Application of a Failure Driven Test Profile in Random Testing

Tsong Yueh Chen, *Member*, *IEEE*, Fei-Ching Kuo, *Member*, *IEEE*, and Huai Liu
Faculty of Information and Communication Technologies
Swinburne University of Technology
Hawthorn, Victoria 3122, Australia
{*tchen, dkuo, hliu*}*@ict.swin.edu.au*

Abstract

Random testing techniques have been extensively used in reliability assessment, as well as in debug testing. When used to assess software reliability, random testing selects test cases based on an operational profile; while in the context of debug testing, random testing often uses a uniform distribution. However, generally neither an operational profile nor a uniform distribution is chosen from the perspective of maximizing the effectiveness of failure detection. Adaptive random testing has been proposed to enhance the failure detection capability of random testing by evenly spreading test cases over the whole input domain. In this paper, we propose a new test profile, which is different from both the uniform distribution, and operational profiles. The aim of the new test profile is to maximize the effectiveness of failure detection. We integrate this new test profile with some existing adaptive random testing algorithms, and develop a family of new random testing algorithms. These new algorithms not only distribute test cases more evenly, but also have better failure detection capabilities than the corresponding original adaptive random testing algorithms. As a consequence, they perform better than the pure random testing.

Key Words: Random testing, adaptive random testing, test profile, uniform distribution, operational profile.

Acronym[1]

RT	Random Testing
ART	Adaptive Random Testing
FSCS-ART	Fixed-Sized-Candidate-Set Adaptive Random Testing
RRT	Restricted Random Testing
ART-DNC	Adaptive Random Testing with Dynamic Non-Uniform Candidate Distribution
pdf	Probability Density Function
PDF	Probability Distribution Function

Notation

E	The set of all executed test cases
I	The input domain
N	The dimension of input domain
ND	N-dimension, where $N = 1, 2, 3, 4, \cdots$
$\lvert\cdot\rvert$	The size of a set
θ	The failure rate of a program
F-measure	The expected number of test cases required to detect the first program failure
F_{RT}	F-measure of random testing
F_{ART}	F-measure of adaptive random testing
ART F-ratio	$\frac{F_{ART}}{F_{RT}}$

[1] The singular and plural of an acronym are always spelled the same.

1. Introduction

Random testing (RT) is a standard software testing technique which simply generates *test cases* (that is, program inputs for testing) at random from the whole *input domain* (that is, the set of all possible inputs for the program under test) [15], [21]. RT has been widely used for assessing software reliability [14], [23], where test cases are often selected according to an operational profile. The operational profile refers to a probability distribution, over the input domain, which characterizes how a program is operated by end-users [20]. Another application of RT is debug testing, which aims at detecting software failures so that program bugs can be removed, and thus software reliability can be improved [13]. When used as a debug testing method, RT often selects test cases based on a uniform distribution; that is, all program inputs have the same probability to be selected as test cases, to ensure that every possible bug could be detected.

Inputs that cause the program under test to exhibit failure behaviors are known as *failure-causing inputs*. A testing method is said to detect a failure if it picks a failure-causing input as a test case. When a testing method is capable of detecting software failures more effectively, it is more likely that program bugs can be removed as early as possible, or that as many bugs may be removed as possible, and hence we may be more confident of the software's reliability. Some researchers [21] argued that RT is a poor debug testing method because it does not make use of any information to guide the selection of test cases. The operational profile, which is used by RT in reliability assessment, is constructed with respect to the usage frequencies of different functions in the program, not from the perspective of how likely the inputs are failure-causing. The uniform distribution, the most commonly used test profile for RT in debug testing, treats all possible program inputs equally, regardless whether some inputs may have higher probabilities to be failure-causing than

other inputs. Briefly speaking, both test profiles for RT are not failure oriented, and hence are not expected to ensure an optimal failure detection capability for RT. This lack has motivated us to investigate whether different test profiles can enhance the effectiveness of RT.

