Best Improvement, and Best Improvement over *RT* for the fault-seeded programs, using *NRRT*.

Program Name	**D**	**Max Target Exclusion (*Max R*)**	**Improvement over *RT* at *Max R***	***R* for Best Improvement over *RT***	**Best Improvement over *RT***
airy	1	100%	42.86%	100%	42.86%
bessj	2	150%	18.70%	**140%**	**20.42%**
bessj0	1	100%	42.73%	100%	42.73%
cel	4	430%	79.51%	430%	79.51%
el2	4	460%	78.50%	460%	78.50%
erfcc	1	100%	45.67%	100%	45.67%
gammq	2	150%	25.85%	150%	25.85%
golden	3	270%	8.57%	270%	8.57%
plgndr	3	270%	79.77%	270%	79.77%
probks	1	100%	46.57%	100%	46.57%
sncndn	2	150%	-0.06%	**60%**	**1.71%**
tanh	1	100%	45.02%	100%	45.02%



	10%	20%	30%	40%	50%	60%	70%	80%	90%	100%	110%	120%	130%	140%	150%
RT	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000
RRT	936.85	880.71	825.28	790.15	747.47	722.14	696.60	666.70	652.04	636.59	630.31	613.89	602.80	596.47	580.03
RRT-SQ	935.83	880.07	833.16	789.05	756.73	720.20	699.03	674.17	658.64	643.04					
Ideal	900.10	800.20	700.30	600.40	500.50	400.60	300.70	200.80	100.90						

Fig. 12. Expected *F-measures* for Random Testing (*RT*) and *Ideal RRT* compared with the *F-measure*, averaged over 10,000 trials, for the original, circular exclusion version of Restricted Random Testing (*RRT*), and the square exclusion version (*RRT-SQ*). The failure region (θ) is 0.1% of the entire input domain. The target exclusion (R) varies from 10% to 150%.

shape). Although the square exclusion results show a similar curve to that for the circular exclusion method (with decreasing *F-measures* for increasing R), the *Max R* was reached far earlier (at 100%), and therefore the best failure-finding result (at *Max R*, *F-measure* of about 640) was less good than that for the circular exclusion method (at *Max R* of 150%, *F-measure* of about 580).

We also applied the *ORRT* and *NRRT* methods with square exclusion regions

Table 6. Program name, Dimension (D), *Max R*, Improvement over *RT* at *Max R*, *R* for Best Improvement, and Best Improvement over *RT* for the fault-seeded programs, using *ORRT-SQ*.

Program Name	**D**	**Max Target Exclusion (*Max R*)**	**Improvement over *RT* at *Max R***	***R* for Best Improvement over *RT***	**Best Improvement over *RT***
airy	1	100%	42.86%	100%	42.86%
bessj	2	220%	53.67%	**210%**	**54.06%**
bessj0	1	100%	42.73%	100%	42.73%
cel	4	51,400%	62.69%	**51,300%**	**62.77%**
el2	4	130%	35.40%	130%	35.40%
erfcc	1	100%	45.67%	100%	45.67%
gammq	2	210%	19.25%	210%	19.25%
golden	3	100%	1.24%	100%	1.24%
plgndr	3	430%	50.21%	430%	50.21%
probks	1	100%	46.57%	100%	46.57%
sncndn	2	100%	-0.24%	**30%**	**1.32%**
tanh	1	100%	45.02%	100%	45.02%

(*ORRT-SQ* and *NRRT-SQ*) to the 12 fault-seeded programs, again with comparable, though slightly poorer, results compared with the original, circular exclusion versions.

A summary of the results of the comparison between *ORRT-SQ* and Random Testing (*RT*) is given in Figs. 13 and 14. Fig. 15 shows the comparison between *NRRT-SQ* and *RT*. The figures show the percentage improvement of the square exclusion shape *RRT* versions over *RT*. Tables 6 and 7 summarize the *Max R* and best improvement over *RT* information for *ORRT-SQ* and *NRRT-SQ*, respectively.

