On Adaptive Random Testing

Fei-Ching Kuo

Faculty of Information and Communication Technologies
Swinburne University of Technology
Hawthorn, Victoria 3122, Australia

A thesis submitted for the degree of PhD in 2006

Abstract

Adaptive random testing (ART) has been proposed as an enhancement to *random testing* for situations where failure-causing inputs are clustered together. The basic idea of ART is to evenly spread test cases throughout the input domain. It has been shown by simulations and empirical analysis that ART frequently outperforms random testing. However, there are some outstanding issues on the cost-effectiveness and practicality of ART, which are the main foci of this thesis.

Firstly, this thesis examines the basic factors that have an impact on the fault-detection effectiveness of adaptive random testing, and identifies favourable and unfavourable conditions for ART. Our study concludes that favourable conditions for ART occur more frequently than unfavourable conditions. Secondly, since all previous studies allow duplicate test cases, there has been a concern whether adaptive random testing performs better than random testing because ART uses fewer duplicate test cases. This thesis confirms that it is the even spread rather than less duplication of test cases which makes ART perform better than RT. Given that the even spread is the main pillar of the success of ART, an investigation has been conducted to study the relevance and appropriateness of several existing metrics of even spreading. Thirdly, the practicality of ART has been challenged for nonnumeric or high dimensional input domains. This thesis provides solutions that address these concerns. Finally, a new problem solving technique, namely, mirroring, has been developed. The integration of mirroring with adaptive random testing has been empirically shown to significantly increase the cost-effectiveness of ART.

In summary, this thesis significantly contributes to both the foundation and the practical applications of adaptive random testing.

Acknowledgments

Firstly, I wish to express my sincere gratitude to my supervisor, Professor Tsong Yueh Chen. Without his encouragement, I might not either have undertaken such a challenging study (PhD) or have been so motivated throughout my candidature. Because of his excellent guidance, technical expertise and professionalism, I could learn and achieve so much in my PhD study.

I am grateful to my research colleagues in Software Testing Group for their invaluable comments and help in my study. I am also grateful to the staff in the Faculty of Information and Communication Technologies, Swinburne University of Technology for their support which provide an excellent research environment. I would like to thank Dr. Dave Towey for his help in improving the presentation of this thesis.

Finally, I would like to thank my family members and friends for their continuing care and support, which enabled me to complete my study

Declaration

I declare that the work presented in this thesis is original and my own work, and that it has not been submitted for any other degree. Parts of this thesis are the results of my research on adaptive random testing with others [10, 12, 16, 17, 18, 19, 20, 21, 22].

Signed:

Dated:

Contents

List of Tables

Table **Page**

List of Figures

Figure **Page**

Chapter 1

Introduction

One of the major challenges in software development is establishing the correctness of software. There exist many methods to prove the total correctness of a program, but these methods cannot be fully automated and are impractical for even moderately complex software. This approach to software quality assurance is called *program proving* [39, 47].

Unlike program proving, *software testing* is unable to prove that the program under test is free of errors. It assures software quality by actively detecting bugs before serious software failures actually take place. Once failures are detected in testing, the problem is handed over to the debugging team to fix. Software testing is a major part of software quality assurance [38] and accounts for over 50% of the total software development cost.

It has long been known that, in the process of software development, the later in the process that a bug is discovered, the more it costs to repair [6]. As a result, testing is no longer a standalone phase as suggested by some traditional software development life cycle models. Nowadays, testing takes place in every stage of the software development process in order to detect failures as early as possible.

1.1 Testing life cycle

Software testing has its own life cycle [55], consisting, basically, of 6 different stages: *unit testing*, *integration testing*, *function testing*, *performance testing*, *acceptance testing* and *installation testing*.

In unit testing, every software component is tested alone to ensure that it operates correctly on its own. In integration testing, components are tested together to ensure their interfaces are defined and handled properly. In function testing, the focus is on whether the integrated system performs the functions as described in the system functional requirements. In performance testing, the goal is to check whether the system meets the nonfunctional requirements set by the customers. *Usability tests*, *stress tests*, *configuration tests* and *timing tests* are examples of performance testing. In acceptance testing, customers are invited to participate in the testing process. It is possible that the software passes all the tests, but fails to meet customers' expectations. This can happen, for example, when software developers have incorrectly interpreted the customers' requirements for the system. When the software passes the acceptance testing, it is ready to be installed for use. In the final stage of testing (the installation testing), the objective is to ensure that when the system is in operation, it functions as it should.

1.2 Testing activities and approaches

Every stage of testing involves some of the following activities: (*i*) defining testing objectives, (*ii*) designing test cases, (*iii*) generating test cases, (*iv*) executing the test cases and (*v*) analyzing test results. Most of these activities are laborious and expensive but the testing resources are often limited, hence there is a genuine demand for a good test plan and automation of the testing process.

Note that Activity (*iv*) involves program execution. The approach of testing with program execution is called the *dynamic approach*, and the approach of testing without program execution is called the *static approach*. Hereafter, all the discussions will be within the context of the dynamic approach. For convenience, the following terminologies are now explained.

Test cases for the dynamic approach are those selected for program execution from the input domain, D (the input domain is the collection of all possible inputs to the program under test). Testing which involves every element of D is called *exhaustive testing*. Unfortunately, exhaustive testing is normally impractical, hence a subset of D (called *test suite* T) is normally designed to achieve certain testing

objectives. Inside D, an element which is able to reveal program failures is called a *failure-causing input*. The objective of testing is to uncover the failure-causing inputs in D.

1.3 Problems of testing

Two problems exist in software testing. One is known as the *test oracle problem* and the other is known as the *reliable test set problem*.

Suppose p is a program computing a function f on D. In general, for some t $\in$ D, when the expected outcome of f(t) is known, a failure is revealed if the actual program output, p(t), is not equal to f(t). Such a mechanism to verify the program correctness is known as the *test oracle*. The *test oracle problem* refers to the situation where no test oracle exists for p, or it is very expensive to verify the outputs of p [63]. Much works has been done to tackle or alleviate this problem. For instance, *metamorphic testing* has been proposed to test programs without the need of an oracle. It employs properties of the function under test, known as *metamorphic relations*, to verify the outputs [13].

Most of the time, the size of D is very large. Testers can only afford to test a subset of D, that is T . However, it is extremely difficult to design T such that its size is manageable and ($\forall$ t $\in$ T, p(t) = f(t)) $\Rightarrow$ ($\forall$ t' $\in$ D, p(t') = f(t')). This problem is called the *reliable test set problem* [37, 42, 71]. Obviously, the quality of T is crucial to the quality of software testing. Many testing techniques have been developed for the design of T, but none of them can detect all failures of software. Since different testing techniques are designed for different objectives, it is common to use more than one technique in testing in order to enhance the quality of the program under test.

1.4 Testing effectiveness and completeness

In this thesis, a testing technique is said to be more effective if, by its generated test suite (T), the method can detect more failures, have a higher chance to detect a failure, or detect a failure faster in terms of requiring fewer test cases to detect the first failure. Basically, "effectiveness" refers to how well a testing technique can

detect software failures. Testing is said to be more complete or comprehensive if more software functionalities, program components or wider ranges of faults could be covered during the test.

Software testing aims to effectively detect failures and to be as complete as possible, hence aspects of "effectiveness" and "completeness" are often used for comparing various testing techniques.

There exist three measures for evaluating the effectiveness of a testing technique: *P-measure*, *E-measure* and *F-measure*. P-measure is defined as the probability of detecting at least one failure, E-measure is defined as the expected number of failures detected, and F-measure is defined as the number of test cases required to detect the first failure. Basically, given a program under test, a testing technique with higher P-measure, higher E-measure or lower F-measure is more effective in detecting failures. Since it is common for debugging to take place once a failure is detected, F-measure is intuitively more appealing to testers and more realistic and informative from a practical point of view. Therefore, F-measure is chosen for measuring testing techniques in this thesis.

There exist many criteria (known as the *adequacy criteria*) to determine how completely a test is performed [64, 70]. Many of them refer to how much of the software functionality and aspects have been tested. A lot of testing techniques test software against these criteria. Therefore, once the software passes the test, its quality can be described in terms of these criteria. Well-known coverage criteria include statement coverage, branch coverage and path coverage.

1.5 Selection of test cases

Testing techniques which focus on the design of test cases are called *test case selection strategies*. Without ambiguity, in the rest of this thesis, unless otherwise specified, both "testing methods" and "testing techniques" will mean "test case selection strategies". The following presents two basic classifications of test case selection strategies.

1.5.1 Black or white box testing

Test case selection strategies can be further classified into two categories: *black box* and *white box*. The former designs tests from the *functional* perspective and ignores the implementation details; the latter takes a *structural* point of view and generates test cases based on program code. *Path coverage testing* is a typical example of white box testing, and requires all program paths to be tested at least once. It belongs to a subcategory of white box test, namely *control-flow coverage testing*[40]. Other than the technique of control-flow coverage testing, white box testing also includes various types of data-flow coverage testing [57] and various types of mutation testing [31, 43, 67]. One black box testing technique is the *category partition method* [54]. In the category partition method, testers need to analyze the program specifications, construct a set of categories and partitions, then finally form test specifications from which test cases are generated [54, 25]. Other examples of black box testing techniques include *random testing*, *equivalence partitioning* and *boundary-value analysis* [51].

1.5.2 Systematic or random testing

As discussed in Section 1.3, the search space (D) for failures is normally very large, hence searching the entire space, called *exhaustive testing*, is impossible. Because of this constraint, many techniques divide the input domain into subdivisions (called *subdomains* or *partitions*) within which they search for failures. This "systematic" approach is known as *partition testing*. The motivation behind partition testing is to make all the elements in a partition somehow "homogeneous". A partition is said to be homogeneous if either all of its elements reveal failures, or none do. For every homogeneous partition, it is sufficient to test any one of its elements. Although there exist some informal guidelines for creating such homogeneous partitions [66, 58], in practice, this goal is very difficult to achieve, particularly for a large and complex program [66]. Partition testing consists of two components: a *partitioning scheme* (used to put elements into groups, in other words, how to partition the input domain), and an *allocation scheme* (used to select test cases from partitions). It should be noted that many partitioning schemes are defined based on adequacy criteria.

In many ways, random testing is the complete opposite to partition testing. It treats all inputs independently from each other, and randomly selects test cases from the input domain according to a predefined distribution. Because random testing somehow detects failures by luck, some researchers have been questioning its effectiveness. Random testing has been criticized as "probably the poorest testing methodology" by Myers [51]. This point of view was not changed until the 1980s. The most surprising results were from the work of Duran and Ntafos [32]. Through simulations, they found that the effectiveness of partition testing is not, or just slightly superior to random testing. Since partition testing often requires more efforts than random testing [1], Duran and Ntafos considered random testing more cost-effective than partition testing. Hamlet and Taylor conducted more extensive simulations to verify the finding of Duran and Ntafos, and obtained similar results [61]. Weyuker and Jeng compared these two testing approaches from an analytical point of view. They found that partition testing can be an excellent testing strategy or a poor one, depending on how failure-causing inputs are grouped within the partitions [65]. Obviously, there is no guarantee that a favourable partitioning scheme can always be constructed for partition testing.

Even now, the comparison between partition testing and random testing still attracts great interest, but no research has been able to conclude which approach is definitely better than the other.

1.6 Focus and organization of this thesis

Compared to systematic approaches, random testing (RT) is simple and is found to be very effective in detecting failures which are not revealed by deterministic approaches. Because of these advantages, RT has been popularly used in industry for revealing software failures [29, 30, 35, 48, 49, 50, 52, 59, 60, 68]. Chen *et al.* [23, 45] have proposed an enhancement to RT. Their proposed approach is called *adaptive random testing* (ART), which both randomly selects and evenly spreads test cases. It is found that ART outperforms pure random testing under certain

[1] Partition testing requires effort in dividing the input domain, determining the conditions for each partition from which a test can be generated, searching the entire input domain for elements which satisfy the partition condition (this task is unsolvable in general [41]), and keeping track of which partitions have been tested or not.

circumstances [23, 45]. This thesis addresses some of the outstanding issues of ART. The thesis content is organized as follows:

Chapter 2 introduces ART and outlines the contributions of this thesis. Chapter 3 studies the factors which have an impact on the fault-detection effectiveness of ART. Chapter 4 studies the impact of selection with replacement (where duplication of test cases is allowed) and without replacement (where duplication is not allowed) on ART. In Chapter 5, the test case distributions of various implementations of ART are presented. Chapter 6 investigates how ART can be applied to nonnumeric programs. In Chapter 7, a cost-effective way of implementing ART, namely mirror ART, is presented. Chapter 8 studies the problems of ART in high dimensional space, and provides solutions to these problems. Chapter 9 concludes the thesis.

Chapter 2

Preliminary

This chapter defines the notations and assumptions used in this thesis. In addition, it presents the technique and some outstanding issues of adaptive random testing (ART) in detail. Finally, it outlines the contributions of this thesis.

2.1 Notations and assumptions

Unless otherwise specified, it is assumed that the program under test is a numeric program, the input parameters of the numeric program are real numbers, and the value range of each parameter is bounded. Such a kind of input domain (D) is said to be of *real type*.

A program under test is N dimensional (or simply, NĐ), if it accepts N input parameters. An input parameter may be called an "axis" or a "dimension". Let I = $\{I_1, I_2, ..., I_N\}$ be the set of input parameters; D = $\{(I_1, I_2, ..., I_N) | \min I_i \leq I_i \leq \max I_i, \forall 1 \leq i \leq N\}$ is used to describe the input domain. Within D, the collection of all failure-causing inputs is denoted by F.

In this thesis, a set symbol embraced by | | denotes the size of that set. For example, |D| denotes the size of D and |F| denotes the size of F. The program *failure rate* (θ) is $\frac{|F|}{|D|}$.

It is well known that failure-causing inputs tend to cluster together [2, 5, 34]. A *failure region* is a region where these inputs are clustered [2]. A *failure pattern* (ρ) refers to the geometric shape of the failure regions, and the distribution of these regions within the input domain. The failure rate and failure pattern are both

important attributes of a program, and are fixed, but unknown to testers before testing.

Chan *et al.* [9] have coarsely classified the failure pattern into three categories: point pattern, strip pattern and block pattern. Some sample faulty programs which lead to these failure patterns are shown in Programs 2.1-2.3. The corresponding patterns are illustrated in Figure 2.1.

Program 2.1 A program fault causing a block pattern.

```
INPUT X, Y
IF (X > 10 AND X < 15 AND Y > 10 AND Y < 15)
{  Z = X/2  } /* ERROR: Should be Z = 2 * X */
ELSE
{  Z = 2*Y  }
OUTPUT Z
```

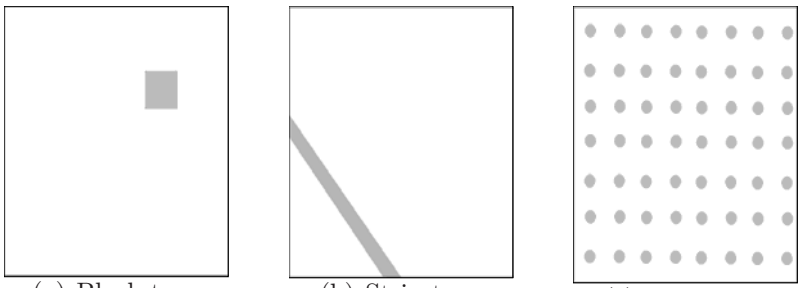
Program 2.2 A program fault causing a strip pattern.

```
INPUT X, Y
IF (X + Y <= -4) /* ERROR: Should be IF(X + Y <= -8) */
{  Z = X - Y  }
ELSE
{  Z = X + Y  }
OUTPUT Z
```

Program 2.3 A program fault causing a point pattern.

```
INPUT X, Y
IF (X mod 3 = 0 AND Y mod 3 = 0)
{  Z = X - Y  } /* ERROR: Should be Z = X + Y */
ELSE
{  Z = X * Y  }
OUTPUT Z
```

Random testing (RT) can be used with an operational profile to estimate the reliability of software under test. The accuracy of the estimation depends on how accurately the operational profile reflects the actual usage of the software. However, in the context of testing, it is common to apply RT with inputs of uniform distribution, that is, inputs having the same chance of being selected as test cases. Therefore, in this thesis, random testing and its variants are assumed to select test cases with uniform distribution and with replacement (thus, test cases may be du-



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

Figure 2.1: Three types of failure patterns

plicated). Like in other studies [28], this assumption is made mainly because it has a simpler mathematical model, and makes the analysis more tractable.

Finally, it is assumed that the test oracle does exist for verifying the test outcomes.

2.2 Adaptive random testing (ART)

There exist many efficient algorithms to select inputs randomly from an input domain. This feature makes RT a popular method, particularly when other testing techniques are too expensive, or too difficult to apply because formal specifications, or the source code on which they rely, are unavailable. In reality, because users often crash systems in many unexpected ways, test cases generated by RT may best reflect these unexpected events or inputs. Because of this, RT can detect certain failures unable to be revealed by deterministic approaches. Due to these merits, RT has been widely used in many real-life applications [29, 30, 35, 48, 49, 50, 52, 60, 68, 59]. RT is not only an important testing technique when applied alone, but also a core component of many other testing techniques. For example, many partition testing techniques select test cases randomly from each partition. It is interesting to note that RT has been adopted in many software testing tools available in the industry, for example, those developed by IBM [4] and Bell Laboratories (Directed Automated Random Testing (DART)) [36]. As pointed out in [36], RT "can be remarkably effective at finding software bugs".

Despite the popularity of RT, some people criticize it for using little or no information about program specifications or code to guide the selection of test cases. Chen *et al.* [23, 45] made use of certain aspects of faulty programs to enhance the

effectiveness of RT. They found that when failure-causing inputs were clustered (like the block pattern in Figure 2.1(a) or strip pattern in Figure 2.1(b)), the effectiveness of RT could be greatly improved if a more even and widespread distribution of test cases was used. They named this approach *adaptive random testing* (ART)[23, 45].

2.2.1 Methods of ART

ART can be implemented in various ways. These implementations can be roughly classified into 3 major categories.

1. **The best of all approach**.

   The principle is to find the best candidate, from a pool of potential test cases, as the next test case. The best may be the one farthest apart from all the executed test cases. A typical example of this approach is the *Fixed-Size-Candidate-Set ART (FSCS-ART)* method.

2. **The exclusion approach**.

   The principle is to define exclusion regions around each executed test case. Random test cases which fall inside these exclusion regions will not be chosen as test cases. This approach keeps new test cases away from the executed test cases. The *Restricted Random Testing (RRT)* [11] method belongs to this approach.

3. **The partitioning approach**.

   Random Testing by Random Partitioning (RPRT) [14] and *Random Testing by Bisection (BPRT)* [14] belong to this approach. Both methods partition the input domain into subdomains, from which they pick one subdomain for testing (that is, they generate a test case from that subdomain). Although their approaches are the same, they are different in the way of dividing the input domain and selecting subdomains for testing.

Basically, the above approaches attempt to avoid selection of similar inputs in a sequence of tests. In this thesis, unless otherwise specified, FSCS-ART is the method of ART studied. In this section, only the FSCS-ART algorithm is presented. The algorithms for RRT, BPRT and RPRT are explained in Section 5.1.

FSCS-ART makes use of two sets of test cases, namely, the *executed set* (E) and the *candidate set* (C). E holds all test cases that have been executed without revealing any failure. C stores k randomly generated inputs, from which the next test case will be chosen. The element in C, which has the largest distance from its nearest neighbour in E, is selected for testing. Figure 2.2 gives the FSCS-ART algorithm, which returns n as its F-measure once the first failure is detected.

As suggested by Chen *et al.* [23, 45], the larger k is, the smaller the F-measure of ART (F_{ART}) is, but values greater than 10 do not bring any significant reduction to F_{ART}. In this thesis, 10 is the default value of k in the experiments, unless otherwise specified.

1. Set $n = 0$ and E = { }.
2. Randomly select a test case, t, from the input domain according to the uniform distribution.
3. Increment n by 1.
4. If t reveals a failure, go to Step 9; otherwise, store t in E.
5. Randomly generate C = $\{c_1, c_2, ..., c_k\}$ according to the uniform distribution.
6. For each element in C, find its nearest neighbour in E.
7. In C, find which element, c_j, has the longest distance to its nearest neighbour in E.
8. Let t = c_j and go to Step 3.
9. Return n and t. EXIT.

Figure 2.2: The FSCS-ART algorithm

2.2.2 Comparison between RT and ART

Since ART is developed to enhance the fault-detection effectiveness of RT, the effectiveness of these two random methods have often been compared. The effectiveness metric for the comparison is the F-measure, which is the number of test cases required to detect the first failure. Given a faulty program for testing, whose failure rate is θ, theoretically, the F-measure of RT (F_{RT}) is $\frac{1}{\theta}$. However, it is extremely difficult to analytically derive the F-measure of ART (F_{ART}). Therefore, F_{ART} is obtained through experiments. Using the central limit theorem [33], the sample size required to estimate the mean F-measure with an accuracy range of $\pm$r% and a

confidence level of $(1-\alpha)\times 100\%$ is given by Formula 2.1.

$$S = \left(\frac{100 \cdot z \cdot \sigma}{r \cdot \mu}\right)^2 \tag{2.1}$$

where z is the normal variate of the desired confidence level, μ is the population mean and σ is the population standard deviation. Since μ and σ are unknown, the mean ($\bar{\chi}$) and standard deviation (s) of the collected F-measure data are used instead. Hence, we have the following formula.

$$S = \left(\frac{100 \cdot z \cdot s}{r \cdot \bar{\chi}}\right)^2 \tag{2.2}$$

Figure 2.3 describes the procedure to obtain F_{ART}. In this thesis, the accuracy range is set to $\pm 5\%$, and the confidence level is set to 95% (thus, the value of z is 1.96).

1. Set M = { }.
2. Test the faulty program with an ART method to find the F-measure.
3. The F-measure obtained in Step 2 is stored in M.
   (The sample size is the number of elements in M, that is |M|.)
4. Calculate the mean $\bar{\chi}$ and standard deviation s based on the data stored in M.
5. Calculate S using Formula 2.2
6. If |M| $\geq$ S, then report the average F-measure = $\frac{\sum_{i=1}^{|\mathrm{M}|} m_i}{|\mathrm{M}|}$, where $\mathrm{m}_i \in M$; otherwise, go to Step 2.

Figure 2.3: The procedure to obtain F_{ART}

2.2.2.1 12 numeric programs

To compare ART with RT, an empirical study was conducted by Chen *et al.* [23]. They used both RT and ART to test the mutants of twelve published programs[1] and then compared the effectiveness of both techniques.

Table 2.1 reports some background information about these twelve programs with seeded faults and previous experiment results. Among these twelve programs, there are five programs with 1Đ input domains; three programs with 2Đ input domains; two programs with 3Đ input domains; and two programs with 4Đ input domains. The ranges for each program's input parameter are listed in the columns 'From' and

[1]These programs were initially selected from [1] and [56]

'To'. X, Y, Z and W denote the input parameters (axis) whenever relevant. Table 2.2 summarizes the failure patterns of these twelve programs with seeded faults.

Program	D	**Input domain**				$\mathbf{F}_{RT}$	$\mathbf{F}_{ART}$	$\mathbf{F}_{ART}/$
name		Axis	Type†	From	To			F_{RT}
Airy	1	X	Real	-5000.0	5000.0	1381.44	799.29	0.58
Bessj0	1	X	Real	-300000.0	300000.0	733.96	423.93	0.58
Erfcc	1	X	Real	-30000.0	30000.0	1803.62	1004.68	0.56
Probks	1	X	Real	-50000.0	50000.0	2634.86	1442.50	0.55
Tanh	1	X	Real	-500.0	500.0	557.96	306.86	0.55
Bessj	2	X	Int	2.0	300.0	802.17	466.66	0.58
		Y	Real	-1000.0	15000.0			
Gammq	2	X	Real	0.0	1700.0	1220.28	1081.43	0.89
		Y	Real	0.0	40.0			
Sncndn	2	X	Real	-5000.0	5000.0	636.19	628.68	0.99
		Y	Real	-5000.0	5000.0			
Golden	3	X	Real	-100.0	60.0	1860.81	1829.74	0.98
		Y	Real	-100.0	60.0			
		Z	Real	-100.0	60.0			
Plgndr	3	X	Int	10.0	500.0	2741.66	1806.94	0.66
		Y	Int	0.0	11.0			
		Z	Real	0.0	1.0			
Cel	4	X	Real	0.001	1.0	3065.25	1607.80	0.52
		Y	Real	0.001	300.0			
		Z	Real	0.001	10000.0			
		W	Real	0.001	1000.0			
El2	4	X	Real	0.0	250.0	1430.76	686.48	0.48
		Y	Real	0.0	250.0			
		Z	Real	0.0	250.0			
		W	Real	0.0	250.0			

†"Real" and "Int" means that the input parameter is real number and integer type, respectively.

Table 2.1: Information about previous studies in 12 programs

Their study shows that ART requires fewer test cases (with up to 50% saving) to detect the first failure than RT in nine of the twelve programs. For the remaining three programs (Gammq, Sncndn and Golden), it was found that their failure patterns do not belong to the type of failure pattern targeted by ART.

2.2.2.2 Simulations

Many comparisons between RT and ART were carried out through simulations. For each simulation study, failure rate and pattern were predefined, but the failure regions were randomly placed in the input domain. A failure is said to be found if a point inside the failure regions is selected as a test case.

Program	Failure pattern
Airy	A block in the centre of the input domain.
Bessj0	A block in the centre of the input domain.
Erfcc	A block in the centre of the input domain.
Probks	A block in the centre of the input domain.
Tanh	A block in the centre of the input domain.
Bessj	Strips.
Gammq	A long narrow strip
Sncndn	Points scattered over the entire input domain.
Golden	Points scattered over around very large hyperplanes.
Plgndr	Strips near the edge of the input domain
Cel	One failure region on the whole edge side of the input domain
El2	Strips near the edges

Table 2.2: Failure patterns of 12 programs

All previous simulation studies were restricted to the 3 types of failure patterns as illustrated in Figure 2.1. The block pattern consists of a single square or circle, the strip pattern is a narrow block, and the point pattern is made up by 10 equal-sized squares or circles. These simulation studies all confirm that ART performs best when the failure pattern is of block type ($F_{ART} \approx 60\%$ of F_{RT}) and worst when the failure pattern is of point type (however, $F_{ART} \approx F_{RT}$)[14, 20, 15].

2.3 Contribution of this thesis

Since the most time-consuming and labour intensive activity in RT (as well as in many other testing methods) is often related to verifying the outputs, ART alleviates this problem by reducing the number of test cases required to detect failures and consequently, the number of required output verifications. When the execution of the program is expensive, ART saves further cost. Since ART preserves the random nature of RT, it is an intuitively appealing alternative to RT. The above points have motivated further investigation of ART.

In this thesis, answers to the following issues are provided.

1. What are the factors affecting ART's effectiveness? (Chapter 3)

2. What are the favourable situations where ART should be applied? (Chapter 3)

3. How does selection with and without replacement affect the effectiveness of

ART? (Chapter 4)

4. How to measure the even spread of test cases for various ART methods? (Chapter 5)

5. How can ART be applied to nonnumeric programs, and what is its performance? (Chapter 6)

6. How can ART be implemented cost-effectively? (Chapter 7)

7. How can the cost-effectiveness of ART be enhanced in high dimensional space? (Chapter 8)

Chapter 3

Favourable and unfavourable conditions for adaptive random testing

It is of a great interest to know under what conditions ART will significantly outperform RT, because such information would enable testers to decide when to use ART instead of RT. In this chapter, an investigation into the fault-detection effectiveness of ART under various scenarios is undertaken. Without ambiguity, ART refers to the FSCS-ART method in this chapter and the size of the candidate set is assumed to be 10, unless otherwise specified.

Here, *ART F-ratio*, the ratio of ART's F-measure (F_{ART}) to RT's F-measure (F_{RT}), is used to indicate how much improvement ART has over RT. A smaller ART F-ratio implies a better improvement of ART over RT.

3.1 Factors influencing the effectiveness of ART

In this section, an investigation into the impact that individual factors (related to the distribution of failure-causing inputs) have on the effectiveness of ART is reported. The investigation was conducted through simulation experiments. Simulation experiments, unlike experiments using real-life programs, make it possible to control the experiment environment and parameters (failure rate, number of failure regions, etc), and thus facilitate the study of the effect that each of these parameters has.

In each experiment, all edge lengths of the N dimensional input domain are assumed to be of equal magnitude, unless otherwise specified. For example, when

$N = 2$, the input domain is a square, and when $N = 3$, the input domain is a cube.

3.1.1 Experiment 1: impact of the failure rate and the dimensions of the input domain

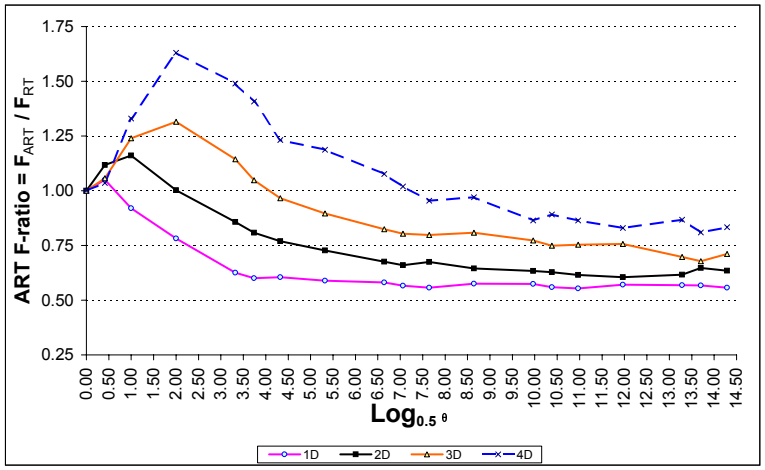


Figure 3.1: The relationship between the ART F-ratio and the failure rate (θ) in N-dimensional input domains, where $N = 1$, 2, 3 and 4

θ	$log_{0.5}\theta$	Expected $F_{RT} = \frac{1}{\theta}$	**ART F-ratio**=(F_{ART}/F_{RT})			
			1Đ	2Đ	3Đ	4Đ
1	0.00	1	1.00	1.00	1.00	1.00
0.75	0.42	1.3	1.05	1.12	1.06	1.04
0.5	1.00	2	0.92	1.16	1.24	1.33
0.25	2.00	4	0.78	1.00	1.32	1.63
0.1	3.32	10	0.63	0.86	1.14	1.49
0.075	3.74	13.3	0.60	0.81	1.05	1.41
0.05	4.32	20	0.61	0.77	0.97	1.23
0.025	5.32	40	0.59	0.73	0.90	1.19
0.01	6.64	100	0.58	0.68	0.82	1.08
0.0075	7.06	133.3	0.57	0.66	0.80	1.02
0.005	7.64	200	0.56	0.67	0.80	0.95
0.0025	8.64	400	0.58	0.65	0.81	0.97
0.001	9.97	1000	0.57	0.63	0.77	0.86
0.00075	10.38	1333.3	0.56	0.63	0.75	0.89
0.0005	10.97	2000	0.55	0.62	0.75	0.86
0.00025	11.97	4000	0.57	0.61	0.76	0.83
0.0001	13.29	10000	0.57	0.62	0.70	0.87
0.000075	13.70	13333.3	0.57	0.65	0.68	0.81
0.00005	14.29	20000	0.56	0.64	0.71	0.83

Table 3.1: Data corresponding to Figure 3.1

Experiment 1 investigates the relationship between the ART F-ratio under various failure rates (θ) and dimensionality (N). In the experiment, θ was varied from

0.00005 to 1. The failure region was a single square randomly placed within the input domain. The dimensions of the input domain were varied from 1 to 4. The results of the experiment are summarized in Figure 3.1, with corresponding data listed in Table 3.1.

This experiment revealed that, for a given θ, the F-ratio of ART is larger for input domains of higher dimensions. When θ gets smaller, however, this disparity of ART F-ratio in different dimensions becomes less significant. It should be noted that at large θ, ART uses more test cases to detect the first failure than does RT. A detailed study of ART's problems in higher dimensional input domains will be presented in Chapter 8.

Experiment 1 shows that when the failure-causing inputs form a single square region, and the failure rate (θ) is small enough, the effectiveness of ART improves as θ decreases in value. In contrast, the effectiveness of RT is not affected by the failure pattern, and the expected F_{RT} remains $\frac{1}{\theta}$. Obviously, the smaller the failure rate θ, the greater the savings ART provides in terms of the number of test cases needed to detect the first failure. For example, in the case of a 2Đ input domain, when θ=0.1, ART may only save about 1.4 test cases; but when θ=0.0001, ART can save about 3800 test cases.

3.1.2 Experiment 2: impact of the failure region shape

In Experiment 1, the failure region was a square. In real life, however, the failure region can be of many different kinds of shape. In Experiment 2, we investigated the relationship between the shape of a failure region and the ART F-ratio. Different shapes have different degrees of compactness, which indicate how densely the failure-causing inputs are clustered. The hypothesis of this study is that more compact the failure region is, the lower the ART F-ratio will be.

To test this hypothesis, Experiment 2 was conducted as follows. Only one failure region in the input domain was simulated with θ being set to 0.005, 0.001 and 0.0005. The experiment was conducted in 2Đ and 3Đ input domains, and failure regions were in rectangular shapes. The edge lengths of the failure regions were in the ratio of 1:r in 2Đ, and 1:r:r in 3Đ where r was varied from 1 to 100.

Many metrics have been proposed to measure the compactness of a shape [53,

62, 69]. One of these measures [53, 62] compares the shape's size with the size of the circular shape having an equal boundary. According to this measurement, Formulae 3.1 and 3.2 give the compactness ratio for 2Đ and 3Đ geometric shapes, respectively:

$$\frac{4A\pi}{P^2} \tag{3.1}$$

$$\frac{6A\sqrt{\pi}}{\sqrt{P^3}} \tag{3.2}$$

where A denotes the area (Formula 3.1) or the volume (Formula 3.2) of the shape; P denotes the perimeter (Formula 3.1) or the surface area (Formula 3.2) of the shape. The ratio ranges from 0 to 1, with the larger values implying a more compact shape. Because a circle/sphere encloses the largest region among all shapes having an equal boundary [8], it has a value of 1. Theorems 3.1 and 3.2 show that the larger the value of r, the less compact is the shape. The results of the experiment are shown in Figures 3.2(a) and 3.2(b).

Theorem 3.1. *Let R_1 and R_2 be two rectangles in the 2Đ space and the ratios of their widths to their lengths be 1: r_1 and 1: r_2, respectively, where $1 \leq r_1 < r_2$. Let their perimeters be P_1 and P_2 and their areas be A_1 and A_2, respectively, where both A_1 and $A_2 > 0$. If $A_1 = A_2$, then $\frac{4A_2\pi}{{P_2}^2} < \frac{4A_1\pi}{{P_1}^2}$*

Proof. For a rectangle with width x and length rx, its area $A = rx^2$ and perimeter $P = 2x(r+1)$. Hence, $P = 2(r+1)\sqrt{\frac{A}{r}}$. For rectangles R_1 and R_2, we have $r_2 - r_1 > 0$ and $1 - \frac{1}{r_1 r_2} > 0$

$\Rightarrow (r_2 - r_1)(1 - \frac{1}{r_1 r_2}) > 0$

$\Rightarrow r_2 - r_1 + \frac{1}{r_2} - \frac{1}{r_1} > 0$

$\Rightarrow r_2 + 2 + \frac{1}{r_2} > r_1 + 2 + \frac{1}{r_1}$

$\Rightarrow \frac{1}{r_2}(r_2+1)^2 > \frac{1}{r_1}(r_1+1)^2$

$\Rightarrow 4\frac{A_2}{r_2}(r_2+1)^2 > 4\frac{A_1}{r_1}(r_1+1)^2$ (because $A_1 = A_2$)

$\Rightarrow (2(r_2+1)\sqrt{\frac{A_2}{r_2}})^2 > (2(r_1+1)\sqrt{\frac{A_1}{r_1}})^2$

$\Rightarrow {P_2}^2 > {P_1}^2$

$\Rightarrow \frac{4A_2\pi}{{P_2}^2} < \frac{4A_1\pi}{{P_1}^2}$. □

Theorem 3.2. *Let C_1 and C_2 be two cuboids in the 3Đ space and the ratios of their edge lengths be 1: r_1: r_1 and 1: r_2 : r_2, respectively, where $1 \leq r_1 < r_2$. Let their*

surface areas be P_1 and P_2 and their volumes be A_1 and A_2, respectively, where both A_1 and $A_2 > 0$. If $A_1 = A_2$, then $\frac{6A_2\sqrt{\pi}}{\sqrt{P_2^3}} < \frac{6A_1\sqrt{\pi}}{\sqrt{P_1^3}}$.

Proof. For a cuboid whose edge lengths are x, rx and rx, its volume $A = r^2x^3$ and surface area $P = 2x^2(r^2 + 2r)$. Hence, $P = 2(\frac{A}{r^2})^{\frac{2}{3}}(r^2 + 2r)$. For cuboids C_1 and C_2 we have ${r_2}^{\frac{1}{3}} - {r_1}^{\frac{1}{3}} > 0$ and $1 - \frac{1}{r_1^{\frac{1}{3}} r_2^{\frac{1}{3}}} > 0$.

$\Rightarrow 2({r_2}^{\frac{1}{3}} - {r_1}^{\frac{1}{3}})(1 - \frac{1}{r_1^{\frac{1}{3}} r_2^{\frac{1}{3}}}) > 0$

$\Rightarrow 2({r_2}^{\frac{1}{3}} - {r_1}^{\frac{1}{3}}) - 2(\frac{r_2^{\frac{1}{3}} - r_1^{\frac{1}{3}}}{r_1^{\frac{1}{3}} r_2^{\frac{1}{3}}}) > 0$

$\Rightarrow ({r_2}^{\frac{1}{3}} + {r_1}^{\frac{1}{3}})({r_2}^{\frac{1}{3}} - {r_1}^{\frac{1}{3}}) - 2(\frac{r_2^{\frac{1}{3}} - r_1^{\frac{1}{3}}}{r_1^{\frac{1}{3}} r_2^{\frac{1}{3}}}) > 0$ (because $({r_2}^{\frac{1}{3}} + {r_1}^{\frac{1}{3}}) > 2$)

$\Rightarrow {r_2}^{\frac{2}{3}} + 2{r_2}^{\frac{-1}{3}} - {r_1}^{\frac{2}{3}} - 2{r_1}^{\frac{-1}{3}} > 0$

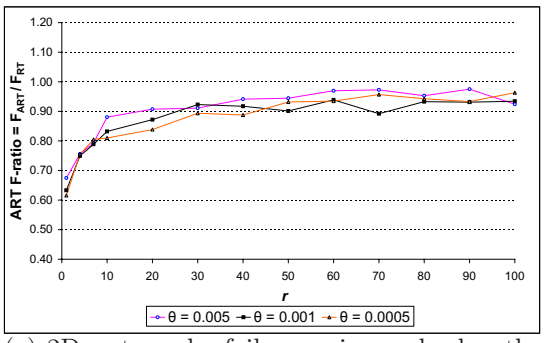
$\Rightarrow {r_2}^{\frac{-4}{3}}({r_2}^2 + 2r_2) - {r_1}^{\frac{-4}{3}}({r_1}^2 + 2r_1) > 0$

$\Rightarrow 2(\frac{A_2}{{r_2}^2})^{\frac{2}{3}}({r_2}^2 + 2r_2) > 2(\frac{A_1}{{r_1}^2})^{\frac{2}{3}}({r_1}^2 + 2r_1)$ (because $A_1 = A_2$)

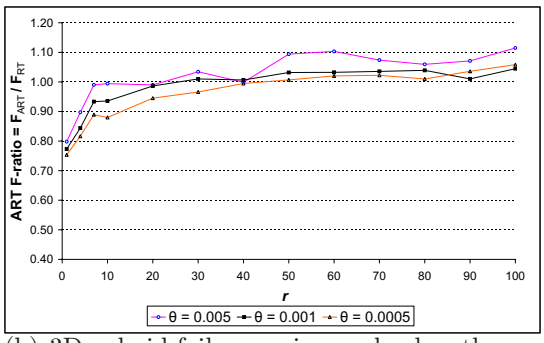
$\Rightarrow P_2 > P_1$

$\Rightarrow P_2^{\frac{3}{2}} > P_1^{\frac{3}{2}}$

$\Rightarrow \frac{6A_2\sqrt{\pi}}{\sqrt{P_2^3}} < \frac{6A_1\sqrt{\pi}}{\sqrt{P_1^3}}$. □



(a) 2Đ rectangular failure regions; edge lengths are in the ratio 1:r



(b) 3Đ cuboid failure regions; edge lengths are in the ratio 1:r:r

Figure 3.2: The relationship between the ART F-ratio and the failure region compactness

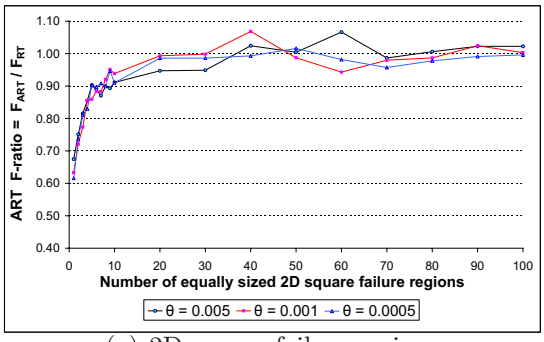
This experiment shows that the ART F-ratio increases when r increases. In other words, when the failure region becomes less compact, the fault-detection effectiveness of ART also decreases, regardless of the value of θ, or the dimensions of the input domain.