Several studies [1], [2], [12], [25] have independently made a common observation that failure-causing inputs tend to cluster into contiguous regions (known as *failure regions* [1]) in the input domain. Chen *et al.* [7] have attempted to improve the effectiveness of RT by means of this characteristic of failure-causing inputs. Their intuition is that, if a test case t does not reveal any software failure, then an input that is away from t is more likely to cause a failure than t's neighbors. Based on such an intuition, they proposed a novel approach, namely *adaptive random testing* (ART), which not only randomly selects, but also more evenly spreads test cases over the input domain. Various algorithms have been developed to implement the basic "even spread" intuition of ART, such as *fixed-sized-candidate-set ART* (FSCS-ART) [7], *restricted random testing* (RRT) [3], and *lattice-based ART* [17]. Most ART algorithms consist of two independent processes. One process (known as *candidate generation process*) randomly generates program inputs as *test case candidates*, or briefly *candidates*. The other process (known as *test case identification process*) applies some criteria to identify test cases among these candidates to ensure an even spread of test cases across the input domain. Because ART was originally proposed as a debug testing method, the random generation of candidates has always been conducted according to a uniform distribution.

In ART, the goal is to achieve an even spread of test cases, but this was not fully realized in previous studies. Therefore, some have attempted [5], [18] to make test cases more evenly spread. But all of these studies were focused on the enhancement of the test case identification process, either by developing new test case identification criteria, or by improving the existing criteria. They

all used the uniform distribution as the test profile for the candidate generation process. However, as mentioned before, such a uniform distribution is not chosen specifically to help improve the effectiveness of RT. We are motivated to look at what test profile can be used to enhance the failure detection capability of RT/ART. In this paper, we propose to develop another test profile, which is different from the uniform distribution, and the operational profile, for the candidate generation process in ART. Our approach is to design a test profile for the candidate generation process such that the test case identification process would deliver a more even distribution of test cases, and consequently, a likely improvement of the failure detection capability. We call such a profile as *failure driven test profile*. We conduct investigations on two particular ART algorithms: FSCS-ART [7], and RRT [3].

Section 2 introduces the background information on FSCS-ART, and RRT. Some concepts used in this paper are also discussed in this section. In Section 3, we investigate how to select a test profile that can be used to guide the candidate generation in FSCS-ART, and RRT, while keeping the test case identification criteria unchanged. The resultant new algorithms are evaluated via some simulations, whose results are also reported in this section. Section 4 concludes the paper. The appendix contains details of some calculations.

2. Background

In this paper, for convenience of illustration, we assume that the program under test only has numeric inputs. Applications of RT, and ART on non-numeric programs have been studied in [22], [26], and [9], [10], [16], [19], respectively.

2.1. Fixed-sized-candidate-set adaptive random testing

Fixed-sized-candidate-set ART (FSCS-ART) [7] makes use of two sets of test cases: the *executed set* $E = \{e_1, e_2, \cdots, e_n\}$, which contains the executed test cases; and the *candidate set* $C = \{c_1, c_2, \cdots, c_k\}$, which contains k randomly generated candidates. A candidate c_b will be chosen as the next test case if for all $j = 1, 2, \cdots, k$,

$$\min_{i=1}^{n} dist(c_b, e_i) \geq \min_{i=1}^{n} dist(c_j, e_i), \tag{1}$$

where *dist* is the Euclidean distance between two points. The details of the FSCS-ART algorithm can be found in [7]. Although the performance of FSCS-ART improves with the increase of k, any $k > 10$ will not significantly improve the effectiveness of FSCS-ART [7]. Therefore, in this paper, we will use FSCS-ART where k has a value of 10.

2.2. Restricted random testing

Restricted random testing (RRT) [3] creates an *exclusion region* around each element of E. Only the randomly generated candidates that fall outside of all excluded areas will be selected as test cases. The exclusion zone for each element of E is of the same size $\frac{R \cdot |I|}{|E|}$, where R is defined as the *target exclusion ratio*. Some simulations [3] have shown that the failure detection capability of RRT improves with the increase of R. However, a larger R means that a larger part of I would be excluded, and hence more computation is required to generate a test case (that is, to generate a random input outside the exclusion regions of all executed test cases). Chen *et al.* [5] have proposed a mechanism to dynamically adjust R to balance the computation time, and the performance. In this paper, we will also adopt such an approach. In addition, we set the initial value of R at 1, 1.7,

3.3, and 6.4 for RRT in 1D, 2D, 3D, and 4D spaces, respectively, as suggested in [3].