Present in the figures for both *ORRT-SQ* and *NRRT-SQ* is the characteristic curve showing an increase in improvement over *RT* as the target exclusion ratio (*R*) is increased, for most of the fault-seeded programs. As with the circular exclusion version, for those programs whose input domain was already relatively normal (*el2*, *golden*, *sncndn* and the 1D programs *airy*, *bessj0*, *erfcc*, *probks*, and *tanh*), the performance for the Ordinary (*ORRT-SQ*) and Normalized (*NRRT-SQ*) versions was identical. Additionally, with the exception of *el2*, the *Max R* was found to be 100% for these programs. For *NRRT-SQ*, all the programs whose input domains were in less than 4D had *Max R* of 100%, and those in 4D had a *Max R* of 130%. The *sncndn*, as with other testing methods, showed no overall improvement over *RT*. The other programs did show improvement, with the best improvement usually being when *Max R* was used.

A comparison of the best *RRT* and *FSCS* results is given in Fig. 16. The figure shows the best improvement over *RT* for Ordinary Restricted Random Testing (*ORRT*), Normalized Restricted Random Testing (*NRRT*), the square exclusion versions of *ORRT* and *NRRT* (*ORRT-SQ* and *NRRT-SQ*), and the Fixed Size Candidate Set version of *ART* (*FSCS*).

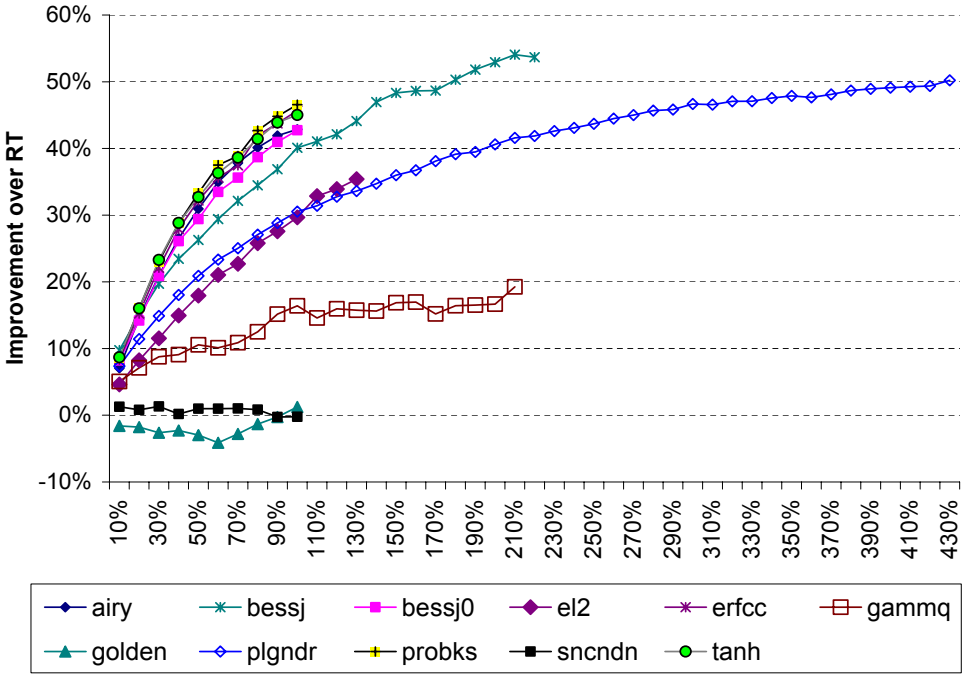


Fig. 13. Comparison of the *F-measures* for the square exclusion shape version of Ordinary Restricted Random Testing (*ORRT-SQ*) when applied to 11 of the fault-seeded programs. The target exclusion (R) varies from 10% to 430%. The figures show the improvement of *ORRT-SQ* over *RT*.

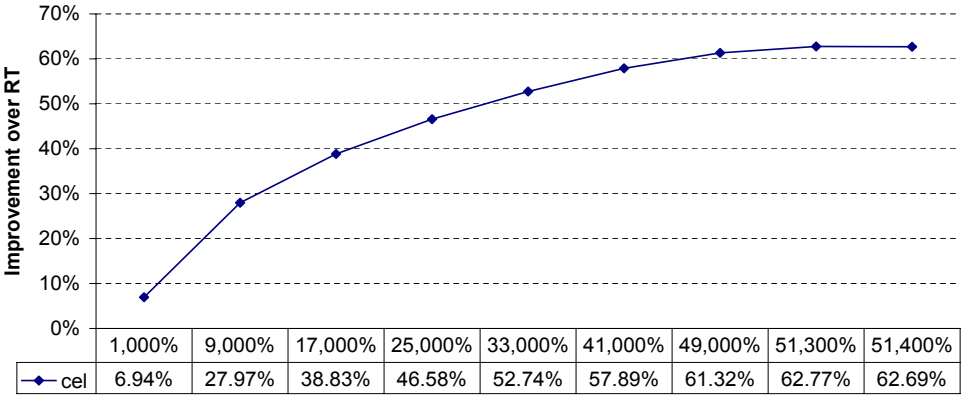


Fig. 14. Improvement in the *F-measures* for the square exclusion shape version of Ordinary Restricted Random Testing (*ORRT-SQ*) compared with *RT*, when applied to the *cel* program. The target exclusion (R) varies from 1,000% to 51,400%.