Note that the use of rectangular shapes in Experiment 2 is for illustration only: failure regions may be of various shape. Investigations into other shapes [21] have also confirmed the hypothesis (that the more compact the failure region is, the

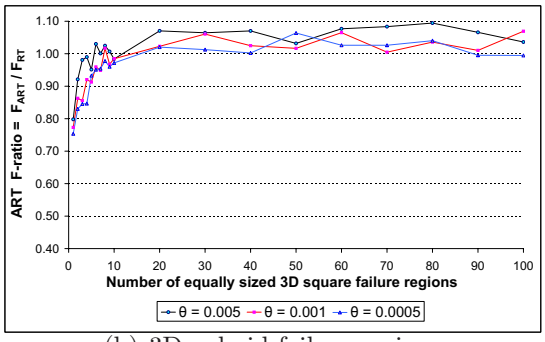
better is the ART's fault-detection effectiveness). These findings show that the failure region shape has an impact on the ART F-ratio.

3.1.3 Experiment 3: impact of the number of failure regions

Previous studies [9, 45, 23] showed that ART has a significant improvement over RT when the failure-causing inputs are clustered in a block region, but not when the failure-causing inputs are scattered over the input domain. Nevertheless, the relationship between the number of failure regions and the ART F-ratio has never been precisely reported. Experiment 3 was designed to investigate this relationship. In this experiment, the number of failure regions (n) was varied from 1 to 100, where all n failure regions were square and were of equal size. Both 2Đ and 3Đ input domains were investigated, with θ being set to 0.005, 0.001 and 0.0005. The results are shown in Figures 3.3(a) and 3.3(b).



(a) 2Đ square failure regions



(b) 3Đ cuboid failure regions

Figure 3.3: The relationship between the ART F-ratio and the number (n) of equally sized failure regions

The results show that for both 2Đ and 3Đ input domains, the ART F-ratio increases with n and reaches a plateau very quickly (around $n = 20$ in 2Đ and $n = 30$ in 3Đ), while keeping the failure rate unchanged. This result confirms the intuition that ART favours the situations where failure-causing inputs are clustered together, because more failure regions mean weaker clustering of failure-causing inputs.

3.1.4 Experiment 4: impact of the presence of a large failure region

Experiment 3 reported that the ART F-ratio increases with the number of failure regions (n) and it reaches a plateau when n is around 20 in 2Đ input domains and 30 in 3Đ input domains. All the failure regions in Experiment 3 were equally sized. In reality, however, equally sized failure regions are unlikely to occur. Experiment 4 investigated the situation where the input domain contained failure regions of varied sizes.

In this experiment, the input domain was set to be 2Đ and to contain n square failure regions, where $n > 1$. The failure rate (θ) was set to 0.001. The following method was used to assign the size to the various regions: after one of the n failure regions is assigned q% of θ, where $q > 0$, the remaining $n - 1$ failure regions share the $(100 - q)$% of θ as follows:

1. Randomly generate $n - 1$ numbers X_1, X_2, ..., X_{n-1} within the range (0, 1).
2. Define θ_i for each of the $n - 1$ failure regions to be $\frac{X_i}{\sum_{i=1}^{n-1} X_i}\ \theta\ (100 - q)\%$.

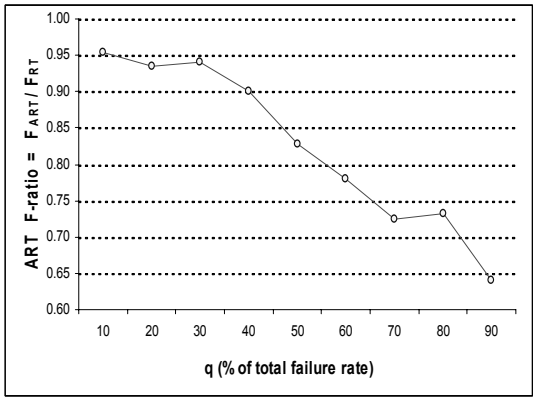


Figure 3.4: The relationship between the ART F-ratio and q, where $\theta = 0.001$ and the number of failure regions is 100

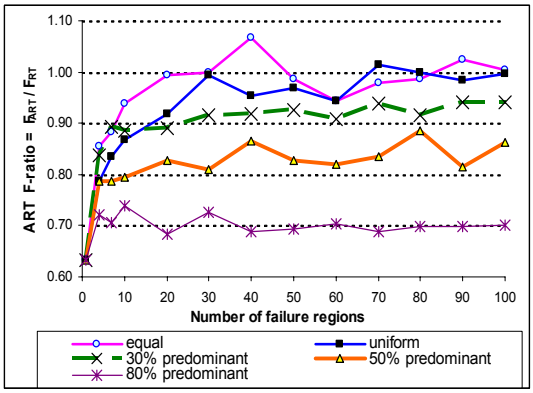


Figure 3.5: The relationship between the ART F-ratio and the number (n) of failure regions for various region size distributions, where θ=0.001

In the first series of experiments, n was set to 100, and q was varied from 10 to 90. The results are shown in Figure 3.4. The figure clearly shows that the ART F-ratio increases when q decreases. In other words, the larger the dominating failure region is, the better the fault-detection effectiveness of ART is.

The second series of experiments was conducted using q =30, 50 and 80; and n was varied from 1 to 100. Note that a predominant failure region exists when q =30, 50 or 80. The results of the experiment are shown in Figure 3.5. Also included are the case where all failure regions are of the same size, and the case where there is no predominant failure region (that is, n instead of $n-1$ random numbers (X_i's) were generated to share 100% θ). For ease of discussion, "equal distribution" refers to the former case, and "uniform distribution" refers to the latter case. On the other hand, "30% predominant distribution", "50% predominant distribution" and "80% predominant distribution" refer to the cases where q is 30, 50 and 80, respectively.

As shown in Figure 3.5, the favourable distributions for ART are in the following order: 80% predominant distribution is the most favourable, 50% predominant distribution, 30% predominant distribution, uniform distribution, and equal distribution is the least favourable. Interestingly, the ART F-ratio is very steady with respect to n for the 80% (around 0.7), 50% (around 0.85) and 30% (around 0.95) predominant distributions. Furthermore, when n is small, the ART F-ratio for the uniform distribution is slightly smaller than for the equal distribution; but as n increases, they become closer to each other. This is understandable because the larger n for the uniform distribution, the smaller the size differences among failure regions and, hence, the uniform distribution becomes more similar to the equal distribution.

The results of Experiment 4 indicate that the presence of a predominant failure region has a more significant impact on the ART F-ratio than does the number of failure regions.

3.2 Analysis of some experiments on real programs and the boundary effect

The results of the simulations reported in Section 3.1 present a clear picture of the impact of individual factors on the effectiveness of ART. In this section, the simulation results are compared with ART's performance on real programs. As mentioned in Section 2.2.2.1, there was an empirical study about the performance of ART on 12 programs. Table 2.1 gives detailed information about that study. Table 3.2 extracts the ART F-ratio from Table 2.1. Also listed are the estimated

ART F-ratios for a single square failure region with the same θ and dimensions as the faulty program, which can be regarded roughly as an estimated lower bound[1]. This value was obtained using linear estimation based on the data from Table 3.1. For example, Airy is a 1Ð program whose θ is 0.000716. Table 3.1 shows that for a 1Ð input domain, when $\theta = 0.00075$, the ART F-ratio for a single square failure pattern is 0.56; and when $\theta = 0.0005$, the ART F-ratio is 0.55. The estimated lower bound for Airy's ART F-ratio is therefore approximately 0.56.

Program Name	No. of dimensions of the input domain	Failure rate (θ)	Actual ART F-ratio	Estimated ART F-ratio for a square failure region with the same failure rate
Airy	1	0.000716	0.5786	0.56
Bessj0	1	0.001373	0.5776	0.57
Erfcc	1	0.000574	0.5570	0.56
Probks	1	0.000387	0.5475	0.56
Tanh	1	0.001817	0.5500	0.57
Bessj	2	0.001298	0.5817	0.64
Gammq	2	0.000830	0.8862	0.63
Sncndn	2	0.001623	0.9882	0.64
Golden	3	0.000550	0.9833	0.75
Plgndr	3	0.000368	0.6591	0.76
Cel	4	0.000332	0.5245	0.84
El2	4	0.000690	0.4798	0.89

Table 3.2: Results of the experiment on 12 numerical programs

For the first 5 programs in Table 3.2, the actual performance of ART was close to the estimated lower bound; for the programs Gammq, Sncndn and Golden, the actual ART F-ratio was worse than the estimated lower bound. The performance of ART on these 8 programs is understandable because their failure patterns are not as compact as the single square region.

For the remaining 4 programs (Bessj, Plgndr, Cel and El2), however, the actual ART F-ratios were significantly lower than the corresponding estimated lower bounds. In other words, the fault-detection effectiveness of ART was significantly better than the estimated optimal performance of ART obtained from the simulation in Section 3.1.1. This interesting phenomenon led to further investigation of these 4 faulty programs. It was found that their failure patterns have a common

[1] ART has about the lowest F-measure when the failure-causing inputs are clustered in a single, very compact region. It should be noted of course, that real life programs may not have failure patterns as compact as the single square region.

characteristic: some of their failure regions are near to or on the edge of the input domain. Since the FSCS-ART algorithm has a bias towards generating test cases near the edge[2], this explains why the ART F-ratios in these 4 programs were smaller than the simulated lower bound.

Since FSCS-ART is not the only ART algorithm that favours edges, it was desirable to obtain better understanding of ART's fault-detection effectiveness under the situations where failure regions are near to the boundary. Therefore, the following two experiments were conducted using the same assumptions as in the previous experiments, unless otherwise specified.

3.2.1 Experiment 5: impact of the location of the failure regions

In this experiment, the range for each dimension of the input domain was set to be [0, 100); and the input domain was partitioned into 5 areas. Such partitioning in 2Đ input domain is depicted in Figure 3.6. It was assumed that the coordinates of the bottom leftmost corner of the square input domain are at (0, 0). Area 1 was nearest to the boundary of the input domain, and was of width 10 units, that is, any point in Area 1 has at least one coordinate in the range [0, 10) or [90, 100). Similarly, Area 2 was in the range [10, 20) or [80, 90); Area 3 was in the range [20, 30) and [70, 80); and Area 4 was in the range [30, 40) and [60, 70). On the other hand, any point in Area 5 has all coordinates in the range [40, 60).

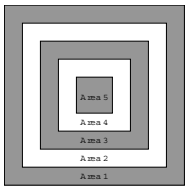


Figure 3.6: A 2Đ input domain with 5 areas

	θ	ART F-ratio				
		Area 1	Area 2	Area 3	Area 4	Area 5
1Đ	0.1	0.38	0.65	0.65	0.65	0.62
2Đ	0.01	0.54	0.67	0.74	0.66	0.66
3Đ	0.001	0.62	0.72	0.75	0.74	0.74
4Đ	0.0001	0.71	0.84	0.84	0.82	0.82

Table 3.3: Results of Experiment 5 (a square region attaches to the boundary)

In this experiment, a square failure region was randomly placed inside one of these areas. The edge length of the failure region was 10. Hence, for the 1Đ input domain, θ=0.1; for 2Đ, θ=0.01; for 3Đ, θ=0.001; and for 4Đ, θ=0.0001. The results

[2]FSCS-ART is one of ART methods which have test cases more frequently selected from the edge area of the input domain (refer to Chapter 5 for details)

of the experiment (Table 3.3) show that the performance of ART was better when the failure region is near the input domain boundary. This observation leads to the following study: the impact of the proximity of failure regions to the input domain boundary on the ART F-ratio (Section 3.2.2).

3.2.2 Experiment 6: impact of the proximity of failure regions to the input domain boundary

Experiment 6 is a modification of Experiments 1-3, the difference between the modified experiments and the experiments in Sections 3.1.1- 3.1.3 being the location of the failure regions. The failure regions were randomly attached to the boundary of the input domain instead of being randomly placed anywhere within the input domain. The results for the modified version of Experiment 1 are shown in Figure 3.7. These results show that when the failure region is on the edge, the ART F-ratio decreases for input domains of all dimensions when compared with Experiment 1 (Figure 3.1).

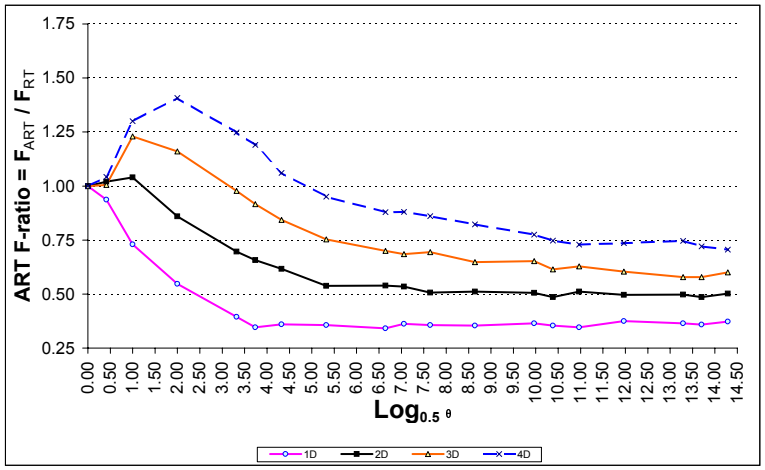


Figure 3.7: The results of the modified Experiment 1

The results of the modified version of Experiment 2 are summarized in Figure 3.8. Since the rectangular/cuboid failure regions have different edge lengths, depending on which side of the region attached to the boundary, there may be a different impact on the ART F-ratio. Figures 3.8(a) and 3.8(b) show that when any side of the region can be attached to the boundary, the ART F-ratio goes up as r increases, but with a slower rate as compared with Figures 3.2(a) and 3.2(b). Figures 3.8(c)

and 3.8(d) show that when only the longest side of the failure region is attached to the boundary, ART F-ratio remains steadily (0.5 for a rectangular failure region and 0.55 for a cuboid failure region) irrespective of the failure rate. Figures 3.8(e) and 3.8(f) show that when only the shortest side of the failure region is attached to the boundary, the results are very similar to those reported in Figures 3.2(a) and 3.2(b). Clearly, when the longest side of the failure region is attached to the boundary, more failure-causing inputs will be at the edges. This explains why attaching the longest side to the boundary results in a much lower ART F-ratio than attaching the shortest side.

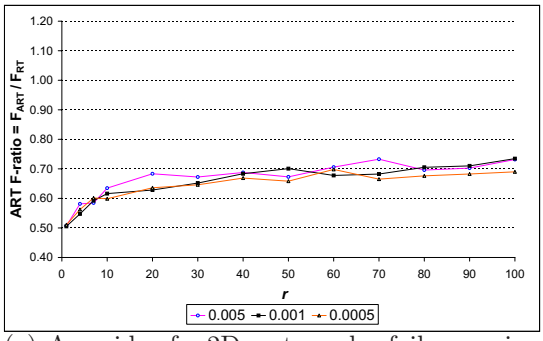
The results of the modified version of Experiment 3 are shown in Figure 3.9. The results show that when the failure regions are attached to the boundary, as n increases, the ART F-ratio approaches 0.5 and 0.6 for 2Đ and 3Đ space, respectively.

In summary, the experiments here show that ART F-ratio can be significantly improved when the failure regions are attached to the boundary of the input domain, regardless of the failure pattern type. Indications are that the proximity of failure regions to the boundary of the input domain has a stronger impact on the ART F-ratio than do the other factors.

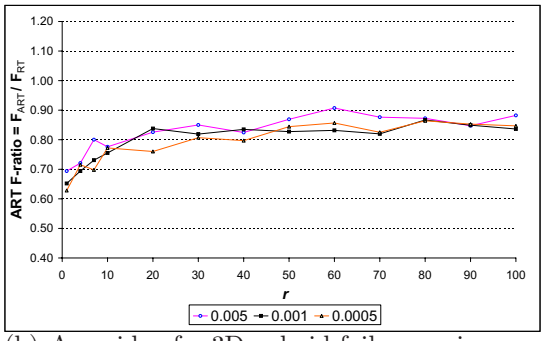
3.3 Discussion and conclusion

In this chapter, a series of studies conducted to investigate the fundamental factors influencing the fault-detection effectiveness of ART was reported. Based on the results of Experiments 1 to 6, it was concluded that the ART F-ratio depends on: (1) the dimensions of the input domain; (2) the failure rate; (3) the compactness of the failure region; (4) the number of failure regions; (5) the size of the predominant failure region (if any); and (6) the proximity of the failure region to the boundary of the input domain.

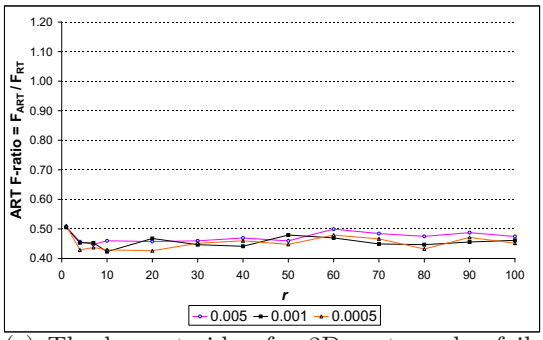
Six favourable conditions for ART have been identified: (1) when the number of dimensions is small; (2) when the failure rate is small; (3) when the failure region is compact; (4) when the number of failure regions is low; (5) when a predominant region exists among all the failure regions; and (6) when the failure region is near to the boundary of the input domain.



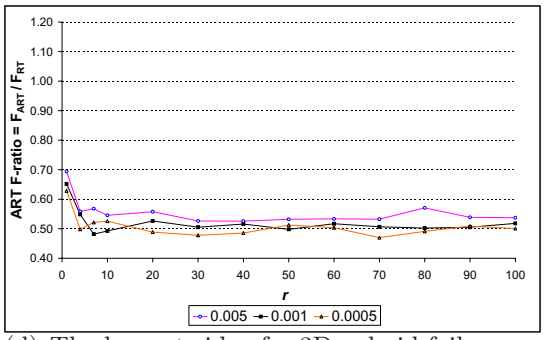
(a) Any side of a 2Đ rectangular failure region can be attached to the boundary; edge lengths are in the ratio $1 : r$



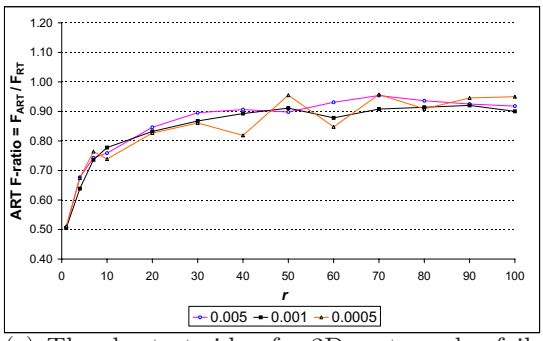
(b) Any side of a 3Đ cuboid failure region can be attached to the boundary; edge lengths are in the ratio $1 : r : r$



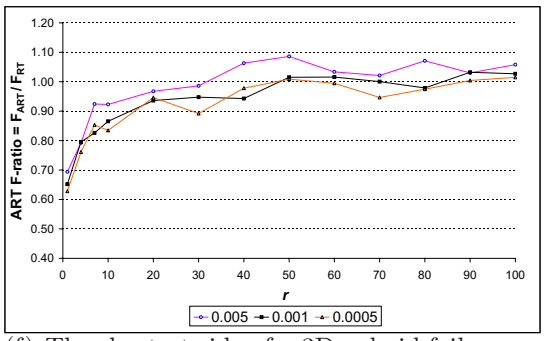
(c) The longest side of a 2Đ rectangular failure region is attached to the boundary; edge lengths are in the ratio $1 : r$



(d) The longest side of a 3Đ cuboid failure region is attached to the boundary; edge lengths are in the ratio $1 : r : r$

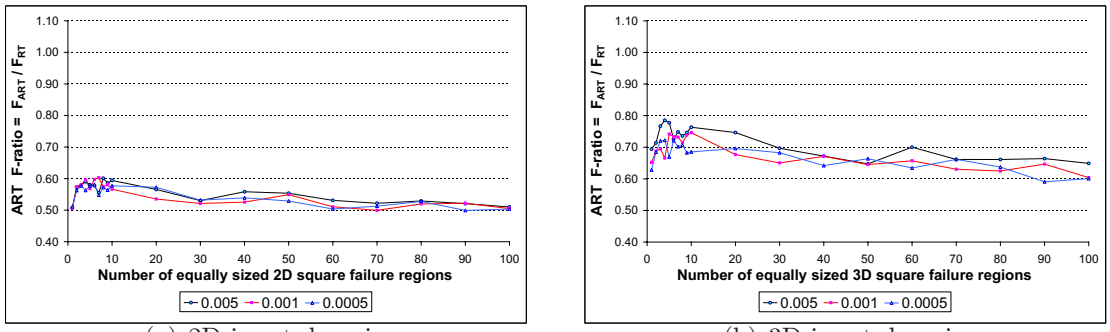


(e) The shortest side of a 2Đ rectangular failure region is attached to the boundary; edge lengths are in the ratio $1 : r$



(f) The shortest side of a 3Đ cuboid failure region is attached to the boundary; edge lengths are in the ratio $1 : r : r$

Figure 3.8: The results of the modified Experiment 2



(a) 2Đ input domain (b) 3Đ input domain

Figure 3.9: The results of the modified Experiment 3

There are many methods to implement ART. These methods not only share common attributes but also have their own distinct characteristics. Hence, not all the results about FSCS-ART reported in this chapter are applicable to other ART methods. However, it is important to note that all ART methods favour the situation where failure-causing inputs are clustered together, hence favourable condition (3), (4) and (5) apply to all ART methods. Obviously, FSCS-ART is not the only ART method which favours small failure rates and failure regions close to the boundary of the input domains. Other examples include Restricted Random Testing (RRT), which will be discussed in Chapter 5.

The performance of ART deteriorates as the number of dimensions increases because existing ART methods have not taken dimensionality into consideration. This topic will be studied further in Chapter 8 where it will be shown how ART can possibly be enhanced to perform well in any dimensional space.

In reality, we have neither control nor sufficient knowledge of the above five factors (excluding the dimensions of the input domain) for the program under test. This inability, however, should not stop us from applying ART because some of the favourable conditions for ART occur frequently. As reported in [2, 5, 34], failure-causing inputs tend to form clusters. This clustering is fundamental to the favourable conditions for ART. Hence, it is anticipated that favourable conditions for applying ART do occur frequently. Most importantly, the presence of some favourable conditions may outweight the adverse effect of some unfavourable conditions on the ART F-ratio, as shown in this study. For example, it has been shown that the presence of a predominant failure region is more influential on the ART F-ratio than the number of failure regions present. Hence, even when there are many failure regions (an

unfavourable condition for ART), ART can still perform well as long as there exists a predominant failure region. Another unfavourable condition for ART is when the failure region is not compact; yet, it was found that if some of the failure regions are near the input domain boundary, then FSCS-ART is still effective, regardless of how compact the failure region is. Finally, it is important to point out that the failure rate, number of faults and number of failure regions are likely to decrease when the software goes through a serious quality assurance process and, hence, the gain of using ART instead of RT will become greater.

In conclusion, the simulation experiments have provided new insights into the factors that influence the effectiveness of ART, and identified the favourable and unfavourable conditions for ART. Knowledge about these conditions can help practitioners decide when to use ART instead of RT.

Chapter 4

Test case selection with or without replacement

Chapter 3 showed that the F-measure of ART depends on many factors. In that study, selection with replacement was assumed. In other words, there could be duplicate test cases during the testing process.

Intuitively speaking, ART should have fewer duplicate test cases than RT, because whenever the candidate set (C) has some elements not appearing in the set of executed test cases (E), the next test case selected from C is guaranteed not to be an element of E. However, it should be noted that along the growth of E, the chance of getting duplicate test cases for ART will increase.

People may wonder whether the improvement of ART over RT is in fact due to there being fewer duplicate test cases with ART than with RT. This has inspired a comparison between RT and ART, both for selection with replacement, and without.

Obviously, when the input domain size (D) is extremely large and only a small portion of inputs is tested, it is very unlikely to have duplicate test cases. Therefore, a finite set of inputs of appropriate size was chosen for this study.

Apart from investigating the impact of test case duplication on the F-measures of testing methods, this study also answers the following questions:

- What actually makes ART outperform RT - the even spread of test cases or smaller amount of duplication of test cases?
- Do the size and type of input domains make ART perform differently?

- Does ART still outperform RT when the program under test contains only one single failure-causing input? Note that there is no clustering of failure-causing inputs in this case, whereas ART is known to outperform RT when failure-causing inputs are clustered together.

Without loss of generality, all input parameters are assumed to accept either integers or real numbers. When any of the input parameters accept real numbers, the input domain is said to be of *real type*. When all input parameters accept integers, the input domain is said to be of *integer type*. Obviously, |D| for the real type input domain is infinitely large, and |D| for the integer type input domain is finite. Again, ART in this chapter refers to the FSCS-ART method, and the size of the candidate set is set to be 10, unless otherwise specified. In this chapter, RT and ART refer to the methods with replacement of test cases, and RT-Wout and ART-Wout refer to the methods without. F_{RT} denotes the F-measure of RT, which is equal to the reciprocal of the failure rate ($\frac{1}{\theta}$).

4.1 Key findings

A series of simulations was conducted to investigate the impact of selection with and without replacement on the F-measures of RT and ART. The impact of the input domain size and type on the F-measures of ART was also studied. Finally, another series of simulations was run to compare the different testing methods in the presence of a very small failure region.

4.1.1 The impact of selection with and without replacement

The following simulations were carried out to compare the F-measures of ART, ART-Wout and RT-Wout with different values for θ. The failure pattern was a single square failure region of edge length l. 2Đ and 3Đ square input domains of integer type were studied. The edge length L of the input domain was set to be either 56, 60 or 64 in 2Đ space; and 8, 12 or 16 in 3Đ space. Obviously, $|D| = L^2$ and $|D| = L^3$ for the 2Đ and 3Đ input domains, respectively. The value of l determined the value of θ. The maximum and minimum values of l were set to be L and 1, respectively. Therefore, the maximum θ was 1 and the minimum θ was $\frac{1}{|D|}$. The

results of the simulations are summarized in Figure 4.1. Note that the data with the largest θ and the smallest θ are presented at the leftmost and rightmost ends of the curve, respectively.

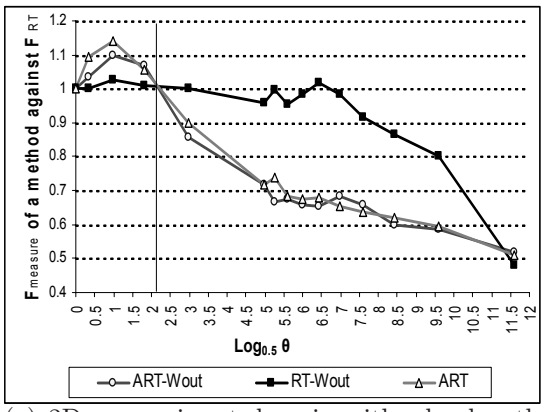
Based on these results, the following conclusions for input domains of various sizes can be drawn:

- ART and ART-Wout have very similar F-measures for all values of θ.
- Irrespective of the size of the input domain, when $l > 5$ for 2Ð space and $l > 3$ for 3Ð space, the F-measures of RT-Wout are very similar to F_{RT}.
- When input domains contained only one failure-causing input, RT-Wout, ART and ART-Wout all have F-measures close to 50% of F_{RT}.

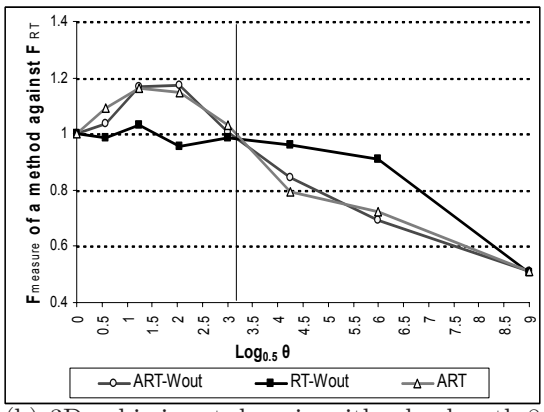
The minimum θ to have similar F-measures for RT-Wout and RT decreases as |D| increases. Furthermore, the number of failure-causing inputs required to get such a minimum θ decreases as the dimensionality increases. Therefore, it is conjectured that RT and RT-Wout will have comparable F-measures for most values of θ because |D| is normally very large in real life applications. Since ART and ART-Wout perform almost identically for all values of θ, and RT and RT-Wout have comparable F-measures for the majority of θ values, it can be concluded that, for most values of θ, selection without replacement does not significantly reduce the F-measures of random testing or adaptive random testing. Clearly, methods without replacement are more expensive than those with replacement, hence it is more cost effective to use RT and ART than RT-Wout and ART-Wout, respectively.

4.1.2 The impact of the input domain size and type

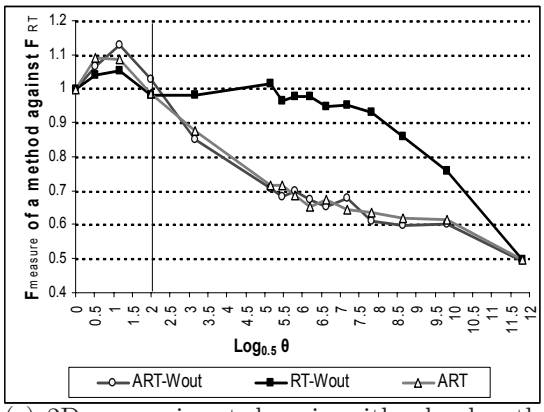
Chapter 3 showed that ART performs well for small values of θ. Except for cases of large θ in high dimensional space, the F-measure for ART decreases as θ decreases. In that study, the input domain size was infinitely large (because the input domain was of real type), therefore, the minimum limiting value of θ was 0, and hence, it was impossible to use simulations to show whether the F-measure of ART would converge to a specific value. In this section, this will be re-investigated. Note that the simulation settings in Section 4.1.1 were designed in such a way that they could



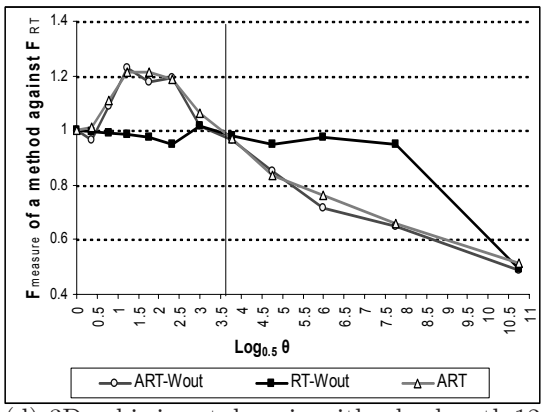
(a) 2Đ square input domain with edge length 56



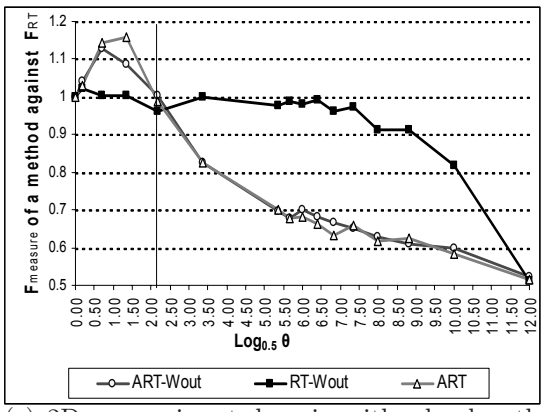
(b) 3Đ cubic input domain with edge length 8



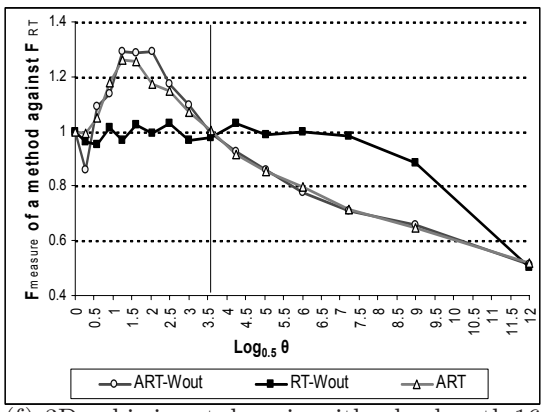
(c) 2Đ square input domain with edge length 60



(d) 3Đ cubic input domain with edge length 12

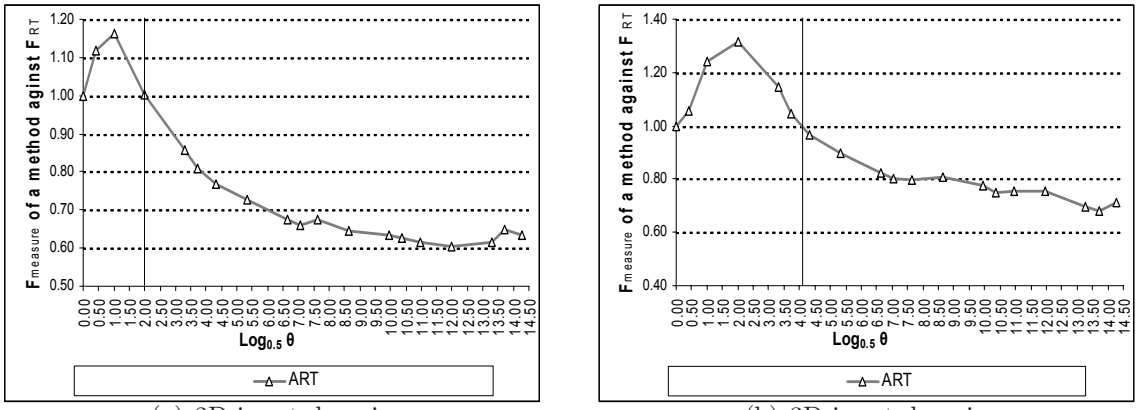


(e) 2Đ square input domain with edge length 64



(f) 3Đ cubic input domain with edge length 16

Figure 4.1: Simulation results for integer type input domains



(a) 2Đ input domain (b) 3Đ input domain

Figure 4.2: Simulation results for real type input domains

be compared with the simulation study in Section 3.1.1. The only difference between them is the input domain type and input domain size.

Figures 4.2 re-summarizes the simulation results for the 2Đ and 3Đ real type input domains (previously reported in Section 3.1.1). The observation from the comparison was that the relationships between θ and the ratio of F-measure of ART to F_{RT} (the ART F-ratio) were almost identical, and the values of θ, at which the ART F-ratio became less than 1, were also similar. Basically, the types of input domains have very little or no effect on the performance of ART. Based on the above observations, it is conjectured that when ART is used in real type input domains, the ART F-ratio will also approach 0.5 as θ approaches 0.

4.1.3 Comparison between testing methods for a small number of failure-causing inputs

A set of simulations was conducted to investigate the F-measures of RT-Wout, ART and ART-Wout for very small values of θ, that is, a very small number m of failure-causing inputs. In these simulations, the failure pattern was not fixed to be square and m was restricted not to exceed 4. All possible patterns based on these m failure-causing inputs are depicted in Figure 4.3. The simulations were conducted in 1Đ to 4Đ square input domains of integer type. All input domains consisted of 4096 elements. Table 4.1 reports the results. Because some patterns are not possible in certain dimensional input domains, Table 4.1 contains some empty cells.

The results show that only for Pattern 1A does RT-Wout have a similar F-

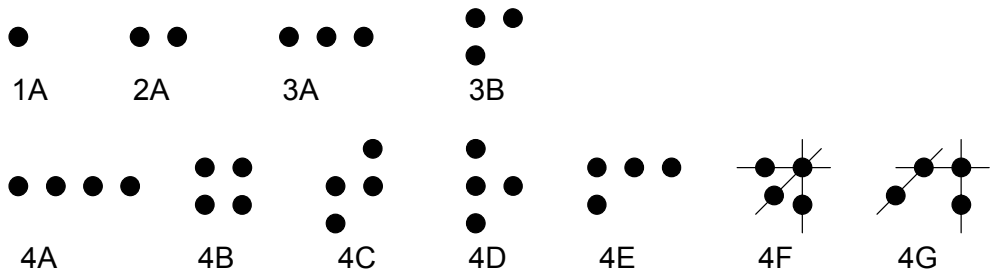


Figure 4.3: Failure patterns composed of at most 4 failure-causing inputs

	F-measure of ART				F-measure of RT-Wout				F-measure of ART-Wout			
	1Đ	**2Đ**	**3Đ**	**4Đ**	**1Đ**	**2Đ**	**3Đ**	**4Đ**	**1Đ**	**2Đ**	**3Đ**	**4Đ**
1A	0.50	0.52	0.52	0.51	0.50	0.51	0.50	0.49	0.47	0.52	0.51	0.53
2A	0.57	0.61	0.64	0.64	0.66	0.67	0.70	0.67	0.53	0.62	0.59	0.64
3A	0.55	0.62	0.68	0.68	0.74	0.79	0.76	0.78	0.53	0.63	0.67	0.67
3B		0.65	0.70	0.69		0.77	0.72	0.75		0.63	0.68	0.68
4A	0.55	0.66	0.70	0.74	0.82	0.80	0.79	0.81	0.54	0.63	0.71	0.71
4B		0.60	0.64	0.73		0.77	0.81	0.83		0.59	0.68	0.68
4C		0.70	0.72	0.73		0.82	0.82	0.80		0.69	0.76	0.74
4D		0.68	0.73	0.73		0.80	0.79	0.83		0.64	0.70	0.75
4E		0.67	0.71	0.71		0.80	0.78	0.76		0.66	0.70	0.71
4F			0.70	0.76			0.81	0.82			0.72	0.72
4G			0.75	0.72			0.83	0.81			0.72	0.73

Table 4.1: The ratio of the F-measure of a testing method to F_{RT}

measure to ART and ART-Wout (about 50% of F_{RT}). When failure patterns are of other types, RT-Wout always has F-measures higher than those of ART and ART-Wout. Note that these results are consistent with those reported in Figure 4.1, where the performance for higher failure rates was investigated.

In fact, it is possible to calculate the expected F-measure of RT-Wout when $m = 1$. Assuming that all inputs have an equal probability of being selected, then, the probability that this single failure-causing input is selected from the input domain is $\frac{1}{|D|}$ (equivalent to θ). Let $P(i)$ be the probability that the failure-causing input will be selected as the test case at the i^{th} trial but not before, therefore,

$P(1) = \frac{1}{|D|}$

$P(2) = \frac{|D|-1}{|D|} \cdot \frac{1}{|D|-1} = \frac{1}{|D|}$

$P(3) = \frac{|D|-1}{|D|} \cdot \frac{|D|-2}{|D|-1} \cdot \frac{1}{|D|-2} = \frac{1}{|D|}$

...

$P(|D|) = \frac{|D|-1}{|D|} \cdot \frac{|D|-2}{|D|-1} \cdot \frac{|D|-3}{|D|-2} \cdot ... \cdot \frac{1}{1} = \frac{1}{|D|}$

Therefore, the average F-measure of RT-Wout is $\sum_{i=1}^{|D|}(i \cdot P(i)) = \sum_{i=1}^{|D|}(i \cdot \frac{1}{|D|}) = \frac{1}{|D|} \cdot \sum_{i=1}^{|D|} i = \frac{1}{|D|} \cdot \left(\frac{1+|D|}{2}\right) \cdot |D| = \frac{1+|D|}{2}$ ($\approx \frac{|D|}{2}$, when $|D|$ is very large).

Since the expected value of F_{RT} is $\frac{1}{\theta}$ which is equal to $|D|$, the F-measure of

RT-Wout is about $\frac{F_{RT}}{2}$. The above theoretical analysis is consistent with the experimental analysis that when $m = 1$, RT-Wout has an F-measure around 50% of F_{RT}.

Table 4.1 shows that, except for $m = 1$, the F-measure of ART is always smaller than the F-measure of RT-Wout. Therefore, the better performance of ART over RT has nothing to do with the duplication of test cases because ART has duplicate test cases but RT-Wout does not.

4.2 Discussion and conclusion

It has long been known that ART outperforms RT when there exist clusters of failure-causing inputs. It had, however, remained unknown whether ART was still superior to RT when there was only one single failure-causing input (that is, no clusters of failure-causing inputs). Although it is too complex to calculate the expected value of the F-measure of ART for the case of $m = 1$, the simulation results (refer to row 1A in Table 4.1) show that the F-measure of ART is about half of F_{RT}, irrespective of the dimensions of the input domains. Note that Chen and Merkel [24] recently found that the F-measure of any testing method cannot be less than 50% of F_{RT} unless the testing method makes use of the information about the location of failure-causing inputs. In this study, it was found by simulations that, when programs contain only one single failure-causing input, the F-measures of RT-Wout, ART-Wout and ART are all about 50% of F_{RT}, but all these methods do not make use of any information about the location of failure-causing inputs.

This study confirms that the input domain size and input domain type have very little or no impact on the F-measure of ART. Furthermore, in general, selection without replacement does not significantly reduce the F-measures of random testing or adaptive random testing. From the viewpoint of cost-effectiveness, it is better to use random testing and adaptive random testing with replacement rather than without replacement.

As mentioned before, ART is expected to generate fewer duplicate test cases than RT for the same size of test suite. Hence, people may wonder whether the improvement of ART over RT is due to the fact that RT yields more duplicate test

cases than ART. In this study, it has been demonstrated that it is the even spread of test cases rather than duplication of test cases that contributes to the significant improvement of ART over RT.

Chapter 5

Test case distributions of adaptive random testing

The principle of ART is to both randomly select and evenly spread test cases. Many ART methods have been proposed, but no comparison has yet been made among these methods in terms of their test case distributions. In this chapter, test case distributions of various ART methods are examined. Besides, the appropriateness of various test case distribution metrics are studied.