2.3. Failure detection capability of adaptive random testing

The F-measure, the expected number of test cases required to detect the first software failure, has been used for measuring the failure detection capability of ART. This paper will follow all previous studies of ART in using the F-measure as the effectiveness metric. Readers may refer to [8] for an explanation for why the F-measure is more appropriate than other metrics in evaluating ART. In this paper, F-measures of RT, and ART are denoted by F_{RT}, and F_{ART}, respectively. The ART F-ratio, the ratio between F_{ART} and F_{RT}, is used to show the improvement of ART over RT.

Generally speaking, failure-causing inputs determine two basic features of a faulty program. One feature is the failure rate (denoted by θ), which refers to the ratio between the number of failure-causing inputs, and the number of all possible inputs. The *failure pattern*, the other basic feature, refers to the failure regions together with their distribution over I. Both θ, and the failure pattern are unknown before testing, although they are fixed after coding. Theoretically, $F_{RT} = 1/\theta$ when test cases are randomly selected according to uniform distribution, and with replacement. Obviously, a theoretical analysis of F_{ART} is extremely difficult, and thus previous studies of ART [4], [5], [6] have estimated F_{ART} via simulations, using the basic procedure as follows.

For simulation studies, to simulate faulty programs, θ and the failure pattern are set first. Then, failure regions, whose size, and shape are decided by θ, and the failure pattern, respectively, are then randomly placed inside I. An ART algorithm is applied to continuously select test cases. If a point inside the failure regions is picked up, it is said that a failure is detected. The number of test cases required to detect the first failure, referred to as the F-count [8], is recorded. Such a process

is repeated for a sufficiently large number (S) of times to ensure that the average value of F-counts can be regarded as a reliable estimate for F_{ART} within a specified confidence level, and an accuracy range (details of deciding S can be found in [7]). In this paper, the default values of confidence level, and accuracy range are set as 95%, and $\pm$5%, respectively.

Previous studies [4], [6] showed that FSCS-ART, and RRT perform best when failure-causing inputs are well clustered into one single compact region. However, their failure detection capabilities deteriorate as θ becomes higher, N becomes higher, the failure region becomes less compact, or the number of failure regions becomes larger.

2.4. Test case distribution of adaptive random testing

Because the basic intuition of ART is the even spread of test cases, some research [4], [19] has been conducted to measure how evenly ART algorithms spread test cases from different perspectives. In [19], the test case distributions of various ART algorithms were coarsely described by some 2D spatial distribution graphs. A more precise approach for measuring test case distributions was proposed by Chen *et al.* [4], where some distribution metrics were employed to measure the evenness of test case distribution. Among these metrics, discrepancy, and dispersion are two commonly used measurement metrics for the equidistribution of sample points. Discrepancy indicates the maximal difference of points' densities for various regions in I, while dispersion indicates the size of the largest empty spherical region (containing no point) in I. Smaller discrepancy or smaller dispersion implies better equidistribution of sample points.

Chen *et al.* [4] found that all ART algorithms under their investigation exhibit various degrees of uneven test case distribution. For example, FSCS-ART, and RRT usually have fairly small dispersion, but the values of their discrepancy will become larger when N is higher, or $|E|$ is

smaller. As mentioned in Section 2.3, FSCS-ART, and RRT have poor failure detection capabilities for high N, or high θ cases. Such a correlation between the ART effectiveness, and the test case distribution has motivated us to look at how to enhance the failure detection capability of ART through the improvement of the evenness of test case distribution.

3. Adaptive Random Testing using a Non-Uniform Distribution as a Test Profile

As mentioned before, neither the uniform distribution (for debug testing), nor the operational profile (for reliability assessment) is designed specifically to help RT achieve an optimal failure detection capability. ART improves the failure detection capability of RT by evenly spreading random test cases. There are two independent processes in most ART algorithms: the candidate generation process, which ensures the randomness of test cases; and the test case identification process, which ensures the even spread of random test cases. All previous studies of ART were focused on the test case identification process, while the candidate generation process always used a uniform distribution, just like RT as a debug testing method. This study attempts to investigate whether applying a different test profile in the candidate generation process can enhance the even spread of test cases, aiming at improving the failure detection capability of RT/ART.