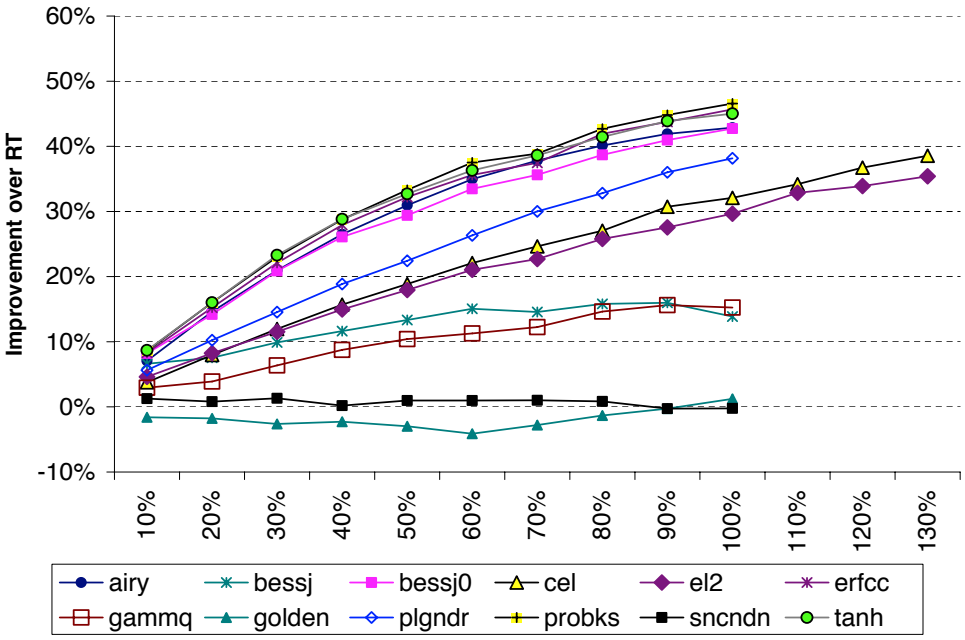
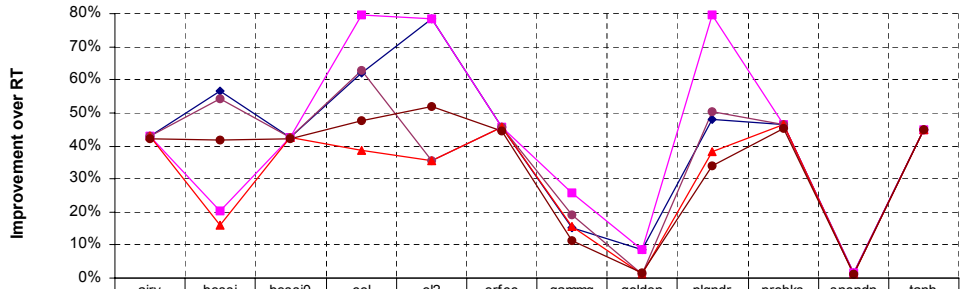


Fig. 15. Comparison of the *F-measures* for the square exclusion shape version of Normalized Restricted Random Testing (*NRRT-SQ*) when applied to the fault-seeded programs. The target exclusion (R) varies from 10% to 130%. The figures show the improvement of *NRRT-SQ* over *RT*.

Table 7. Program name, Dimension (D), *Max R*, Improvement over *RT* at *Max R*, *R* for Best Improvement, and Best Improvement over *RT* for the fault-seeded programs, using *NRRT-SQ*.