As a reminder, θ denotes the failure rate, E denotes the set of executed test cases, D denotes the input domain of N dimensions, and |E| and |D| denote the size of E and D, respectively.

5.1 Various ART methods

In Section 2.2.1, three major approaches to implementing ART were discussed: (*i*) *the best of all*, (*ii*) *by exclusion* and (*iii*) *by partitioning*. Also discussed includes the algorithm of FSCS-ART. In this section, the algorithms of restricted random testing (RRT), random testing by bisection (BPRT) and random testing by random partitioning (RPRT) will be described. Note that RRT belongs to the approach of ART by exclusion, while BPRT and RPRT belong to the the approach of ART by partitioning.

- **RRT** [11]

  In RRT, a circular zone x_i (an *exclusion region*) is drawn around each element

of E, and the next test case is restricted to coming from outside these regions. The size of the exclusion zones is kept the same for each element of E, and set to be $\frac{R \cdot |\mathrm{D}|}{|\mathrm{E}|}$ where R is the control parameter of RRT, called the *target exclusion ratio*. Since exclusion regions may overlap, it is meaningful to set R greater than 100%.

The RRT algorithm is basically as follows. Keep track of what inputs have been tested, and then determine what regions will be excluded before selecting the next test case. Continue selecting inputs randomly from D until one outside all exclusion regions is found, and take it as the next test case. The above process is repeated until the first failure is detected.

It has been found from simulation studies that the larger the value of R, the smaller the F-measure of RRT. Since the current version of RRT is to have one fixed R throughout the whole testing process, when E grows, more and more parts of D will be excluded, and hence getting a test cases outside all exclusion regions becomes more and more difficult. Several investigations have been concluded to investigate the setting of R. These studies were carried out using simulations with certain assumptions on |E|, the shape of input domains, etc. Applying RRT with fixed R to real life cases is not so straightforward.

The RRT used in this thesis is slightly different from the original one. In this algorithm (refer to Figure A.1 in Appendix A for details), R is dynamically adjusted during the testing process from large to small, in order to keep the computation time manageable for obtaining a test case outside all exclusion regions. The initial values of R were set as 100%, 170%, 330% and 640% for the 1Đ, 2Đ, 3Đ and 4Đ square input domains, respectively. As suggested by Chen *et al.* [11] that at these values of R, RRT can perform well for $|\mathrm{E}| \leq 100$.

- **BPRT** [14]

  Let L be the set of all partitions generated in the testing process. An element of L is marked "tested" if there is an executed test case inside that partition. Initially, L = {D}. From L, one untested partition is selected for testing. BPRT keeps track of which elements of L have been tested. When all partitions are tested without revealing any failure, BPRT bisects each element of L to

double the number of partitions, and then updates the "tested" status of all resultant partitions. After the updating, half of the partitions are untested. Repeat the above process until a failure is revealed. Figure A.2 in Appendix A shows the detailed algorithm of BPRT.

- **RPRT** [14]

  Let L be the set of all partitions generated in the testing process. Initially, L = {D}. From L, RPRT selects the "largest partition" l for testing. A test case t is generated from l. If t reveals a failure then testing is halted; otherwise t is used as the intersection centre to divide l into 2^N rectangular partitions. L is then updated by replacing l by its resultant partitions. Repeat the above process until a failure is revealed. Figure A.3 in Appendix A shows the detailed algorithm of RPRT.

5.2 The effectiveness of various ART methods

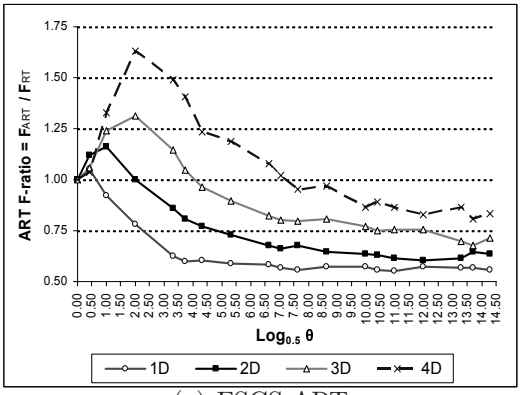
RRT, BPRT and RPRT were studied using the same experimental setting as in Section 3.1.1. The results are summarized in Figures 5.1[1] and 5.2, where Figure 5.1 groups the data by ART method and Figure 5.2 groups the data by dimension. From these results, the following observations can be made:

1. The performance of FSCS-ART, RRT, BPRT and RPRT deteriorates with the increase of N. For the same θ, almost all their F-measures are higher in high dimensional spaces than in low dimensional spaces.

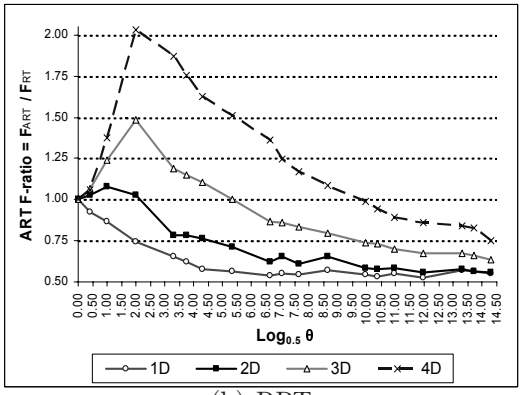
2. FSCS-ART and RRT have very similar performance trends. When θ is large, both FSCS-ART and RRT perform worse than RT and their F-measures increase as θ decreases. When θ drops to a certain value v, their F-measures start to decrease as θ decreases. Note that the value for v is smaller when N is larger.

3. BPRT and RPRT have opposite performance trends compared with FSCS-ART and RRT.

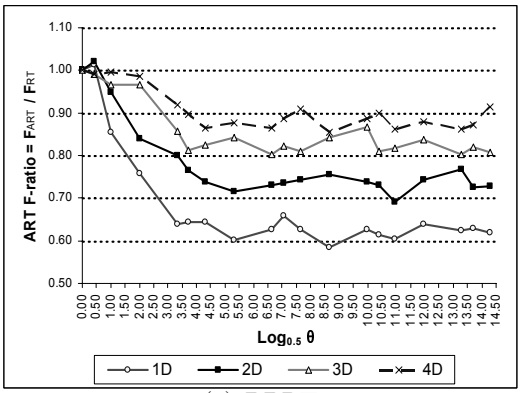
[1]Figure 5.1(a) is identical to Figures 3.1



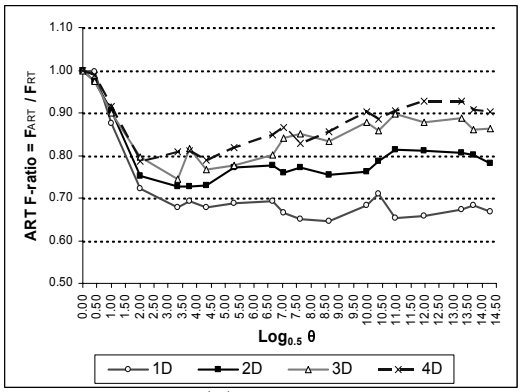
(a) FSCS-ART



(b) RRT

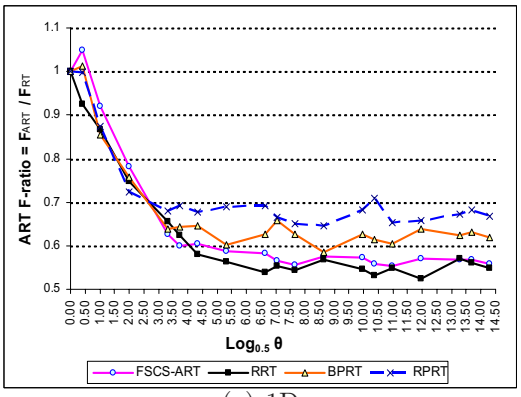


(c) BPRT

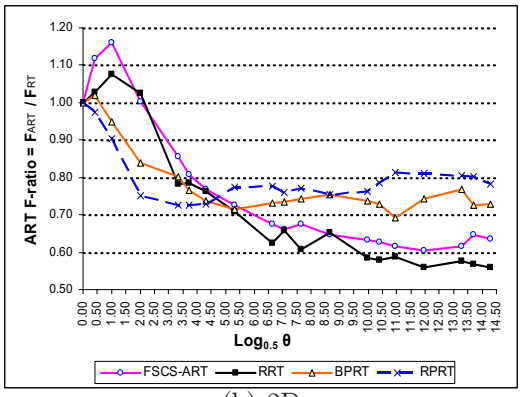


(d) RPRT

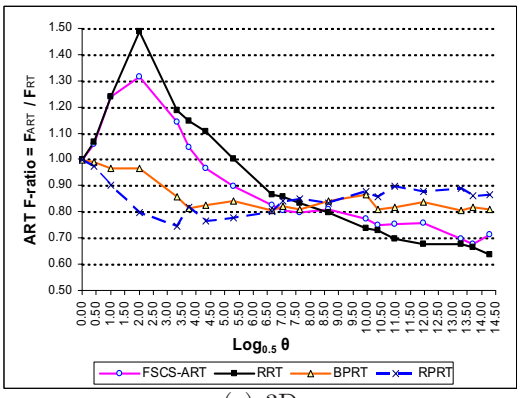
Figure 5.1: The effectiveness of various ART methods



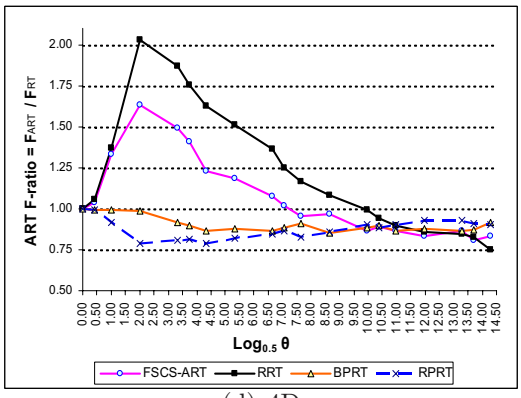
(a) 1Đ



(b) 2Đ



(c) 3Đ



(d) 4Đ

Figure 5.2: The effectiveness of various ART methods in different dimensions

When θ is large, the F-measures of BPRT and RPRT are smaller than those of RT, and decrease as θ decreases. After θ drops to a specific value v, the F-measures of BPRT and RPRT stop decreasing. Note that the value of v increases with the increase of N. When $\theta < v$, the F-measure of BPRT fluctuates slightly around a certain value f_b; while the F-measure of RPRT increases steadily towards a stable value f_r. As shown in Figure 5.1, the performance trends of FSCS-ART and RRT are completely opposite to that of RPRT, and partially opposite to that of BPRT. Note that f_b and f_r grow with N and appear to approach to F_{RT} (the F-measure of RT) when N is very large.

4. There exists a specific failure rate r, which depends on N, such that RRT has higher F-measures than FSCS-ART when $\theta > r$; and RRT has lower F-measures than FSCS-ART when $\theta \leq r$.

5. There exists a specific failure rate r, which depends on N, such that BPRT has higher F-measures than RPRT when $\theta > r$; and BPRT has lower F-measures than RPRT when $\theta \leq r$.

Although FSCS-ART, RRT, BPRT and RPRT all aim at evenly spreading test cases, this study shows that their fault-detection effectiveness vary. The differences are due to their ways of distributing test cases. In the next sections, the distribution of test cases generated by these testing methods will be measured.

5.3 Measurement of test case distribution

For ease of discussion, the following notations are introduced. Suppose p' and p'' are two elements of E. *dist*(p', p'') denotes the distance between p' and p''; and ɲ(p, E\{p}) denotes the nearest neighbour of p in E. Without loss of generality, the range of values for each dimension of D is set to [0, 1). In other words, D = $\{(\mathrm{I}_1, \mathrm{I}_2, ..., \mathrm{I}_N) | 0 \leq \mathrm{I}_i \leq 1, \forall 1 \leq i \leq N\}$, or simply D = $[0, 1)^N$.

5.3.1 Metrics

In this study, four metrics were used to measure the test case distributions (the distribution of E in D). The following outlines the definitions of these four metrics.

- **Discrepancy**

$$M_{Discrepancy} = \max_{i=0...m} \left| \frac{|\mathrm{E}_i|}{|\mathrm{E}|} - \frac{|\mathrm{D}_i|}{|\mathrm{D}|} \right| \tag{5.1}$$

  where D_1, D_2, ...D_m denote m randomly defined subsets of D, with their corresponding sets of test cases being denoted by E_1, E_2, ...E_m, which are subsets of E. Note that m is set to 1000 in this study.

  Intuitively speaking, $M_{Discrepancy}$ indicates whether regions have an equal density of the points. E is considered reasonably equidistributed if $M_{Discrepancy}$ is close to 0.

- **Dispersion**

$$M_{Dispersion} = \max_{i=1...|\mathrm{E}|} dist(e_i, \mathfrak{n}(e_i, \mathrm{E}\backslash\{e_i\})) \tag{5.2}$$

  where $\mathrm{e}_i \in \mathrm{E}$.

  Intuitively speaking, $M_{Dispersion}$ indicates whether any point in E is surrounded by a very large empty spherical region (containing no points other than itself). In other words, $M_{Dispersion}$ indicates how large is the largest empty spherical region in the space. A smaller $M_{Dispersion}$ indicates that E is more equidistributed.

- **The ratio of the number of test cases in the edge of the input domain (E_{edge}) to the number of test cases in the central region of the input domain (E_{centre})**

$$M_{Edge:Centre} = \frac{|\mathrm{E}_{edge}|}{|\mathrm{E}_{centre}|} \tag{5.3}$$

  where E_{edge} and E_{centre} denote two disjoint subsets of E locating in D_{edge} and D_{centre}, respectively; $\mathrm{E} = \mathrm{E}_{edge} \cup \mathrm{E}_{centre}$. Note that $|\mathrm{D}_{centre}| = \frac{|\mathrm{D}|}{2}$ and $\mathrm{D}_{edge} =$

$\mathrm{D}\backslash\mathrm{D}_{centre}$. Therefore, $\mathrm{D}_{centre} = \left[\frac{1}{2} - \sqrt[N]{\frac{|\mathrm{D}|}{2^{N+1}}}, \frac{1}{2} + \sqrt[N]{\frac{|\mathrm{D}|}{2^{N+1}}}\right)^N$.

Clearly, in order for $M_{discrepancy}$ to be small, $M_{Edge:Centre}$ should be close to 1; otherwise, different parts of D have different densities of points.

- **Average distance (abbreviated as "AveDist") among points to their nearest neighbours**

$$M_{AveDist} = \frac{\sum_{i=1...|\mathrm{E}|} dist(e_i, \mathfrak{n}(e_i, \mathrm{E}\backslash\{e_i\}))}{|\mathrm{E}|} \tag{5.4}$$

where $\mathrm{e}_i \in \mathrm{E}$.

It is worthwhile to compare $M_{AveDist}$ and $M_{Dispersion}$ for a given E. Let G denote the set of $dist(\mathrm{e}_i, \mathfrak{n}(\mathrm{e}_i, \mathrm{E}\backslash\{\mathrm{e}_i\}))$ for all elements of E. The value $(M_{AveDist} - M_{Dispersion})$ roughly indicates how far the data in G deviate from the mean. Intuitively speaking, the smaller the value $(M_{Dispersion} - M_{AveDist})$ is, the more evenly spread E is.

Discrepancy and dispersion are two commonly used metrics for measuring sample point equidistribution. More details of these two metrics can be found in Branicky *et al.* [7].

5.3.2 Data collection

The same technique as described in Section 2.2.2 was used to collect a statistically reliable mean value of a test case distribution metric. The experiment procedure is described in Figure 5.3.

The experiments involved 3 parameters: (*i*) the number of points in E (|E|); (*ii*) the testing methods used to generate points; and (*iii*) the dimensionality of D (that is, N).

Here, the test case distributions of RT is also studied. RT in this thesis is assumed to have a uniform distribution of test cases (points, in this context). A "uniform distribution of points" means that all points have an equal chance of being selected. The goal here is to investigate how random points are distributed with respect to the metrics defined in Section 5.3.1.

M_{tcdist} denotes one of the measures given in Formulae 5.1-5.4.
{
1. Input n and set G = { }.
2. Generate n test cases (called test suite E), using the chosen testing method.
3. Calculate M_{tcdist} for the test suite generated in Step 2. Add M_{tcdist} in G. (The sample size is the number of elements in G, i.e. |G|.)
4. Calculate the mean $\bar{\chi}$ and standard deviation s for G.
5. Calculate S according to Formula 2.2
6. If |G| $\geq$ S, then report the average $M_{tcdist} = \frac{\sum_{i=1}^{|G|} g_i}{|G|}$ where $g_i \in$ G; otherwise, go to Step 2.

}

Figure 5.3: Procedure to obtain the mean value of the test case distribution metric

5.4 Analysis of test case distributions

The results of comparing testing methods using $M_{Edge:Centre}$ are summarized in Figures 5.4 and 5.5. Figure 5.4 groups the data by testing method, and Figure 5.5 groups the data by dimensionality. From these data, the following observations can be made:

- When $N = 1$, $M_{Edge:Centre}$ for all ART methods under study is close to 1. In other words, there is no bias towards generating test cases in either D_{edge} or D_{centre} (refer to Figure 5.4(a)).

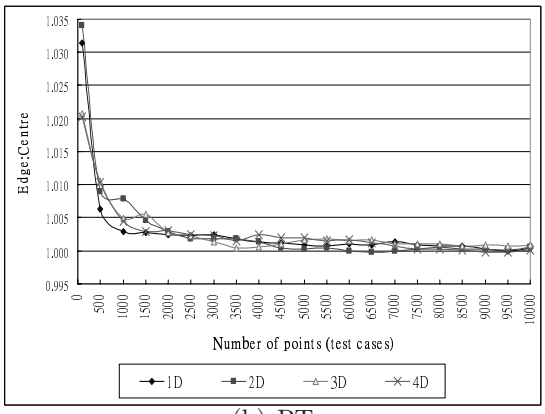
- When $N > 1$, FSCS-ART and RRT tend to generate more test cases in D_{edge} than in D_{centre} (or simply, FSCS-ART and RRT have *edge bias*)
  Evidence: Figure 5.4(a) shows that when $N > 1$, all $M_{Edge:Centre}$ for FSCS-ART and RRT are larger than 1.

- When $N > 1$, RPRT allocates more test cases in D_{centre} than in D_{edge} (or simply, RPRT has *centre bias*)
  Evidence: Figure 5.4(a) shows that when $N > 1$ all $M_{Edge:Centre}$ for RPRT are smaller than 1.

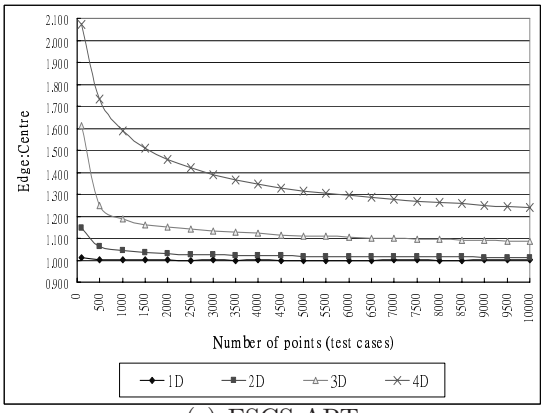
- The edge bias (for FSCS-ART and RRT) and centre bias (for RPRT) increase as N increases.
  Evidence: Figures 5.4(c) and 5.4(d) show that $M_{Edge:Centre}$ for FSCS-ART and

RT	1D	2D	3D	4D
Max	1.0315	1.0341	1.0208	1.0202
Min	1.0000	0.9998	1.0005	0.9998
Max-Min	0.0315	0.0343	0.0203	0.0204
FSCS-ART	1D	2D	3D	4D
Max	1.0103	1.1442	1.6105	2.0733
Min	0.9997	1.0132	1.0863	1.2409
Max-Min	0.0106	0.1310	0.5241	0.8323
RRT	1D	2D	3D	4D
Max	1.0114	1.2103	2.1943	4.7346
Min	0.9995	1.0201	1.1358	1.3444
Max-Min	0.0119	0.1901	1.0585	3.3902
BPRT	1D	2D	3D	4D
Max	1.0085	1.0456	1.0482	1.0250
Min	0.9991	0.9975	0.9963	0.9979
Max-Min	0.0094	0.0480	0.0519	0.0271
RPRT	1D	2D	3D	4D
Max	1.0001	0.9995	0.9923	0.9780
Min	0.9812	0.9522	0.9236	0.9012
Max-Min	0.0190	0.0473	0.0688	0.0768

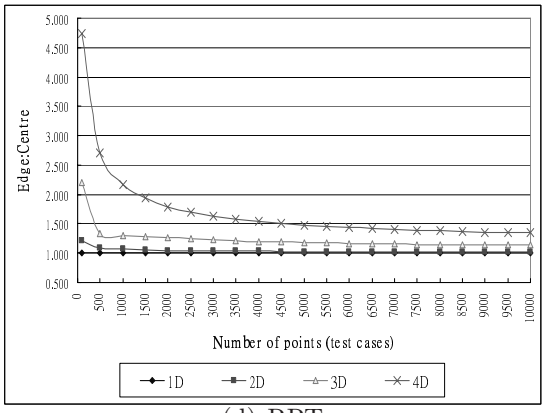
(a) Range of $M_{Edge:Centre}$ for each testing method and dimension where $|E| \leq 10000$



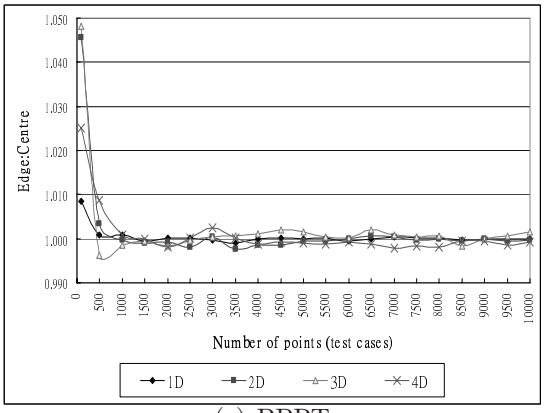
(b) RT



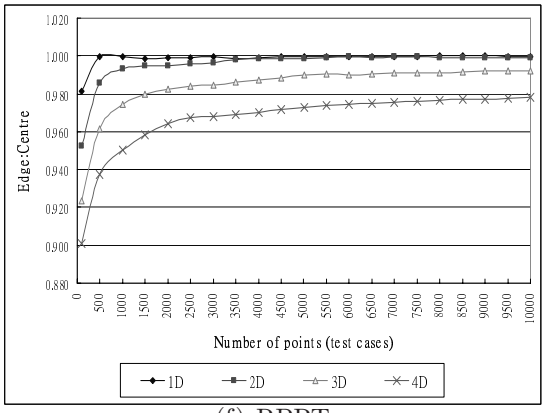
(c) FSCS-ART



(d) RRT

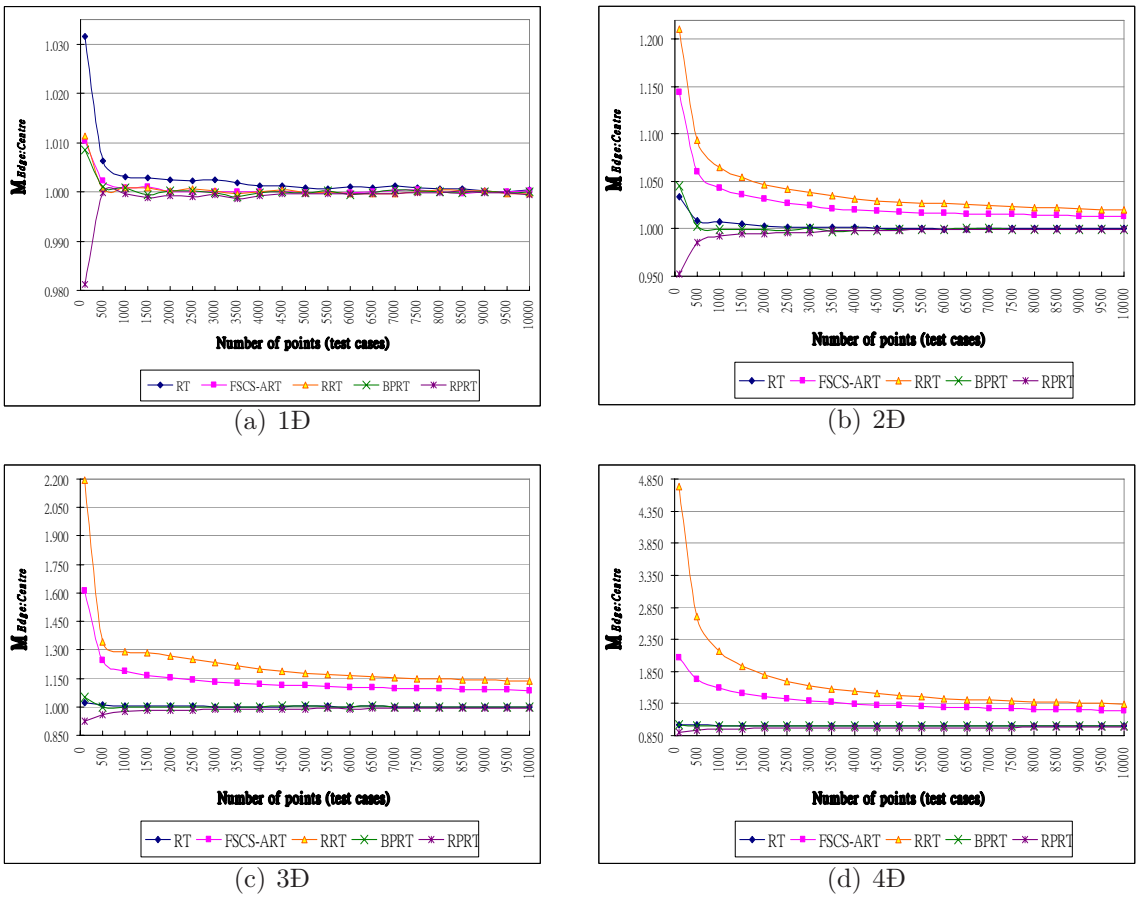


(e) BPRT



(f) RPRT

Figure 5.4: $M_{Edge:Centre}$ with respect to the testing method



(a) 1Đ (b) 2Đ

(c) 3Đ (d) 4Đ

Figure 5.5: $M_{Edge:Centre}$ with respect to the dimension

RRT increase as N increases. Figure 5.4(f) shows that $M_{Edge:Centre}$ for RPRT decreases as N increases.

- RT and BPRT have neither edge bias nor centre bias, for all N.

  Evidence: Figures 5.4(b) and 5.4(e) show that $M_{Edge:Centre}$ for BPRT and RT is very close to 1.

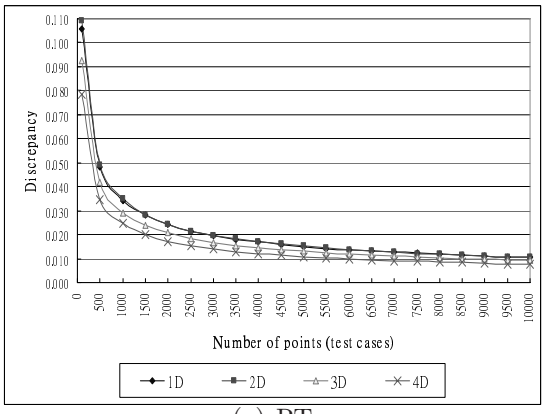
- The edge bias is stronger with RRT than with FSCS-ART.

  Evidence: Figure 5.5 shows that when $N \geq 2$, RRT always has a higher $M_{Edge:Centre}$ than FSCS-ART.

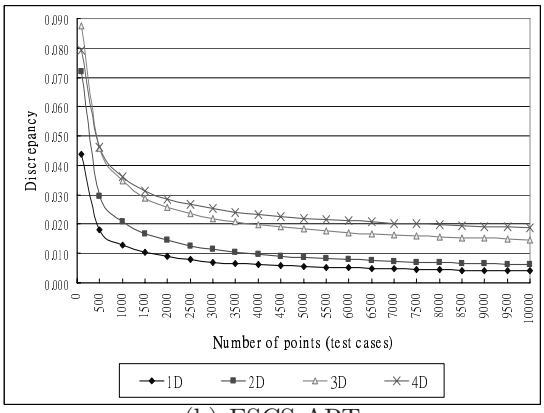
- The centre bias of RPRT is much less significant than the edge bias of FSCS-ART or RRT.

  Evidence: Figure 5.5 shows that when $N \geq 2$, $M_{Edge:Centre}$ for RPRT is significantly closer to 1 than that for RRT or FSCS-ART.

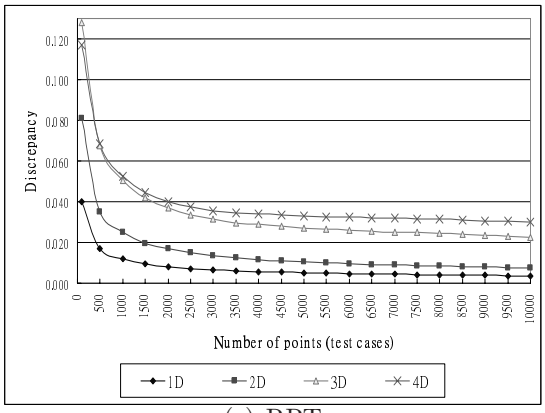
The results of comparing testing methods using $M_{Discrepancy}$ are summarized in Figures 5.6 and 5.7. Figure 5.6 groups the data by the testing methods. Figure 5.7 groups the data by the dimensions. Based on these data, the following observations can be made.



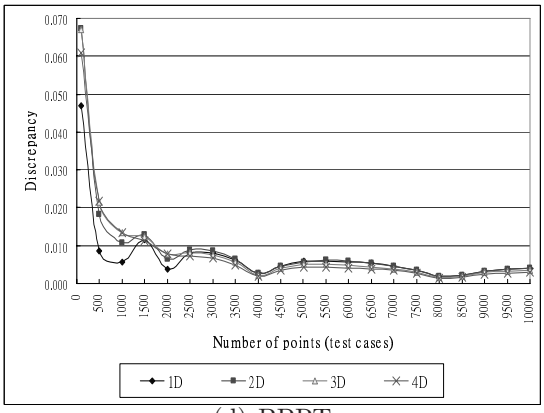
(a) RT



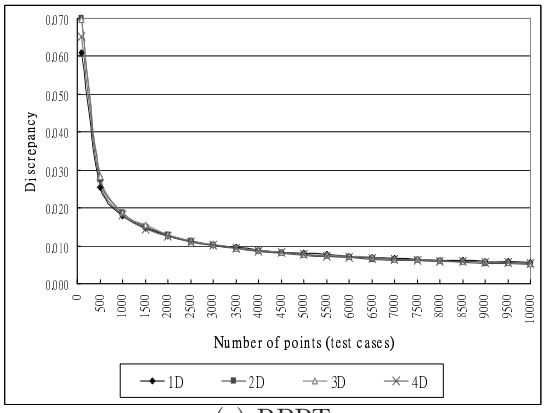
(b) FSCS-ART



(c) RRT



(d) BPRT

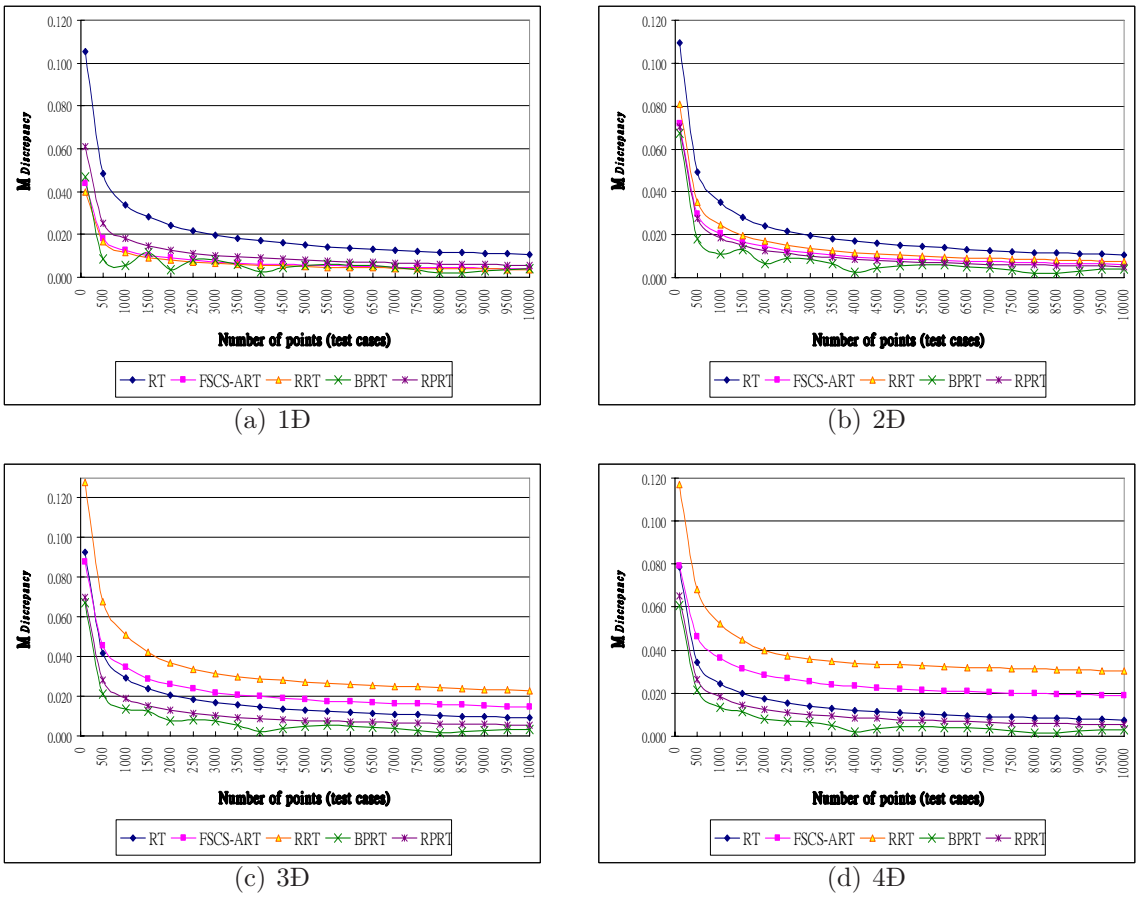


(e) RPRT

Figure 5.6: $M_{Discrepancy}$ with respect to the testing method

- The impact of N on $M_{Discrepancy}$ of RT, FSCS-ART and RRT is different.

  Evidence: As N increases, $M_{Discrepancy}$ for RT decreases (Figure 5.6(a)), but



(a) 1Đ (b) 2Đ

(c) 3Đ (d) 4Đ

Figure 5.7: $M_{Discrepancy}$ with respect to the dimension

for FSCS-ART and RRT, it increases (Figures 5.6(b) and 5.6(c)). The change in $M_{Discrepancy}$ from low dimension to high dimension, however, is negligible for RT when compared with the change in $M_{Discrepancy}$ for FSCS-ART and RRT.

- Figures 5.6(d) and 5.6(e) show that $M_{Discrepancy}$ for RPRT and BPRT are independent of the dimensions under study.
- Figure 5.7 shows that in general BPRT has the lowest $M_{Discrepancy}$ for all N.
- Figure 5.7(a) shows that when $N = 1$, FSCS-ART, RRT and BPRT have almost identical values for $M_{Discrepancy}$.

Clearly, measuring the density of points in two partitions (D_{Edge} and D_{Centre}) is only part of the measuring by $M_{Discrepancy}$ (which measures the density of points in 1000 randomly defined partitions of D). Hence, values of $M_{Edge:Centre}$ close to 1 do not imply a low $M_{Discrepancy}$, but a low $M_{Discrepancy}$ does imply that $M_{Edge:Centre}$

will be close to 1. This explains why the value of $M_{Edge:Centre}$ for RT is close to 1 for all N, but its $M_{Discrepancy}$ is not the lowest. In other words, although RT is neither centre nor edge biased, its generated points are not as evenly spread as those of BPRT.

The results of comparing testing methods using $M_{Dispersion}$ are summarized in Figure 5.8. From these data, the following observations can be made.

- For $N = 1$, Figure 5.8(a) shows that all ART methods have smaller $M_{Dispersion}$ values than RT.
- For $N > 1$, Figure 5.8 shows that normally, RRT has the lowest $M_{Dispersion}$ values, followed in ascending order by FSCS-ART, BPRT, RPRT and then RT.

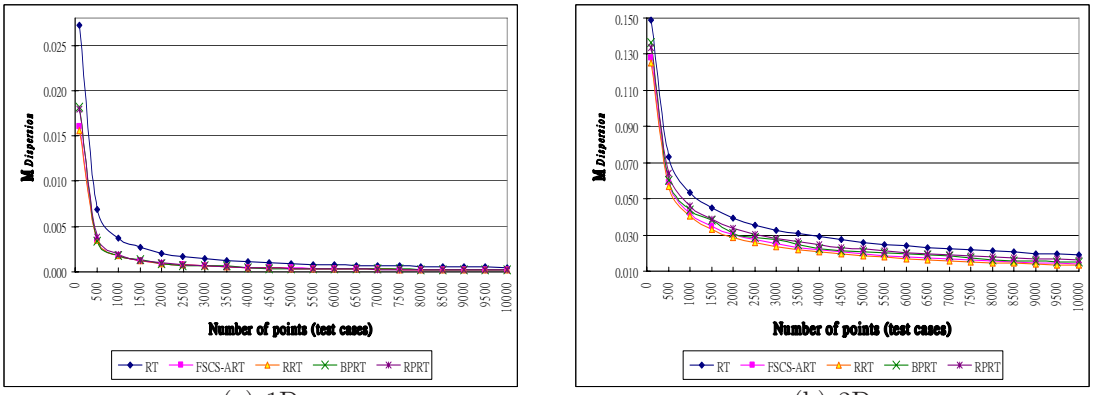
The results of comparing testing methods using $M_{AveDist}$ are summarized in Figure 5.9. The data show that RRT always has the highest $M_{AveDist}$, followed in descending order by FSCS-ART, BPRT, RPRT and then RT, for all dimensions.

Note that among those 5 testing methods studied, the methods with lower $M_{Dispersion}$ have larger $M_{AveDist}$. Clearly, RRT has the smallest ($M_{Dispersion} - M_{AveDist}$), followed in ascending order by FSCS-ART, BPRT, RPRT and RT. As mentioned in Section 5.3.1, the smaller the value ($M_{Dispersion} - M_{AveDist}$) is, the more even spread E is. From this perspective, RRT is considered to have the most even spread of test cases, followed by FSCS-ART, BPRT, RPRT and then RT.

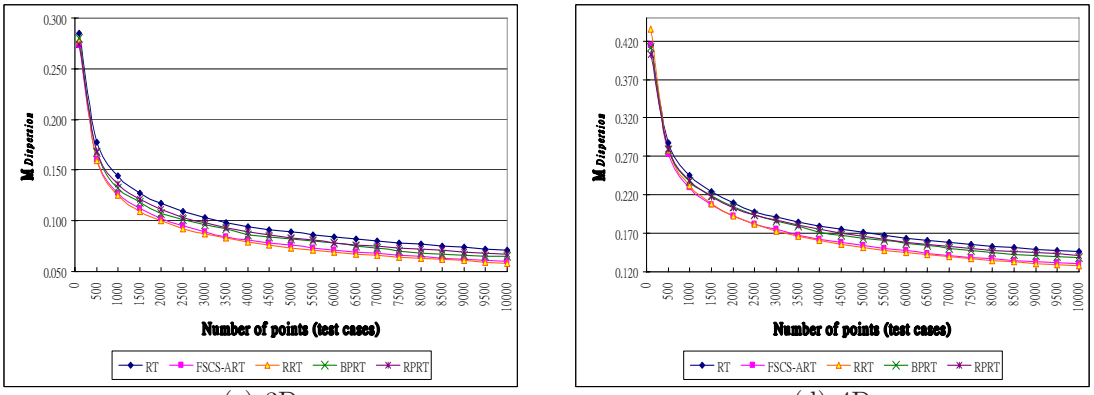
5.5 Discussion and conclusion

The concept of an even spread of test cases is simple but vague. In this chapter, several metrics were used to measure the test case distribution of ART. The relevance and appropriateness of these metrics were investigated.