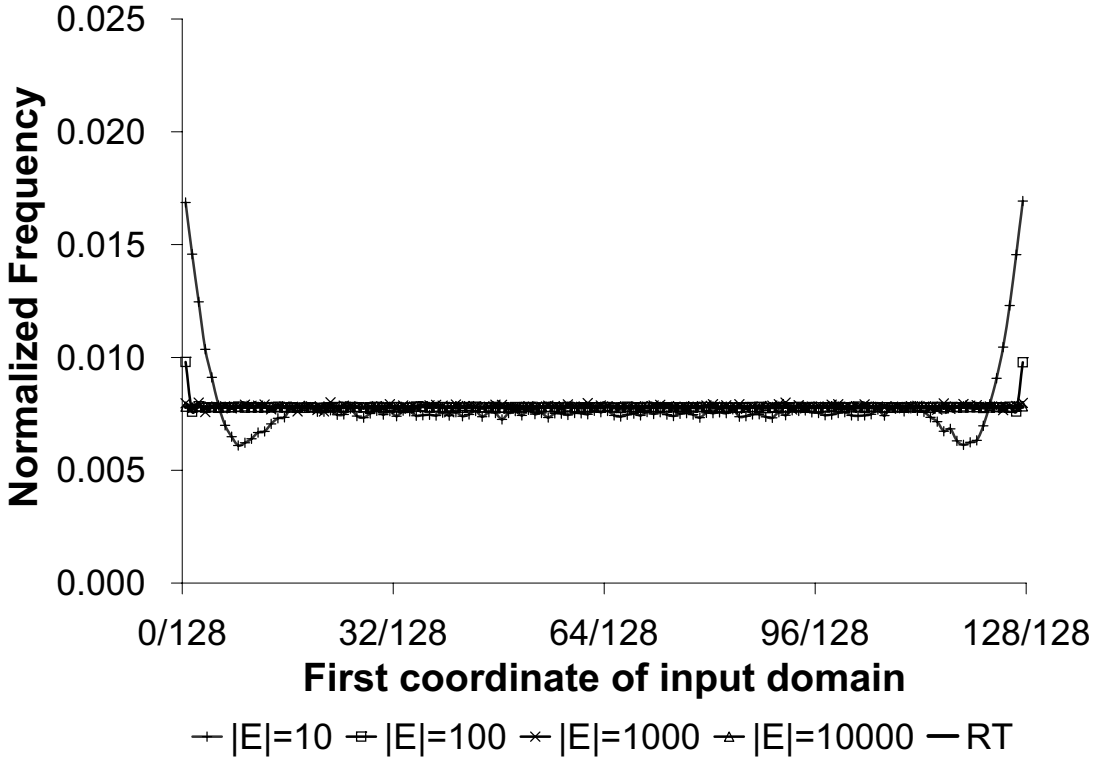
FSCS-ART, and RRT have been found to have a bias of selecting test cases from certain parts of I [4], [19]. Such a bias (hereafter referred to as *tc-bias*) may result in an uneven distribution of test cases. Intuitively speaking, if the distribution of candidates has a "reverse" effect to the distribution of test cases in FSCS-ART, and RRT, the tc-bias may be offset, and hence test case distributions may become more even. However, neither the spatial distribution graphs in [19] nor the distribution metrics in [4] have given a precise measurement for the tc-bias. In this paper, we employ a new method to quantitatively describe the tc-bias of an ART algorithm.