Program Name	**D**	**Max Target Exclusion (*Max R*)**	**Improvement over *RT* at *Max R***	***R* for Best Improvement over *RT***	**Best Improvement over *RT***
airy	1	100%	42.86%	100%	42.86%
bessj	2	100%	13.87%	**90%**	**15.99%**
bessj0	1	100%	42.73%	100%	42.73%
cel	4	130%	38.55%	130%	38.55%
el2	4	130%	35.40%	130%	35.40%
erfcc	1	100%	45.67%	100%	45.67%
gammq	2	100%	15.24%	**90%**	**15.63%**
golden	3	100%	1.24%	100%	1.24%
plgndr	3	100%	38.17%	100%	38.17%
probks	1	100%	46.57%	100%	46.57%
sncndn	2	100%	-0.24%	**30%**	**1.32%**
tanh	1	100%	45.02%	100%	45.02%



	airy	bessj	bessj0	cel	el2	erfcc	gammq	golden	plgndr	probks	sncndn	tanh
ORRT	42.86%	56.62%	42.73%	61.96%	78.50%	45.67%	15.05%	8.57%	47.93%	46.57%	1.71%	45.02%
NRRT	42.86%	20.42%	42.73%	79.51%	78.50%	45.67%	25.85%	8.57%	79.77%	46.57%	1.71%	45.02%
ORRT-SQ	42.86%	54.06%	42.73%	62.77%	35.40%	45.67%	19.25%	1.24%	50.21%	46.57%	1.32%	45.02%
NRRT-SQ	42.86%	15.99%	42.73%	38.55%	35.40%	45.67%	15.63%	1.24%	38.17%	46.57%	1.32%	45.02%
FSCS	42.14%	41.83%	42.24%	47.55%	52.02%	44.30%	11.38%	1.67%	34.09%	45.25%	1.18%	45.00%

Fig. 16. Comparison of best results for Ordinary Restricted Random Testing (*ORRT*), Normalized Restricted Random Testing (*NRRT*), the square exclusion versions of *ORRT* (*ORRT-SQ*) and *NRRT* (*NRRT-SQ*), and the Fixed Size Candidate Set version of Adaptive Random Testing (FSCS), when applied to the seeded programs. The figures show the improvement over the calculated *RT*.

From the figure, it is can be seen that the *RRT* methods compare favorably with the *FSCS* method, in many cases significantly outperforming it. As expected, the square exclusion versions of *RRT* (*ORRT-SQ* and *NRRT-SQ*) have a slightly poorer performance compared with the circular exclusion versions (*ORRT* and *NRRT*). For almost all of the fault-seeded programs, the *NRRT* results are very similar to, or better than, the *ORRT* results; only for the *bessj* program does the *NRRT* method perform less well than the other methods. For the square exclusion versions, however, it is *ORRT-SQ* which appears to consistently outperform *NRRT-SQ*. Both the *golden* and *sncndn* programs showed little improvement over *RT* with the *FSCS* method; this remained the case with *RRT* for the *sncndn* program, but not for *golden*, which improved by about 9%. For the 1D fault-seeded programs (*airy*, *bessj0*, *erfcc*, *probks*, and *tanh*), all five methods achieved similar improvement over *RT*. Because the implementation of *RRT* in 1D is identical, regardless of whether *ORRT*, *NRRT*, *ORRT-SQ* or *NRRT-SQ*, the results for all four variatiosn are identical. For the three programs whose input domains were already normalized, but greater than 1D (i.e., *el2*, *golden*, and *sncndn*), the performances were the same for the Ordinary and Normalized versions, in other words, *ORRT* and *NRRT* achieved identical results, and *ORRT-SQ* and *NRRT-SQ* achieved identical results.

4.8. *Overheads*

The *RRT* methods incur potentially significant overheads in the generation of the $(m+1)^{th}$ test case. At this instant, when generating the $(m+1)^{th}$ test case, 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 each executed test case than the radius of the exclusion region. For two points, P and Q ($(p_1, p_2, \ldots, p_N)$ and $(q_1, q_2, \ldots, q_N)$), the Euclidean distance between the points can be calculated from the following expression:

$$Distance(P, Q) = \sqrt{\sum_{i=1}^{N} (p_i - q_i)^2} \tag{2}$$

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 that the $(m + 1)^{th}$ test case is acceptable. In practice, 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 be within the final exclusion region checked. Normally, a constraint on the maximum number of attempts to generate a single acceptable test case is imposed.

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/hypercube 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 relative size of the bounding region (in the following equations, m is the number of executed test cases, which is also the number of exclusion regions).

$$\begin{matrix} \textit{Expected Total} \\ \textit{Number of Test Cases} \\ \textit{Within Bounding Region} \end{matrix} = m \times \frac{Size_{Bounding\ Region}}{Size_{Input\ Domain}} \tag{3}$$

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

$$Size_{Bounding\ Region} = [2r]^N \tag{4}$$

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).