For ease of discussion, in Table 5.1, the testing methods are ranked according to their test case distribution metrics collected in Section 5.4. Different metrics have different definitions about an even spread of test cases. For the same metric, the method which most satisfies the definition is ranked 1, and the one which least satisfies the definition is ranked 5. When two methods satisfy the definition to more

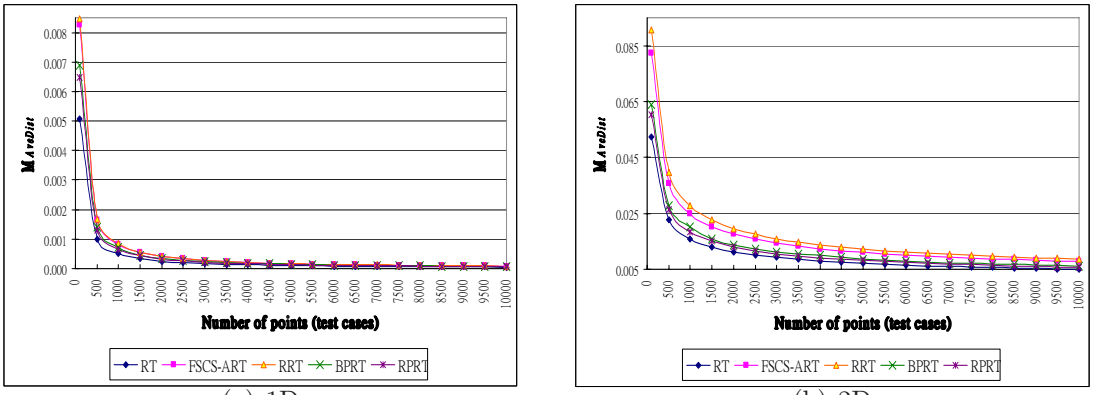


(a) 1Đ (b) 2Đ

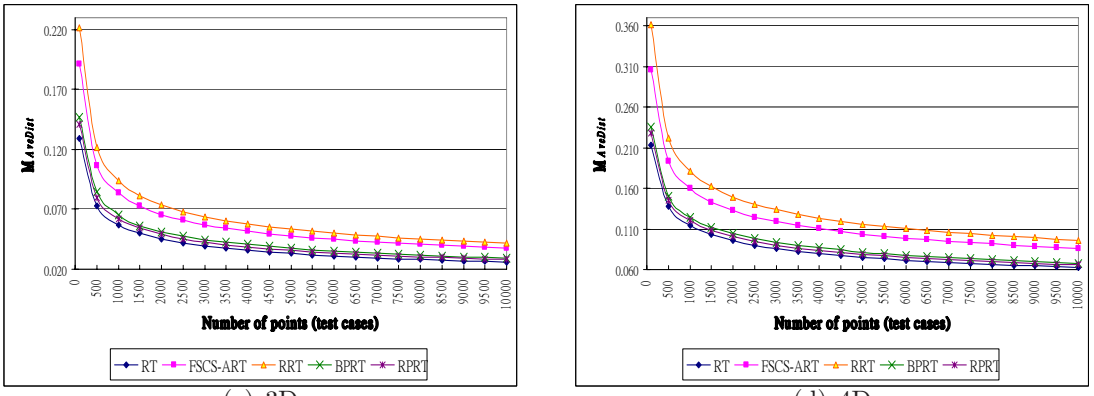


(c) 3Đ (d) 4Đ

Figure 5.8: $M_{Dispersion}$ with respect to various dimensions



(a) 1Đ (b) 2Đ



(c) 3Đ (d) 4Đ

Figure 5.9: $M_{AveDist}$ with respect to various dimensions

or less the same degree, they are given the same ranking. Note that ($M_{Dispersion} - M_{AveDist}$) is considered a separate metric in Table 5.1.

	$M_{Edge:Centre}$				$M_{Discrepancy}$				$M_{Dispersion}$		$M_{AveDist}$	$M_{Dispersion} - M_{AveDist}$
Definition	The closer to 1 is better				The closer to 0 is better				The smaller is better		The larger is better	The smaller is better
Dimension	1Đ	2Đ	3Đ	4Đ	1Đ	2Đ	3Đ	4Đ	1Đ	2-4Đ	1-4Đ	1-4Đ
RRT	1	5§	5§	5§	2	4	5	5	1	1	1	1
FSCS-ART	1	4§	4§	4§	2	3	4	4	1	2	2	2
BPRT	1	1	1	1	1	1	1	1	1	3	3	3
RPRT	1	3†	3†	3†	4	2	2	2	1	4	4	4
RT	1	1	1	1	5	5	3	3	5	5	5	5

† The $M_{Edge:Centre}$ is smaller than 1, so the testing method has a centre bias.
§ The $M_{Edge:Centre}$ is larger than 1, so the testing method has an edge bias.

Table 5.1: Testing methods ranked according to test case distribution metrics

For the 1Đ case, it has been observed that RRT has the best performance, followed by FSCS-ART, BPRT, RPRT and then RT (the same order as listed in Table 5.1). However, when looking at the 2Đ, 3Đ and 4Đ cases, the same performance ordering is observed for small values of θ, but not for large values of θ.

Basically, a testing method with the best performance for low failure rates becomes the worst performer for high failure rates, and a method with the worst performance for low failure rates becomes the best performer for high failure rates. This observation suggests that there are some factors (unrelated to how evenly a method spreads test cases) influencing the performance of ART at high failure rates. It will be shown in Chapter 8 that N and θ have an strong impact on the performance of ART. Only when θ is small enough, the performance of ART will depend only on how evenly spread the generated test cases are. To sensibly match the test case distribution to the performance of ART methods, without being influenced by external factors, the rest of the discussion will assume small failure rates.

Table 5.1 shows that the $M_{AveDist}$ and ($M_{Dispersion} - M_{AveDist}$) metrics indicate that RRT has the most even spread of test cases, followed by FSCS-ART, BPRT and RT. In other words, the rankings according to these two metrics are consistent with the ranking according to the F-measures.

Basically, how test cases are distributed by a testing method depends on its algorithm. FSCS-ART and RRT have an edge bias because they physically spread test cases apart in order to avoid the clustering. Since the number of corners and edges of the input domain grows exponentially with the increase of N, the edge bias

of FSCS-ART and RRT becomes more serious with the growth of N.

Interestingly, among all ART methods under study, the one with the highest $M_{Edge:Centre}$ (that is, RRT) has the largest $M_{AveDist}$ and the lowest $M_{Dispersion}$ while the one with the lowest $M_{Edge:Centre}$ (that is, RPRT) has the lowest $M_{AveDist}$ and the largest $M_{Dispersion}$. In other words, a high $M_{Edge:Centre}$ implies a high $M_{AveDist}$ and a low $M_{Dispersion}$. As discussed before, the smaller the value $(M_{Dispersion} - M_{AveDist})$ is, the more evenly spread E is. Hence, it can be concluded that pushing test cases away (so that $M_{Edge:Centre} > 1$) is not a bad approach to evenly spreading test cases, but nor is it the best approach because its edge bias is not small. In Chapter 8, some work on the enhancement of FSCS-ART to address this issue will be examined.

Finally, it should be pointed out that even though $M_{Discrepancy}$ and $M_{Dispersion}$ are two commonly used metrics for measuring sample point equidistribution, in this study, $M_{Dispersion}$ appears to be more appropriate than $M_{Discrepancy}$, particularly when it is used conjunction with another metric $M_{AveDist}$.

Chapter 6

Application of adaptive random testing to nonnumeric programs

ART has been successfully applied to numeric programs, but not yet to nonnumeric programs. The work of this chapter is to extend the application of ART to nonnumeric programs. A case study and series of experiments were conducted to investigate the fault-detection effectiveness of ART for nonnumeric programs.

6.1 The meaning of evenly spreading test cases in nonnumeric programs

Section 2.1 shows that failure-causing inputs cluster together to form certain patterns. When the failure pattern consists of a few compact and contiguous regions, it has been found that keeping test cases further apart (in other words, evenly spreading test cases) can enhance the fault-detection effectiveness of RT. This idea forms the basis of ART.

When attempting to apply ART to nonnumeric programs, it is important to know what it is meant by an even spread of test cases in these applications. Unlike numeric programs, the input domain of nonnumeric programs cannot be represented like a Cartesian space. It is difficult to visualize how nonnumeric inputs are placed in the input domain and to understand whether failure-causing inputs of nonnumeric programs cluster together or not. In the context of software testing, inputs are likely to have the same failure behaviour (revealing failures or no failures) if they trigger

the same functionality and computation. In other words, failure-causing inputs are *similar* to each other in terms of their associated functionality and computations. Intuitively speaking, ensuring that successive test cases are different from each other in terms of their associated functionality and computations should detect failures more effectively.

The above finding leads to a better understanding about contiguous failure regions in numeric programs. Obviously, many numeric programs implement continuous functions or nearly continuous function. Hence, numerically adjacent inputs (that is, neighbouring inputs in the Cartesian space) are likely to trigger similar functionality and computations, and hence similar failure behaviour. This explains why failure-causing inputs are physically next to each other and hence form contiguous regions in the input domains of numeric programs. Although ART implementations in numeric programs aim to keep test cases physically apart, they effectively enforce different functionality or computations to be tested. This study implies that the actual meaning of evenly spreading test cases in the context of software testing is to make the successive tests as different as possible in terms of functionality and computations tested.

Having identified the rationale for why ART has been so successful for numeric programs, it is now possible to propose an innovative approach to applying ART to nonnumeric programs.

There are two implementation issues in this study. One is related to the random selection of test cases. The other is related to the even spread of test cases. Since RT has been popularly used in many real-life applications, there exist many efficient algorithms [68, 60] to generate random inputs for nonnumeric programs. Hence, the outstanding issue is how test cases can be evenly spread in the input domains of nonnumeric programs. Obviously, this task involves the tracking of previously executed test cases (or tested inputs), and requires an appropriate *dissimilarity metric* to decide which untested input is the most dissimilar to the previously tested inputs, and hence should be used as the next test case.

Note that in numeric programs, keeping test cases apart naturally involves some kind of *distance measurement between inputs*. This is why all existing implementations of ART involve the Euclidean distance or other similar metric in its test

case selection. Obviously, the distance metric is not applicable in nonnumeric programs. As explained above, it is necessary to design an appropriate dissimilarity metric which can capture the difference between inputs in terms of their associated functionality and computations. This task requires some analysis of the program specifications and source code to understand how inputs are used by the software under test.

6.2 A case study of the International Postage Calculation System

In this section, a case study is conducted to show the feasibility of application of ART to nonnumeric programs. The system chosen for study is called the International Postage Calculation System (IPCS), which calculates the postage for the following international postal services.

Service 1. [Parcel] delivered by [Air]
Service 2. [Parcel] delivered by [Economy]
Service 3. [Parcel] delivered by [Sea]
Service 4. [Parcel] delivered by [EMS document]
Service 5. [Parcel] delivered by [EMS merchandise]
Service 6. [Parcel] delivered by [EMS prepaid]
Service 7. [Parcel] delivered by [Air prepaid]
Service 8. [Letter] delivered by [Air]
Service 9. [Letter] delivered by [Air prepaid]
Service 10. [Letter] delivered by [Registered]
Service 11. [Letter] delivered by [Express prepaid]
Service 12. [Letter] delivered by [EMS prepaid]
Service 13. [Postcard] delivered by [Prepaid]
Service 14. [Postcard] delivered by [Non-prepaid]
Service 15. [Greeting card] delivered by [Air]
Service 16. [Aerogramme] delivered by [Air]

Clearly, handling these various services requires various data and involves different computations. The IPCS has 16 distinct functionality, each of which is denoted

by IPCS$_i$, $1 \leq i \leq 16$. The detailed IPCS specification is given in Appendix B.

Name	**(Abbreviation)**	**Meaning**
GType	(GDT)	Type of goods to be delivered
DSystem	(DSM)	Delivery system
Country	(CTR)	Country to which the good is delivered
Weight	(WGT)	Weight of goods
Insured	(INS)	Goods insured (true or false)
Value	(VAL)	Insured value
Confirm	(CFR)	To receive confirmation? (true or false)
Quantity	(QTY)	Number of goods to be delivered
Month	(MTH)	In which month the goods is delivered

Table 6.1: A complete set of input parameters to IPCS

All *input parameters* of IPCS are summarized in Table 6.1. Note that only certain parameters are relevant for each IPCS$_i$. For example, IPCS$_{16}$ (function for aerogramme delivered by air) requires only the 'GType', 'DSystem', 'Weight' and 'Quantity' parameters. Table 6.2 summarizes the relevant parameters of each IPCS$_i$ ($\surd$ indicates relevance).

Table 6.3 outlines the input domain of the IPCS. It details what values each input parameter can take for each IPCS$_i$. Table 6.3 apparently suggests that there should be 16 frames of inputs. However, there are rules on constructing valid and meaningful inputs. Therefore, it turns out that there are 24 frames of inputs, which give rise to 24 mutually exclusive subsets of the input domain. For simplicity, it is assumed that all numeric data are of integer type so as to have a manageable input domain size, as will be explained later. Table 6.3 also contains the input domain and subdomain sizes.

The following shows the procedure for random selection of test cases for the IPCS. Since all subdomains are disjoint, selection of subdomains for testing is according to the probability that is defined as the ratio of the subdomain size to the entire input domain size. Effectively, selecting test cases according to this procedure implements the assumption that all inputs have an equal chance of being selected as a test case.

$IPCS_i$	GDT	DSM	CTR	WGT	INS	VAL	CFR	QTY	MTH
$i = 1$	√	√	√	√				√	
$i = 2$	√	√	√	√				√	
$i = 3$	√	√	√	√				√	
$i = 4$	√	√	√	√	√	√	√	√	
$i = 5$	√	√	√	√	√	√	√	√	
$i = 6$	√	√	√	√	√	√	√	√	
$i = 7$	√	√	√	√				√	
$i = 8$	√	√	√	√				√	
$i = 9$	√	√		√				√	
$i = 10$	√	√		√			√	√	
$i = 11$	√	√		√				√	
$i = 12$	√	√	√	√	√	√	√	√	
$i = 13$	√	√		√				√	
$i = 14$	√	√		√				√	
$i = 15$	√	√		√				√	√
$i = 16$	√	√		√				√	

Table 6.2: The relevance of each input parameter to each $IPCS_i$

6.3 ART method for nonnumeric programs

As mentioned in Section 2.2.1, there exist many implementations of ART for numeric programs. FSCS-ART selects the best candidate for testing among k randomly generated inputs. A candidate is considered the best if it has the maximum shortest distance away from all the executed test cases. Through this selection process, FSCS-ART is able to keep test cases apart from each other, in other words, to keep them as dissimilar as possible.

It is found that FSCS-ART can easily be modified to deal with nonnumeric programs. The part which needs modifications is the metric used to measure how far apart the inputs are, the *dissimilarity metric*.

It is possible to generalize parts of the FSCS-ART procedure (that is, parts related to selecting the best candidate) to cater for nonnumeric application. First, generate k random inputs to construct the candidate set (C). For each element c of C, compare the dissimilarity between c and each element of the executed set (E). In total, there will be |E| number of comparisons, which will return |E| values of the dissimilarity metric (or simply *dissimilarity measures*) for c. The smallest value among these values is assigned to c as its *fitness value*. Finally, the element of C with the highest fitness value is selected as the next test case.

Unless otherwise specified, ART in the rest of the discussion refers to the above

$IPCS_i$	GDT	DSM	CTR	WGT	INS	VAL	CFR	QTY	MTH	Sub-domain size	No. of C.C.
$i = 1$	Parcel	Air	191	(0,500]				=1		95500	36
$i = 2$	Parcel	Economic	191	(0,500]				=1		95500	36
$i = 3$	Parcel	Sea	191	(0,500]				=1		95500	36
$i = 4$	Parcel	EMS document	191	(0,500]	False	=0	False	=1		95500	36
					True	(0,100]	True	=1		9550000	36
					True	(0,100]	False	=1		9550000	36
$i = 5$	Parcel	EMS merchandise	191	(0,500]	False	=0	False	=1		95500	36
					True	(0,100]	True	=1		9550000	36
					True	(0,100]	False	=1		9550000	36
$i = 6$	Parcel	EMS prepaid	191	(0,250]	False	=0	False	(0,10]		477500	8
					True	(0,100]	True	(0,10]		47750000	8
					True	(0,100]	False	(0,10]		47750000	8
$i = 7$	Parcel	Air prepaid	191	(0,100]				(0,10]		191000	8
$i = 8$	Letter	Air	191	(0,50]				(0,100]		955000	8
$i = 9$	Letter	Air prepaid		(0,25]				(0,100]		2500	9
$i = 10$	Letter	Registered		(0,50]			True/ False	(0,100]		10000	12
$i = 11$	Letter	Express prepaid		(0,50]				(0,100]		5000	6
$i = 12$	Letter	EMS prepaid	191	(0,50]	False	=0	False	(0,100]		955000	4
					True	(0,100]	True	(0,100]		95500000	4
					True	(0,100]	False	(0,100]		95500000	4
$i = 13$	Post card	Prepaid		(0,30]				(0,1000]		30000	1
$i = 14$	Post card	Non-prepaid		(0,30]				(0,1000]		30000	1
$i = 15$	Greeting card	Air		(0,30]				(0,1000]	Jan - Dec	360000	2
$i = 16$	Aerogramme	Air		(0,30]				(0,1000]		30000	1
Input domain size = 328223500											
Total C.C. (combinations of choices) = 408											

The parameter 'CTR' accepts 191 countries. The country list is given in Table B.14 in Appendix B.
For a numeric data x, (a,b] and =a mean that a $<$ x $\leq$ b and x = a, respectively.

Table 6.3: Input domain of IPCS

FSCS-ART implementation for nonnumeric programs. Following the previous experiments for numeric programs, k is set to 10, unless otherwise specified.

For nonnumeric inputs, a dissimilarity metric should consist of two components: (i) an input classification scheme capturing the relationship between inputs and their associated functionality or computations, and (ii) a formula for measuring the dissimilarity between inputs.

6.3.1 Classification of inputs

Because it is the specifications that describe the system functionality and the role of the inputs, analyzing these specifications helps in the derivation of an input classification scheme. The *category partition method* (CPM) is a "systematic, specification-based method that uses partitioning to generate functional tests for complex software systems" [54]. In CPM, testers analyze the specification and construct *categories* and *choices* to form test frames from which test cases are generated. There exist many guidelines and tools to assist in the construction of categories and partitions [25, 26, 27].

It should be clearly noted that, in this work, specifications were analyzed and categories and choices were constructed in order to capture the relationship between inputs and their associated functionality, but not to construct test cases (because test cases are randomly selected and evenly spread). Furthermore, the system code is not used to derive the input classification scheme in this case study of applying ART to IPCS, hence the rest of this discussion does not refer to "associated program computation".

Table 6.4 summarizes the definitions of categories and choices for the IPCS, as well as the values associated with each choice. There are 408 possible valid combinations of choices in total. Table 6.5 summarizes the relationship between input parameters and categories for each $IPCS_i$.

Table 6.4: Definition of categories and choices for IPCS

ID	**Category**	**ID**	**Choice**	**Values**
A_1	Goods' type	O_1^1	Parcel	Parcel
		O_2^1	Letter	Letter
		O_3^1	Postcard	Postcard
		O_4^1	Greeting card	Greeting card
		O_5^1	Aerogramme	Aerogramme
A_2	Delivery system for parcel	O_1^2	Air	Air
		O_2^2	Economy	Economy
		O_3^2	Sea	Sea
		O_4^2	EMS document	EMS document
		O_5^2	EMS merchandise	EMS merchandise
		O_6^2	EMS prepaid	EMS prepaid
		O_7^2	Air prepaid	Air prepaid
A_3	Delivery system for letter	O_1^3	Air	Air normal
		O_2^3	Air prepaid	Air prepaid
		O_3^3	Registered	Registered
		O_4^3	Express prepaid	Express prepaid
		O_5^3	EMS prepaid	EMS prepaid
A_4	Delivery system	O_1^4	Prepaid	Prepaid

Continued on next page

Table 6.4 – continued from previous page

ID	Category	ID	Choice	Values
	for postcard	O_2^4	Non-Prepaid	Non-Prepaid
A_5	Other delivery system	O_1^5	Air	Air
A_6	Country category 1	O_1^6	Zone A	1 country
		O_2^6	Zone B	34 countries
		O_3^6	Zone C	23 countries
		O_4^6	Zone D	133 countries
A_7	Country category 2	O_1^7	Zone AP	35 countries
		O_2^7	Zone RW	156 countries
A_8	Weight category 1	O_1^8	Choice 1	> 0 and ≤ 25[1]
		O_2^8	Choice 2	$> 25$and ≤ 50[1]
		O_3^8	Choice 3	> 50 and ≤ 75[1]
		O_4^8	Choice 4	> 75 and ≤ 100[1]
		O_5^8	Choice 5	> 100 and ≤ 125[1]
		O_6^8	Choice 6	> 125 and ≤ 150[1]
		O_7^8	Choice 7	> 150 and ≤ 175[1]
		O_8^8	Choice 8	> 175 and ≤ 200[1]
		O_9^8	Choice 9	> 200 and ≤ 500[1]
A_9	Weight category 2	O_1^9	Choice 1	> 0 and ≤ 100
		O_2^9	Choice 2	>100 and ≤ 250
A_{10}	Weight category 3	O_1^{10}	Choice 1	> 0 and ≤ 50
		O_2^{10}	Choice 2	> 50 and ≤ 100
A_{11}	Weight category 4	O_1^{11}	Choice 1	> 0 and ≤ 5
		O_2^{11}	Choice 2	> 5 and ≤ 12
		O_3^{11}	Choice 3	> 12 and ≤ 25
		O_4^{11}	Choice 4	> 25 and ≤ 50
A_{12}	Weight category 5	O_1^{12}	Choice 1	> 0 and ≤ 5
		O_2^{12}	Choice 2	> 5 and ≤ 12
		O_3^{12}	Choice 3	> 12 and ≤ 25
A_{13}	Weight category 6	O_1^{13}	Choice 1	> 0 and ≤ 25
		O_2^{13}	Choice 2	> 25 and ≤ 50

Continued on next page

[1] in the unit of 50g.

Table 6.4 – continued from previous page

ID	Category	ID	Choice	Values
A_{14}	Weight category 7	O_1^{14}	Choice 1	> 0 and ≤ 50
A_{15}	Weight category 8	O_1^{15}	Choice 1	> 0 and ≤ 30
A_{16}	Insured	O_1^{16}	Choice 1	True
		O_2^{16}	Choice 2	False
A_{17}	Value	O_1^{17}	Choice 1	$= 0$
		O_2^{17}	Choice 2	> 0 and ≤ 100
A_{18}	Confirm	O_1^{18}	Choice 1	True
		O_2^{18}	Choice 2	False
A_{19}	Quantity category 1	O_1^{19}	Choice 1	> 0 and ≤ 1
A_{20}	Quantity category 2	O_1^{20}	Choice 1	> 0 and ≤ 4
		O_2^{20}	Choice 2	> 4 and ≤ 10
A_{21}	Quantity category 3	O_1^{21}	Choice 1	> 0 and ≤ 100
A_{22}	Quantity category 4	O_1^{22}	Choice 1	> 0 and ≤ 9
		O_2^{22}	Choice 2	> 9 and ≤ 49
		O_3^{22}	Choice 3	> 49 and ≤ 100
A_{23}	Quantity category 5	O_1^{23}	Choice 1	> 0 and ≤ 9
		O_2^{23}	Choice 2	> 9 and ≤ 99
		O_3^{23}	Choice 3	> 99 and ≤ 100
A_{24}	Quantity category 6	O_1^{24}	Choice 1	> 0 and ≤ 4
		O_2^{24}	Choice 2	> 4 and ≤ 100
A_{25}	Quantity category 7	O_1^{25}	Choice 1	> 0 and ≤ 1000
A_{26}	Month	O_1^{26}	Choice 1	January, ... , October
		O_2^{26}	Choice 2	November, December

6.3.2 Measuring the dissimilarity between inputs

Given an input classification scheme, the next step is to compare the dissimilarities between inputs. For ease of discussion, the following notations are used: A = {A_1, A_2 ...,A_h} denotes a set of c**A**tegories. For each A_i, O_i denotes its ch**O**ices {O_1^i, O_2^i ..., O_l^i}. For an input t, A(t) = {A_1^t, A_2^t ..., A_q^t} denotes its corresponding set of categories and O(t) = {O_1^t, O_2^t ..., O_q^t} denotes its corresponding set of choices,

$IPCS_i$	GDT	DSM	CTR	WGT	INS	VAL	CFR	QTY	MTH
$i = 1$	A_1	A_2	A_6	A_8				A_{19}	
$i = 2$	A_1	A_2	A_6	A_8				A_{19}	
$i = 3$	A_1	A_2	A_6	A_8				A_{19}	
$i = 4$	A_1	A_2	A_6	A_8	A_{16}	A_{17}	A_{18}	A_{19}	
$i = 5$	A_1	A_2	A_6	A_8	A_{16}	A_{17}	A_{18}	A_{19}	
$i = 6$	A_1	A_2	A_7	A_9	A_{16}	A_{17}	A_{18}	A_{20}	
$i = 7$	A_1	A_2	A_7	A_{10}				A_{20}	
$i = 8$	A_1	A_3	A_7	A_{11}				A_{21}	
$i = 9$	A_1	A_3		A_{12}				A_{22}	
$i = 10$	A_1	A_3		A_{13}			A_{18}	A_{23}	
$i = 11$	A_1	A_3		A_{13}				A_{23}	
$i = 12$	A_1	A_3	A_7	A_{14}	A_{16}	A_{17}	A_{18}	A_{24}	
$i = 13$	A_1	A_4		A_{15}				A_{25}	
$i = 14$	A_1	A_4		A_{15}				A_{25}	
$i = 15$	A_1	A_5		A_{15}				A_{25}	A_{26}
$i = 16$	A_1	A_5		A_{15}				A_{25}	

Table 6.5: The relationship between an input parameter and a category

where q is the number of input parameters required in defining the input t.

Consider 2 inputs x and y. Intuitively speaking, if a category is associated with one and only one of these inputs, these 2 inputs differ with respect to this category. If both inputs have parameters belonging to the same category, but their parameter values fall into different choices, then these two inputs should be considered different as different functionalities may be triggered. The above intuition forms the basis of the dissimilarity metric, which measures the number of categories that are not in common, and the number of categories that are in common but have different choices for inputs x and y. The algorithm of the dissimilarity measurement (used by ART to determine the dissimilarity between x and y) is given in Figure 6.1.

1. Construct A(x), A(y), O(x) and O(y).
2. Construct $\widehat{O}(x, y) = O(x) \bigcup O(y) \setminus (O(x) \bigcap O(y))$ where $\widehat{O}(x, y)$ denotes the number of distinct choices for inputs x and y.
3. Construct $\widehat{A}(x, y) = \{A_l \mid A_i, \text{if } \exists\ O_j^i \in \widehat{O}(x, y)\}$ where $\widehat{A}(x, y)$ denotes the number of distinct categories which cover different choices for inputs x and y.
4. The dissimilarity measure (v) between x and y is $|\widehat{A}(x, y)|$.

Figure 6.1: The algorithm of the dissimilarity comparison

Note that when $\widehat{O}(x, y) = \oslash$, x and y "overlap" completely in terms of choices, and hence the dissimilarity between x and y is zero. When there exists an element e of E which is very similar to c (in other words, e overlaps with c significantly) in

terms of choices, the fitness value of c will be small.

The following example illustrates how the dissimilarity metric is measured, and how ART selects a test case from a set of candidates. Suppose E = $\{e_1, e_2\}$ and C = $\{c_1, c_2\}$, where

e_1 = (GType = Parcel, DSystem = Air, Country = USA, Weight = 250, Quantity = 1),

e_2 = (GType = Letter, DSystem = Registered, Weight = 20, Confirm = False, Quantity = 1),

c_1 = (GType = Letter, DSystem = EMS prepaid, Country = New Zealand, Weight = 30, Insured = False, Value = 0, Confirm = False, Quantity = 5), and

c_2 = (GType = Greeting card, DSystem = Air, Weight = 10, Quantity = 100, Month = December).

e_1, e_2, c_1 and c_2 will trigger $IPCS_1$, $IPCS_{10}$, $IPCS_{12}$ and $IPCS_{15}$, respectively. Based on Tables 6.4 and 6.5, the following are immediate from the definition of O(t) and A(t),

$$\begin{aligned}
A(e_1) &= \{A_1, A_2, A_6, A_8, A_{19}\}\\
O(e_1) &= \{O_1^1, O_1^2, O_3^6, O_9^8, O_1^{19}\}\\
A(e_2) &= \{A_1, A_3, A_{13}, A_{18}, A_{23}\}\\
O(e_2) &= \{O_2^1, O_3^3, O_1^{13}, O_2^{18}, O_1^{23}\}\\
A(c_1) &= \{A_1, A_3, A_7, A_{14}, A_{16}, A_{17}, A_{18}, A_{24}\}\\
O(c_1) &= \{O_2^1, O_5^3, O_1^7, O_1^{14}, O_2^{16}, O_1^{17}, O_2^{18}, O_2^{24}\}\\
A(c_2) &= \{A_1, A_5, A_{15}, A_{25}, A_{26}\}\\
O(c_2) &= \{O_4^1, O_1^5, O_1^{15}, O_1^{25}, O_2^{26}\}
\end{aligned}$$

The smallest dissimilarity measure of c_1 can then be determined by first computing the following:

$$\begin{aligned}
\widehat{O}(e_1, c_1) &= \{O_1^1, O_1^2, O_3^6, O_9^8, O_1^{19}, O_2^1, O_5^3, O_1^7, O_1^{14}, O_2^{16}, O_1^{17}, O_2^{18}, O_2^{24}\}\\
\widehat{A}(e_1, c_1) &= \{A_1, A_2, A_6, A_8, A_{19}, A_3, A_7, A_{14}, A_{16}, A_{17}, A_{18}, A_{24}\}\\
\widehat{O}(e_2, c_1) &= \{O_3^3, O_1^{13}, O_1^{23}, O_5^3, O_1^7, O_1^{14}, O_2^{16}, O_1^{17}, O_2^{24}\}\\
\widehat{A}(e_2, c_1) &= \{A_3, A_{13}, A_{23}, A_7, A_{14}, A_{16}, A_{17}, A_{24}\}
\end{aligned}$$

Obviously, $|\widehat{A}(e_1, c_1)| = 12$ and $|\widehat{A}(e_2, c_1)| = 8$. Therefore, the smallest dissimilarity measure between c_1 and E (that is, fitness value for c_1) is 8. Similarly,

$$\begin{aligned}
\widehat{O}(e_1, c_2) &= \{O_1^1, O_1^2, O_3^6, O_9^8, O_1^{19}, O_4^1, O_1^5, O_1^{15}, O_1^{25}, O_2^{26}\}\\
\widehat{A}(e_1, c_2) &= \{A_1, A_2, A_6, A_8, A_{19}, A_5, A_{15}, A_{25}, A_{26}\}\\
\widehat{O}(e_2, c_2) &= \{O_2^1, O_3^3, O_1^{13}, O_2^{18}, O_1^{23}, O_4^1, O_1^5, O_1^{15}, O_1^{25}, O_2^{26}\}\\
\widehat{A}(e_2, c_2) &= \{A_1, A_3, A_{13}, A_{18}, A_{23}, A_5, A_{15}, A_{25}, A_{26}\}
\end{aligned}$$

Obviously, $|\widehat{A}(e_1, c_2)| = 9$ and $|\widehat{A}(e_2, c_2)| = 9$. Therefore, the smallest dissimilarity measure between c_2 and E (that is, fitness value for c_2) is 9. Since the fitness value of c_1 is smaller than that of c_2, the next test case selected is c_2.

6.4 Issues concerning test case selection

In this section, some important issues related to the selection of test cases in the ART method (proposed in Section 6.3) will be addressed.

Although the inputs of the IPCS are assumed to be uniformly distributed, the subdomains have different probabilities of being selected for testing because some contain more elements than others. Since C consists of randomly selected inputs, the likelihood of an IPCS function being included in C depends on the size of the corresponding subdomain. Obviously, the elements defining C determine what will be the next test case and hence which ICPS function will be triggered next.

Consider the case where $C = \{c_1, c_2\}$ and c_1 has more categories (or choices) than c_2, that is, $|O(c_1)| > |O(c_2)|$ and $|A(c_1)| > |A(c_2)|$. Obviously, if E has no elements like c_1 and c_2 yet in terms of choices (this situation is more likely to occur when $|E|$ is small), then c_1 should have a larger fitness value than c_2; otherwise, c_1 is not necessarily to have a larger fitness value than c_2.

Initially, some IPCS functions may have a higher chance of being tested than others because they have a larger subdomain size, or they have more categories and choices than others. But after some tests, these functions may no longer have a higher probability of being tested because E may have already covered many of their categories or choices. Note that these IPCS functions may be considered "complex or frequently operated", and the others may be considered "simple or rarely operated". If simple or rarely operated IPCS functions have their categories and choices not appearing in other IPCS functions, they would effectively be given a higher priority for testing by ART because ART attempts to get successive tests which are as different from each other as possible.

When all elements of C have the same fitness value, an arbitrary element of C will be chosen as the next test case. With the growth of E, the 408 combinations of choices will eventually be covered. Obviously, if this scenario occurs, then E will

contain at least 408 elements. Then, no more candidates can have a non-zero fitness value. As a consequence, ART becomes incapable of evenly spreading test cases, and is said to have the *input comparison problem*.

6.5 Refinement of ART method

As mentioned in the previous section, an input comparison problem may occur with ART, especially when the categories and choices are not properly defined. Here, this problem is addressed by refining the ART method.

Two parameters are introduced for the purpose of refinement, namely AgeOption and FVOption. The AgeOption defines which elements of E are to be considered in the dissimilarity comparisons, and the FVOption defines how fitness values are to be calculated. Each parameter has two possible options, which are summarized in Table 6.6. Figure 6.2 shows the enhanced ART algorithm, which incoperates these two additional features. It should be pointed out that the ART method proposed in Section 6.3 is equivalent to the enhanced ART with the 'No Age' and 'MIN' options. Hereafter, ART refers to this enhanced ART method.

Parameter	**Option**	**Meaning**
AgeOption	No Age	All the executed test cases would be considered
	Recent n	Only the most recent n executed test cases would be considered
FVOption	SUM	The sum of all dissimilarity measures between a candidate and E.
	MIN	The smallest dissimilarity measure between a candidate and E.

Table 6.6: Options for AgeOption and FVOption parameters

Clearly, if ART is used with MIN option, when fewer executed test cases are considered in the dissimilarity comparison, it is less likely that all combinations of choices would be covered, and less likely that all candidates have zero fitness value. From this perspective, MIN with Recent n option should help ART alleviate the input comparison problem. However, if not all executed test cases are considered in the dissimilarity comparisons, some of those not considered may be very similar to the next test case.

ART with SUM option will alleviate the input comparison problem, because with

Define v_i as the fitness value of a candidate c_i.
Let $E_{\check{C}}$ denote the subset of E being considered in the dissimilarity comparisons.
{
1. Set $n = 0$, E = { }.
2. Randomly select a test case, t, from the input domain according to the uniform distribution.
3. Increment n by 1.
4. If t reveals a failure, go to Step 13; otherwise, add t to E.
5. Randomly generate C = {c_1, c_2, ..., c_k} according to the uniform distribution.
6. Determine $E_{\check{C}}$ based on AgeOption.
7. For each $c_i \in C$, do Step 8-10.
8. Conduct a dissimilarity comparison for each pair of c_i and e_j where $e_j \in E_{\check{C}}$ to obtain $|E_{\check{C}}|$ dissimilarity measures for c_i.
9. If FVOption = 'SUM', then
   let v_i = the sum of the dissimilarity measures of c_i (obtained in Step 8).
10. If FVOption = 'MIN', then
    let v_i = the smallest dissimilarity measures of c_i (obtained in Step 8).
11. Rank all the elements of C in descending order by their v_i.
    Candidates which have the same fitness values, are given the same rank.
12. Let t be the first candidate with the smallest rank, and then go to Step 3.
13. Return n and t. EXIT.
}

Figure 6.2: Nonnumeric version of FSCS-ART

SUM option, it is less frequently that the sum of the dissimilarity measures of all candidates are identical.

6.6 Experiment

In this section, the effectiveness of ART with the IPCS program is studied.

6.6.1 Mutant design

To test the IPCS using ART, 15 mutants were introduced. The mutants were created in a random manner, but those with a very small failure rate ($\theta < 0.000049$) were discarded. Each mutant was given an unique index, m_i where $1 \leq i \leq 15$.

Chapter 3 reported that when testing a numeric program, the F-measure of ART (F_{ART}) depended on the failure pattern (number of failure regions, the shape of failure regions, the existence of predominant failure region, and the proximity of failure regions to the input domain boundary). Unfortunately, some of these concepts have

not yet been properly defined for nonnumeric applications (for example, the failure region shape and the location of failure regions in the input domain). Since failure-causing inputs are similar to each other in terms of their associated functionality and computations, failure patterns will be described in terms of the following two characterizations: (*i*) the number of IPCS functions with which failures are associated (for ease of discussion, these are called 'blocks'); and (*ii*) whether there exists any predominant block (only relevant if failures are associated with more than 1 block) whose size is at least 10 times larger than other blocks. For each mutant, its failure-causing inputs and the analysis of its failure pattern are reported in Table 6.7.

	Failure-causing inputs	**Failure pattern**	
		No. of blocks	predominant blocks
m1	All inputs for $IPCS_3$	1	
m2	Inputs for $IPCS_4$, whose Insured = True Inputs for $IPCS_5$, whose Insured = True Inputs for $IPCS_6$, whose Insured = True	3	exist
m3	Inputs for $IPCS_6$, whose Weight is either 100 or 250	1	
m4	Inputs for $IPCS_7$, whose Quantity > 5	1	
m5	Inputs for $IPCS_5$, whose Zone = 'C' and whose Weight > 50	1	
m6	Inputs for $IPCS_1$, whose Weight $\in$ W Inputs for $IPCS_2$, whose Weight $\in$ W Inputs for $IPCS_3$, whose Zone is either 'C' or 'D' and whose Weight $\in$ W Inputs for $IPCS_4$, whose Weight $\in$ W Inputs for $IPCS_5$, whose Weight $\in$ W , where W denotes the set of {25, 50, 75, 100, 125, 150, 175, 200, 250, 300, 350, 400, 450}	5	exist
m7	Inputs for $IPCS_{12}$, whose Country = 'New Zealand'	1	
m8	All inputs for $IPCS_1$ All inputs for $IPCS_8$	2	not exist
m9	All inputs for $IPCS_{15}$ All inputs for $IPCS_{16}$	2	exist
m10	All inputs for $IPCS_8$	1	
m11	Inputs for $IPCS_9$, whose Quantity ≥ 5 Inputs for $IPCS_{10}$, whose Quantity ≥ 5 Inputs for $IPCS_{11}$, whose Quantity ≥ 5	3	no exist
m12	Inputs for $IPCS_{12}$, whose Value = 0	1	
m13	All inputs for $IPCS_{13}$ All inputs for $IPCS_{14}$	2	no exist
m14	All inputs for $IPCS_{15}$, whose Month = November	1	
m15	Inputs for $IPCS_{16}$, whose Quantity is indivisible by 10	1	

Table 6.7: Design of mutants

6.6.2 Simulation results

In this experiment, the following parameters were used.

- AgeOption - No Age or Recent 10
- FVOption - SUM or MIN

There are 4 different scenarios in total. The results are summarized in Table 6.8.

	RT	**No Age**		**Recent 10**	
		SUM	MIN	SUM	MIN
m1	3259.30	359.74	355.44	367.80	369.76
m2	4.77	3.16	2.93	3.16	2.93
m3	401.44	473.84	453.83	443.81	337.89
m4	2848.82	297.72	300.36	304.84	300.62
m5	154.90	30.74	28.71	39.73	34.52
m6	462.78	138.69	178.24	154.49	176.40
m7	314.64	207.24	375.26	232.93	383.82
m8	316.30	33.95	34.84	33.95	34.53
m9	3692.58	369.69	386.75	368.63	382.48
m10	353.13	37.75	35.39	37.75	36.48
m11	20144.10	1987.02	1987.18	2018.18	1987.86
m12	345.34	40.68	42.1	38.73	47.18
m13	5632.53	556.07	540.64	542.48	535.73
m14	10836.60	1061.84	6194.17	1096.93	1234.53
m15	11623.80	1230.25	2246.64	1213.21	1334.91

Table 6.8: F_{ART} under $k = 10$

In the comparisons between RT and ART, the following observations were made:

- For m1, m4 and m8-15, the F-measure of RT (F_{RT}) is about 10 times larger than F_{ART}.
- For m2, m5 and m6, F_{RT} is considerably larger than F_{ART}.
- For m3, MIN with Recent 10 is the only setting under which ART outperforms RT.
- For m7, ART outperforms RT with the SUM option but not with the MIN option.

An investigation into cases where $F_{ART} \approx \frac{F_{RT}}{10}$ revealed that mutants have homogeneous subdomains (all elements of an IPCS function have the same failure

behaviour), or failure-causing inputs concentrated in a small number of IPCS functions.

When the failure rate (θ) is high, like that of m2 ($\theta \approx 0.2$), we found that F_{ART} is still about 60% of F_{RT}. Such a significant improvement is mainly due to the fact that m2 has a predominant failure block.

The failure-causing inputs in m5 (a single block) are more concentrated than those in m6 (5 blocks with the existence of a predominant block). Hence, the results that the ART F-ratio ($= \frac{F_{ART}}{F_{RT}}$) in m5 is smaller than that in m6, reflect a difference in the degree of concentration of failure-causing inputs in m5 and m6.

Since RT randomly selects test cases from the input domain, the likelihood of functionality or computation being tested depends solely on the number of associated inputs. Because of this randomness, inputs which trigger the same functionality or computation can possibly be tested consecutively. Obviously, it requires more test cases to detect the first failure when program failures are associated with only a few functionality. This is why ART can detect failures more effectively than RT in such cases. Obviously, there will be some cases where RT may outperform ART, like m3 and m7 in this study. Both of these two mutants have high θ, and failure-causing inputs being too specific (in other words, they are similar to a 'point pattern'). However, the experiments show that with an appropriate setting, ART can still outperform RT.

It remains to find which setting is the most preferable for the mutants studied in this chapter. This information is particularly useful when there is no information about θ or the failure pattern of the program under test. The data in Table 6.8 were re-analyzed and Table 6.9, 6.10, 6.11 and 6.12 were constructed to show the improvement of various combinations of options. From these tables, the following observations were made.

- When SUM is applied with No Age, its F_{ART} can be much smaller than that when SUM is applied with Recent 10.
- When MIN is applied with Recent 10, its F_{ART} can be much smaller than that when MIN is applied with No Age.
- Both Tables 6.11 and 6.12 show that the F_{ART} using SUM can be much smaller

than the F_{ART} using MIN. Furthermore, it is more frequent to get a smaller F_{ART} using SUM than using MIN.

- MIN with No Age can perform extremely badly, compared with other settings even though F_{ART} in that setting (MIN with No Age) is still smaller than F_{RT}.