3.1. Describing the tc-bias of an adaptive random testing algorithm

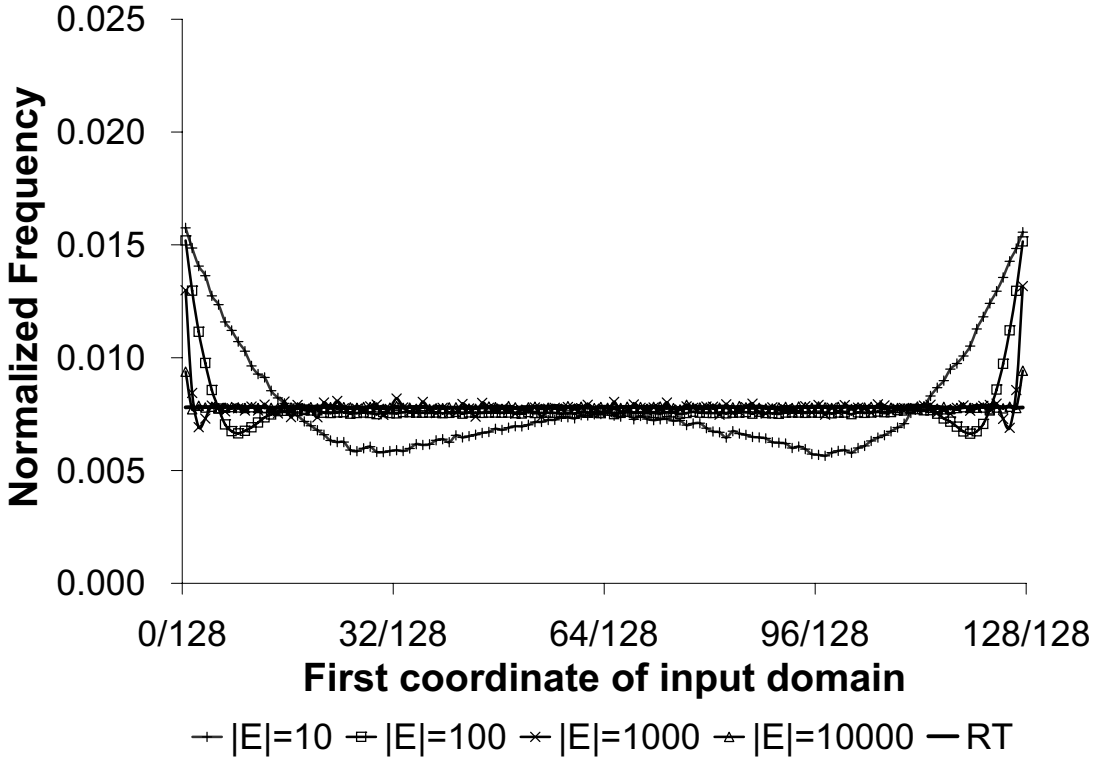
If points are equidistributed, each coordinate of them will be equidistributed. Hence, if the projections of points in any dimension are non-equidistributed, we can say that these points are not equidistributed. Therefore, if we check just one coordinate of the test cases selected by an ART algorithm, and find any bias, then we may judge that the ART algorithm has tc-bias.

Our method for measuring the tc-bias is as follows. Suppose that I is a unit square such that each dimension of I has the range of value as $[0,1)$. We choose a certain coordinate, say the 1st coordinate, and divide it into m equal-sized subdomains, $[0,1/m),[1/m,2/m),\cdots,[(m-1)/m,1)$. Then, a set of test cases are generated using an ART algorithm. For each subdomain, we record the number of test cases whose 1st coordinates are inside this subdomain. The normalized frequency of points inside each subdomain is then calculated. Such a process will be repeated for a sufficient number of times so that reliable average values of frequencies are obtained within a 95% confidence level, and a $\pm$5% accuracy range. Based on the values of the collected average normalized frequencies for these m subdomains, we calculate two more statistics: i) the standard deviation of these m average normalized frequencies, denoted by *stdev*; and ii) the difference between the maximal and minimal values of these m average normalized frequencies, denoted by *max–min*. The smaller *max–min* and *stdev* are, the lower tc-bias an ART algorithm has.