Table 8. 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 exectued test cases; R is the Exclusion Ratio; and r is the radius of the exclusion regions.

N	**Area/Volume/ Hyper Volume (for circle, sphere, etc)**	**Expected number of test cases inside Bounding Region**
2	$\pi r^2 = \frac{A \times R}{m}$	$\frac{4R}{\pi}$
3	$\frac{4}{3}\pi r^3 = \frac{A \times R}{m}$	$\frac{6R}{\pi}$
4	$\frac{1}{2}\pi^2 r^4 = \frac{A \times R}{m}$	$\frac{32R}{\pi^2}$

$$Size_{Exclusion\ Region} = \frac{R \times Size_{Input\ Domain}}{m} \tag{5}$$

The formula to calculate the radius changes according to the dimensions of the input domain. Table 8 summarizes the expected number of test cases to fall inside a bounding region for 2, 3 and 4 dimensions.

Experiments have verified the filtering method's speed compared with the ordinary, circular exclusion implementation of RRT, while maintaining identical failure-finding results. Indeed, as Table 8 shows, the (maximum) number of test cases on which the more expensive circular exclusion method will be applied is a small constant, determined by the size of the exclusion regions: for example, with target exclusion rate (R) of 150%, in 2D, it is expected to be applied to less than 2, all other test cases being filtered.

Obviously, when larger proportions of the input domain are excluded, it is more difficult to randomly generate a test case lying in a non-excluded area. In this case, the number of attempts to generate an acceptable test case increases. Approximately, if 99% of the input domain is excluded, leaving only 1% from which the test case may be drawn, it should take an average of 100 (1/0.01) attempts to generate an acceptable test case.

4.9. *Discussion*

Simulation and experimental results indicate that the proposed Restricted Random Testing (RRT) method does indeed improve over the basic Random Testing (RT) method. In the simulation, the method showed improvement over the expected *F-measure* for RT of up to 40%. The calculated RRT *F-measure* lay between the expected *F-measure* for RT and for the *Ideal* RRT (the simplified mathematical

model for *RRT* under ideal conditions), and appeared to improve as the target exclusion rate (R) was increased.

Investigation into the maximum target exclusion rate (*Max R*) suggested that this would be the optimal target exclusion rate, and would produce the best failure-finding results, measured with the *F-measure*. The value of *Max R* depends on the number of dimensions and the shape of the input domain. Empirical studies gave approximate values for *Max R* for different numbers of exclusion regions/points, for 1 to 4 dimensional, homogeneous input domains.

When the original method (*ORRT*) was applied to twelve fault-seeded programs, the results were very encouraging. For most programs, the characteristic slope representing improvement as target exclusion (R) increases was present, and the best failure finding results were mostly when *Max R* (or close to *Max R*) was used. Of all the programs, two (*gammq* and *golden*) showed a less pronounced characteristic slope, and lower overall improvement over *RT* (about 15% and 9% respectively). Only one program, *sncndn*, showed no significant difference when tested with *ORRT*.

When using *ORRT*, it was found that *Max R* varied strongly according to the size and shape of the input domain. A normalized version of *RRT* (*NRRT*) was devised and applied to the fault-seeded programs. Because *NRRT* made use of an initial homogeneous input domain, better estimation of the *Max R* was possible: the results showed the *Max R* for each program to be far more stable than for *ORRT*, and to be similar to the values generated in the investigation of *Max R*. As expected, for those programs whose input domains were already normalized, *el2*, *golden*, *sncndn* and the 1D programs (*airy*, *bessj0*, *erfcc*, *probks*, and *tanh*), there was no difference in performance using *NRRT* compared with *ORRT*.

The characteristic slope representing improvement in failure-finding efficiency as R increases was again present in the *NRRT* results, and the best results were mostly when *Max R* (or close to) was used. The *gammq* program performed better with *NRRT* than with *ORRT*, displaying the characteristic slope more markedly, and having a higher overall improvement over *RT* (25%). The *bessj* program performed less well when tested with *NRRT* than *ORRT*. The characteristic slope was less pronounced, and the best improvement was significantly lower than for *ORRT* (20% for *NRRT*; 56% for *ORRT*). The reason for the better performance of *ORRT* over *NRRT* with the *bessj* program was traced to the failure pattern of the program, being a narrow block, or incomplete strip type pattern.