	No Age (a)	**Recent 10 (b)**	$\frac{a-b}{\min(a,b)}$
m1	359.74	367.8	-2.24%
m2	3.16	3.16	0.00%
m3	473.84	443.81	6.77%
m4	297.72	304.84	-2.39%
m5	30.74	39.73	-29.25%
m6	138.69	154.49	-11.39%
m7	207.24	232.93	-12.40%
m8	33.95	33.95	0.00%
m9	369.69	368.63	0.29%
m10	37.75	37.75	0.00%
m11	1987.02	2018.18	-1.57%
m12	40.68	38.73	5.03%
m13	556.07	542.48	2.51%
m14	1061.84	1096.93	-3.30%
m15	1230.25	1213.21	1.40%

Table 6.9: ART with SUM

	No Age (a)	**Recent 10 (b)**	$\frac{a-b}{\min(a,b)}$
m1	355.44	369.76	-4.03%
m2	2.93	2.93	0.00%
m3	453.83	337.89	34.31%
m4	300.36	300.62	-0.09%
m5	28.71	34.52	-20.24%
m6	178.24	176.4	1.04%
m7	375.26	383.82	-2.28%
m8	34.84	34.53	0.90%
m9	386.75	382.48	1.12%
m10	35.39	36.48	-3.08%
m11	1987.18	1987.86	-0.03%
m12	42.1	47.18	-12.07%
m13	540.64	535.73	0.92%
m14	6194.17	1234.53	401.74%
m15	2246.64	1334.91	68.30%

Table 6.10: ART with MIN

	SUM (a)	**MIN (b)**	$\frac{a-b}{\min(a,b)}$
m1	359.74	355.44	1.21%
m2	3.16	2.93	7.85%
m3	473.84	453.83	4.41%
m4	297.72	300.36	-0.89%
m5	30.74	28.71	7.07%
m6	138.69	178.24	-28.52%
m7	207.24	375.26	-81.08%
m8	33.95	34.84	-2.62%
m9	369.69	386.75	-4.61%
m10	37.75	35.39	6.67%
m11	1987.02	1987.18	-0.01%
m12	40.68	42.1	-3.49%
m13	556.07	540.64	2.85%
m14	1061.84	6194.17	-483.34%
m15	1230.25	2246.64	-82.62%

Table 6.11: ART with No Age

	SUM (a)	**MIN (b)**	$\frac{a-b}{\min(a,b)}$
m1	367.80	369.76	-0.53%
m2	3.16	2.93	7.85%
m3	443.81	337.89	31.35%
m4	304.84	300.62	1.40%
m5	39.73	34.52	15.09%
m6	154.49	176.4	-14.18%
m7	232.93	383.82	-64.78%
m8	33.95	34.53	-1.71%
m9	368.63	382.48	-3.76%
m10	37.75	36.48	3.48%
m11	2018.18	1987.86	1.53%
m12	38.73	47.18	-21.82%
m13	542.48	535.73	1.26%
m14	1096.93	1234.53	-12.54%
m15	1213.21	1334.91	-10.03%

Table 6.12: ART with Recent 10

Intuitively, SUM should perform better with No Age than with Recent 10 because a wider range of dissimilarity values are generated. On the other hand, MIN should perform better with Recent 10 than with No Age, because it is less likely to encounter the input comparison problem. This is why MIN and No Age perform extremely

poorly for m14 and m15. With regard to which FVOption best fits with No Age and Recent 10 to achieve a small F_{ART}, Tables 6.11 and 6.12 show that SUM is the most preferable candidate.

In summary, the results show that SUM and No Age is the most effective combination of options.

6.7 Conclusion

In recent studies, ART has been successfully applied to numeric programs. In this chapter, the extension of the application of ART to nonnumeric programs was demonstrated. A significant contribution to the understandings of the even spread of test cases for nonnumeric programs was made, and a dissimilarity scheme to help enforce an even spread of test cases in nonnumeric input domains was developed. This study shows that, similar to numeric programs, the F-measure of ART is usually smaller than that of RT for nonnumeric programs.

Chapter 7

Mirror adaptive random testing (MART)

In Chapters 3 and 4, ART's performance was studied in many scenarios, and it was showed that ART could save a significant number of test cases used by RT to detect the first failure. However, ART requires additional overheads to evenly spread test cases. Such overheads may render ART less desirabe to use.

Take FSCS-ART as an example. In FSCS-ART, after the first test case, all test cases are selected from a pool of k potential candidates (called *candidate set* C). A candidate c is selected for testing if the distance between c and its nearest neighbour in E (the set of previously executed test cases) is the maximum (compared with other elements of C). In other words, $\min_{i=1}^{|\mathrm{E}|} dist(\mathrm{c}, \mathrm{E}_i) \geq \min_{i=1}^{|\mathrm{E}|} dist(\mathrm{c}_j, \mathrm{E}_i) \forall \mathrm{c}_j \in \mathrm{C}$ where $dist$(p, q) denotes the distance between p and q. Since the number of required distance computations is of the order $|\mathrm{E}|^2$ (a detailed explanation is given in Section 7.1), FSCS-ART may require a significant number of distance computations when |E| is large. Without ambiguity, "ART" in this chapter refers to FSCS-ART with the value of k being 10, unless otherwise specified.

In this chapter, a technique called mirroring is proposed to reduce the distance computation of ART. The integration of mirroring and ART is is called *mirror adaptive random testing* (abbreviated as MART). Its efficiency and fault detection effectiveness are examined in the following sections. Guidelines of using MART are given at the last section of this chapter.

7.1 Basic idea of MART

The basic idea of mirror adaptive random testing (MART) is explained as follows. Firstly, the input domain D is partitioned into disjoint subdomains. ART is used to generate a test case t in one of the subdomains; then, a simple mapping function is used to generate an image of t in each remaining subdomain. The resultant test cases (t and its images) are tested in a sequential order. The following elaborates this method using a 2 dimensional input domain.

D is partitioned into disjoint subdomains of identical size and shape. Figure 7.1 illustrates some examples of partitioning. The notation $\mathrm{X}_m\mathrm{Y}_n$ refers to the partitioning that has m and n equal sections in X-axis and Y-axis, respectively, with examples given in Figure 7.1.

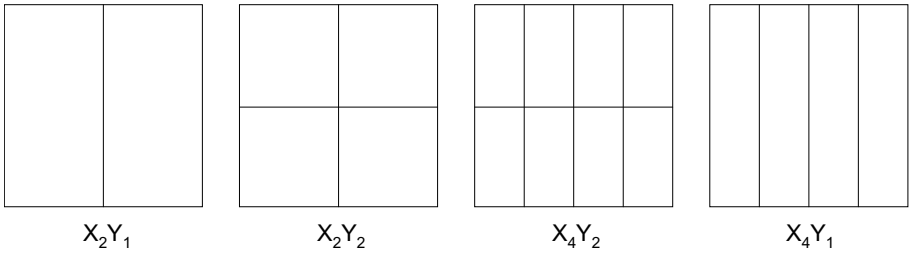


Figure 7.1: Some simple partitioning schemes

Suppose $\mathrm{D} = \{(X,Y)|0 \leq X < u, 0 \leq Y < v\}$. As shown in Figure 7.2, the $\mathrm{X}_2\mathrm{Y}_1$ mirror partitioning is used to generate two subdomains: $\mathrm{D}_1 = \{(X,Y)|0 \leq X < \frac{u}{2}, 0 \leq Y < v\}$ and $\mathrm{D}_2 = \{(X,Y)|\frac{u}{2} \leq X < u, 0 \leq Y < v\}$; and every test case t generated by ART in D_1 is linearly translated to D_2. Suppose $\mathrm{t} = (x,y)$, then its image will be $(x + \frac{u}{2}, y)$. Obviously, the distribution of test cases in D_1 is identical to the distribution of their images in D_2. Thus, test cases in D_2 will be as evenly spread as those in D_1. Since ART is only applied to D_1, only test cases in D_1 require distance computations.

Clearly, linear translation is less expensive than using ART to generate test cases. For a mirroring scheme with h subdomains, if MART has executed l test cases, ART would have been invoked $\lceil \frac{l}{h} \rceil$ times because ART is only applied to one of the h subdomains ($\lceil \frac{l}{h} \rceil$ denotes the integer by rounding up $\frac{l}{h}$). If C contains k elements, when ART is used without mirroring and has executed the same amount of test cases as MART, the required number of distance calculations will be $k \cdot \sum_{i=1}^{l} i = \frac{k \cdot l \cdot (l+1)}{2}$.

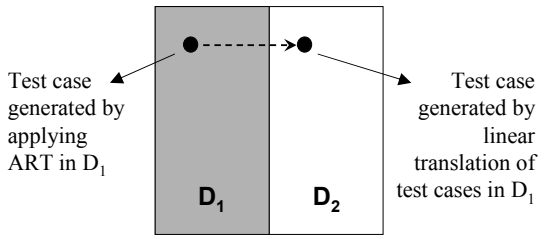


Figure 7.2: Basic idea of MART

Hence, the distance calculations in ART are of the order of l^2. On the other hand, the number of distance calculations for MART will be $k \cdot \sum_{i=1}^{\lceil \frac{l}{h} \rceil} i \approx \frac{k \cdot l \cdot (l+h)}{2h^2}$. In other words, MART needs only about $\frac{1}{h^2}$ as many distance calculations as ART.

The following experiment was conducted to compare the effectiveness of MART and ART. In this simulation experiment, there were the following parameters:

1. Partitionings - X_2Y_1, X_3Y_1, X_1Y_2 and X_1Y_3.

2. Failure rates (θ) - 0.0005, 0.001, 0.005 and 0.01.

3. Failure patterns - block, strip and point patterns.

Without loss of generality, ART was applied to the top leftmost subdomain. The test case t generated by ART was always tested before its images. The images are tested in such an order that the left image is tested before the right image in a row by row manner from top to bottom. Hereafter, this particular order of testing images of t is named the *sequential order* in the context of this chapter.

The classification of failure patterns given in Figure 2.1 is applied here. The block failure pattern here was set like that in Section 3.1.1 (a single square). The strip pattern was simulated by randomly choosing a point on the vertical boundary of D (Y-axis), and a point on the horizontal boundary of D (X-axis), and plotting the boundaries of a strip centred on them, with width chosen according to the specified failure rate. With regard to the simulation of point patterns, 10 circular failure regions were randomly located in D, without overlapping each other.

There are 48 different scenarios in total; these were grouped according to the type of failure patterns, and then further grouped according to the failure rates, into Tables 7.1-7.4. Based on these results, it can be observed that the F-measure for ART and the various MART are similar for various values of θ, and various

failure patterns. In other words, although MART requires a fraction ($\frac{1}{h^2}$) of the distance computations of ART, it achieves similar fault-detection effectiveness (in terms of the F-measure).

	Block	Strip	Point
ART	1231.46	1475.14	1817.42
X2Y1	1305.81	1475.63	1810.70
X3Y1	1238.54	1452.61	1902.14
X1Y2	1238.26	1402.67	1855.49
X1Y3	1304.49	1442.79	1838.51

Table 7.1: F-measures of ART and MART when $\theta = 0.0005$

	Block	Strip	Point
ART	633.08	781.68	927.15
X2Y1	635.08	787.21	937.50
X3Y1	631.09	783.25	929.20
X1Y2	625.58	827.41	917.78
X1Y3	648.58	815.81	892.89

Table 7.2: F-measures of ART and MART when $\theta = 0.001$

	Block	Strip	Point
ART	134.93	171.43	191.19
X2Y1	125.59	180.88	193.89
X3Y1	138.15	182.93	195.33
X1Y2	130.14	190.59	191.75
X1Y3	134.21	176.57	192.19

Table 7.3: F-measures of ART and MART when $\theta = 0.005$

	Block	Strip	Point
ART	67.54	91.44	93.13
X2Y1	70.99	85.04	102.37
X3Y1	70.30	93.11	96.92
X1Y2	66.95	87.15	97.20
X1Y3	71.84	93.40	89.59

Table 7.4: F-measures of ART and MART when $\theta = 0.01$

After understanding how mirroring works to improve the efficiency of ART, interesting characteristics of MART and related issues will now be studied in detail.

7.2 Key components and algorithm of MART

Basically, there are three major components in mirroring, namely *mirror partitioning* (defining how D is partitioned into subdomains), *mirror function* (defining how to map an element from one subdomain to the other subdomains) and *mirror selection order* (defining the testing order of images). These three components form the *mirroring scheme*, which should be defined prior to the use of MART. For example, one of the mirroring schemes used in the previous section consists of: $\mathrm{X}_2\mathrm{Y}_1$ as the mirror partitioning; linear translation as the mirror function; and sequential order as the mirror selection order.

Since ART does not employ any partitioning scheme, the $\mathrm{X}_m\mathrm{Y}_n$ mirror partitioning that yields subdomains of identical size and shape will be used. The following gives a formal definition of this kind of mirror partitioning using a 2Đ input domain. Suppose $\mathrm{D} = \{(X, Y) | x_1 \leq X < x_2,\ y_2 \leq Y < y_1\}$. This rectangular input domain D is said to have (x_1, y_1) as its top leftmost corner, and (x_2, y_2) as its bottom right-

most corner. For the X_mY_n partitioning in Figure 7.3, each subdomain is denoted as D_i, $1 \leq i \leq m \cdot n$. The index i is ordered from left to right in a row by row manner from top to bottom. Each D_i is assigned a tuple of integers (i_x, i_y) to present its location in D, where $(i_x, i_y) = (\lfloor \frac{m+i-1}{m} \rfloor, i - m(\lfloor \frac{m+i-1}{m} \rfloor - 1))$, and $\lfloor \frac{m+i-1}{m} \rfloor$ denotes the integer by rounding down $\frac{m+i-1}{m}$. For example, the top leftmost subdomain (D_1) and bottom rightmost subdomain ($D_{m \cdot n}$) are given indexes (1, 1) and (m, n), respectively.

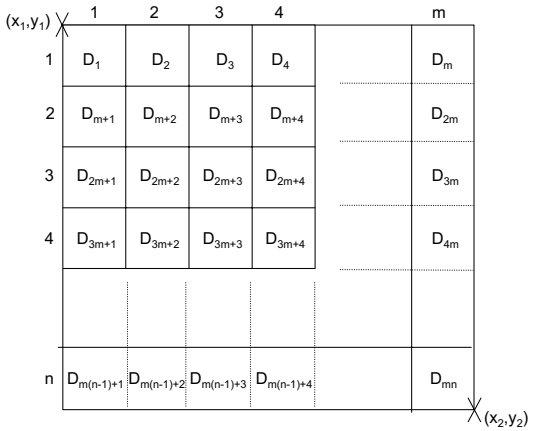
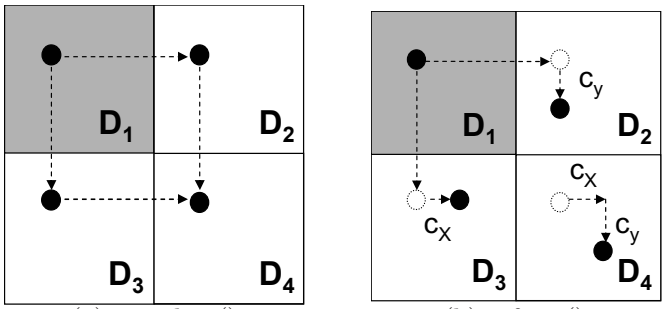


Figure 7.3: Coordination of subdomains for the mirror partitioning of X_mY_n

In MART, ART is only applied to a selected subdomain (namely, the *source domain*) to generate test cases, which are in turn mapped to other subdomains (namely, the *mirror domains*) to generate further test cases. Without loss of generality, D_1 is designated as the source domain, unless otherwise specified.

For any point, t = (x, y) in D_1, two mapping functions, namely *translate()* and *refract()* are investigated in this thesis. These functions are one-to-one mapping functions and yield an image of t in the D_i according to Formulae 7.1 and 7.2. Figure 7.4 schematically illustrates these two mirror functions. Basically, translate() linearly translates t from one subdomain to another. refract() is slightly more sophisticated than translate(): given a predefined vector c = (c_x, c_y) where both c_x and c_y are greater than or equal to 0, refract() gets images by linear translation first, and then shifts the x-coordinate and y-coordinate of the image in D_i by $(c_x \cdot (i_y - 1)$, $c_y \cdot (i_x - 1))$, subject to wrapping (that is, the bottom border is connected to the top border, and the right border is connected to the left border). Obviously, some settings of c may not have actual shift effect. For example, when c_y is 0 and the

$X_n Y_1$ mirror partitioning is used where $n > 1$, there is no actual shift on the images of t. In this example, refract() is identical to the translate() function.



(a) translate() (b) refract()

Figure 7.4: Various mirror functions

$$\text{translate}(t, i) = t + (\frac{(i_x - 1) \cdot (x_2 - x_1)}{m}, \frac{(i_y - 1) \cdot (y_1 - y_2)}{n}) \tag{7.1}$$

$$\text{refract}(t, i, c) = (T_x, T_x), \text{where}(T_x, T_y) \text{ can be calculated as follows:} \tag{7.2}$$

$$(T'_x, T'_x) = \text{translate}(t, i) + (c_x \cdot (i_y - 1), c_y \cdot (i_x - 1))$$

$$T_x = \begin{cases} T_x = T'_x & \text{if } (T'_x < x_1 + \frac{i_x \cdot (x_2 - x_1)}{m}) \\ T_x = \frac{(i_x - 1) \cdot (x_2 - x_1)}{m} + \left(T'_x - \frac{x_2 - x_1}{m} \cdot \left\lfloor \frac{\frac{T'_x - x_1}{x_2 - x_1}}{m} \right\rfloor \right) & \text{otherwise} \end{cases}$$

$$T_y = \begin{cases} T_y = T'_y & \text{if } (T'_y < y_1 + \frac{i_y \cdot (y_1 - y_2)}{n}) \\ T_y = \frac{(i_y - 1) \cdot (y_1 - y_2)}{n} + \left(T'_y - \frac{y_1 - y_2}{n} \cdot \left\lfloor \frac{\frac{T'_y - y_2}{y_1 - y_2}}{n} \right\rfloor \right) & \text{otherwise} \end{cases}$$

where $\left\lfloor \frac{a}{b} \right\rfloor$ denotes the integer by rounding down $\frac{a}{b}$.

Note that test cases generated in D_1 are always tested before their images. The images of t (called *sibling images*) can be tested according to one of the following orders, described in terms of the mirror domains where they lie.

1. Sequential order: D_i is always tested before D_{i+1}, where $1 \leq i < mn$.

2. Random order: Mirror domains are selected for testing in a random manner without replacement.

3. Adaptive random order: Similar to ART, this selection order attempts to test mirror domains in a random manner, but prevents adjacent (neighbouring)

subdomains being tested one after another. As a consequence, some existing approaches of ART can be used to implement adaptive selection order. Here, FSCS-ART is used to determine the testing order. Note that when the number of untested subdomains is smaller than 10 (the size of the candidate set C), then the remaining subdomains are tested randomly.

Figure 7.5 shows the detailed algorithm of MART. Basically, the algorithm of MART has three additional steps (Steps 2, 6 and 7) compared with the basic ART algorithm, given in Figure 2.2.

Let M(t, D_i) be the image of t in D_i generated by the chosen mirror function.
Let L be an array, storing the mirror domains in the order to be tested.
L[x] is used to denote the x^{th} element of L.
Suppose there are 3 mirror domains {D_2, D_3, D_4} to be tested in the following order: D_2, D_4, D_3. Thus, L[1], L[2] and L[3] contains D_2, D_3 and D_4, respectively.

{

1. Set $n = 0$, E = { } and C = { }.
2. Partition the input domain according to the chosen mirror partitioning.
3. Randomly select a test case, t, from D_1 according to the uniform distribution.
4. Increment n by 1.
5. If t reveals a failure, go to Step 12; otherwise, store t in E.
6. Determine L based on the chosen mirror selection order function
7. For every L[x], where x from 1 to $(h - 1)$, do Steps 7.1-7.2
   7.1. Increment n by 1.
   7.2. If M(t, L[x]) reveals a failure, go to Step12.
8. Randomly generate C = {c_1, c_2,..., c_k} from D_1 according to the uniform distribution.
9. For each element in C, find its nearest neighbour in E.
10. In C, find which element, c_j, has the longest distance to its nearest neighbour in E.
11. Let t = c_j and go to Step 4.
12. Return n and t. EXIT.

}

Figure 7.5: The algorithm of MART

7.3 The characteristics of mirroring

MART consists of two parts: ART and mirroring. It is important to present some important characteristics of mirroring before studying MART in detail.

7.3.1 Effective failure-causing inputs

Through the mirror function, some non-failure-causing inputs in D_1 are associated with failure-causing inputs in mirror domains. In this way, selection of these non-failure-causing inputs by ART may lead to detection of failures by their *failure-causing images*. Elements in D_1, which can detect failures either by themselves, their images or by both, are known as *effective failure-causing inputs*. $\ddot{E}$ is used to denote the set of effective failure-causing inputs. In brief, selecting an effective failure-causing input guarantees the detection of a failure.

Let F_i denote the failure-causing inputs inside D_i, $\forall 1 \leq i \leq h$. ρ and ρ_i are used to denote the failure patterns formed by F and F_i, respectively. θ and θ_i are used to denote the failure rates in D and D_1 ($\frac{|F|}{|D|}$ and $\frac{|F_i|}{|D_i|}$), respectively.

Obviously, $\ddot{E} = \bigcup_{i=1}^{h} \ddot{E}_i$, where $\ddot{E}_1 = \{t \in D_1 \mid t \in F_1\}$ and $\ddot{E}_i = \{t \in D_1 \mid$ the image of t in D_i is an element of $F_i\}$ $\forall i, 2 \leq i \leq h$. By definition, $|\ddot{E}| \leq |D_1|$. When there exist distinct $\ddot{E}_i$ and $\ddot{E}_j$ such that $\ddot{E}_i \bigcap \ddot{E}_j \neq \varnothing$, overlapping of effective failure-causing inputs (or simply *overlapping*) is said to occur. Based on the notion of effective failure-causing inputs, the *effective failure rate*, $\ddot{E}\theta$, is defined as $\frac{|\ddot{E}|}{|D_1|}$, and the *effective failure pattern*, $\ddot{E}\rho$, is defined as the pattern formed by $\ddot{E}$ in D_1.

Consider Program 7.1 which accepts one integer parameter, namely, "deposit". Suppose D = {deposit $|0 <$ deposit $\leq 1000\}$, and the mirroring scheme consists of translate() mirror function and X_2 mirror partitioning which generates two subdomains, namely, D_1 = {deposit $|0 <$ deposit $\leq 500\}$ and D_2 = {deposit $|500 <$ deposit $\leq 1000\}$. Then, $F_1 = \{10 \leq$ deposit $< 50\}$ and $F_2 = \{750 \leq$ deposit $\leq 1000\}$. $\ddot{E}$ $=\ddot{E}_1 \cup \ddot{E}_2$ where $\ddot{E}_1 = F_1$ and $\ddot{E}_2 = \{250 \leq$ deposit $\leq 500\}$. $\ddot{E}\theta = \frac{291}{500}$ and $\ddot{E}\rho$ consists of two blocks.

7.3.2 Some properties of mirroring schemes

In this section, the properties of mirroring schemes introduced in Section 7.2 are studied. The mirror partitioning scheme is assumed to yield h subdomains, unless otherwise specified.

Proposition 7.1.

(i) If all θ_i are equal, then $\theta_i = \theta$, otherwise, there is at least one θ_i which is greater

Program 7.1 Simple program under test

```
INPUT deposit
IF (deposit < 10)    /* ERROR: Should be if(deposit < 50) */
{
  OUTPUT Deposit does not meet the minimum requirement of the bank.
  EXIT
}
ELSE IF(deposit < 750)
{
  balance = balance + deposit.
  reward_points = reward_points + deposit
  OUTPUT balance, reward_points
}
ELSE IF /* reward customer to deposit more than $750 per time */
{
  balance = balance + deposit.
  reward_points = reward_points + deposit * 1.1
  //ERROR: should be reward_points = reward_points + deposit * 1.15
  OUTPUT balance, reward_points
}
```

than θ

(ii) $\theta_i \leq \ddot{E}\theta \leq 1$, *where* $1 \leq i \leq h$,

(iii) $\theta \leq \ddot{E}\theta$ *(*$\theta < \ddot{E}\theta$ *if not all* θ_i *are equal).*

(iv) $\ddot{E}\theta \leq h\theta$

Proof. (i) Suppose all θ_i are equal. It follows from the definition that all $|\mathrm{F}_i|$ are equal because all $|\mathrm{D}_i|$ are equal. It is obvious that $\theta = \theta_i$.

Suppose some θ_i are not equal and there does not exist any $\theta_i > \theta$. Without loss of generality, assume that $\theta_1 < \theta$ and $\theta_i \leq \theta \ \forall i \geq 2$. Since $|\mathrm{F}_i| = \theta_i \cdot |\mathrm{D}_i|$, $\forall i$, then $\sum_{i=1}^{h} |\mathrm{F}_i| = \sum_{i=1}^{h} (\theta_i \cdot |\mathrm{D}_i|)$. Since $\theta_1 < \theta$ and $\theta_i \leq \theta \ \forall i \geq 2$, then $\sum_{i=1}^{h} |\mathrm{F}_i| < \sum_{i=1}^{h} (\theta \cdot |\mathrm{D}_i|)$. By definition, $|\mathrm{F}| = \sum_{i=1}^{h} |\mathrm{F}_i|$ and $|\mathrm{D}| = \sum_{i=1}^{h} |\mathrm{D}_i|$. Therefore, $|\mathrm{F}| < \theta \cdot |\mathrm{D}|$. However, $|\mathrm{F}| < \theta \cdot |\mathrm{D}|$ is contradictory to the definition that $|\mathrm{F}| = \theta \cdot |\mathrm{D}|$. In other words, either all θ_i are equal, or at least one of them is greater than θ.

(ii) Obviously, $\ddot{E}\theta \leq 1$. By definition, $\theta_i = \frac{|\mathrm{F}_i|}{|\mathrm{D}_i|} = \frac{|\ddot{E}_i|}{|\mathrm{D}_i|} = \frac{|\ddot{E}_i|}{|\mathrm{D}_1|}$ because $|\mathrm{D}_i| = |\mathrm{D}_1|$ and the mirror function is a one-to-one mapping. Since $|\bigcup_{i=1}^{h} \ddot{E}_i|$ is never smaller than $|\mathrm{F}_i|$, $\ddot{E}\theta$ $(= \frac{|\bigcup_{i=1}^{h} \ddot{E}_i|}{|\mathrm{D}_1|})$ can never be smaller than any θ_i.

(iii) It follows immediately from (i) and (ii).

(iv) Since the mirror function is a one-to-one mapping, $|\ddot{E}_i| = |\mathrm{F}_i|$, for all i.

Since $|\ddot{E}| = \bigcup_{i=1}^{h} |\ddot{E}|_i \leq \sum_{i=1}^{h} |\ddot{E}_i|$, this means $|\ddot{E}| \leq |\mathrm{F}|$. Therefore, $\ddot{E}\theta = \frac{|\ddot{E}|}{|\mathrm{D}_1|} \leq \frac{|\mathrm{F}|}{|\mathrm{D}_1|} = \frac{h|\mathrm{F}|}{|\mathrm{D}|} = h\theta$ □

Proposition 7.1 shows that the partitioning of D does make at least one θ_i greater than θ, unless all θ_i are equal. In addition, it shows that $\ddot{E}\theta$ will be at least as large as θ and the maximum θ_i, but it will never be larger than $h\theta$.

As discussed above, some non-failure-causing inputs in D_1 could be associated with failure-causing images in mirror domains. From this perspective, in addition to the "physical" grouping of the failure-causing inputs by the mirror partitioning scheme, mirroring can be considered to "virtually" group failure-causing inputs through mirror functions into D_1. Clearly, $\ddot{E}\theta$ depends on how a mirroring scheme groups failure-causing inputs. Next, the effect of overlapping on $\ddot{E}\theta$ will be shown.

Lemma 7.1. *There is no overlapping if and only if* $|\ddot{E}| = |\mathrm{F}|$.

Proof. Since the mirror function is a one-to-one mapping, $|\ddot{E}_i| = |\mathrm{F}_i|$ for all i. Hence, $\sum_{i=1}^{h} |\ddot{E}_i| = |\mathrm{F}|$. There exists no distinct $\ddot{E}_i$ and $\ddot{E}_j$ such that $\ddot{E}_i \bigcap \ddot{E}_j \neq \varnothing \iff |\ddot{E}| = |\bigcup_{i=1}^{h} E_i| = \sum_{i=1}^{h} |\ddot{E}_i|$ Therefore, there is no overlapping $\iff |\ddot{E}| = |\mathrm{F}|$. □

An immediate consequence from Lemma 7.1, and the definitions of $\ddot{E}$ and F, is that overlapping occurs if and only if $|\ddot{E}| < |\mathrm{F}|$. Lemma 7.1 leads to Proposition 7.2 which gives two necessary conditions for non-overlapping: (i) the size of D_1 is at least equal to the size of F; and (ii) $h \leq \frac{1}{\theta}$.

Proposition 7.2. *If no overlapping occurs,* $|\mathrm{D}_1| \geq |\mathrm{F}|$ *and* $h \leq \frac{1}{\theta}$.

Proof. Assume that there is no overlapping. It follows immediately after Lemma 7.1 that $|\ddot{E}| = |\mathrm{F}|$. Since $\ddot{E} \subset \mathrm{D}_1$, we have $|\mathrm{D}_1| \geq |\ddot{E}|$, and hence $|\mathrm{D}_1| \geq |\ddot{E}| = |\mathrm{F}|$. $|\mathrm{D}_1| \geq |\mathrm{F}|$ implies $\frac{|\mathrm{D}_1|}{|\mathrm{D}|} \geq \frac{|\mathrm{F}|}{|\mathrm{D}|}$ which in turn implies $h \leq \frac{1}{\theta}$. □

Proposition 7.2 implies that if h is too large, such that $h > \frac{1}{\theta}$, then overlapping occurs.

Proposition 7.3. *There is no overlapping if and only if* $\ddot{E}\theta = h\theta$.

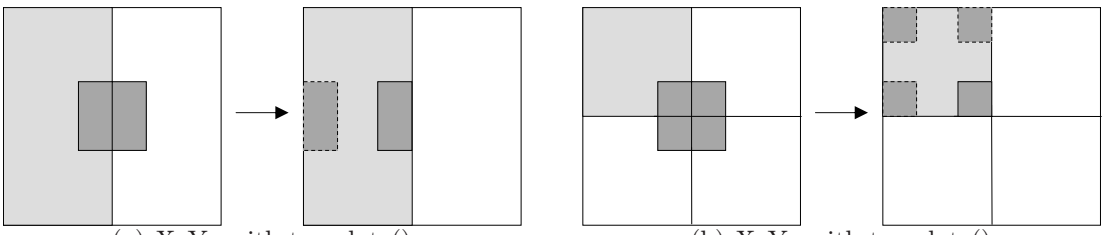
Proof. $\ddot{E}\theta = h\theta \iff \frac{|\ddot{E}|}{|\mathrm{D}_1|} = \frac{h|\mathrm{F}|}{|\mathrm{D}|} \iff \frac{|\ddot{E}|}{|\mathrm{D}_1|} = \frac{|\mathrm{D}|}{|\mathrm{D}_1|}\frac{|\mathrm{F}|}{|\mathrm{D}|} \iff |\ddot{E}| = |\mathrm{F}|$. It follows immediately from Lemma 7.1 that there is no overlapping if and only if $\ddot{E}\theta = h\theta$. □

It follows immediately from Propositions 7.1 and 7.3 that overlapping occurs if and only if $\ddot{E}\theta < h\theta$. Proposition 7.3 also implies that when there is no overlapping, $\ddot{E}\theta = 1$ if and only if h is equal to $\frac{1}{\theta}$.

Next, other factors which affect $\ddot{E}\theta$ and $\ddot{E}\rho$ will be investigated. As explained above, mirror partitioning schemes determine the distribution of failure-causing inputs in each subdomain, and mirror functions may turn a non-failure-causing input in D_1 into an effective failure-causing input. Therefore, $\ddot{E}\theta$ and $\ddot{E}\rho$ depend on mirror functions and mirror partitionings. The following two examples illustrate this dependency. It should be noted that D_1 is highlighted with a light grey colour, and failure regions with a dark grey colour.

In the first example, the effect of mirror partitioning is studied. $\theta = 0.1$ and there is a square failure region located at the centre of D. In both X_2Y_1 (Figure 7.6(a)) and X_2Y_2 (Figure 7.6(b)), translate() is the mirror function used. $\ddot{E}\theta$ in X_2Y_1 and X_2Y_2 are 0.2 and 0.4, respectively (both greater than θ, which is 0.1); and $\ddot{E}\rho$ in X_2Y_1 and X_2Y_2 consist of two identical slices and four identical slices of the original failure region, respectively.

Clearly, when a failure region spans more than one subdomain, translate() effectively divides the original failure region into several sub-regions and maps these sub-regions to the border or corners of D_1. Note that these mapped sub-regions may completely fill D_1.

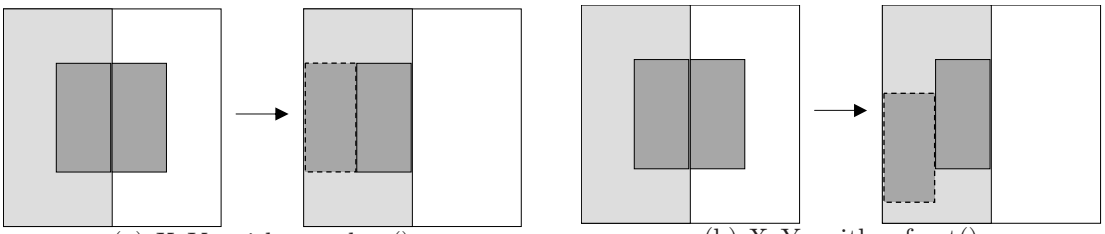


(a) X_2Y_1 with translate() (b) X_2Y_2 with translate()

Figure 7.6: Effect of mirror partitionings on $\ddot{E}\theta$ and $\ddot{E}\rho$

The second example illustrates the effect of mirror functions. $\theta = 0.25$, and there is a square failure region located at the centre of D. translate() (shown in Figure 7.7(a)) and refract() with the shift vector c = (0, 1) (shown in Figure 7.7(b)) are the mirror functions used in an X_2Y_1 partitioning. With translate(), $\ddot{E}\rho$ is identical to the original square failure region, but $\ddot{E}\theta = 0.5$ ($> \theta$ which is 0.25). With refract(), a half of the square failure region (formed by $\ddot{E}_2$) is moved down by 1 unit relative

to the other half (formed by $\ddot{E}_1$), and $\ddot{E}\theta = 0.5$ ($> \theta$ which is 0.25).



(a) X_2Y_1 with translate() (b) X_2Y_1 with refract()

Figure 7.7: Effect of mirror function on $\ddot{E}\theta$ and $\ddot{E}\rho$

Apart from mirror partitionings and mirror functions, θ and ρ also affect $\ddot{E}\theta$ and $\ddot{E}\rho$. Consider the scenario in Figure 7.6(a) and Figure 7.7(a) where the same mirror partitioning and mirror function are used. Their ρ's are both squares, but their θ's are different. The resultant $\ddot{E}\rho$'s are very different. Further consider the scenario as illustrated in Figure 7.8 where $\theta = 0.25$ and ρ consists of two identical rectangles. Each of these rectangles is a half square. Although Figure 7.8 and Figure 7.7(a) have the same θ and use the same mirror partitioning and mirror function, the $\ddot{E}\theta$ and $\ddot{E}\rho$ are different in Figure 7.8 and Figure 7.7(a): $\ddot{E}\theta = 0.25$ and $\ddot{E}\rho$ is a rectangle in Figure 7.8 while $\ddot{E}\theta = 0.5$ and $\ddot{E}\rho$ is a square in Figure 7.7(a). Overlapping occurs in Figure 7.8 but not in Figure 7.7(a).

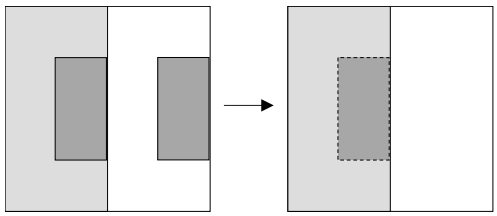


Figure 7.8: Effect of ρ on $\ddot{E}\theta$ and $\ddot{E}\rho$

Note that although mirror partitioning, mirror function, θ and ρ all affect $\ddot{E}\theta$ and $\ddot{E}\rho$, the former two can be controlled by the testers but not the latter two. This issue will be revisited in Section 7.5.

7.4 F-measure of MART

In this section, the factors determining the F-measure of MART, abbreviated as F_{MART}, will be investigated. F_{MART} refers to the number of test cases used by MART to detect the first failure, including test cases both in D_1 and mirror domains.

Whenever MART selects an element e of $\ddot{E}$ from D_1, a failure will be revealed by itself, or by at least one of its $(h-1)$ images. This observation leads to the following conclusions:

1. The earlier MART selects an effective failure-causing input e from D_1, the smaller F_{MART} will be.

2. If e is not a failure-causing input itself (that is, e$\notin F_1$), the earlier an actual failure-causing image of e is selected, the smaller F_{MART}.

3. Before e is selected, MART has already executed a multiple of h test cases. Therefore, when $\ddot{E}\theta < 1$, a larger h results in a larger F_{MART}.

The above conclusions lead to the following issues.

- **Factors determining the speed of selecting an element from $\ddot{E}$**
  When ART is applied to the whole D, its F_{ART} depends on N (the number of dimensions of D), θ, ρ, and the proximity of failure regions (formed by F) to the edges of D. Similarly, the number of attempts (denoted as $F_{\widetilde{MART}}$) required by MART to select an element from $\ddot{E}$ depends on N, $\ddot{E}\theta$, $\ddot{E}\rho$ and the proximity of the failure regions (formed by $\ddot{E}$) to the edges of D_1. Since F_{MART} depends on $F_{\widetilde{MART}}$, therefore F_{MART} depends on these factors.

- **Significance of impact of each component of mirroring scheme**
  The mirror partitioning directly determines the value of h and how failure-causing inputs are grouped in each subdomain. The mirror function virtually groups failure-causing inputs into D_1 and has an impact on how evenly test cases are distributed in each mirror domain. Thus, the choice of mirror partitioning and mirror function has a significant impact on F_{MART}. Obviously, when an e from $\ddot{E}$ is selected and e$\notin F_1$, how fast an actual failure-causing image of e is identified depends on the selection order. Hence, the impact of the mirror selection order on F_{MART} is more significant for the situation where h is large and $\ddot{E}\theta$ is close to 1, than the situation where h is small and $\ddot{E}\theta$ is close to 0.

- **Effect of overlapping**
  Obviously, when $\ddot{E}\theta = 1$, F_{MART} will be at most h (because $F_{\widetilde{MART}}$ is 1).

However, when $\ddot{E}\theta < 1$, F_{MART} will be about $h \cdot F_{\widetilde{MART}}$. As a reminder, if overlapping does not occur, $h \leq \frac{1}{\theta}$ and $\ddot{E}\theta = h\theta$; otherwise, h may be larger than $\frac{1}{\theta}$ and $\ddot{E}\theta < h\theta$. Therefore, when $\ddot{E}\theta = 1$, overlapping may increase F_{MART} (because h may be larger than $\frac{1}{\theta}$), and when $\ddot{E}\theta < 1$, overlapping will increase $F_{\widetilde{MART}}$ (because $F_{\widetilde{MART}} \propto \frac{1}{\ddot{E}\theta}$ but $\ddot{E}\theta < h\theta$) and hence will increase F_{MART}. In summary, overlapping tends to increases F_{MART}.

Note that MART is proposed to effectively implement ART[1] without significant deterioration of its fault-detection effectiveness. As shown above, there are many factors that can affect the performance of MART. In the rest of this chapter, guidelines for obtaining an F_{MART} smaller than or approximately equal to F_{ART} are derived and explained.

7.5 Design issues

To design a mirroring scheme, the following need to be considered:

- **Not every dimension of the input domain can be partitioned**
  Consider the faulty program listed in Program 7.2. Its input parameters are X and Y, but its failures depend only on Y. X is called an *uncritical parameter (or axis)* because X is uncritical to the failures. The corresponding failure pattern looks like Figure 7.9: a very large failure region spanning the entire X axis of D. Clearly, if a translate() mirror function is used, partitioning along X can create great overlapping and hence will yield a large F_{MART}. Therefore, it is suggested to employ as much information as possible to guide the partitioning of the input domain.

Program 7.2 Program failures depend on one single parameter.

```
INPUT X, Y
IF (Y <= 0)    /* ERROR: Should be if(Y <= 1) */
{  Z = X - Y   }
ELSE
{  Z = X + Y  }
OUTPUT Z
```

[1]In Section 7.1, it was shown that MART needs only about $\frac{1}{h^2}$ as many distance calculations as the ART when both have executed the same number of test cases.