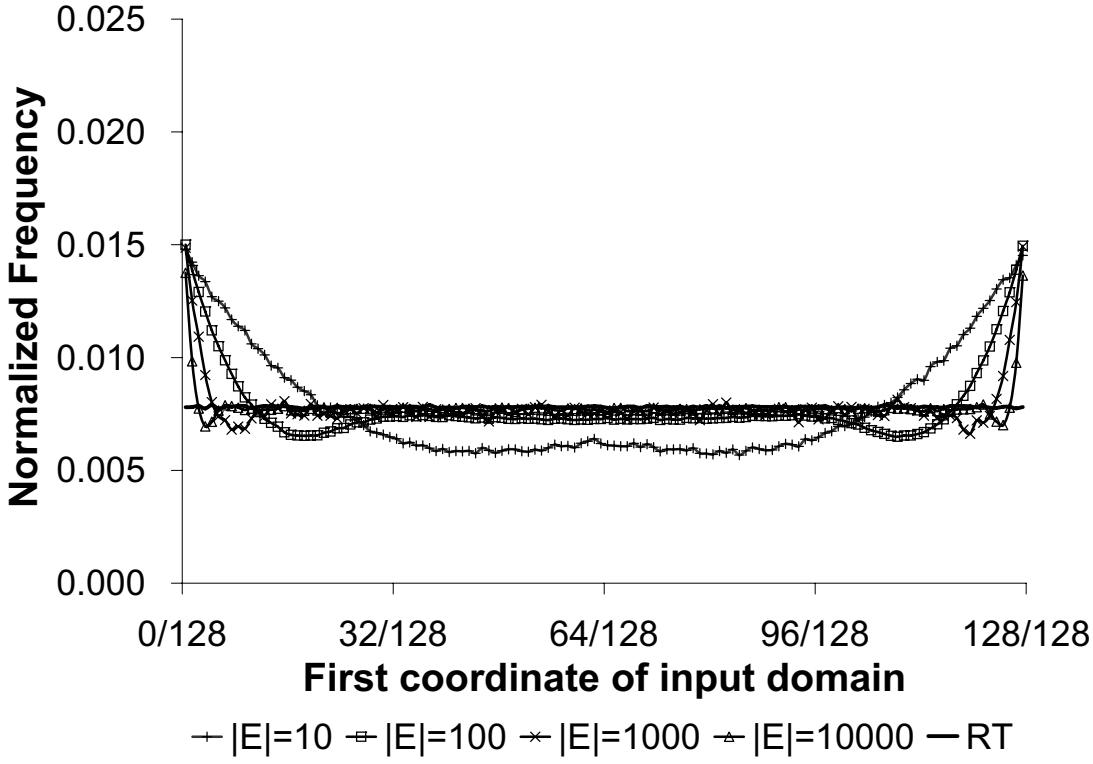
We used the above method to measure the tc-biases of FSCS-ART, RRT, and pure RT in 1D, 2D, 3D, and 4D spaces. m is set to 128, and $|E|$ is set to 10, 100, 1000, and 10,000. Because the frequency distributions of test cases of FSCS-ART and RRT are similar to each other, we only plot the frequency distribution graph of FSCS-ART in Fig. 1. The values of *max–min*, and *stdev* for FSCS-ART, RRT, and RT (which quantitatively describe the degrees of tc-biases for these three



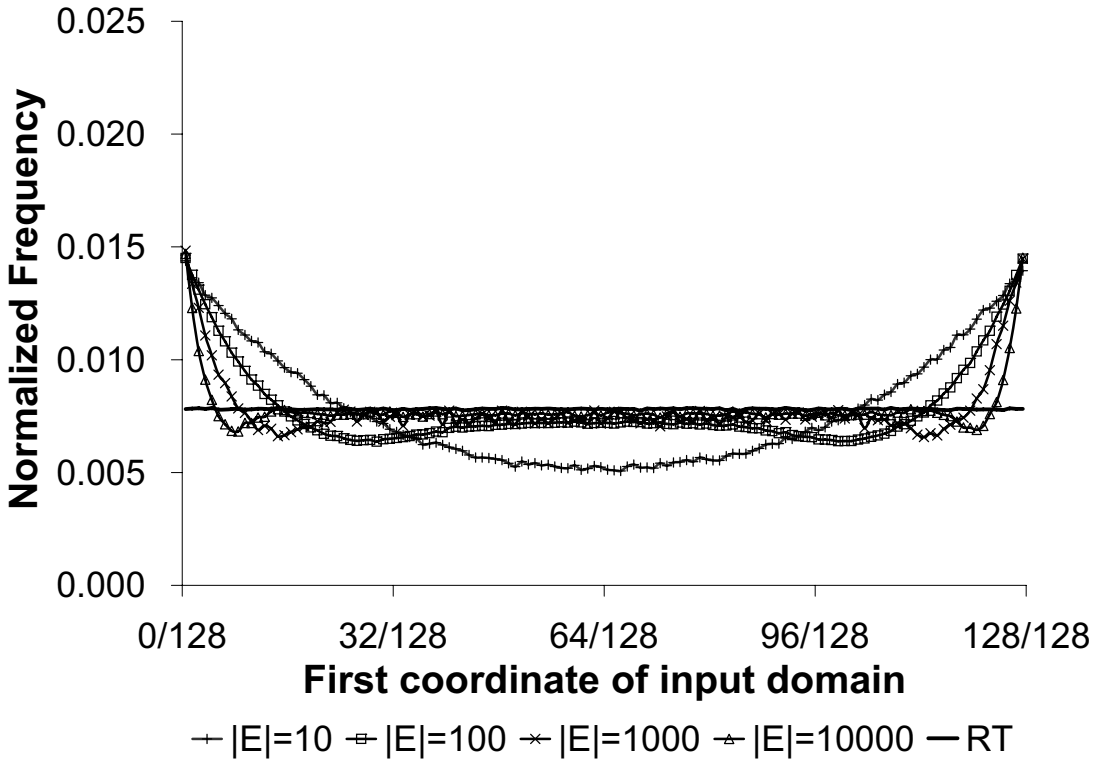
1.a Frequency distribution for 1D FSCS-ART