Results for the square exclusion shape version of *RRT* (*RRT-SQ*) again showed the characteristic curve with an increase in improvement over *RT* as the target exclusion ratio (R) increased, for most of the fault-seeded programs. The best results, as with the circular exclusion version, were usually when the *Max R* was used. Overall, the results for *RRT-SQ* were comparable, but slightly less good than those for the circular *RRT* methods.

The results for both *RRT* methods were compared with those from Adaptive

Random Testing (ART), which had also been applied to the fault-seeded programs. The results showed RRT to compare very favorably with $FSCS$, with $NRRT$ outperforming $FSCS$ for the three programs *cel*, *el2* and *plgndr*, and $ORRT$ outperforming for these three and *bessj*.

The five 1-dimensional input domain programs (*airy*, *bessj0*, *erfcc*, *probks* and *tanh*) all showed similar results for the different testing methods, and all showed significant improvement (40% to 45%) over the RT results.

When $FSCS$ was applied to the programs, 2 showed no significant improvement over the RT failure-finding efficiency, *golden* and *sncndn*. When RRT was applied to *golden*, there was some improvement (about 9%), but the characteristic curve was less pronounced than for most other programs. When applied to the *sncndn* program though, the RRT methods also failed to improve over RT. Investigation into the failure patterns of *sncndn* revealed its pattern to be of point type, making *sncndn* a program which, according to our hypothesis, was not expected to respond well to RRT.

Overall, the experimental results show that, for non-point type failure patterns, the RRT methods do improve over Random Testing methods, and that the improvement tends to increase as the target exclusion ratio is increased, with best failure finding efficiency usually when the maximum target exclusion rate ($Max\ R$) is used.

The main overhead in the Restricted Random Testing (RRT) method, common to all versions, is in the generation of a valid test case. When preparing a test case, the algorithm first generates a random point within the input domain, and then checks if this point lies within any exclusion zone. If the generated point is within an exclusion zone, it is discarded and another random point is generated. This repeated generation and checking of test cases continues until a valid point (one lying outside all exclusion zones) is found, or until a predefined, maximum number of attempts to find one have been made. The number of attempts required to generate an acceptable test case generally increases as the target exclusion (R) increases.

One way of reducing the overheads of the circular exclusion region versions of the RRT methods is to reduce the number of distance calculations necessary to verify whether or not a point is excluded. One simple, but very effective way of doing this is to filter the executed test cases through a Bounding Square/Cube/Hypercube (requiring only the cheaper inequality operations), and calculate the distance only for those executed test cases inside the Bounding Region. The number of test cases inside the Bounding Region is significantly less than the total number in the entire input domain.

5. Summary and Conclusion

In this paper, we have presented the Restricted Random Testing (RRT) method, a method motivated by the proposition that Random Testing (RT) failure detection

capabilities, for some failure patters types, could be improved by ensuring a more evenly spread distribution of test cases over the input domain. The *RRT* method makes use of exclusion zones around previously executed, but non-failure-revealing test cases, restricting subsequent cases to be drawn from outside these exclusion zones.

Two versions of *RRT* were developed, the Original (*ORRT*), and a Normalized version (*NRRT*). Both methods share the same fundamental algorithm, but differ in their management of non-homogeneous input domains. Specifically, *NRRT* makes use of an intermediate, normalized domain, and scales the initial points and exclusion zones from this domain to the actual input domain. Although usually implemented with a circular exclusion region, an alternative implementation using square regions was also developed and investigated.

In addition to simulation studies, *ORRT* and *NRRT* (both for circular and square exclusion) were applied to twelve fault-seeded programs, with the resulting *F-measure* data compared to *RT* and *FSCS*, an alternative testing strategy. Results showed a relationship between the increase in failure-finding efficiency, measured with the *F-measure*, as the target exclusion rate (R) for *RRT* was increased. In most cases, the best results were found when the maximum target exclusion rate (*Max R*) was used.

Advance knowledge of the *Max R* is difficult to obtain, particularly for non-homogeneous input domains, but is extremely useful to the tester using the *RRT* strategy. From simulation data in several dimensions, we were able to approximate what *Max R* would be for different dimensions and numbers of exclusion zones. The *NRRT* results further strengthened the predictions, with *Max R* for the different fault-seeded programs being similar or identical according to the dimensionality of the program's input domain.

An examination of the results for *RRT* compared with *FSCS* and *RT* revealed that Restricted Random Testing does improve significantly over *RT* for non-point type failure patterns, and that it compares very favorably with *FSCS*.