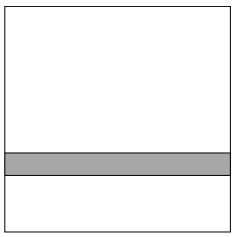
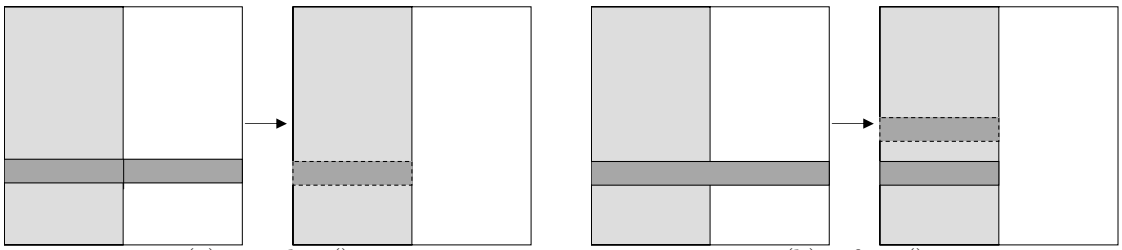


Figure 7.9: Failure pattern for program 7.2

- **Design a mirror function to avoid or reduce overlapping**

  Clearly, it is impossible to design a mirror partitioning that will guarantee no overlapping. Since the mirror function determines the relationship between elements of D_1 and elements of mirror domains, more complicated mirror functions may be used to avoid or reduce overlapping. A comparison between translate() and refract() shows that when the X_2Y_1 mirror partitioning is used to partition D as shown in Figure 7.9, translate() is more likely to have overlapping than refract(), as shown in Figure 7.10.



(a) translate() (b) refract()

Figure 7.10: The impact of translate() and refract() functions on overlapping

- **Spreading test cases evenly**

  Since ART is only applied to D_1, the mirror function determines how test cases are evenly spread within each mirror domain, and the mirror selection order determines how sibling images are evenly spread (in an incremental sense) in the entire input domain. Note that in translate() and refract(), the "distance" between two test cases in D_1 is preserved between the corresponding images in the same mirror domains. However, adaptive random order is better than other selection orders in evenly spreading the sibling images.

7.6 Experiments

A series of experiments were conducted to investigate the impact of various factors on F_{MART}. These experiments were done in a 2Đ square input domain (denoted by D), unless otherwise specified.

7.6.1 Impact of $\ddot{E}\theta$ and $\ddot{E}\rho$ on $\mathbf{F}_{\widetilde{MART}}$ and $\mathbf{F}_{MART}$

A series of experiments were conducted to investigate F_{MART} and $\mathrm{F}_{\widetilde{MART}}$ under various mirror functions, mirror partitioning, ρ and θ. The number of subdomains (h) was kept small, hence $F_{MART} \approx h \cdot F_{\widetilde{MART}}$. In such a setting, there is no significant difference between various mirror selection orders (this will be explained in detail in Section 7.6.2), therefore, the sequential order was used in this experiment.

It is known that ART performs differently with different failure patterns. When ART is applied to the entire input domain, its F-measure (F_{ART}) depends on ρ. For MART, ART is only applied to D_1, the number of attempts taken to select an effective failure-causing input (that is, $\mathrm{F}_{\widetilde{MART}}$) depends on $\ddot{E}\rho$. It is interesting to compare the effectiveness of ART and MART in the situations where ρ and $\ddot{E}\rho$ are different. Also investigated was the impact of overlapping on $\mathrm{F}_{\widetilde{MART}}$ and F_{MART}.

Two experiments were conducted to investigate the situation where ρ is of point type, but $\ddot{E}\rho$ is of block type. In these two experiments, there were 4 square failure regions in D; θ was set to 0.0016, the edge length of D was set to 1000; and the mirroring scheme had $\mathrm{X}_2\mathrm{Y}_2$ mirror partitioning and translate() as mirror function. The difference between the two experiments is on the degree of overlapping.

In the first experiment, the coordinates of the bottom leftmost corner of the input domain were (0, 0), and the coordinates of top leftmost corners of these squares were (50, 620), (570, 620), (50, 100) and (570, 100), respectively, as shown in Figure 7.11. Each of these squares is of size 20×20. $\ddot{E}\theta$, $\ddot{E}\rho$ and the simulation results are reported in Table 7.5. Obviously, no overlapping occurs in this case. Also investigated was the F_{ART} for the same failure pattern types. Table 7.5 shows clearly that F_{MART} is smaller than F_{ART}. This study shows that if the mirroring scheme transforms an undesirable ρ to a "desirable" $\ddot{E}\rho$ (from a point pattern to a block pattern in this case), without overlapping, MART may perform better than

ART.

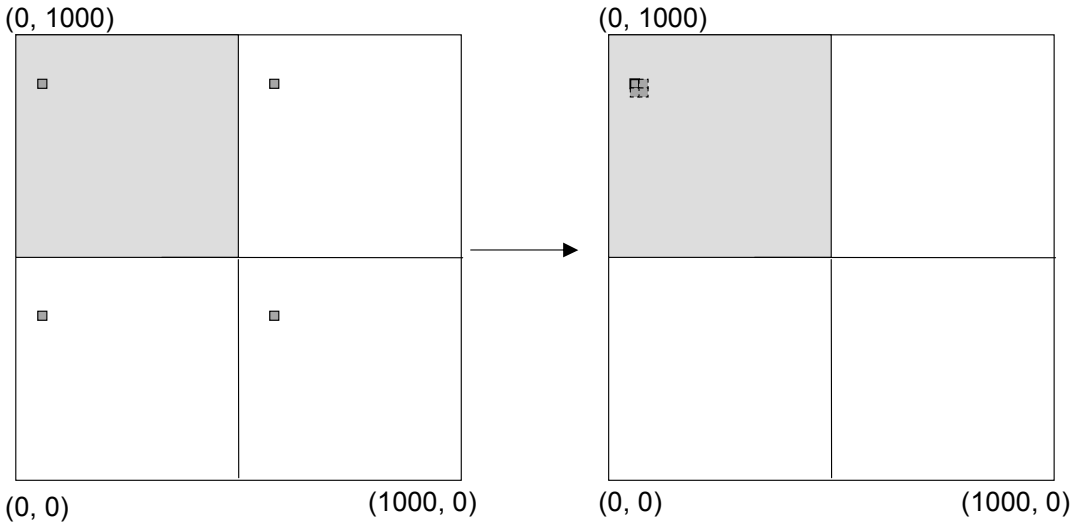


Figure 7.11: ρ consists of 4 identical square blocks, but $\ddot{E}\rho$ consists of a single square block whose size is the total size of 4 square blocks.

Setting	$\ddot{E}\theta$	$\ddot{E}\rho$	**F-measure**
X_2Y_2 + translate()	4θ	A whole square	413.68
ART	not applicable	not applicable	514.54

Table 7.5: Results for experiment 2

The second experiment was similar to the first experiment except that the top leftmost corners of these square failure regions were (50, 600), (550, 600), (50, 100) and (550, 100), respectively, as shown in Figure 7.12. $\ddot{E}\theta$, $\ddot{E}\rho$ and the results of the experiment are reported in Table 7.6. In this case, four square failure regions completely overlapped each other. Note that $\ddot{E}\theta = \theta$ and $\ddot{E}\rho$ is a single square block. The simulation result shows that F_{MART} is 1589.25 (about $\frac{0.64h}{\ddot{E}\theta}$). Despite the fact that $\ddot{E}\rho$ is a desirable pattern for MART, F_{MART} is much greater than F_{ART} because of overlapping.

Setting	$\ddot{E}\theta$	$\ddot{E}\rho$	**F-measure**
X_2Y_2 + translate()	θ	A quarter of ρ	1589.25
ART	not applicable	not applicable	537.04

Table 7.6: Results for experiment 2

Another experiment was conducted to compare F_{ART} and F_{MART} for the situations where ρ is a block pattern, but $\ddot{E}\rho$ is a point pattern. In this experiment,

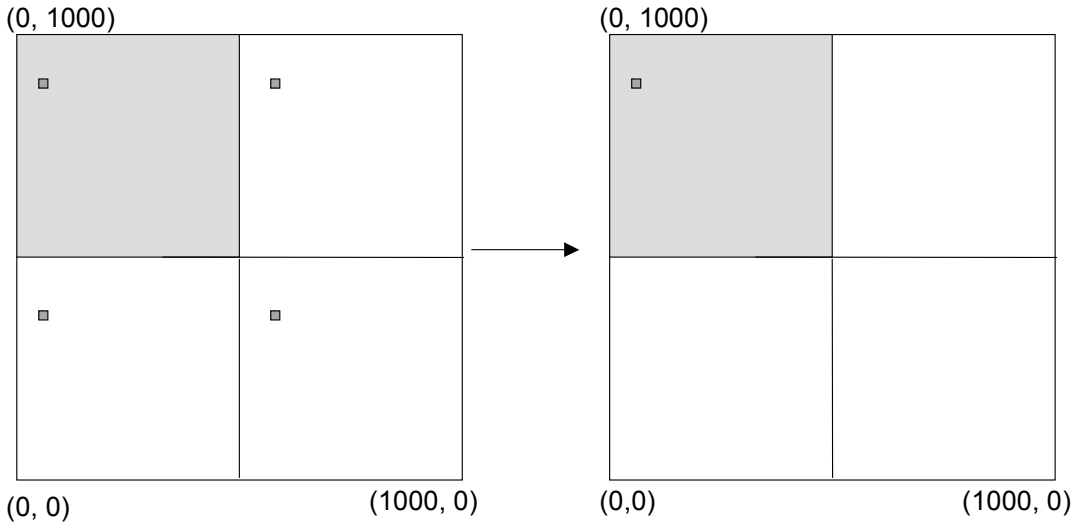


Figure 7.12: ρ consists of 4 identical square blocks, but $\ddot{E}\rho$ consists of a single square block whose size is identical to any one of these 4 square blocks.

$\theta = 0.0025$ and ρ was a single square failure region lying at the centre of D. Two mirroring schemes, X_2Y_1 with translate() and X_2Y_2 with translate() (Figure 7.6) were used. $\ddot{E}\theta$, $\ddot{E}\rho$ and the results of the experiment are reported in Table 7.7. Since the failure regions formed by $\ddot{E}$ are next to the borders or the corners of D_1, the average $F_{\widetilde{MART}}$ is expected to be small (because FSCS-ART performs well when failure regions are close to the boundary). For $\theta = 0.0025$ and ρ a square failure region, F_{ART} is 252 (equivalent to F_{MART} for X_1Y_1 mirror partitioning as reported in Table 7.9). Clearly, F_{ART} is larger than F_{MART} collected in this experiment (205.29 and 157.22, in Table 7.7). This study shows that even if mirroring turns a "desirable block pattern" ρ into an "undesirable point pattern" $\ddot{E}\rho$, MART may still outperform ART if the failure regions formed by $\ddot{E}$ are close to the boundary of D_1.

Setting	$\ddot{E}\theta$	$\ddot{E}\rho$	$\mathbf{F}_{MART}$
X_2Y_1 + translate()	2θ	2 half of the square reside on the sides	205.29
X_2Y_2 + translate()	2θ	4 squares reside on the corners	157.22

Table 7.7: Results for experiment 1

In summary, mirroring schemes have an impact on $F_{\widetilde{MART}}$ and F_{MART} because they can change the $\ddot{E}\theta$ and $\ddot{E}\rho$.

7.6.2 Impact of h and mirror selection order on $\mathrm{F}_{\widetilde{MART}}$ and F_{MART}

A series of experiments were conducted to investigate F_{MART} under various h and mirror selection orders. The relationship between h and the degree of overlapping was also investigated.

7.6.2.1 Block pattern

In the first experiment, F_{MART} was studied using three failure rates (0.001, 0.0025 and 0.01) and three mirror selection orders (sequential, random and adaptive random order). translate() was chosen as the mirror function and the $\mathrm{X}_n\mathrm{Y}_n$ partitioning was chosen to generate square subdomains, where n was 1 or an even number from 2 to 80. As a reminder, when the number of subdomains (h) is smaller than 10 (the size of C), the implementation of adaptive random order is effectively the same as the random order. Furthermore, MART using $\mathrm{X}_1\mathrm{Y}_1$ partitioning scheme is identical to the original ART, so the F_{MART} of $\mathrm{X}_1\mathrm{Y}_1$ setting is equivalent to F_{ART}, and serves as a benchmark.

In each simulation run, a square failure region was randomly located in D. Hence, $\ddot{E}\rho$ was either a square region or several sub-regions next to the borders or corners of D_1, depending on whether or not the failure region spanned more than one subdomain.

Table 7.8 reports h, $\ddot{E}\theta$ and $\frac{|\mathrm{F}|}{|\mathrm{D}_1|}$ for various θ and mirror partitionings. Clearly, $\ddot{E}\theta$ increases with the increase of h. However, when h reaches or exceeds $\frac{1}{\theta}$, $\ddot{E}\theta$ reaches its maximum value of 1. Any further increase in h will not increase $\ddot{E}\theta$, but will increase overlapping. Since the translate() function is used, and all D_i are square (just like ρ), $\frac{|\mathrm{F}|}{|\mathrm{D}_1|} \leq 1$ implies no overlapping; and $\frac{|\mathrm{F}|}{|\mathrm{D}_1|} > 1$ implies that there is overlapping. In fact, $\frac{|\mathrm{F}|}{|\mathrm{D}_1|}$ can be used as a measure of the degree of overlapping when $\frac{|\mathrm{F}|}{|\mathrm{D}_1|}$ is greater than 1.

From Table 7.8 (refer to the data marked by *), the following is observed: When $\theta = 0.01$, $\mathrm{X}_{10}\mathrm{Y}_{10}$ has $\ddot{E}\theta = 1$ without overlapping. Similarly, when $\theta = 0.0025$, $\mathrm{X}_{20}\mathrm{Y}_{20}$ has $\ddot{E}\theta = 1$ without overlapping. When θ= 0.001, there is no mirror partitioning scheme in the study having $\ddot{E}\theta = 1$ without overlapping; $\mathrm{X}_{30}\mathrm{Y}_{30}$ is the mirror partitioning having $\ddot{E}\theta$ close to 1 without overlapping, and $\mathrm{X}_{32}\mathrm{Y}_{32}$ is the

mirror partitioning having $\ddot{E}\theta = 1$ with the smallest degree of overlapping.

The simulation results are reported in Figure 7.13 (the raw data are given in Table 7.9). Obviously, when $h = 1$, the selection orders are not applicable and hence all F_{MART} are identical for the same θ. When h is small, various selection orders have very similar F_{MART} (note that data for F_{MART} in Table 7.9 are truncated).

Based on Tables 7.8 and 7.9, and Figure 7.13, the following three conclusions can be made.

1. When $1 < h < \frac{1}{\theta}$ (that is, $1 < h < 100$, $1 < h < 400$ and $1 < h < 1000$ for $\theta = 0.01$, $\theta = 0.0025$ and $\theta = 0.001$, respectively), $\ddot{E}\theta$ is equal to $h\theta$ but smaller than 1, there is no overlapping, and F_{MART} is within the range $[\frac{0.75}{\theta}, \frac{0.5}{\theta}]$.

2. When $h = 100$, $h = 400$ and $h = 1024$ for $\theta = 0.01$, $\theta = 0.0025$ and $\theta = 0.001$, respectively, $\ddot{E}\theta$ is equal to 1. Under these scenarios, there is little or no overlapping. Furthermore, for all mirror selection orders, these scenarios also have the smallest F_{MART} ($\approx \frac{1}{2\theta}$) (marked by * in Table 7.9).

3. When $h > \frac{1}{\theta}$, $\ddot{E}\theta$ is 1 and overlapping occurs. F_{MART} is proportional to $\frac{h}{2}$ for the sequential order; and F_{MART} approaches F_{RT} and F_{ART} for the random order and adaptive random order, respectively.

An increase of h has conflicting impacts on $F_{\widetilde{MART}}$. It increases both $\ddot{E}\theta$ and the chance of failure regions lying across several subdomains. Since MART in D_1 is implemented using FSCS-ART, $F_{\widetilde{MART}}$ is large for higher $\ddot{E}\theta$, but small in the case where failure regions are close to the edge. This explains why before $\ddot{E}\theta$ reaches 1, F_{MART} fluctuates as h changes. However, $F_{\widetilde{MART}}$ is still close to F_{ART}.

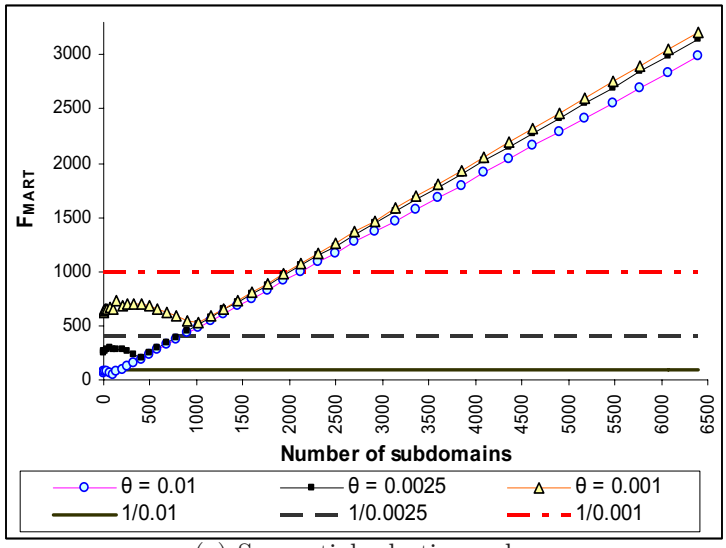
Suppose every element of D_1 belongs to $\ddot{E}$, a test case t is selected from D_1 and L consists of t and $(h-1)$ images of t. In this case, detecting a failure by MART is equivalent to searching L for an actual failure-causing input. Chapter 4 shows that the F-measures of RT without replacement and ART without replacement approach $\frac{1}{2\theta}$ when the number of failure-causing inputs approaches 1. This is why random order and adaptive random order both have F_{MART} close to $\frac{1}{2h}$ ($\approx \frac{1}{2\theta}$) when $\ddot{E}\theta = 1$, and there is little or no overlapping (little overlapping means that L contains a very small number of failure-causing inputs).

$\mathbf{X}_n\mathbf{Y}_n$	h	θ= **0.01**		θ= **0.0025**		θ= **0.001**	
n =		$\frac{\lvert F\rvert}{\lvert D_1\rvert}$	$\ddot{E}\theta$	$\frac{\lvert F\rvert}{\lvert D_1\rvert}$	$\ddot{E}\theta$	$\frac{\lvert F\rvert}{\lvert D_1\rvert}$	$\ddot{E}\theta$
1	1	0.01	0.01	0.0025	0.0025	0.0010	0.0010
2	4	0.04	0.04	0.01	0.01	0.0040	0.0040
4	16	0.16	0.16	0.04	0.04	0.0160	0.0160
6	36	0.36	0.36	0.09	0.09	0.036	0.036
8	64	0.64	0.64	0.16	0.16	0.064	0.064
10	100	***1.00**	***1.00**	0.25	0.25	0.10	0.10
12	144	1.44	1.00	0.36	0.36	0.144	0.144
14	196	1.96	1.00	0.49	0.49	0.196	0.196
16	256	2.56	1.00	0.64	0.64	0.256	0.256
18	324	3.24	1.00	0.81	0.81	0.324	0.324
20	400	4.00	1.00	***1.00**	***1.00**	0.400	0.400
22	484	4.84	1.00	1.21	1.00	0.484	0.484
24	576	5.76	1.00	1.44	1.00	0.576	0.576
26	676	6.76	1.00	1.69	1.00	0.676	0.676
28	784	7.84	1.00	1.96	1.00	0.784	0.784
30	900	9.00	1.00	2.25	1.00	***0.900**	***0.900**
32	1024	10.24	1.00	2.56	1.00	***1.024**	***1.000**
34	1156	11.56	1.00	2.89	1.00	1.156	1.000
36	1296	12.96	1.00	3.24	1.00	1.296	1.000
38	1444	14.44	1.00	3.61	1.00	1.444	1.000
40	1600	16.00	1.00	4.00	1.00	1.600	1.000
42	1764	17.64	1.00	4.41	1.00	1.764	1.000
44	1936	19.36	1.00	4.84	1.00	1.936	1.000
46	2116	21.16	1.00	5.29	1.00	2.116	1.000
48	2304	23.04	1.00	5.76	1.00	2.304	1.000
50	2500	25.00	1.00	6.25	1.00	2.500	1.000
52	2704	27.04	1.00	6.76	1.00	2.704	1.000
54	2916	29.16	1.00	7.29	1.00	2.916	1.000
56	3136	31.36	1.00	7.84	1.00	3.136	1.000
58	3364	33.64	1.00	8.41	1.00	3.364	1.000
60	3600	36.00	1.00	9.00	1.00	3.600	1.000
62	3844	38.44	1.00	9.61	1.00	3.844	1.000
64	4096	40.96	1.00	10.24	1.00	4.096	1.000
66	4356	43.56	1.00	10.89	1.00	4.356	1.000
68	4624	46.24	1.00	11.56	1.00	4.624	1.000
70	4900	49.00	1.00	12.25	1.00	4.900	1.000
72	5184	51.84	1.00	12.96	1.00	5.184	1.000
74	5476	54.76	1.00	13.69	1.00	5.476	1.000
76	5776	57.76	1.00	14.44	1.00	5.776	1.000
78	6084	60.84	1.00	15.21	1.00	6.084	1.000
80	6400	64.00	1.00	16.00	1.00	6.400	1.000

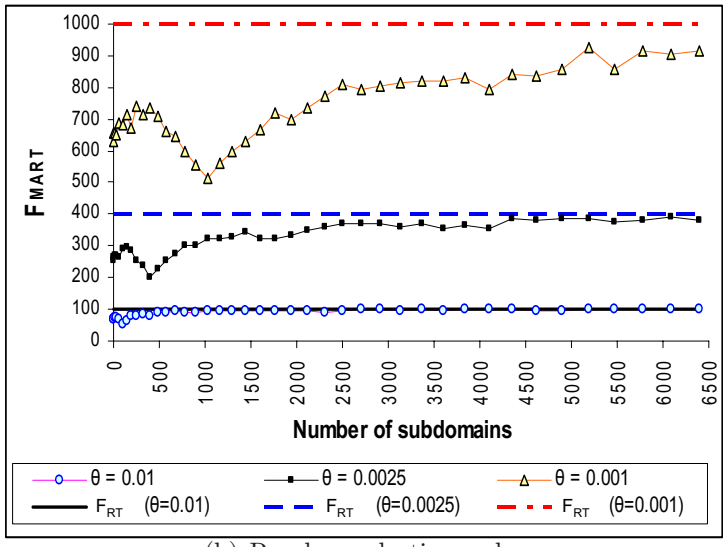
Table 7.8: h, $\ddot{E}\theta$, $\frac{\lvert F\rvert}{\lvert D_1\rvert}$ under different mirror partitionings

$\mathbf{X}_n\mathbf{Y}_n$	h	θ = **0.01**			θ = **0.0025**			θ = **0.001**		
		SEO	**RO**	**ARO**	**SEO**	**RO**	**ARO**	**SEO**	**RO**	**ARO**
1	1	66	66	66	252	252	252	628	628	628
2	4	70	68	68	264	264	264	640	658	658
4	16	70	72	73	270	271	271	648	657	661
6	36	74	75	74	276	270	265	647	651	655
8	64	66	66	65	298	267	274	662	686	678
10	100	***52**	***51**	***51**	275	292	272	656	683	670
12	144	73	66	61	283	299	280	726	715	729
14	196	98	77	69	284	287	283	684	674	705
16	256	126	79	75	266	256	264	695	742	666
18	324	158	83	72	229	238	234	702	715	717
20	400	193	81	63	***205**	***200**	***204**	705	735	695
22	484	234	91	66	247	229	226	691	710	665
24	576	276	92	70	293	256	252	654	660	680
26	676	324	95	69	341	275	260	627	645	634
28	784	374	92	69	394	300	291	596	596	586
30	900	428	91	67	453	303	300	541	557	560
32	1024	486	94	67	513	323	299	***524**	***514**	***531**
34	1156	548	95	70	579	325	294	590	562	570
36	1296	614	93	70	647	327	269	660	597	557
38	1444	682	95	70	722	346	263	734	632	597
40	1600	754	96	68	798	324	239	813	668	645
42	1764	832	93	68	876	324	252	894	717	674
44	1936	912	96	70	963	334	260	982	701	687
46	2116	995	95	69	1051	349	264	1069	733	684
48	2304	1082	92	73	1144	361	266	1163	774	733
50	2500	1172	95	69	1237	372	271	1261	810	725
52	2704	1269	102	70	1337	373	269	1363	795	737
54	2916	1367	100	67	1443	373	273	1470	803	712
56	3136	1468	97	72	1550	359	269	1581	814	720
58	3364	1575	102	70	1661	371	268	1693	818	694
60	3600	1684	97	65	1776	354	248	1813	822	645
62	3844	1797	101	67	1897	364	265	1935	832	604
64	4096	1912	100	71	2021	356	258	2058	796	615
66	4356	2034	101	69	2147	384	262	2188	840	616
68	4624	2159	93	70	2277	384	276	2322	837	667
70	4900	2286	94	70	2413	388	268	2460	856	624
72	5184	2418	103	68	2553	384	262	2601	924	653
74	5476	2554	99	70	2694	377	268	2749	855	662
76	5776	2694	99	70	2843	379	265	2898	915	692
78	6084	2834	103	68	2992	390	261	3051	904	640
80	6400	2981	99	69	3146	383	257	3208	915	673

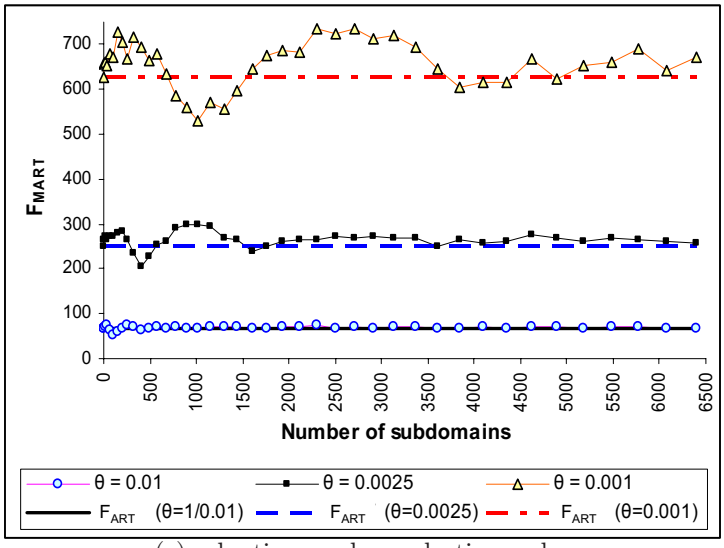
Table 7.9: F_{MART} for various numbers of subdomains (h), failure rates (θ) and mirror selection orders (SEQ, RO and ARO denote sequential, random and adaptive random order, respectively), where the program failure pattern consists of a square failure region



(a) Sequential selection order



(b) Random selection order



(c) adaptive random selection order

Figure 7.13: F_{MART} for various numbers of subdomains ($1 \leq h \leq 6400$) when ρ is a square region

A linear search for a failure-causing input in L will make F_{MART} proportional to h. When ρ is one single square failure region, the simulation data shows that F_{MART} for the sequential order is about $\frac{h}{2}$.

Basically, the above results show that when $\ddot{E}\theta$ is approximately equal to 1, and little or no overlapping occurs, F_{MART} can be around $\frac{1}{2\theta}$, which is smaller than F_{ART}. Before $\ddot{E}\theta$ reaches 1, F_{MART} mainly depends on $F_{\widetilde{MART}}$. When $\ddot{E}\theta = 1$, even in the presence of overlapping, a good choice of selection order may make F_{MART} close to F_{ART} when $h > \frac{1}{\theta}$.

7.6.2.2 Strip and point pattern

Though the above simulations only examines the square failure region type, they do provide some insights into the impact that h, $\ddot{E}\theta$, degree of overlapping and selection order have on F_{MART}.

The same experiment was conducted to examine the case where ρ was a strip type (a rectangle whose edge lengths are in the ratio of 1:5) and point type (10 equally-sized squares). θ was set to 0.01. Note that in this experiment, the failure patterns and the input domain have the same orientation. The results are plotted in Figure 7.14, and the raw data are given in Table 7.10.

Obviously, the degree of overlapping, $\ddot{E}\theta$, and $\ddot{E}\rho$ for strip and point pattern are very complex to analyze. Since the shapes of failure regions are set to be rectangular in this study, when the edge length of the subdomains is the same as the shortest edge length of a failure region, it is known that $\ddot{E}\theta$ will be 1. Such scenarios occur with $X_{24}Y_{24}$ and $X_{32}Y_{32}$ mirror partitionings for the strip and point pattern, respectively.

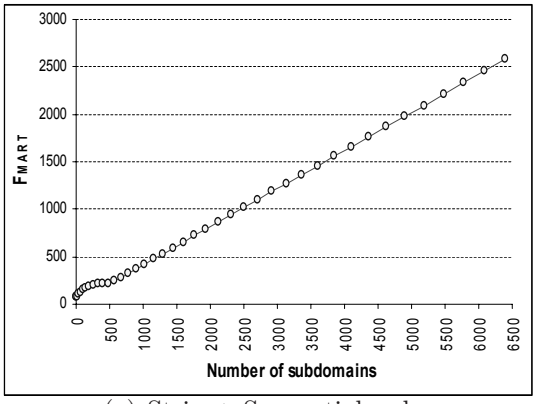
In this study, obtaining $\ddot{E}\theta = 1$ without overlapping is extremely difficult for the point pattern (because 10 square regions are randomly placed in D), but impossible for the strip pattern (because all subdomains are square but ρ is a rectangle). With a substantial amount of overlapping, neither strip nor point pattern will have $F_{MART} \approx \frac{1}{2\theta}$ for any h.

Figure 7.14 shows that when $\ddot{E}\theta$ is 1 [2], depending on the selection order used, F_{MART} is either proportional to h (about $\frac{h}{2.5}$ and $\frac{h}{14}$ for the strip and point pattens, respectively), or approaching F_{RT} (theoretically known to be 100) or F_{ART} (equiva-

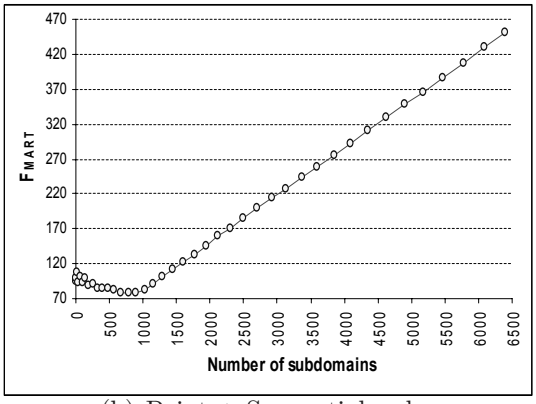
[2] $\ddot{E}\theta$ will definitely be 1 when $h > 576$ and $h > 1024$ for the strip and point pattern, but could be 1 for other mirror partitionings, too.

$\mathbf{X}_n\mathbf{Y}_n$	h	**Strip**			**Point**		
n =		**SEO**	**RO**	**ARO**	**SEO**	**RO**	**ARO**
1	1	75.89	75.89	75.89	95.90	95.90	95.90
2	4	83.84	78.75	78.72	99.28	104.60	104.58
4	16	82.68	80.53	78.54	107.73	116.97	110.94
6	36	101.72	97.43	96.93	94.06	98.73	99.16
8	64	127.70	119.24	120.25	102.11	108.42	105.55
10	100	154.74	133.33	136.91	92.42	100.85	90.66
12	144	173.67	142.14	149.96	98.36	99.51	99.54
14	196	184.13	144.68	144.69	89.15	94.49	91.48
16	256	193.75	146.69	136.18	90.30	96.22	98.45
18	324	210.28	140.72	131.32	84.55	93.43	89.23
20	400	210.90	116.38	108.59	83.86	95.96	90.69
22	484	215.88	91.27	77.12	83.97	90.82	89.24
24	576	241.44	93.12	75.67	83.13	93.14	93.20
26	676	283.36	94.50	77.18	78.30	87.51	93.44
28	784	324.84	99.47	82.39	78.26	92.12	94.22
30	900	375.31	95.23	82.42	79.16	94.15	90.96
32	1024	421.55	102.20	85.01	82.86	91.24	95.01
34	1156	475.11	103.66	82.64	91.32	92.88	91.02
36	1296	531.37	102.05	83.93	101.22	97.33	96.07
38	1444	591.95	104.85	84.82	111.37	100.65	97.81
40	1600	654.70	103.45	83.58	122.02	95.91	94.44
42	1764	721.24	100.17	80.28	133.12	102.18	92.25
44	1936	789.58	98.84	75.80	145.90	96.91	99.43
46	2116	860.81	94.40	83.53	159.13	94.88	95.23
48	2304	939.45	99.29	83.34	171.47	100.25	92.03
50	2500	1017.42	98.23	82.07	185.76	97.09	90.67
52	2704	1099.12	101.31	80.66	199.27	103.08	93.00
54	2916	1184.59	102.25	81.59	213.49	94.83	97.39
56	3136	1272.88	100.21	81.17	227.80	96.39	92.73
58	3364	1363.90	102.02	80.62	242.86	95.52	95.73
60	3600	1455.65	101.43	81.32	258.92	99.07	94.39
62	3844	1554.40	102.51	84.97	275.25	98.40	92.34
64	4096	1657.48	100.19	80.12	291.78	95.77	93.42
66	4356	1761.47	96.48	78.28	309.96	100.69	95.76
68	4624	1868.72	99.42	80.52	328.89	97.25	96.76
70	4900	1979.43	98.92	79.41	347.49	101.03	98.61
72	5184	2092.09	99.23	80.42	365.98	103.01	95.14
74	5476	2208.02	100.98	81.82	386.27	102.35	95.83
76	5776	2329.75	100.18	83.11	407.61	101.38	99.06
78	6084	2453.06	102.64	79.22	429.33	98.23	95.41
80	6400	2577.28	103.47	80.80	450.44	98.61	98.37

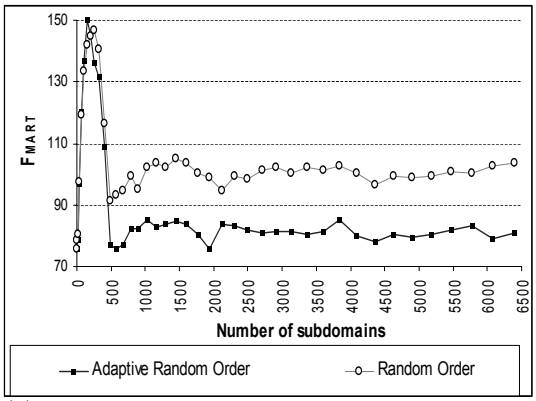
Table 7.10: F_{MART} for various numbers of subdomains (h) and mirror selection orders (SEQ, RO and ARO denote sequential, random and adaptive random order, respectively), where the failure pattern is of strip or point type



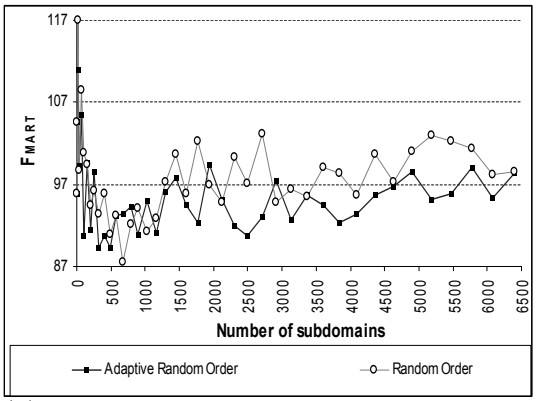
(a) Strip + Sequential order



(b) Point + Sequential order



(c) Strip + Random and Adaptive Random order



(d) Point + Random and Adaptive Random order

Figure 7.14: F_{MART} for various numbers of subdomains ($1 \leq h \leq 6400$) for strip and point patterns when $\theta = 0.01$.

lent to F_{MART} for X_1Y_1 mirror partitioning). Obviously, the adaptive random order is the best choice for MART when h is large, followed by the random order, and then the sequential order.

Before $\ddot{E}\theta$ reaches 1, there is a high peak in the curve of Figure 7.14(c), which is due to a large amount of overlapping. This result shows that when $\ddot{E}\theta < 1$, F_{MART} is significantly affected by the degree of overlapping.

7.6.3 Summary

Previous subsections show that when $\ddot{E}\theta$ is 1, a good choice of selection order may avoid a large F_{MART}. But, before $\ddot{E}\theta$ reaches 1, it is necessary to reduce or avoid overlapping in order to have a small F_{MART}. Obviously, reducing overlapping is not a easy task because it depends on how mirror partitioning and mirror function group failure-causing inputs, which in turn depends on θ and ρ.

Intuitively speaking, the larger h is, the more likely it is that $\ddot{E}\theta$ will be 1.

However, if h is very large, then evenly spreading sibling images by a selection order could be costly. In summary, it is better not to use a very large h, and to reduce overlapping as much as possible through a good selection of mirror partitioning and mirror function.

7.7 An empirical evaluation of MART

In this section, the effectiveness of MART is studied through its application to real life programs.

In the past, Chen *et al.* [23] carried out the study of ART on 12 numeric programs. Their study and findings are briefly summarized in Table 2.1 in Section 2.2.2. The same programs are used to test the performance of MART. X, Y, Z and W denote different program input parameters whenever relevant. As a reminder, ρ in Sncndn and Golden are point type, hence ART and RT have very similar F-measures with these two programs.

	Airy	Bessj0	Erfcc	Probks	Tanh
F_{RT}	1381.44	733.96	1803.62	2634.86	557.96
F_{ART}	799.29	423.93	1004.68	1442.50	306.86
F_{MART}	X_h	X_h	X_h	X_h	X_h
$h = 2$	506.83	319.43	706.85	1040.53	228.85
$h = 3$	807.57	439.34	1045.30	1488.30	327.67
$h = 4$	491.86	312.80	725.56	1067.91	229.87
$h = 5$	811.81	418.25	993.11	1443.30	325.30
$h = 6$	532.21	307.30	721.63	1077.01	228.92
$h = 7$	786.79	451.31	1006.73	1435.97	313.03
$h = 8$	510.71	310.73	696.20	1025.98	231.41
$h = 9$	771.84	424.71	1056.59	1476.50	313.25
$h = 10$	519.18	305.63	712.10	1060.55	218.98

Table 7.11: F-measures of RT, ART and MART with translate() in 1Ð programs

Two sets of experiments were conducted, differing only in their mirror functions. One experiment used the translate() mirror function, while the other used refract(). In these two experiments, the following settings were used: h was set to be less than or equal to 10 (such settings of h result in $|D_1| >> |F|$); and partitioning was only performed in one dimension of the input domain (either X, Y, Z or W), and is termed *one-dimensional partitioning* in this context. Since h was small,

the sequential selection order was employed. Note that refract() is identical to translate() when dealing with 1Đ input domains. The results for the 1Đ programs are given in Table 7.11. Results for 2Đ, 3Đ and 4Đ programs are summarized in Tables 7.12-7.17. Tables 7.12, 7.14 and 7.16 are for the translate(), while Tables 7.13, 7.15 and 7.17 are for the refract() function. Also included are F_{RT} and F_{ART} (previously reported in Table 2.1) which serve as benchmarks in this study. For refract(), c was arbitrarily set to be (2, 2), (2, 2, 2) and (2, 2, 2) for 2Đ, 3Đ and 4Đ programs (refer to Formula 7.2 for the refract() function).

	Bessj		**Gammq**		**Sncndn**	
F_{RT}	802.17		1220.28		636.19	
F_{ART}	466.66		1081.43		628.68	
F_{MART}	X_hY_1	X_1Y_h	X_hY_1	X_1Y_h	X_hY_1	X_1Y_h
$h = 2$	492.82	453.12	1059.12	1109.53	625.15	606.28
$h = 3$	541.73	450.12	1082.18	1083.16	638.86	640.66
$h = 4$	553.10	441.35	1046.84	1077.72	627.03	615.26
$h = 5$	609.81	441.26	1129.23	1059.60	626.89	611.34
$h = 6$	622.91	437.46	1061.24	1076.67	613.79	632.26
$h = 7$	716.22	452.86	1045.83	1022.54	619.69	642.00
$h = 8$	722.35	438.13	1108.44	1004.45	646.23	640.99
$h = 9$	770.23	451.21	1096.41	947.42	636.87	611.57
$h = 10$	874.67	448.05	1151.50	889.68	653.94	623.38

Table 7.12: F-measures of RT, ART and MART with translate() in 2Đ programs

	Bessj		**Gammq**		**Sncndn**	
F_{RT}	802.17		1220.28		636.19	
F_{ART}	466.66		1081.43		628.68	
F_{MART}	X_hY_1	X_1Y_h	X_hY_1	X_1Y_h	X_hY_1	X_1Y_h
$h = 2$	492.82	453.12	1059.12	1074.53	613.38	650.71
$h = 3$	541.73	450.12	1082.18	1137.19	645.14	634.06
$h = 4$	554.19	441.35	1046.84	1084.24	645.63	650.53
$h = 5$	579.22	441.26	1129.23	1059.98	646.42	625.53
$h = 6$	623.07	437.46	1061.24	1030.85	649.75	627.98
$h = 7$	681.65	452.86	1045.83	915.71	635.25	622.53
$h = 8$	708.71	438.13	1108.44	869.96	627.43	610.36
$h = 9$	781.36	451.21	1096.41	796.71	633.26	633.82
$h = 10$	819.02	448.05	1151.50	681.62	622.85	635.31

Table 7.13: F-measures of RT, ART and MART with refract() in 2Đ programs

The following are the observations about the F_{MART} for the translate() mirror function.