1.b Frequency distribution for 2D FSCS-ART



1.c Frequency distribution for 3D FSCS-ART



1.d Frequency distribution for 4D FSCS-ART

Figure 1. Frequency distribution of the first coordinate of test cases generated by FSCS-ART.

testing methods) are summarized in Table 1. In Fig. 1, the x-, and y-axes denote the locations of the first dimension's subdomains, and the normalized frequencies of test cases inside these subdomains, respectively. We found that the test case distribution of RT is always uniform no matter how many test cases were generated, which is theoretically expected. Therefore, we only plot the frequency distribution of RT where $|E| = 10000$. Note that, although RT does not have any tc-bias, that does not mean that RT is better than ART in terms of the F-measure.

Table 1. Values of *max–min*, and *stdev* for FSCS-ART, RRT, and RT

N	testing strategy	$\|E\| = 10$		$\|E\| = 100$		$\|E\| = 1000$		$\|E\| = 10000$	
		max–min	*stdev*	*max–min*	*stdev*	*max–min*	*stdev*	*max–min*	*stdev*
1	RT	4.70E-04	8.70E-05	1.60E-04	3.04E-05	9.90E-04	1.86E-04	3.47E-04	6.14E-05
	FSCS-ART	1.08E-02	1.66E-03	2.20E-03	2.52E-04	4.45E-04	8.55E-05	1.00E-04	2.07E-05
	RRT	9.09E-03	1.44E-03	2.04E-03	2.35E-04	4.05E-04	7.83E-05	1.20E-04	2.27E-05
2	RT	4.25E-04	9.01E-05	1.58E-04	2.68E-05	9.20E-04	2.20E-04	3.50E-04	5.78E-05
	FSCS-ART	1.01E-02	2.38E-03	8.57E-03	1.29E-03	6.28E-03	7.01E-04	1.76E-03	2.05E-04
	RRT	1.14E-02	2.65E-03	1.34E-02	1.88E-03	1.01E-02	1.06E-03	2.09E-03	2.50E-04
3	RT	3.31E-04	7.42E-05	1.72E-04	3.20E-05	1.02E-03	1.93E-04	3.12E-04	5.89E-05
	FSCS-ART	9.10E-03	2.51E-03	8.52E-03	1.70E-03	8.29E-03	1.20E-03	6.81E-03	8.00E-04
	RRT	1.20E-02	3.22E-03	1.78E-02	3.18E-03	1.79E-02	2.37E-03	1.38E-02	1.52E-03
4	RT	4.90E-04	9.35E-05	1.52E-04	2.82E-05	8.55E-04	2.04E-04	3.61E-04	6.45E-05
	FSCS-ART	9.31E-03	2.59E-03	8.14E-03	1.89E-03	8.26E-03	1.48E-03	7.86E-03	1.13E-03
	RRT	1.31E-02	3.67E-03	1.93E-02	3.93E-03	2.06E-02	3.17E-03	1.78E-02	2.32E-03

The experimental data of Table 1, and Fig. 1 show that both FSCS-ART, and RRT have certain tc-biases, and the tc-biases become higher with the increase of N, as well as with the decrease of $|E|$. Further investigation of the frequency distribution graphs show that points from the boundary part of I have higher probabilities to be selected as test cases than those from the central part of I. Moreover, all frequency distributions are symmetric with respect to the center of I.

Having quantified the tc-biases of FSCS-ART, and RRT, we can then design a new test profile for their candidate generation processes. The aim of such a test profile is to improve the even distribution of test cases for FSCS-ART, and RRT, to improve their failure detection capabilities. The following section describes how to design a proper test profile for such a purpose.

3.2. Designing a non-uniform distribution as a test profile

To offset the tc-biases of FSCS-ART, and RRT, the new test profile for the candidate generation process should have the following essential features.