The overheads for the *RRT* method are associated with the verification of a potential test case's position lying outside all exclusion regions. Investigation into the use of exclusion shapes alternative to the original circle/sphere/hypersphere resulted in a filtering mechanism which significantly reduces the number of distance calculations necessary. In general though, the execution costs for the method increase as the target exclusion rate (R) is increased. Because of this, when the cost of execution of the program is lower than the cost of generating a test case, lower target exclusion rates (and hence lower test case generation times) may be used. It is in those situations where the execution costs of the program are far greater than the test case generation overheads that *RRT* offers the best advantage.

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

1. Association for Computer Machinery, *Collected Algorithms from ACM, Vol. I, II, III*, Association for Computer Machinery, 1980.
2. T. A. Budd, "Mutation Analysis: Ideas, Examples, Problems and Prospects", *Computer Program Testing*, B. Chandrasekaran and S. Radicci (eds.), North-Holland, Amsterdam, pp. 129-148, 1981.
3. 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. 28, No. 12, pp. 775-782, December 1996.
4. K. P. Chan, T. Y. Chen, and D. Towey, "Adaptive Random Testing with Filtering: An Overhead Reduction Technique", Proceedings *17th International Conference on Software Engineering and Knowledge Engineering (SEKE'05)*, Taipei, Taiwan, Republic of China, July 14-16, 2005.
5. 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, pp. 368-381, Springer-Verlag, Berlin Heidelberg 2003.
6. K. P. Chan, T. Y. Chen, and D. Towey, "Restricted Random Testing", *Software Quality - ECSQ 2002, 7th European Conference*, Helsinki, Finland, June 9-13, 2002, Proceedings. LNCS 2349, pp. 321-330, Springer-Verlag, Berlin Heidelberg 2002.
7. 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.
8. T. Y. Chen and Y. T. Yu, "On the Relationship Between Partition and Random Testing", *IEEE Transactions on Software Engineering*, Vol. 20, No. 12, pp. 977-980, December 1994.
9. B. S. Everitt, *The Cambridge Dictionary of Statistics*, Cambridge University Press, 1998.
10. P. S. Loo and W. K. Tsai, "Random Testing Revisited", *Information and Software Technology*, Vol. 30, No. 9, pp. 402-417, September 1988.
11. I. K. Mak, "On the Effectiveness of Random Testing", Masters Thesis. Department of Computer Science, The University of Melbourne, Australia, 1997.
12. B. P. Miller, L. Fredriksen, and B. So. "An Empirical Study of the Reliability of UNIX Utilities". *Communications of the ACM*, Vol. 33, No. 12, pp. 32-44, December 1990.
13. B. P. Miller, D. Koski, C. Lee, V. Maganty, R. Murthy, A. Natarajan, and J. Steidl. "Fuzz Revisited: A Re-examination of the Reliability of UNIX Utilities and Services". Technical Report CS-TR-1995-1268, University of Wisconsin, 1995.
14. W. H. Press, B. P. Flannery, S. A. Teulolsky, and W. T. Vetterling, *Numerical Recipes*, Cambridge University Press, 1986.
15. C. P. Robert and G. Casella, *Monte Carlo Statistical Methods*, Springer, New York, 1999.
16. D. Slutz, "Massive Stochastic Testing of SQL". *24th International Conference on Very Large Databases (VLDB 98)*, August 24-27, 1998, New York City, New York,

USA, Proceedings, pp. 618-622, 1998.
17. T. Yoshikawa, K. Shimura, and T. Ozawa, "Random Program Generator for Java JIT Compiler Test System". *3rd International Conference on Quality Software (QSIC 2003)*, pp. 20-24. IEEE Computer Society Press, 2003.

Kwok Ping Chan received his B.Sc.(Eng.) and Ph.D. degrees from the University of Hong Kong. He is currently an Associate Professor in Computer Science and Information Systems at the same university. His main research interest is in pattern recognition, image processing, fuzzy set theory, and in particular, their applications to software testing.

Tsong Yueh Chen received the B.Sc. and M.Phil. degrees from the University of Hong Kong, M.Sc. degree and DIC from the Imperial College of Science and Technology, and the Ph.D. degree from the University of Melbourne. He is currently a Professor of Software Engineering and the Director of Centre for Software Engineering, Swinburne University of Technology, Australia. Prior to joining the Swinburne University of Technology, he taught at the University of Hong Kong and the University of Melbourne. His research interests include software testing, debugging, software maintenance and software design.