	Golden			**Plgndr**		
F_{RT}	1860.81			2741.66		
F_{ART}	1829.74			1806.94		
F_{MART}	$X_hY_1Z_1$	$X_1Y_hZ_1$	$X_1Y_1Z_h$	$X_hY_1Z_1$	$X_1Y_hZ_1$	$X_1Y_1Z_h$
$h = 2$	1887.70	1525.00	1826.99	1626.10	1637.30	3162.27
$h = 3$	1848.06	2553.26	1827.28	1563.34	1771.12	4768.42
$h = 4$	1794.50	3016.02	1803.97	1574.27	1863.81	6118.38
$h = 5$	1874.24	3564.46	1703.57	1624.38	1828.62	7828.19
$h = 6$	1782.65	3877.72	1840.69	1591.62	1646.02	9373.61
$h = 7$	1933.48	4259.07	1900.55	1612.91	1579.51	10534.10
$h = 8$	1895.48	4357.91	1830.41	1635.27	1433.37	13072.90
$h = 9$	1770.35	4935.02	1801.37	1590.44	1273.61	14646.60
$h = 10$	1842.86	4889.21	1824.40	1598.64	1105.33	16274.70

Table 7.14: F-measures of RT, ART and MART with translate() in 3Ð programs

	Golden			**Plgndr**		
F_{RT}	1860.81			2741.66		
F_{ART}	1829.74			1806.94		
F_{MART}	$X_hY_1Z_1$	$X_1Y_hZ_1$	$X_1Y_1Z_h$	$X_hY_1Z_1$	$X_1Y_hZ_1$	$X_1Y_1Z_h$
$h = 2$	1904.67	2123.14	1914.70	1609.45	1544.61	1804.49
$h = 3$	1915.16	2458.63	1889.90	1592.29	1917.65	1958.39
$h = 4$	1906.84	2902.70	1826.73	1644.93	2153.90	2023.78
$h = 5$	1862.84	3642.48	1870.09	1596.54	2004.72	2168.76
$h = 6$	1848.49	3871.07	1951.35	1520.60	1937.17	2025.59
$h = 7$	1867.74	4191.88	1855.63	1588.71	1828.32	2134.59
$h = 8$	1885.62	4505.65	1849.03	1570.47	1845.48	2224.34
$h = 9$	1815.83	4837.41	1912.47	1534.58	1830.88	2196.33
$h = 10$	1818.27	5174.09	1900.90	1607.13	1679.25	2161.83

Table 7.15: F-measures of RT, ART and MART with refract() in 3Ð programs

- For all 1Ð programs, F_{MART} is very similar to F_{ART} when h is an odd number, but much smaller than F_{ART} ($F_{MART} \approx 40\% F_{ART}$) when h is an even number.

- For all 2Ð programs, MART performs similarly to ART for most of the partitionings, except that the Bessj program has F_{MART} higher than F_{ART} for the X_hY_1 partitioning.

- Consider all 3Ð programs. For Golden, when $X_hY_1Z_1$ and $X_1Y_1Z_h$ partitionings are used, $F_{MART} \approx F_{ART}$; for Plgndr, when $X_hY_1Z_1$ and $X_1Y_hZ_1$ partitionings are used, $F_{MART} \approx F_{ART}$; for one-dimensional partitionings other those mentioned above, F_{MART} increases as h increases.

- For the 4Ð programs, namely Cel and El2, F_{MART} is similar to F_{ART} for the

	Cel				**El2**			
F_{RT}	3065.25				1430.76			
F_{ART}	1607.80				686.48			
F_{MART}	X_hY_1	X_1Y_h	X_1Y_1	X_1Y_1	X_hY_1	X_1Y_h	X_1Y_1	X_1Y_1
F_{MART}	Z_1W_1	Z_1W_1	Z_hW_1	Z_1W_h	Z_1W_1	Z_1W_1	Z_hW_1	Z_1W_h
$h = 2$	3209.55	1650.98	3083.69	3103.29	735.14	1054.19	1473.28	1385.84
$h = 3$	4616.94	2005.89	4645.01	4472.73	767.05	1165.21	2139.66	2253.58
$h = 4$	6155.59	2231.75	6134.08	5840.64	822.86	1355.03	2824.57	2840.34
$h = 5$	7695.95	2507.28	7382.87	7840.12	817.10	1457.27	3637.78	3612.20
$h = 6$	9234.94	2737.87	9082.24	8886.08	877.98	1545.74	4430.86	4375.73
$h = 7$	10773.90	2700.58	10157.60	9997.69	945.01	1635.95	5186.23	5184.84
$h = 8$	12312.90	2839.18	11496.60	12488.50	1078.82	1755.62	5848.35	5813.73
$h = 9$	13851.90	3007.03	13843.60	13787.50	1074.76	1799.47	6697.39	6657.45
$h = 10$	15390.90	2890.33	15646.30	14956.00	1118.95	1853.38	7248.20	7059.08

Table 7.16: F-measures of RT, ART and MART with translate() in 4Ð programs

	Cel				**El2**			
F_{RT}	3065.25				1430.76			
F_{ART}	1607.80				686.48			
F_{MART}	X_hY_1	X_1Y_h	X_1Y_1	X_1Y_1	X_hY_1	X_1Y_h	X_1Y_1	X_1Y_1
F_{MART}	Z_1W_1	Z_1W_1	Z_hW_1	Z_1W_h	Z_1W_1	Z_1W_1	Z_hW_1	Z_1W_h
$h = 2$	1679.59	1691.30	1644.26	1609.88	763.65	750.10	725.78	707.60
$h = 3$	1671.15	1986.27	1621.62	1662.96	754.39	793.09	765.20	765.01
$h = 4$	1718.20	2276.43	1618.68	1701.21	814.83	795.92	763.84	775.94
$h = 5$	1740.04	2449.79	1662.34	1689.93	858.02	823.90	790.44	784.74
$h = 6$	1814.52	2734.76	1652.83	1728.70	887.17	891.41	821.83	805.05
$h = 7$	1845.29	2823.54	1728.50	1766.03	907.36	893.64	840.12	797.36
$h = 8$	1850.39	2808.14	1782.38	1824.33	963.18	891.89	883.82	850.06
$h = 9$	1951.89	2888.96	1796.55	1773.08	1021.01	931.01	917.88	855.01
$h = 10$	1944.14	2871.18	1891.37	1875.06	1070.64	959.83	896.85	905.89

Table 7.17: F-measures of RT, ART and MART with refract() in 4Ð programs

cases where $X_1Y_2Z_1W_1$ is applied to Cel, and $X_2Y_1Z_1W_1$ applied to El2. For other one-dimensional partitionings, F_{MART} increases as h increases, at various rates.

- When partitioning the Y axis, once h reaches 6, increasing h decreases F_{MART} for Plgndr and Gammq.

The following are the observations about F_{MART} for the refract() mirror function.

- For the 2Ð programs, refract() and translate() have comparable F_{MART}, except that when X_1Y_h partitioning is used with $h \geq 7$ for Gammq, the F_{MART} for refract() are much smaller than for translate().

- For Golden, the F_{MART} for the refract() function do not differ much from the

F_{MART} for the translate() function, except that when $X_1Y_2Z_1$ partitioning is used, refract() yields a higher F_{MART} than translate().

- For Plgndr, refract() and translate() have very similar results for $X_hY_1Z_1$. However, when the $X_1Y_1Z_h$ mirror partitioning is used, for all h, refract() has much lower F_{MART} than translate() has, and when the $X_1Y_hZ_1$ partitioning is used, for all $h > 2$, refract() has higher F_{MART} than translate() has.

- For the 4Đ programs, those cases where MART which uses translate() and performs poorly, are significantly improved by using the refract() function.

As studied in Section 7.6, there are many situations which could yield an F_{MART} smaller than F_{ART}. Examples include situations where some failure regions are close to the boundary of D_1, or when there is little overlapping and $\ddot{E}\theta \approx 1$. Referring to the 1Đ programs, with translate() as the mirror function, F_{MART} is about $40\%F_{ART}$ when h is an even number. The failure patterns for these 5 programs were investigated and it was found that they all have one failure region next to the middle point of D (that is, $X = 0$). The translate() mirror function with an even value for h produced a failure region (formed by $\ddot{E}$) attaching to the border of D_1. This explains why F_{MART} is so small for an even h.

Basically, MART performs no worse than ART when the dimensionality of D is small. But with the increase of the dimensionality, it was found that more and more mirror partitioning schemes had overlapping, and F_{MART} was larger than F_{ART}.

Consider some of the 3Đ and 4Đ programs where translate() was used. It was observed that for some one-dimensional partitionings, such as the $X_1Y_1Z_h$ mirror partitioning used in Plgndr, $F_{MART} \approx h \cdot F_{ART}$ for all $h \leq 10$. It was found that the failure patterns of these programs have a very large region spanning the dimension where the one-dimensional partitioning is applied, hence a large overlapping was created when partitioning that particular dimension and translate() function was used. Section 7.5 shows that refract() can reduce or even avoid overlapping for this kind of ρ. The simulation results do show that F_{MART} for refract() is smaller than F_{MART} for translate() function.

For most of the scenarios, refract() performs similarly or better than translate(). However, when the $X_1Y_2Z_1$ mirror partitioning is used for Golden, and when the

$X_1Y_hZ_1$ mirror partitioning is used for Plgndr with $h > 2$, refract() performs worse than translate(). After analyzing the failure patterns of these two programs, they were found to have some failure regions whose orientations coincidentally gave rise to more overlapping with the shifting operation in refract().

It is interesting to notice that the refract() and translate() functions yield many identical F_{MART} for Bessj and Gammq programs. It was found that the majority of failure-causing inputs for these two programs are in D_1. Therefore, the majority of the failures were revealed by the elements in D_1 directly rather than by their images in the mirror domains. Consequently, no matter what mirror function is used, F_{MART} will be very similar, or even identical.

Based on the above observations and discussions, the following can be concluded: (*i*) a slightly complex mirror function may reduce overlapping and hence improve the performance of MART; and (*ii*) by comparing the F_{MART} collected from slightly different mirroring schemes, it is possible to identify failure patterns (size, slope, orientation of failure regions), and to some extent, the location of the majority of the failure-causing inputs. Point (*ii*) is particularly useful for debugging the program.

7.8 Discussion and conclusion

In this chapter, a method called mirroring was proposed to reduce the overhead of ART. Although only the outcome of integrating mirroring and FSCS-ART was reported, the technique of mirroring is obviously applicable to other implementation of ART, and other areas in computer science and software engineering.

As discussed in Section 7.5, partitioning the input domain along the uncritical parameters gives rise to significant overlapping, even when h is small. When programs only involve one input parameter, the parameter is critical to program failures, unless the program is proved to be correct. On the other hand, when programs involve many parameters, it is possible that some of these parameters are uncritical to program failures. This explains why F_{MART} is more sensitive to the partitioning schemes in higher dimensional programs than in lower ones. Whenever possible, prior to the design of mirror partitioning, the program specifications and code should be analyzed to identify potential uncritical parameters. Most likely, an

uncritical parameter is a parameter which is not involved in any decision node of the program control flow, like the parameter X in Program 7.2.

Note that failure regions with the same failure rate and shape appear larger in higher dimensional input domains than in lower ones. Consider the case where both input domain D and failure region F are square. If the edge length of D is d and $|\mathrm{F}| = \theta \cdot |\mathrm{D}|$ where $0 < \theta < 1$, then the edge length f of F will be $\sqrt[N]{\theta} \cdot d$. Clearly, f increases as N increases, if d remains constant.

Intuitively speaking, the smaller the failure region is, the smaller overlapping will be. Since failure regions appear larger in higher dimensions than in lower ones, it is better to apply MART to high dimensional input domains only when θ is small. Note that the failure rates (θ) in these 12-program studies are not very small (all greater than 0.0003). This explains why the performance of MART is less good for some high dimensional programs.

After some tests have been done on the software, it shall be possible to have a better idea about θ and ρ. By then, it may be possible to design a favourable mirroring scheme for MART. Therefore, it is recommended that MART be conducted at later stages of software testing.

In conclusion, the guidelines for using MART can be stated as follows: Use a small h whenever possible, but with a proper design of the mirror partitioning and mirror function. It is okay to use a large h, but an effective mirror selection order is necessary to evenly spread the sibling images. Avoid a very large h because the cost of evenly spreading a very large number of sibling images may be as high as applying ART on the entire input domain. A good design of mirror partitioning and mirror functions requires knowledge of both θ and ρ. Therefore, it is better to apply MART at later stages of software testing, because by then, there may be better knowledge of θ and ρ available. A carefully designed mirror function may significantly reduce the degree of overlapping, and consequently avoid a large F_{MART}.

Chapter 8

Adaptive random testing in high dimensional space

The "curse of dimensionality" is a well-known problem in many disciplines. It was originally defined by Richard Bellman as the remarkable growth in the difficulty of problems as the dimension increases [3]. Consider the development of random number generators. It has long been recognized difficult to obtain a truly uniform distribution of samples in any dimensional space [44], even though recently, there are some good random number generators for reasonably high dimensional spaces, such as the Mersenne Twister algorithm [46], which has a 623-dimensional equidistribution property [46].

The current approaches of evenly spreading test cases in ART have not fully taken dimensionality into consideration. Although they all work well in low dimensions, their performance is less good in high dimensions (refer to Sections 3.1.1 and 5.2 for details).

In this chapter, the problems of ART in high dimensional space are addressed and some solutions to them are proposed. Again, FSCS-ART is chosen for this study. As a reminder, at the same failure rate, its F-measure increases as the dimensionality (N) increases.

As a reminder, θ denotes the failure rate, F denotes the whole set of failure-causing inputs, E and C denote the set of executed test cases and candidate set (a set of k random inputs) in ART, respectively. Unless otherwise specified, k is assumed to be 10 and the input domain (D) is assumed to be rectangular in this

chapter.

8.1 Problem 1 and its solution

For ease of discussion, D_{centre} is defined as the region which is located at the centre of D, and D_{edge} is defined as $\mathrm{D} \backslash \mathrm{D}_{centre}$. The shape of D_{centre} is identical to D (both rectangular) and $|\mathrm{D}_{centre}| = a|\mathrm{D}|$ where $0 < a < 1$. Note that the definition of D_{centre} given in Chapter 5 was restricted to $a = 0.5$.

Theorem 8.1. *Assume that failure-causing inputs (*F*) composes one single rectangular region inside* D*. Further assume that* $|\mathrm{D}_{centre}| = a \cdot |\mathrm{D}|$ *and the shapes of* D *and* D_{centre} *are identical.* L_i*,* $\hat{L}_i$ *and* l_i *denote the lengths of* D*,* D_{centre} *and* F *in the* i^{th} *dimension, respectively. For any* $0 < a < 1$*, if* $\forall i, l_i > \frac{L_i}{2}$ *(so* $\theta > \frac{1}{2^N}$*),*
(i) the chance (p) of picking an element of F *from* D_{centre} *is greater than* θ*, and*
(ii) the chance (q) of picking an element of F *from* D_{edge} *is smaller than* θ*.*

Proof. Suppose that for every i, $1 \leq i \leq N$, we have $l_i > \frac{L}{2}$. In other words, $l_i = \frac{x_i + L_i}{2}$, where $0 < x_i \leq L_i$. Let $|\mathrm{F}_{centre}|$ and $|\mathrm{F}_{edge}|$ denote the size of F inside D_{centre} and D_{edge}, respectively. w_i is used to denote the length of F inside D_{centre} in the i^{th} dimension. Clearly, $|\mathrm{F}_{centre}| = \prod_{i=1}^{N} w_i$ and $|\mathrm{F}_{edge}| = |\mathrm{F}| - \prod_{i=1}^{N} w_i$. Since D and D_{centre} are identical in shape, we have $\hat{L}_i = \sqrt[N]{a} \cdot L_i$. When F attaches to a corner of D, $w_i = l_i - \frac{L_i - \hat{L}_i}{2} = \frac{x_i + \hat{L}_i}{2}$. However, F can be any place of D, hence we have $w_i \geq \frac{x_i + \hat{L}_i}{2}$.
Clearly, $(x_i + \sqrt[N]{a} L_i) > (\sqrt[N]{a} x_i + \sqrt[N]{a} L_i)$ (because $0 < a < 1$ and $0 < x_i$)
$\Rightarrow \prod_{i=1}^{N} \frac{x_i + \hat{L}_i}{2} > a \cdot \prod_{i=1}^{N} \frac{x_i + L_i}{2}$
$\Rightarrow \prod_{i=1}^{N} \frac{x_i + \hat{L}_i}{2} > a \cdot \prod_{i=1}^{N} l_i$
$\Rightarrow \prod_{i=1}^{N} \frac{x_i + \hat{L}_i}{2} > a \cdot |\mathrm{F}|$
$\Rightarrow \frac{1}{a|\mathrm{D}|} \prod_{i=1}^{N} w_i > \frac{|\mathrm{F}|}{|\mathrm{D}|}$ (because $w_i \geq \frac{x_i + \hat{L}_i}{2}$)
$\Rightarrow \frac{|\mathrm{F}_{centre}|}{|\mathrm{D}_{centre}|} > \frac{|\mathrm{F}|}{|\mathrm{D}|}$
$\Rightarrow p > \frac{|\mathrm{F}|}{|\mathrm{D}|} = \theta$
As proved above, $\prod_{i=1}^{N} \frac{x_i + \hat{L}_i}{2} > a \cdot |\mathrm{F}|$
$\Rightarrow |\mathrm{F}| - \prod_{i=1}^{N} \frac{x_i + \hat{L}_i}{2} < |\mathrm{F}| - a|\mathrm{F}|$
$\Rightarrow \frac{1}{|\mathrm{D}|} \left(|\mathrm{F}| - \prod_{i=1}^{N} \frac{x_i + \hat{L}_i}{2} \right) < \frac{1-a}{|\mathrm{D}|} |\mathrm{F}|$
$\Rightarrow \frac{1}{(1-a)|\mathrm{D}|} \left(|\mathrm{F}| - \prod_{i=1}^{N} \frac{x_i + \hat{L}_i}{2} \right) < \frac{|\mathrm{F}|}{|\mathrm{D}|}$ (remark: $(1 - a) > 0$)

$\Rightarrow \frac{1}{(1-a)|\mathrm{D}|}(|\mathrm{F}| - \prod_{i=1}^{N} w_i) < \frac{|\mathrm{F}|}{|\mathrm{D}|}$ (because $w_i \geq \frac{x_i + \hat{L}_i}{2}$)
$\Rightarrow \frac{|\mathrm{F}_{edge}|}{|\mathrm{D}_{edge}|} < \frac{|\mathrm{F}|}{|\mathrm{D}|}$
$\Rightarrow q < \frac{|\mathrm{F}|}{|\mathrm{D}|} = \theta$ □

Normally, if we select test cases from D, the chance of detecting failures is θ. If F composes one single rectangular region, when $\theta > \frac{1}{2^N}$, Theorem 8.1 shows that the chance of detecting failures is higher for test cases selected from D_{centre} than those selected from D_{edge}. This theorem is valid irrespective of the size of D_{centre}. Note that Theorem 8.1 does not imply that when $\theta \leq \frac{1}{2^N}$, the test cases from D_{centre} have a lower chance of detecting failures than those from D_{edge}.

With the results of Chapter 5 and Theorem 8.1, it is now possible to study the impact of N on the effectiveness of ART. An increase of dimensionality not only increases the edge bias of FSCS-ART, but also increases the likelihood of satisfying $\theta > \frac{1}{2^N}$ because $\frac{1}{2^N}$ exponentially decreases as N increases. Note that when $\theta > \frac{1}{2^N}$, test cases from D_{centre} will have a higher chance of detecting failures than those from the whole D. Therefore, FSCS-ART performs worse than RT for large values of θ, in high dimensional space. The higher the dimensionality (N), the larger the ART F-ratio is, and the wider range of θ where the ART F-ratio is greater than 1 (note: this scenario is also true for RRT). On the other hand, the centre bias of RPRT increases with N, therefore the opposite performance trends have been observed with RPRT as for FSCS-ART in Section 5.2.

This study shows that different inputs can have different probabilities of detecting failures, and that these failure detection probabilities have an impact on the ART F-ratio. Intuitively speaking, at small values of θ, test cases will have more similar failure detection probabilities, and hence the ART F-ratio will depend more on how evenly spread the test cases are in an ART method.

8.1.1 Solution

In view of the above observations, an attempt was made to reduce the edge bias of FSCS-ART in order to enhance its fault-detection effectiveness. The existing FSCS-ART method was refined as follows.

The candidate set (C) is selected from a sub-region D_i of D, where D_i and D share the same centre (denoted O). Among C, the candidate farthest away from all

the executed test cases (E) is selected as the next test case. The size ($|D_i|$) of D_i determines how far candidates can be spread out from O, and hence how far the next test case can be apart from O. Obviously, in this way, more test cases would be selected from the central part of D than with the original FSCS-ART. This approach of selecting test cases is called *FSCS-ART with filtering by region*.

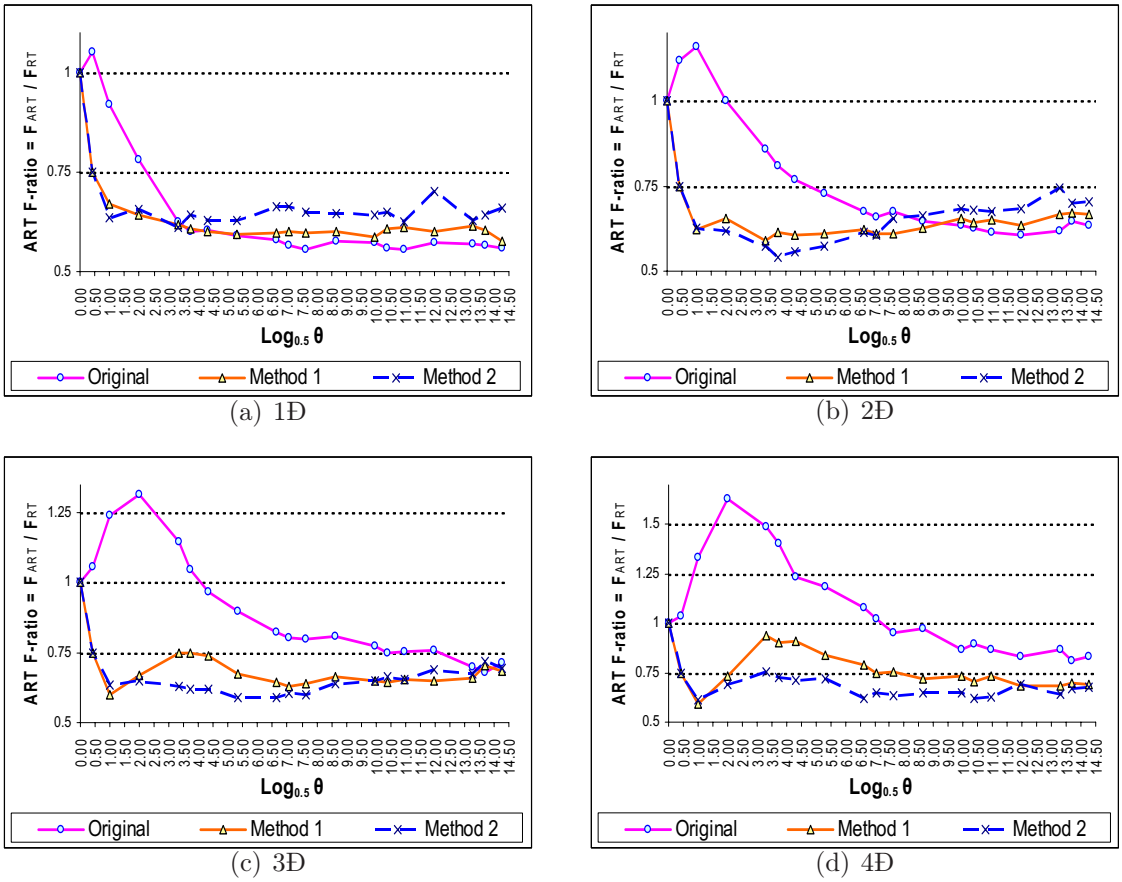
Obviously, there are many ways to implement FSCS-ART with filtering by region. The following introduces two implementations. Ten sub-regions are defined, namely D_i, $1 \leq i \leq 10$. Each D_i is assumed to have its edge length equal to $\sqrt[N]{\frac{i}{10}}$ of the edge length of D in each dimension. Obviously, $|D_i| = \frac{i}{10}|D|$ (so $D = D_{10}$) and $D_i \subset D_{i+1}$ $\forall i = 1...9$.

For the first n test cases where $n \leq N$, C is arbitrarily constructed from D_5. After that, C is constructed either from any of $\{D_5, D_6...D_{10}\}$ (referred to as Method 1), or from any of $\{D_1, D_2...D_{10}\}$ (referred to as Method 2). Clearly, Method 2 is more centrally oriented than Method 1 because the chance of constructing C from D_{10} is lower in Method 2 ($\frac{1}{10}$) than Method 1 ($\frac{1}{6}$). The algorithms of these two methods are given in Figure 8.1 where Methods 1 and 2 differ only in Step 6.

1. Set $n = 0$ and E = { }.
2. Generate 10 subsets of D, where $|D_i| = \frac{i}{10} \cdot |D|$ and D_i is centrally located inside D
3. Randomly select a test case, t, from D_5 according to the uniform distribution
4. Increment n by 1.
5. If t reveals a failure, go to Step 10; otherwise, store t in E.
6. If $|E| \leq N$, let $m = 5$, otherwise let m be any random number in [5, 10].
   /**
   Line 6 becomes if $|E| \leq N$, let $m = 5$, otherwise let m be any random number in [1, 10] for Method 2.
   **/
7. Randomly choose k candidates from D_m according to the uniform distribution in order to form the candidate set C.
8. Find c ∈ C, which has the longest distance to its nearest neighbour in E.
9. Let t = c, go to Step 4.
10. Return n and t. EXIT.

Figure 8.1: Methods 1 and 2 for FSCS-ART with filtering by region

A simulation was conducted to compare the effectiveness of Methods 1 and 2 with that of the original FSCS-ART. A square D was set to contain a single square



(a) 1Đ (b) 2Đ

(c) 3Đ (d) 4Đ

Figure 8.2: Comparison between FSCS-ART with filtering by region and the original FSCS-ART

failure region. The simulation results are reported in Figure 8.2. Based on these results, the following conclusions have been drawn.

- When θ is large enough, both Methods 1 and 2 have lower F-measures than the original FSCS-ART has.
- There exists a value v such that when $\theta > v$, Method 2 performs better than Method 1; while $\theta \leq v$, Method 1 performs better than Method 2. Note that v decreases as N increases.
- When $N \leq 2$, there exists a value v_{m1} such that when $\theta > v_{m1}$, Method 1 performs better than the original FSCS-ART; while $\theta \leq v_{m1}$, the trend is the opposite. There also exists a value v_{m2} such that the performance comparison between Method 2 and the original FSCS-ART is similar to the above comparison for Method 1 and the original FSCS-ART. Note that v_{m1} and v_{m2} can be different, and both v_{m1} and v_{m2} decrease as N increases. Although

such v_{m1} and v_{m2} were not observed for $N > 2$, it is conjectured that these two values do exist, but that they were too small to be inside the θ range studied in this simulation (the studied range of θ is [0.00005, 1]).

The last two points show that assigning more test cases to the central part of D is an effective way of detecting failures for large θ, but an ineffective way for small θ. This result is expected because the smaller the θ, the smaller the difference between the failure detection probabilities of the inputs.

Based on the above observation, testers may be greatly interested to know when they should assign more test cases to the central part of D, and when they should not. Theorem 8.1 shows that when $\theta > \frac{1}{2^N}$, test cases from D_{centre} have a higher chance of detecting failures than those from D_{edge}. However, it does not imply that when $\theta \leq \frac{1}{2^N}$, the test cases from D_{centre} have a lower chance of detecting failures than those from D_{edge}. Figure 8.2 does not show any interesting characterization corresponding to $\theta \approx \frac{1}{2^N}$, except that v, v_{m1} and v_{m2} all depend on N. v, v_{m1} and v_{m2} all are much smaller than $\frac{1}{2^N}$. Theorem 8.1 and the simulation results suggest that the favourable scenarios for Methods 1 and 2 become more likely to occur as N increases.

Although the study here has assumed F to be rectangular in shape, it does not mean Methods 1 and 2 will work poorly for other kinds of failure patterns. It is clear from the proof of Theorem 8.1 that if the proportions of F in D_{centre} and D_{edge} are in a ratio larger than $|\mathrm{D}_{centre}| : |\mathrm{D}_{edge}|$, Methods 1 and 2 will outperform the original FSCS-ART.

8.2 Problem 2 and its solution

Another problem with FSCS-ART is related to the metric, the Euclidean distance, used to select the best candidate test case. When D is 1 dimensional, no matter where points (inputs) are located, they will all appear in one line, therefore, merely keeping test cases apart in *distance* is sufficient to achieve an even spread of test cases. However, when D is N dimensional, the spatial distribution of points is more complicated. If FSCS-ART only aims at keeping test cases apart, it is possible that test cases which are apart from each other, may form certain patterns (such as lines

or hyperplanes) which are considered bad from the perspective of an even spread of test cases.

Consider the case where C = {c_1, c_2} (that is, $k = 2$) and $N = 2$. Assume that c_1 and c_2 are of equal distance from their nearest neighbour in E. In other words, they both are entitled to be the next test case according to the existing selection criterion used in FSCS-ART. Further assume that there exists an element e_i of E such that e_i and c_1 have the same coordinate in a dimension, but no such a relationship exists between c_2 and any element in E. It is argued that c_2 is preferable to c_1 as the next test case, because of the following two reasons.

- In addition to keeping test cases apart, intuitively speaking, getting test cases which are dissimilar in all dimensions should *cover* larger parts of the input space than allowing test cases to be similar in some dimensions. From a *testing coverage* point of view, c_2 should be preferable to c_1.

- Consider the case where the program failures depend on only some of the program input parameters (like Program 7.2 in Section 7.5). Since failures are only sensitive to Y in Program 7.2, if test case t fails to detect a failure, it is better for the later test cases to be different from t, with respect to Y. Since it is not normally known in advance which parameters (dimensions) the program failures depend on, to effectively detect failures in programs, it is better for the next test case to be as different from all elements of E (non failure revealing executed test cases) as possible. This should be not just with respect to the "Euclidean distance", but also with respect to the "value in each program input parameter". From this perspective, c_2 is preferable to c_1.

In summary, it has been shown that when $N > 1$, using only "distance" as the selection criterion may generate test cases which are neither evenly spread nor effective in detecting failures. The following solution is proposed to address this problem.

8.2.1 Solution

For ease of discussion, the following notations and concepts are introduced. In N dimensional D, the co-ordinates of two points A and B are denoted as (a_1, a_2...a_N) and (b_1, b_2...b_N), respectively. dist(A, B) denotes the Euclidean distance between

point A and point B, and dist_i(A, B) denotes $|a_i - b_i|$ with respect to the i^{th} dimension. Among all dist_i(A, B), the shortest and the longest distance are denoted as minDist(A, B) and maxDist(A, B), respectively. Figure 8.3 illustrates the above concepts for a 2 dimensional space, with minDist(A, B) being dist_2(A, B) and maxDist(A, B) being dist_1(A, B). Finally, DistRatio(A, B) is defined as the ratio of minDist(A, B) to maxDist(A, B).

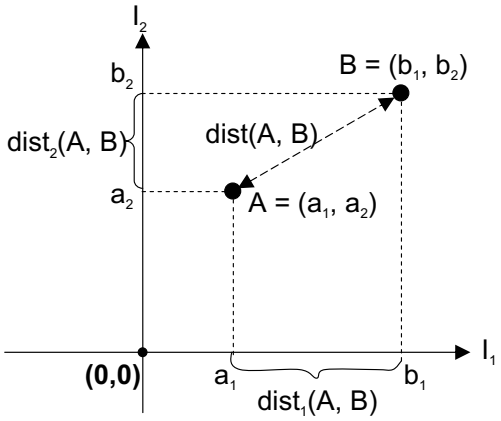


Figure 8.3: Notations

The enhanced FSCS-ART is basically the same as the original FSCS-ART, but with one additional feature: a filtering process which checks the *eligibility* of the elements of C to be a test case. This filtering process is to ensure that the eligible candidates are far apart from previously executed test cases in terms of the "input parameters". A candidate c is eligible if, for every e_i of E, DistRatio(c, e_i) is smaller than v where v is a value chosen in the range [0, 1]. For ease of discussion, the condition that determines the eligibility of a candidate is called the *eligibility criterion*. The above enhanced FSCS-ART is named *FSCS-ART with filtering by eligibility*. In the following paragraphs, this proposed method will be examined in detail. Without loss of generality, the method will be explained using a 2 dimensional space.

First of all, the distribution of eligible inputs inside the input domain is investigated. Consider e as the only element in E, which is intersected by Lines A, B, C and D having the slope of v, $-v$, $\frac{-1}{v}$ and $\frac{1}{v}$, respectively. Figure 8.4 illustrates that the eligible inputs are separated from the ineligible inputs by Lines A, B, C and D associated with v.

Secondly, how v and the size of E (|E|) affect the number of eligible inputs is investigated. Suppose D consists of 49 elements and |E| = 1 as shown in Figure

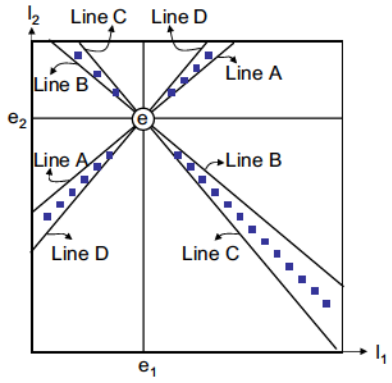


Figure 8.4: Eligible inputs (black spots) with respect to v and e (an element of E)

8.5. There are 0, 20 and 36 elements out of 49 elements, which are eligible when v = tan(45°), tan(30°) and tan(15°), respectively. Obviously, the number of eligible inputs increases as v decreases. On the other hand, for a fixed v, the growth of E will "exclude" more and more elements from being eligible. Figure 8.6 illustrates the case where v remains unchanged but the number of elements in E is different (|E| = 1 in Figure 8.6(a) and |E| = 2 in Figure 8.6(b)).

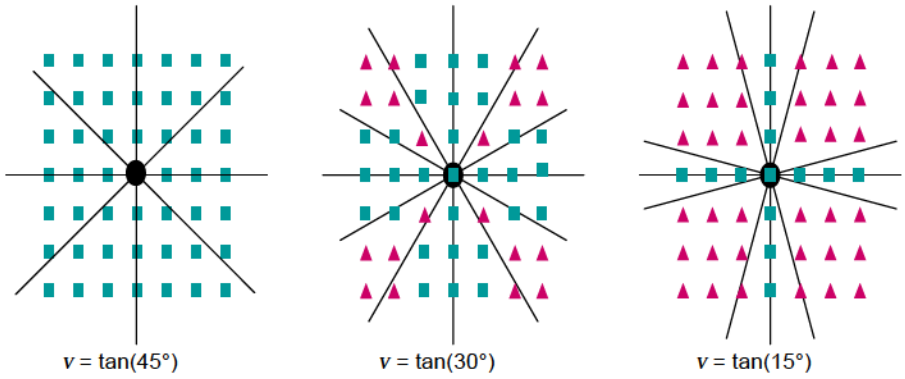
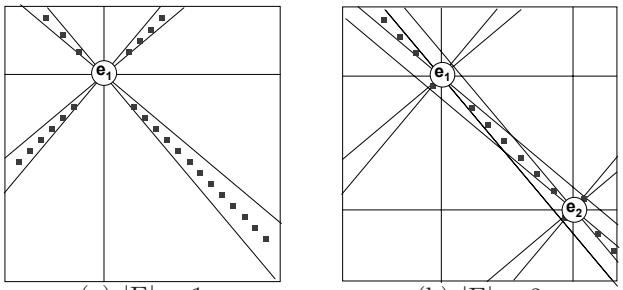


Figure 8.5: The relationship between v and the number of eligible inputs (represented by triangles)

For ease of discussion, an *inclusion region* refers to the region in which eligible inputs lie, while an *exclusion region* refers to the region in which ineligible inputs lie.

The detailed algorithm of FSCS-ART with filtering by eligibility is given in Figure 8.7 where Steps 6-14 are introduced to replace Step 5 of Figure 2.2 (the original FSCS-ART algorithm). Basically, it is necessary to construct a candidate set C such that all its elements are eligible. In C, the element farthest away from



(a) |E| = 1 (b) |E| = 2

Figure 8.6: The relationship between |E| and the number of eligible inputs (black spots)

all elements of E is chosen as the next test case.

To use FSCS-ART with filtering by eligibility, the tester needs to provide the parameters v and r. The role of v has been explained above, and the role of r is explained as follows. Since E expands throughout testing, eventually, either no eligible inputs can be identified as elements of C, or it is too expensive to construct C. To resolve this problem, it is proposed to dynamically relax the eligibility criterion during the testing process when insufficient numbers of eligible candidates have been generated after g attempts. The role of r, which is within the range (0, 1), is to reduce the value of v (by resetting v to be $v \cdot r$) so that the eligibility criterion will be relaxed. The detailed algorithm of FSCS-ART with filtering by eligibility is given in Figure 8.7.

Clearly, the larger the value g is, the longer time it takes to decide whether adjusting v is required or not. g was arbitrarily set to 4. Since the effect of the filtering disappears when $v = 0$, v is adjusted only if necessary. Here, if C has at least 70% of its elements eligible, this means that the eligibility criteria used so far are not too strict and hence there is no need to reduce v. On the other hand, if fewer than 70% candidates are eligible, the eligibility criteria used so far could be too strict and hence there is a need for reducing v.

It is interesting to know how the effectiveness of FSCS-ART with filtering by eligibility is affected by v and r. This was investigated by conducting some simulations.

The results with respect to the setting $v = 0.5$ ($\approx tan(26.57°)$) and $r = 0.5$ for $N = 2$, 3 and 4 are shown in Figure 8.8. Under this setting, the experimental data

Define n as the number of tests conducted so far.
{
1. Input v and r, where $0 < r < 1$ and $0 \leq v \leq 1$
2. Set $n = 0$, E = { } and C = { }.
3. Randomly select a test case, t, from D according to the uniform distribution.
4. Increment n by 1.
5. If t reveals a failure, go to Step 18; otherwise, store t in E.
6. Randomly generate k inputs according to the uniform distribution, in order to construct C.
7. For each $c_i \in$ C, examine the eligibility of c_i.
   mark c_i "eligible" or "ineligible" accordingly.
8. If all elements of C are eligible, go to Step 15.
9. Set nTrial = 0.
10. Do Steps 11-14 until all c_i of C are eligible
11. Replace each ineligible c_i by another random input.
12. Examine the eligibility of all replacements, and mark them "eligible" or "ineligible" according to the updated value of v.
13. Increment nTrial by 1.
14. After 4 attempts (when nTrial = 4),
    if fewer than 70% of candidates are eligible, then set nTrial = 0 and $v = v \cdot r$.
15. For each $c_i \in$ C, calculate the distance d_i between c_i and its nearest neighbour in E.
16. Find $c_b \in$ C such that its $d \geq d_i$ where $k \geq i \geq 1$.
17. Let t = c_b and go to Step 4.
18. Return n and t. EXIT.

}

Figure 8.7: The algorithm of FSCS-ART with filtering by eligibility

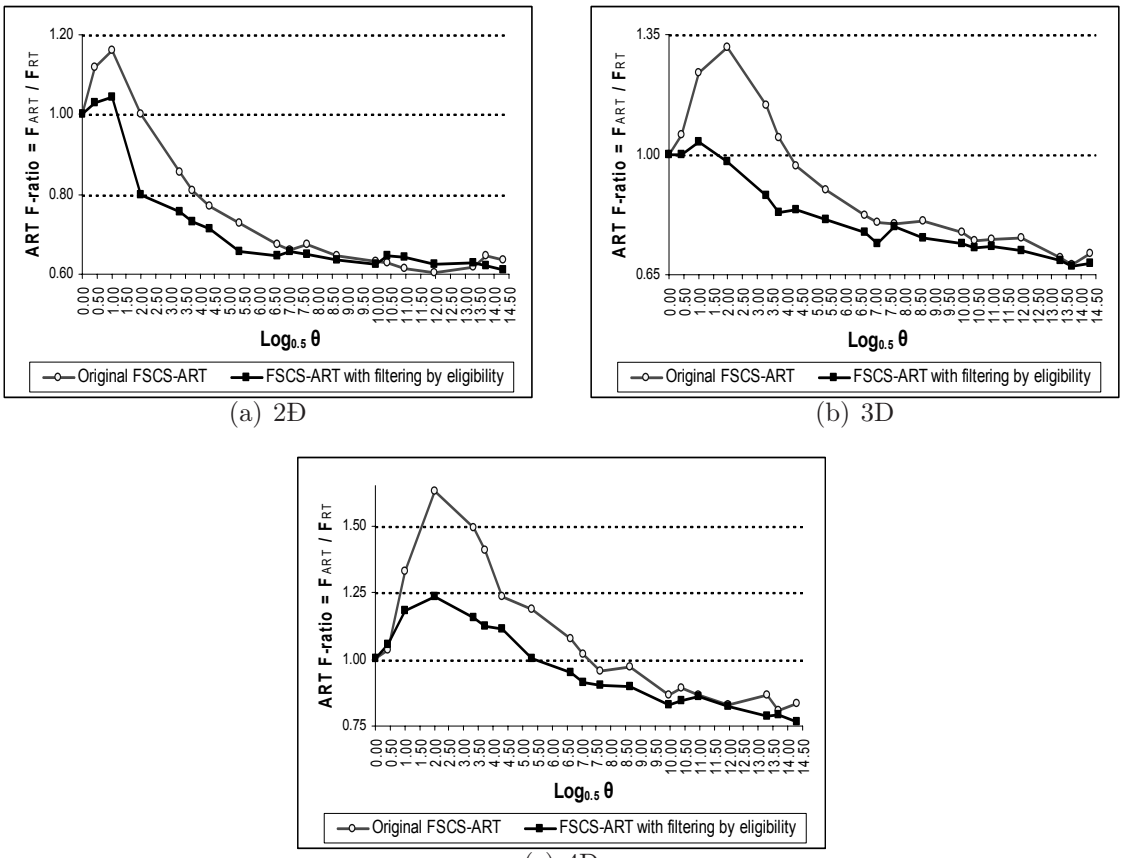
show that the process of filtering does make FSCS-ART with filtering by eligibility more effective than the original FSCS-ART. However, the degree of improvement declines as θ decreases because the F-measure of FSCS-ART approaches $\frac{1}{2\theta}$ while θ approaches 0, as explained in Chapter 4.

Further experiments were conducted with the following settings. In these experiments, D is set to be 4 dimensional.

- v is 0.9 ($\approx \tan(41.99°)$), 0.5 ($\approx \tan(26.57°)$) or 0.1 ($\approx \tan(5.71°)$)
- r is 0.9, 0.5 or 0.1

There are 9 different scenarios in total. The results from the experiments are presented in Figure 8.9. Based on these data, the following observations have been made:

- A higher r is better: it results in a smaller ART F-ratio.



(a) 2Đ (b) 3D

(c) 4Đ

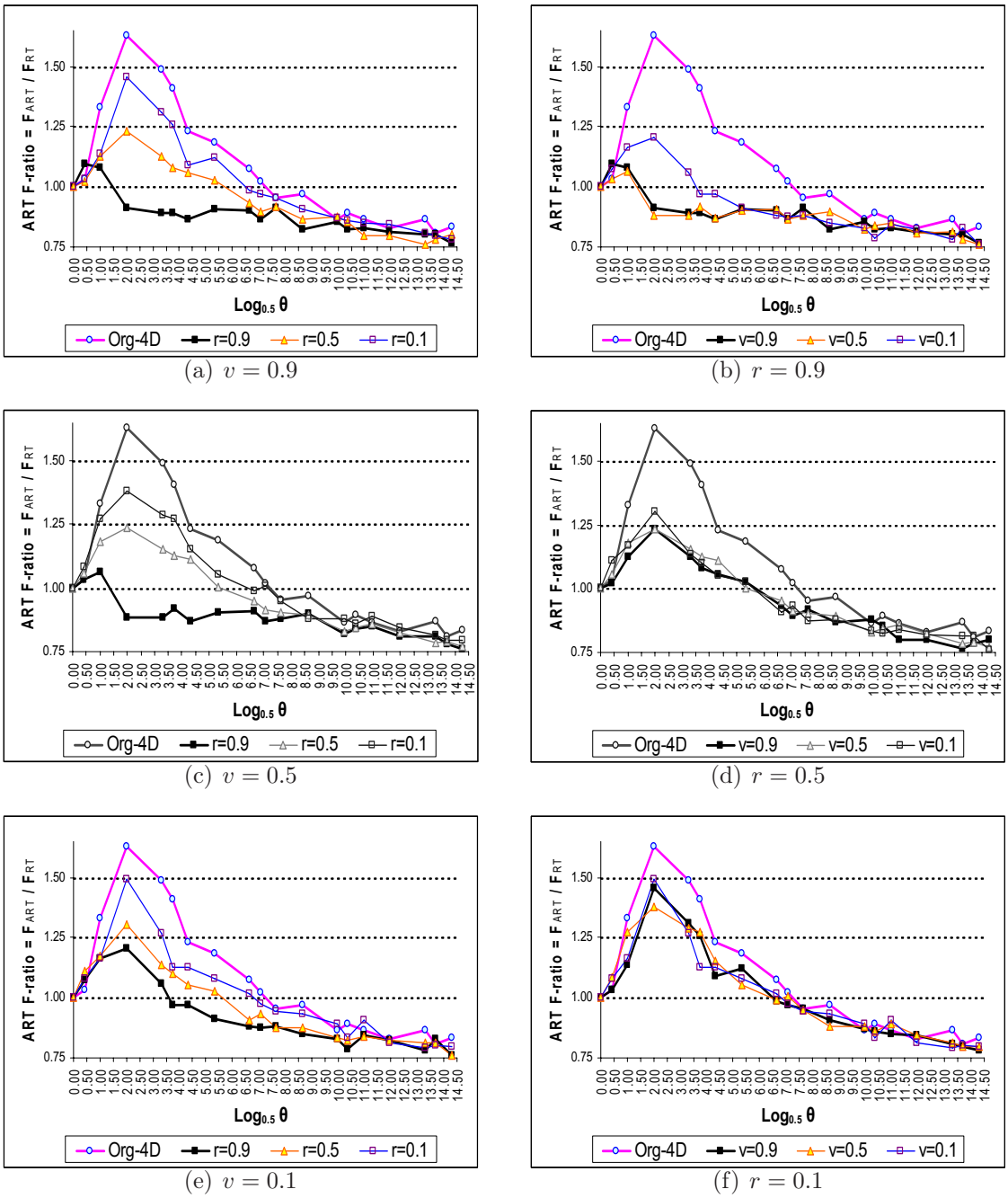
Figure 8.8: Comparison between the FSCS-ART with filtering by eligibility and the original FSCS-ART with $v = 0.5$, $r = 0.5$ and $k = 10$

- When r is 0.5 or 0.1, v seems to have no impact on the F-measure of FSCS-ART with filtering by eligibility, but when $r = 0.9$, a small v (like 0.1) makes FSCS-ART with filtering by eligibility ineffective.

- For a given r, any value of v higher than 0.5 will yield similar F-measures.

The second point is understandable because when r is small, no matter what value for v is used initially, v will become small very quickly (because v will be adjusted by r in Step 14 of the algorithm given in Figure 8.7). The above observation implies that effective FSCS-ART with filtering by eligibility needs a high initial value of v (say 0.9) and a large r (say 0.9) in order to keep v large, and its approach to 0 slow.

Another experiment was conducted to investigate the F-measure of FSCS-ART with filtering by eligibility for the situations where failures depend on some, but not all, input parameters (this is the situation motivating the development of FSCS-ART with filtering by eligibility). One single rectangular failure region was set to lie within a square N dimensional D, with edge length L. θ is 0.01 or 0.005, and N



(a) $v = 0.9$ (b) $r = 0.9$

(c) $v = 0.5$ (d) $r = 0.5$

(e) $v = 0.1$ (f) $r = 0.1$

Figure 8.9: Comparison between the FSCS-ART with filtering by eligibility and the original FSCS-ART

is 2, 3 or 4. For a given N dimensional D, the number m of *failure dependent input parameters* (input parameters which cause program failures) is set to be smaller than N, and the failure region is set to span $(N - m)$ axes. As an example, consider a 3 dimensional D, and let l_i denote the edge length of the failure region in dimension i (l_i is the value range of the failure-causing inputs with respect to the i^{th} parameter). Suppose θ is 0.01. If failures are caused by the 3^{rd} parameter (so $m = 1$), then $l_1 : l_2 : l_3 = L : L : 0.01L$. If failures are caused by the 2^{nd} and 3^{rd} parameters (so $m = 2$), then $l_1 : l_2 : l_3 = L : 0.1L : 0.1L$.

The simulation results are shown in Table 8.1. The results show that the smaller the value of m (fewer failure dependent input parameters), the better improvement of FSCS-ART with filtering by eligibility over FSCS-ART there is.

(a) $\theta = 0.01$

N	m	**Testing method**	**F-measure**
2	1	filtering by eligibility	69.76
		original	96.08
3	1	filtering by eligibility	77.41
		original	103.12
	2	filtering by eligibility	93.81
		original	96.56
4	1	filtering by eligibility	82.31
		original	98.37
	2	filtering by eligibility	97.98
		original	107.40
	3	filtering by eligibility	98.49
		original	104.81

(b) $\theta = 0.005$

N	m	**Testing method**	**F-measure**
2	1	filtering by eligibility	144.00
		original	189.48
3	1	filtering by eligibility	154.63
		original	197.64
	2	filtering by eligibility	175.94
		original	192.79
4	1	filtering by eligibility	160.94
		original	197.98
	2	filtering by eligibility	196.27
		original	216.47
	3	filtering by eligibility	195.59
		original	208.93

Table 8.1: Comparison between FSCS-ART with filtering by eligibility with the original FSCS-ART when failures of an N dimensional program depend on m input parameters

Filtering improves FSCS-ART not just in high dimensional cases, but also when program failures depend on some input parameters. It should be noted that both cases are quite common in real life programs. Therefore, it is suggested that FSCS-ART with filtering by eligibility should be used instead of the original FSCS-ART.

8.3 Summary

In this chapter, two problems associated with applying FSCS-ART to high dimensional space were found. The first one is related to the curse of dimensionality - the growth of dimensionality increases the chance of detecting failures if test cases are selected from D_{centre} instead of D_{edge}, but at the same time, increases the bias of

FSCS-ART to generate test cases from boundaries of D (known as the *edge bias*). The second problem is related to the use of the Euclidean distance in the selection of test cases. Solutions have been proposed to solve these problems, and it was found that these solutions do enhance the original FSCS-ART in high dimensional input domains.

Intuitively speaking, both FSCS-ART with filtering by region and FSCS-ART with filtering by eligibility have less edge bias than the original FSCS-ART. However, filtering by region may become more biased towards the central part of D as the number of elements in E grows. Nevertheless, filtering by eligibility has no tendency towards the central bias. Section 8.1 shows that in high dimensional space, to detect failure effectively for high failure rates, it is essential to reduce the edge bias. The simulation results show that both filtering approaches outperform the original FSCS-ART for high failure rates. Whereas, when θ drops to a specific value, filtering by region will become less effective than the original FSCS-ART; filtering by eligibility, however, continues to outperform FSCS-ART. The simulation data clearly show that filtering by eligibility evenly spreads test cases with less bias and hence outperforms the original FSCS-ART. It can therefore be concluded that FSCS-ART with filtering by eligibility is a more promising solution to the high dimension problem of FSCS-ART.

Although only the high dimension problems of FSCS-ART have been addressed, these problems are also typical of other ART methods. The proposed solutions, namely, selecting test cases more frequently from the central regions and enforcing successive test cases to be far away from each other in terms of other perspectives in addition to the Euclidean distance, are also applicable to other ART methods.

Chapter 9

Conclusion

Adaptive random testing (ART) has been proposed to enhance the fault detection effectiveness of random testing (RT). Its principle is to both randomly select and evenly spread test cases.

ART and RT have often been compared with respect to the F-measure, which is defined as the number of test cases required for detecting the first failure. When test cases are selected according to uniform distribution with replacement (duplicate test cases are allowed), the F-measure of RT (F_{RT}) is theoretically known to be equal to the reciprocal of the program failure rate ($\frac{1}{\theta}$). The experimentally measured F-measures (F_{ART}) of ART are often found to be smaller than F_{RT} when failure-causing inputs are clustered together. However, there remain many issues related to the use of ART in real life applications. The purpose of this thesis is to address these issues and provide solutions to the associated problems. ART can be implemented by various ways; this thesis mainly covers the *Fixed-Size-Candidate-Set ART (FSCS-ART)* implementation. The following summarizes the major findings of this thesis.

All studies conducted so far have demonstrated that ART outperforms RT when failure-causing inputs are clustered; but these studies were not deep enough to precisely describe the conditions for ART to outperform RT. In Chapter 3, the factors which have an impact on the effectiveness of an implementation of ART, namely, FSCS-ART, were studied. It was found that its F-measure depends on (1) the dimensions of the input domain N; (2) program's failure rate, θ; (3) the compactness of the failure region; (4) the number of failure regions; (5) the size of the predominant failure region (if any); and (6) the proximity of the failure region to the boundary of

the input domain. Six favourable conditions for ART have been identified: (1) when N is small; (2) when θ is small; (3) when the failure region is compact; (4) when the number of failure regions is small; (5) when a predominant region exists among the failure regions; and (6) when the failure region is near to the boundary of the input domain. The work of this thesis contributes to the knowledge about the conditions under which ART outperforms RT, and hence helps practitioners understand when it is cost-effective to apply ART instead of RT.

In Chapter 4, it was found that ART with replacement and without replacement always have very similar F-measures. Since selection without replacement is more expensive than selection with replacement, it is more economical to apply ART with replacement. Assuming ART and RT are both implemented with replacement, this study clarifies that even though ART is likely to have fewer duplicate test cases than RT, it is not the main reason for its superiority to RT, which is related to its principle of evenly spreading test cases. Finally, Chapter 4 shows that when there is only one single failure-causing input in the input domain, ART with replacement, ART without replacement and RT without replacement all have F-measures around $\frac{1}{2\theta}$.

Intuitively, ART outperforms RT ($\mathrm{F}_{ART} < \mathrm{F}_{RT}$) because of the effect of evenly spreading test cases. Thus, test case distributions of various ART methods were examined in Chapter 5. Additionally, the appropriateness of various test case distribution metrics were studied. When θ is small, the $M_{AveDist}$ and $(M_{Dispersion} - M_{AveDist})$ metrics appear to be the most appropriate indicators of an even spread of test cases, as they correctly reflect the ranking of testing methods with respect to their F-measures. It was found that the most effective ART method had the largest $M_{AveDist}$ and the smallest $(M_{Dispersion} - M_{AveDist})$.

Practical issues related to the use of ART in nonnumeric programs were addressed in Chapter 6, where the feasibility of applying ART to nonnumeric programs was demonstrated. A breakthrough was made in understanding the even spread of test cases for nonnumeric programs, and a dissimilarity scheme was developed to help ART keep test cases far apart in the nonnumeric input domain. The study showed that ART outperforms RT with respect to the F-measure for the overwhelmingly majority of the mutants created in this study.

In Chapter 7, an innovative approach to efficiently implement ART was proposed. This approach is called Mirror ART (MART). It was found that MART uses less distance computations than ART to evenly spread the test cases over the input domain. When the mirroring scheme is designed well, MART can perform as well as ART or even better. Some guidelines were given for using this method. As a byproduct, it was also found that testing programs with slightly different mirroring schemes could help in debugging the program.

In Chapter 8, two problems of ART about the dimensionality of the input domain were identified. Firstly, although the growth of dimensionality increases the chance of detecting failures for test cases selected from the central part of the input domain, it also increases the edge bias of FSCS-ART. Secondly, since the criterion used by FSCS-ART to select the best candidate only considered the distance, its appropriateness diminishes with the growth of the dimensionality. The following solutions were proposed to address these problems: (*i*) selecting test cases more frequently from the central regions; and (*ii*) enforcing successive test cases to be different from each other, according to metrics other than just the Euclidean distance. The simulation study showed that these solutions do enhance the original FSCS-ART in high dimensional input domains.

This thesis also identifies some potential problems for future research. These problems include: how to evenly spread test cases in high dimensional input domains; how to make use of test case distribution information to guide the selection of test cases; how to design other dissimilarity metrics for nonnumeric problems; how to make use of the failure patterns in debugging; how to apply the technique of mirroring in other software testing areas.

In summary, this thesis has made significant contributions to random testing, and provided a more comprehensive understanding of adaptive random testing and its applicability.

Bibliography

[1] ACM. *Collected Algorithms from ACM*. Association for Computing Machinery, 1980.

[2] P. E. Ammann and J. C. Knight. Data diversity: an approach to software fault tolerance. *IEEE Transactions on Computers*, 37(4):418–425, 1988.

[3] R. Bellman. *Dynamic Programming*. Princeton University Press, New Jersey, 1957.

[4] D. L. Bird and C. U. Munoz. Automatic generation of random self-checking test cases. *IBM Systems Journal*, 22(3):229–245, 1983.

[5] P. G. Bishop. The variation of software survival times for different operational input profiles. In *Proceedings of the 23rd International Symposium on Fault-Tolerant Computing (FTCS-23)*, pages 98–107. IEEE Computer Society Press, 1993.

[6] B. Boehm. *Software Engineering Economics*. Prentice Hall, Englewood Cliffs, NJ, 1981.

[7] M. S. Branicky, S. M. LaV alle, K. Olson, and L. Yang. Quasi-randomized path planning. In *Proceedings of the 2001 IEEE International Conference on Robotics and Automation*, pages 1481–1487. IEEE Computer Society, 2001.

[8] Y. D. Burago and V. A. Zalgaller. *Geometric Inequalities*. Grundlehren der mathematischen Wissenschaften, Vol. 285. Springer-Verlag, 1988. Translated from the Russian by A. B. Sossinsky.

[9] F. T. Chan, T. Y. Chen, I. K. Mak, and Y. T. Yu. Proportional sampling strategy: Guidelines for software testing practitioners. *Information and Software Technology*, 38(12):775–782, 1996.

[10] K. P. Chan, T. Y. Chen, F. C. Kuo, and D. Towey. A revisit of adaptive random testing by restriction. In *Proceedings of the 28th Annual International Computer Software and Applications Conference (COMPSAC 2004)*, pages 78–85. IEEE Computer Society Press, 2004.

[11] K. P. Chan, T. Y. Chen, and D. Towey. Restricted random testing: Adaptive random testing by exclusion. *Accepted to appear in International Journal of Software Engineering and Knowledge Engineering*, 2006.

[12] K. P. Chan, D. Towey, T. Y. Chen, F. C. Kuo, and R. G. Merkel. Using the information: incorporating positive feedback information into the testing process. In *Proceedings of the 11th International Workshop on Software Technology and Engineering Practice (STEP 2003)*, pages 71–76, Amsterdam, The Netherlands, 2004. IEEE Computer Society Press.

[13] T. Y. Chen, S. C. Cheung, and S. M. Yiu. Metamorphic testing: a new approach for generating next test cases. Technical Report HKUST-CS98-01, Department of Computer Science, Hong Kong University of Science and Technology, Hong Kong, 1998.

[14] T. Y. Chen, G. Eddy, R. G. Merkel, and P. K. Wong. Adaptive random testing through dynamic partitioning. In *Proceedings of the 4th International Conference on Quality Software (QSIC 04)*, pages 79–86, Braunschweig, Germany, 2004. IEEE Computer Society Press.

[15] T. Y. Chen and D. H. Huang. Adaptive random testing by localization. In *Proceedings of the 11th Asia-Pacific Software Engineering Conference (APSEC'04)*, pages 292–298. IEEE Computer Society, 2004.

[16] T. Y. Chen and F. C. Kuo. Is adaptive random testing really better than random testing. In *Proceedings of the 1th International Workshop on Random Testing in the 2006 ACM SIGSOFT international symposium on Software testing and analysis*, pages 64–69, Maine, Portland, 2006. ACM Press.

[17] T. Y. Chen, F. C. Kuo, and R. G. Merkel. On the statistical properties of the f-measure. In *Proceedings of the 4th International Conference on Quality Software (QSIC 2004)*, pages 146–153, Brunschwig, Germany, 2004. IEEE Computer Society Press.

[18] T. Y. Chen, F. C. Kuo, and R. G. Merkel. On the statistical properties of testing effectiveness measures. *Journal of Systems and Software*, 79(5):591–601, 2006.

[19] T. Y. Chen, F. C. Kuo, R. G. Merkel, and S. P. Ng. Mirror adaptive random testing. In *3rd International Conference on Quality Software (QSIC 2003)*, pages 4–11. IEEE Computer Society, 2003.

[20] T. Y. Chen, F. C. Kuo, R. G. Merkel, and S. P. Ng. Mirror adaptive random testing. *Information and Software Technology*, 46(15):1001–1010, 2004.

[21] T. Y. Chen, F. C. Kuo, and C. A. Sun. The impact of the compactness of failure regions on the performance of adaptive random testing. *Journal of software*. Acceptance date: 3 April 2006.

[22] T. Y. Chen, F. C. Kuo, and Z. Q. Zhou. On the relationships between the distribution of failure-causing inputs and effectiveness of adaptive random testing. In *Proceedings of the 17th International Conference on Software Engineering and Knowledge Engineering (SEKE 2005))*, pages 306–311, Taipei, Taiwan, 2005.

[23] T. Y. Chen, H. Leung, and I. K. Mak. Adaptive random testing. In *Proceedings of the 9th Asian Computing Science Conference*, volume 3321 of *Lecture Notes in Computer Science*, pages 320–329, 2004.

[24] T. Y. Chen and R. Merkel. An upper bound on software testing effectiveness. submitted for publication.

[25] T. Y. Chen, P. L. Poon, S. F. Tang, and T. H. Tse. On the identification of categories and choices for specification-based test case generation. *Information and software technology*, 46(13):887–898, 2004.

[26] T. Y. Chen, P. L. Poon, and T. H. Tse. An integrated classification-tree methodology for test case generation. *International journal of software engineering and knowledge engineering*, 10(6):647–679, 2000.

[27] T. Y. Chen, P. L. Poon, and T. H. Tse. A choice relation framework for supporting category-partition test case generation. *IEEE transactions on software engineering*, 29(7):577–593, 2003.

[28] T. Y. Chen and Y. T. Yu. On the relationship between partition and random testing. *IEEE Transactions on Software Engineering*, 20(12):977–980, 1994.

[29] R. Cobb and H. D. Mills. Engineering software under statistical quality control. *IEEE Software*, 7(6):45–54, 1990.

[30] T. Dabóczi, I. Kollár, G. Simon, and T. Megyeri. Automatic testing of graphical user interfaces. In *Proceedings of the 20th IEEE Instrumentation and Measurement Technology Conference 2003 (IMTC '03)*, pages 441–445, Vail, CO, USA, 2003.

[31] R. A. DeMillo, R. J. Lipton, and F. G. Sayward. Hints on test data selection: help for the practicing programmer. *IEEE Computer*, 11(4):31–41, 1978.

[32] J. W. Duran and S. C. Ntafos. An evaluation of random testing. *IEEE Transactions on Software Engineering*, 10(4):438–444, 1984.

[33] B. S. Everitt. *The Cambridge Dictionary of Statistics*. Cambridge University Press, 1998.

[34] G. B. Finelli. Nasa software failure characterization experiments. *Reliability Engineering and System Safety*, 32(1–2):155–169, 1991.

[35] J. E. Forrester and B. P. Miller. An empirical study of the robustness of Windows NT applications using random testing. In *Proceedings of the 4th USENIX Windows Systems Symposium*, pages 59–68, Seattle, 2000.

[36] P. Godefroid, N. Klarlund, and K. Sen. Dart: directed automated random testing. In *Proceedings of ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI 2005)*, pages 213–223, 2005.

[37] J. B. Goodenough and S. L. Gerhart. Toward a theory of test data selection. *IEEE Transactions on Software Engineering*, 1(2):156–173, 1975.

[38] B. Hailpern and P. Santhanam. Software debugging, testing, and verification. *IBM Systems Journal*, 41(1):4–12, 2002.

[39] S. L. Hantler and J. C. King. An introduction to proving the correctness of programs. *ACM Computing Surveys*, 8(3):331–353, 1976.

[40] W. Hetzel. *The complete guide to software testing*. Collins, 1984.

[41] W. E. Howden. Methodology for the generation of program test data. *IEEE Transactions on Computers*, 24(5):554–560, 1975.

[42] W. E. Howden. Reliability of the path analysis testing strategy. *IEEE Transactions on Software Engineering*, 2(3):208–215, 1976.

[43] W. E. Howden. Weak mutation testing and completeness of test sets. *IEEE Transactions on Software Engineering*, 8(4):371–379, 1982.

[44] D. Knuth. *The Art of Computer Programming*, volume 2: Seminumerical Algorithms. Addison-Wesley, 3 edition, 1997.

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

[46] M. Matsumoto and T. Nishimura. Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Transactions on Modeling and Computer Simulation*, 8(1):3–30, 1 1998.

[47] A. Mili. *An introduction to Formal Program Verification*. Van Nostrand Reinhold, New York, 1985.

[48] B. P. Miller, L. Fredriksen, and B. So. An empirical study of the reliability of UNIX utilities. *Communications of the ACM*, 33(12):32–44, 1990.

[49] B. P. Miller, D. Koski, C. P. 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.

[50] E. Miller. Website testing. `http://www.soft.com/eValid/Technology/White.Papers/website.testing.html`, Software Research, Inc., 2005.

[51] G. J. Myers. *The Art of Software Testing*. Wiley, New York, second edition, 1979.

[52] N. Nyman. In defense of monkey testing: Random testing can find bugs, even in well engineered software. `http://www.softtest.org/sigs/material/nnyman2.htm`, Microsoft Corporation.

[53] T. R. D. of the Texas Legislative Council. *Data for 2001 Redistricting in Texas*. the Texas Legislative Council, Austin, Texas, 2001. Available at http://www.tlc.state.tx.us/pubspol/red2001data.pdf.

[54] T. J. Ostrand and M. J. Balcer. The category-partition method for specifying and generating functional tests. *Communications of the ACM*, 31(6):676–686, 1988.

[55] S. L. Pfleeger. *Software engineering theory and practice*. Prentic-Hall International, Inc, USA, 1998.

[56] W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling. *Numerical Recipes*. Cambridge University Press, 1986.

[57] S. Rapps and E. J. Weyuker. Selecting software test data using data flow information. *IEEE Transactions on Software Engineering*, 11(4):367–375, 1985.

[58] D. J. Richardson and L. A. Clarke. A partition analysis method to increase program reliability. In *Proceedings of the 5th International Conference on Software Engineering*, pages 244–253, Los Alamitos, California, USA, 1981. IEEE Computer Society Press.

[59] K. Sen, D. Marinov, and G. Agha. Cute: a concolic unit testing engine for c. In *ESEC/FSE-13: Proceedings of the 10th European software engineering conference held jointly with 13th ACM SIGSOFT international symposium on Foundations of software engineering*, pages 263–272, New York, NY, USA, 2005. ACM Press.

[60] D. Slutz. Massive stochastic testing of SQL. In *Proceedings of the 24th International Conference on Very Large Databases (VLDB 98)*, pages 618–622, 1998.

[61] R. Taylor and R. Hamlet. Partition testing does not inspire confidence. *IEEE Transactions on Software Engineering*, 16(12):1402–1411, December 1990.

[62] C. M. Varachiu and N. Varachiu. A fuzzy paradigm approach for the cognitive process of categorization. In *Proceedings of the 1st IEEE International Conference on Cognitive Informatics (ICCI'02)*, pages 229–232. IEEE Computer Society Press, 2002.

[63] E. J. Weyuker. On testing non-testable programs. *The Computer Journal*, 25(4):465–470, 1982.

[64] E. J. Weyuker. Axiomatizing software test data adequacy. *IEEE transactions on software engineering*, 12(12):1128–1138, 1986.

[65] E. J. Weyuker and B. Jeng. Analyzing partition testing strategies. *IEEE transactions on software engineering*, 17(7):703–711, 1991.

[66] E. J. Weyuker and T. J. Ostrand. Theories of program testing and the application of revealing subdomains. *IEEE transactions on software engineering*, 6(3):236–245, 1980.

[67] M. R. Woodward and K. Halewood. From weak to strong, dead or alive? an analysis of some mutation testing issues. In *Proceedings of the 2nd workshop on software testing, verification and analysis*, pages 152–158, Los Alamitos, California, USA, 1988. IEEE Computer Society Press.

[68] T. Yoshikawa, K. Shimura, and T. Ozawa. Random program generator for Java JIT compiler test system. In *Proceedings of the 3rd International Conference on Quality Software (QSIC 2003)*, pages 20–24. IEEE Computer Society Press, 2003.

[69] H. P. Young. Measuring the compactness of legislative districts. *Legislative Studies Quarterly*, 13(1):105–115, 1988.

[70] H. Zhu, P. A. V. Hall, and J. H. R. May. Software unit test coverage and adequacy. *ACM computing surveys*, 29(4):366–42, 1997.

[71] H. Zhu and L. Jin. *Software quality assurance and testing*. Academic Press, Beijing, 1997.

Appendix A

Algorithms of various ART implementations

Define x_i as a circular zone drawn around the executed test case e_i.
The size of each x_i is $\frac{R \cdot |D|}{|E|}$.

{

1. Input maxTrial and R where both maxTrial and $R > 0$.
2. Set $n = 0$ and E = { }.
3. Randomly select a test case, t, from the input domain according to the uniform distribution.
4. Increment n by 1.
5. If t reveals a failure, go to Step 14; otherwise, store t in E.
6. Set noOfTrial = 0.
7. $\forall i$ $e_i \in$ E, determine x_i.
8. Repeat Steps 9-13.
9. If noOfTrial = maxTrial, then set $R = R - 0.1$ and re-determine all x_i.
11. Randomly select a test case, t, from the input domain according to the uniform distribution.
12. If t is outside $\bigcup_{i=1}^{|E|} x_i$, go to Step 4.
13. Increment noOfTrial by 1.
14. Return n and t. EXIT.

}

Figure A.1: The RRT algorithm

{

1. Set $n = 0$, E = { }, L = {D} and mark all elements in L 'untested'.
2. Get one element of L and determine its longest side length h.
   If more than one edge side of this element has the length h, choose one edge randomly.
3. Randomly pick up a untested element l from L.
4. If l exists, mark l 'tested' and randomly generate a test case t in l according to the uniform distribution; otherwise, go to Step 6.
5. If t reveals a failure, go to Step 10; otherwise, store t in E and go to Step 3.
6. Bisect every $l_i \in$ L along the edge side chosen in Step 2.
7. Reset L to contain all partitions generated in Step 6.
8. Update the 'tested' status for all elements of L.
9. Go to Step 2.
10. Return n and t. EXIT.

}

Figure A.2: The BPRT algorithm [14]

{

1. Set $n = 0$, E = { } and L = {D}.
2. Select the element l with the largest area from L. If there are more than one element with the largest area, choose one randomly.
3. Randomly select a test case, t, from l according to the uniform distribution.
4. Increment n by 1.
5. If t reveals a failure, go to Step 8; otherwise, store t in E.
6. Divide l into 2^N partitions based on the co-ordinates of t. Inside L, replace l by these resultant partitions.
7. Go to Step 2.
8. Return n and t. EXIT.

}

Figure A.3: The RPRT algorithm [14]

Appendix B

Specification of International Postage Calculation System (IPCS)

Weight (1 unit = 50g)	**Zone AP**		**Zone RW**	
	Per item	**Per item**	**Per pack of 5**	**Per pack of 5**
Up to 100 units	$50.00	$225.00	$70.00	$315.00
Over 100 up to 250 units	$115.00			

Table B.1: Parcel EMS prepaid mail

Weight (1 unit = 50g)	**Zone AP**		**Zone RW**	
	Per item	**Per pack of 5**	**Per item**	**Per pack of 5**
Up to 50 units	$9.50	$42.75	$13.50	$60.75
Over 50 up to 100 units	$27.00	$121.50	$42.00	$189.00

Table B.2: Parcel Air prepaid mail

Weight (1 unit = 50g)	**Air**	**Economy**	**Sea**	**EMS Document**	**EMS Merchandise**
Zone A					
Up to 25 units	$6.00	$5.50	–	$30.00	$37.00
Over 25 up to 50 units	$9.00	$8.00	–	$30.00	$37.00
Over 50 up to 75 units	$12.50	$11.00	–	$34.50	$41.50
Over 75 up to 100 units	$15.50	$13.50	–	$34.50	$41.50
Over 100 up to 125 units	$19.00	$16.50	–	$39.00	$46.00
Over 125 up to 150 units	$22.00	$19.00	–	$39.00	$46.00
Over 150 up to 175 units	$25.50	$22.00	–	$43.50	$50.50
Over 175 up to 200 units	$28.50	$24.50	–	$43.50	$50.50
Extra 50 units or part thereof	$3.50	$3.00	–	$4.50	$4.50
Zone B					
Up to 25 units	$7.00	$6.50	–	$32.00	$39.00
Over 25 up to 50 units	$11.00	$9.50	–	$32.00	$39.00
Over 50 up to 75 units	$15.50	$13.00	–	$38.50	$45.50
Over 75 up to 100 units	$19.50	$16.00	–	$38.50	$45.50
Over 100 up to 125 units	$24.00	$19.50	–	$45.00	$52.00
Over 125 up to 150 units	$28.00	$22.50	–	$45.00	$52.00
Over 150 up to 175 units	$32.50	$26.00	–	$51.50	$58.50
Over 175 up to 200 units	$36.50	$29.00	–	$51.50	$58.50
Extra 50 units or part thereof	$4.50	$3.50	–	$6.50	$6.50
Zone C					
Up to 25 units	$8.00	$7.00	$6.00	$35.00	$42.00
Over 25 up to 50 units	$13.00	$11.00	$9.00	$35.00	$42.00
Over 50 up to 75 units	$18.50	$15.50	$12.50	$43.50	$50.50
Over 75 up to 100 units	$23.50	$19.50	$15.50	$43.50	$50.50
Over 100 up to 125 units	$29.00	$24.00	$19.00	$52.00	$59.00
Over 125 up to 150 units	$34.00	$28.00	$22.00	$52.00	$59.00
Over 150 up to 175 units	$39.50	$32.50	$25.50	$60.50	$67.50
Over 175 up to 200 units	$44.50	$36.50	$28.50	$60.50	$67.50
Extra 50 units or part thereof	$6.50	$5.50	$3.50	$8.50	$8.50
Zone D					
Up to 25 units	$9.50	$8.00	$6.00	$37.00	$44.00
Over 25 up to 50 units	$15.50	$12.50	$9.00	$37.00	$44.00
Over 50 up to 75 units	$22.00	$17.50	$12.50	$46.50	$53.50
Over 75 up to 100 units	$28.00	$22.00	$15.50	$46.50	$53.50
Over 100 up to 125 units	$34.50	$27.00	$19.00	$56.00	$63.00
Over 125 up to 150 units	$40.50	$31.50	$22.00	$56.00	$63.00
Over 150 up to 175 units	$47.00	$36.50	$25.50	$65.50	$72.50
Over 175 up to 200 units	$53.00	$41.00	$28.50	$65.50	$72.50
Extra 50 units or part thereof	$8.50	$6.50	$3.50	$9.50	$9.50

Table B.3: Parcel non-prepaid mail

Weight (1 unit = 10g)	**Zone AP**	**Zone RW**
Up to 5 units	$1.20	$1.80
Over 5 up to 12 units	$2.40	$3.60
Over 12 up to 25 units	$3.60	$5.40
Up to 25 up to 50 units	$7.20	$10.80

Table B.4: Letter air mail

Weight (1 unit = 10g)	**To any destination**		
	Single envelope	**Per pack of 10**	
		1-4 packs	**Over 4 packs**
Up to 5 units	$2.00	$18.00	$17.00
Over 5 up to 12 units	$3.80	$34.20	$32.30
Over 12 up to 25 units	$6.00	$54.00	$51.00

Table B.5: Letter air prepaid mail

Weight (1 unit = 10g)	**To any destination**		
	Single envelope	**Per pack of 10**	
		1-9 packs	**Over 9 packs**
Up to 25 units	$10.00	$90.00	$85.00
Over 25 up to 50 units	$16.50	$148.50	$140.25

Table B.6: Letter Registered mail

Weight (1 unit = 10g)	**To any destination**		
	Single envelope	**Per pack of 10**	
		1-9 packs	**Over 9 packs**
Up to 25 units	$10.50	$99.75	$94.50
Over 25 up to 50 units	$15.85	$150.58	$142.65

Table B.7: Letter Express prepaid mail

Weight (1 unit = 10g)	**Zone AP**		**Zone RW**	
	Per item	**Per pack of 5**	**Per item**	**Per pack of 5**
Up to 50 units	$34.00	$153.00	$45.00	$202.50

Table B.8: Letter EMS prepaid mails

Weight	**To any destination**	
	Per item	**Per pack of 10**
Up to 30 g	$0.95	$9.00

Table B.9: Aerogramme

Weight	**To any destination**	
	Prepaid	**Non-prepaid**
Up to 30 g	$1.20	$1.10

Table B.10: Postcard

Weight	**To any destination**	
	January-October	**November-December**
Up to 30 g	$1.10	$1.00

Table B.11: Greeting card

Items	**Availability**	**Fee**
Pickup	EMS courier	$9.50
Insurance	EMS courier	Maximum insured value: 100 units* Fee: $5.50 plus $2.00 per 2 units* or part thereof
Delivery confirmation	Registered letter Insured EMS goods	$2.75 per article plus postage

* 1 unit = 100 dollars

Table B.12: Other charges

Zone	**Regions**	**Number of countries**
A	New Zealand	1
B	Asia/Pacific	34
C	USA/Canada/Middle East	23
D	Other countries	133
AP	New Zealand and Asia/Pacific	35
RW	Other countries	156

Table B.13: Zones

Zone A: 1 country	**Zone D: 133 countries**	
New Zealand	Algeria	Finland
	Angola	France
	Benin	Cote d'lvoire
	Botswana	Georgia
	Burkina Faso	Germany
Zone B: 34 countries	Burundi	Greece
Bhutan	Cameroon	Hungary
Cambodia	Cape Verde	Iceland
China	Central African Rep.	Ireland
East Timor	Chad	Italy
India	Congo Dem Rec	Latvia
Indonesia	Congo Rep	Liechtenstein
Japan	Djibouti	Lithuania
Korea Dem	Egypt	Luxembourg
Korea Rep	Equatorial Guinea	Macedonia
Laos	Eritrea	Malta
Malaysia	Ethiopia	Moldova
Maldives	Gabon	Monaco
Nepal	Gambia	Netherlands
Pakistan	Ghana	Norway
Philippines	Guinea	Poland
Singapore	Kenya	Portugal
Sri Lanka	Lesotho	Romania
Thailand	Liberia	San Marino
Vietnam	Malawi	Serbia & Montenegro
Myanmar	Mali	Slovakia
Fiji	Mauritania	Slovenia
Kiribati	Morocco	Spain
Micronesia	Mozambique	Sweden
Nauru	Namibia	Switzerland
Palau	Niger	Ukraine
Papua New Guinea	Nigeria	United Kingdom
Samoa	Rwanda	Antigua and Barbuda
Tonga	Sao Tome & Principe	Bahamas
Tuvalu	Senegal	Barbados
Vanuatu	Seychelles	Belize
Bangladesh	Sierra Leone	Costa Rica
Brunei Darussalam	Somalia	Cuba
Marshall Islands	South Africa	Dominica
Solomon Islands	Sudan	Dominican Rec
	Swaziland	El Salvador
	Tanzania	Grenada
	Togo	Guatemala
Zone C: 23 countries	Tunisia	Haiti
Comoros	Uganda	Honduras
Madagascar	Zambia	Jamaica
Mauritius	Zimbabwe	Nicaragua
Afghanistan	Kazakhstan	Panama
Bahrain	Mongolia	St. Christopher & Nevis
Iran	Russian	St. Lucia
Iraq	Tajikistan	St. Vincent & the Grenadines
Israel	Turkey	Trinidad & Tobago
Jordan	Turkmenistan	Argentina
Kuwait	Uzbekistan	Bolivia
Kyrgyzstan	Albania	Brazil
Lebanon	Andorra	Chile
Oman	Armenia	Colombia
Qatar	Austria	Ecuador
Saudi Arabia	Azerbaijan	Guyana
Syria	Belarus	Paraguay
United Arab Emirates	Belgium	Peru
Yemen Republic	Bosnia & Herzegovina	Suriname
Cyprus	Bulgaria	Uruguay
Canada	Croatia	Venezuela
Mexico	Czech Republic	Libyan Arab Jamahiriya
U.S.A.	Denmark	Guinea-Bissau
Hawaii	Estonia	

Table B.14: Countries of various zones

List of Publications

1. Refereed International Journal Papers

   (a) T.Y. Chen, F.-C. Kuo and C.A. Sun, The impact of the compactness of failure regions on the performance of adaptive random testing, *Journal of software*, Acceptance date: 3 April 2006.

   (b) T.Y. Chen, F.-C. Kuo and R.G. Merkel, On the Statistical Properties of Testing Effectiveness Measures, *Journal of Systems and Software*, 79 (5): 591-601, 2006.

   (c) T.Y. Chen, F.-C. Kuo, R.G. Merkel and S.P. Ng, Mirror Adaptive Random Testing, *Journal of Information and Software Technology*, 46 (15): 1001-1010, 2004.

2. Refereed International Conference/ Workshop Papers

   (a) T.Y. Chen and F.-C. Kuo, Is adaptive random testing really better than random testing, In *Proceedings of the 1th International Workshop on Random Testing in the 2006 ACM SIGSOFT international symposium on Software testing and analysis*, Pages 64-69, Maine, Portland, 2006. ACM Press.

   (b) T.Y. Chen, F.-C. Kuo and Z.Q. Zhou, On the Relationships between the Distribution of Failure-Causing Inputs and Effectiveness of Adaptive Random Testing, In *Proceedings of the 17th International Conference on Software Engineering and Knowledge Engineering (SEKE 2005)*, Pages 306-311, Taipei, Taiwan, 2005.

(c) T.Y. Chen, F.-C. Kuo and R.G. Merkel, On the Statistical Properties of the F-measure, In *Proceedings of the 4th International Conference on Quality Software (QSIC 2004)*, Pages 146-153, Brunschwig, Germany, IEEE Computer Society Press, 2004.

(d) K.P. Chan, T.Y. Chen, F.-C. Kuo and D. Towey, A Revisit of Adaptive Random Testing by Restriction, In *Proceedings of the 28th annual International Computer Software and Applications Conference (COMPSAC 2004)*, Pages 78-85, Hong Kong, China, IEEE Computer Society Press, 2004.

(e) K.P. Chan, D. Towey, T.Y. Chen, F.-C. Kuo and R. Merkel, Using the Information: incorporating Positive Feedback Information into the Testing Process, In *Proceedings of the 11th International Workshop on Software Technology and Engineering Practice (STEP 2003)*, Pages 71-76, Amsterdam, The Netherlands, IEEE Computer Society Press, 2004.

(f) T.Y. Chen, F.-C. Kuo, R.G. Merkel and S.P. Ng, Mirror Adaptive Random Testing, In *Proceedings of the 3rd International Conference on Quality Software (QSIC 2003)*, Pages 4-11 , Dallas, Texas, USA, IEEE Computer Society Press, 2003.