Dave Towey received his B.A. degree (upgraded in 2000 to M.A.) from the University of Dublin, Trinity College. After several years working in Japan in the field of Breast Cancer detection, he joined the University of Hong Kong, where he is currently a Ph.D. candidate. His current research interests include fuzzy set theory and software engineering, particularly software testing.

Appendix - Derivation of Expected F-measure for Ideal RRT

Notation

d	Entire input domain size
m	Size of failure-causing region
R	Total (target) Exclusion Rate as a fraction of entire input domain (d)
m_i	Size of available (non-excluded) failure-causing region when i test cases (and hence i exclusion regions) are present
e_i	Total exclusion region size when i test cases (and hence i exclusion regions) are present
F_q	*F-measure* $= q$
$E(F)$	Expected *F-measure*

Let $P(F_q) =$ (Probability of *F-measure* $= q$).

$$P(F_1) = \frac{m}{d}$$

$$P(F_2) = \left(1 - \frac{m}{d}\right)\left(\frac{m_1}{d - e_1}\right)$$

$$P(F_3) = \left(1 - \frac{m}{d}\right)\left(1 - \frac{m_1}{d - e_1}\right)\left(\frac{m_2}{d - e_2}\right)$$

$$P(F_k) = \begin{cases} \left(1 - \frac{m}{d}\right)\left[\prod_{j=1}^{k-2}\left(1 - \frac{m_j}{d - e_j}\right)\right]\left(\frac{m_{k-1}}{d - e_{k-1}}\right), & \text{when } k \geq 2 \\ \frac{m}{d}, & \text{when } k = 1 \end{cases}$$

For *Ideal RRT*, since there is no overlapping, falling of exclusion region outside of the input domain, or eclipsing, $e_k = Rd$, and $m_k = m$, $\forall k \geq 1$

Therefore,

$$P(F_k) = \begin{cases} \left(1 - \frac{m}{d}\right)\left(1 - \frac{m}{d - Rd}\right)^{k-2}\left(\frac{m}{d - Rd}\right), & \text{when } k \geq 2 \\ \frac{m}{d}, & \text{when } k = 1 \end{cases}$$

$$\begin{aligned} E(F) &= \sum^{i} P(F_q).F_q \\ &= P(F_1).F_1 + \sum_{k=2}^{\infty} P(F_k).F_k \\ &= \frac{m}{d}.1 + \sum_{k=2}^{\infty}\left[\left(1 - \frac{m}{d}\right)\left(1 - \frac{m}{d - Rd}\right)^{k-2}\left(\frac{m}{d - Rd}\right)\right].k \\ &= \frac{m}{d} + (1 - \frac{m}{d})\left(\frac{m}{d - Rd}\right)\sum_{k=2}^{\infty}\left[\left(1 - \frac{m}{d - Rd}\right)^{k-2}\right].k \end{aligned} \quad (2)$$

$$\text{Let } \beta = \left(1 - \frac{m}{d - Rd}\right) \tag{3}$$

$$\text{Therefore, } \sum_{k=2}^{\infty} \left[\left(1 - \frac{m}{d - Rd}\right)^{k-2}\right].k = \sum_{k=2}^{\infty} \beta^{k-2}.k$$

$$= 2.1 + 3.\beta + 4.\beta^2 + 5.\beta^3 + 6.\beta^4 + ...$$
$$= (2 + \beta + \beta^2 + \beta^3 + \beta^4 + ...)(1 + \beta + \beta^2 + \beta^3 + \beta^4 + ...)$$
$$= \left(1 + \frac{1}{1-\beta}\right)\left(\frac{1}{1-\beta}\right) \quad \text{because } \beta < 1$$
$$= \frac{2 - \beta}{(1-\beta)^2} \tag{4}$$

Substituting (3) and (4) into (2), we have

$$E(F) = \frac{m}{d} + \left(1 - \frac{m}{d}\right)\left(\frac{1 + \frac{m}{d-Rd}}{\frac{m}{d-Rd}}\right)$$
$$= \frac{m}{d} + \left(\frac{d-m}{d}\right)\left(\frac{d - dR + m}{m}\right)$$
$$= \frac{m}{d} + \left(\frac{d^2 - d^2R + dRm - m^2}{dm}\right)$$
$$= \frac{d - dR + Rm}{m}$$