- The probability distribution in the test profile must be dynamic throughout the testing process; that is, the test profile should be changeable as E changes because the amount of tc-bias to be offset varies as E changes.
- The elements in the central part must have a higher probability of being selected as candidates than the elements in the boundary part because the candidates from the central part have a lower probability of being selected as the next test case.
- The probability distribution must be symmetric with respect to the center of I because the distribution of test cases is also symmetric with respect to the center of I.

Many non-uniform distributions have the above properties. A simple example is a *linear combination* of two uniform-distributed random variables.

$$Y = \alpha X_1 + (1-\alpha)X_2, \tag{2}$$

where $0 \leq \alpha \leq 0.5$. X_1, and X_2 are two random variables which are both uniformly distributed in $[0,1)$. The probability density function (pdf) of Y, denoted by $f_Y(y)$, is

$$f_Y(y) = \begin{cases} 0 \quad , & \text{when} \quad y < 0 \text{ or } y \geq 1 \\ \dfrac{y}{\alpha(1-\alpha)}, & \text{when} \quad 0 \leq y < \alpha \\ \dfrac{1}{(1-\alpha)} \ , & \text{when} \quad \alpha \leq y < 1-\alpha \\ \dfrac{1-y}{\alpha(1-\alpha)}, & \text{when} \quad 1-\alpha \leq y < 1 \end{cases} \tag{3}$$

Appendix A describes how to derive $f_Y(y)$, and Fig. 2 shows distributions of Y for various α.

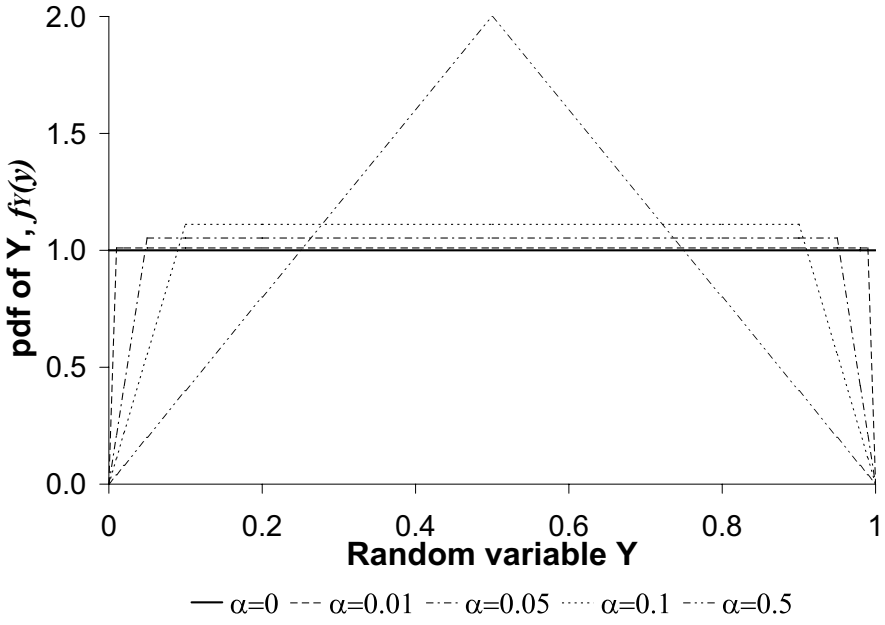


Figure 2. The probability density function of Y.

3.3. Our approach

We propose to generate candidates according to 2 instead of the uniform distribution; as a result, more candidates are likely to be chosen from the center of I than from the edge. The parameter α in 2, and 3 decides how likely candidates are to be selected from the center. In each dimension, for each round of test case selection, α is dynamically chosen as follows.

For ease of illustration, assume that each dimension of I has the value range $[0,1)$, and is equally divided into two subranges: the *central subrange* consisting of $[0.25,0.75)$; and the *boundary subrange* consisting of $[0,0.25)$, and $[0.75,1)$. We then define the following two parameters.

- **The normalized ratio of the l^{th} coordinate of E (the set of all already executed test cases) being located in the boundary subrange of the l^{th} dimension of I** (denoted by $p^l_{tc-boundary}$). For a non-empty set of executed test cases $E = \{e_1, e_2, \cdots, e_n\}$, their l^{th} co-

ordinates are denoted by $e_{1l}, e_{2l}, \cdots, e_{nl}$, respectively, where $0 \leq e_{il} < 1$ ($i = 1, 2, \cdots, n$). We define $E^l_{tc-boundary} = \{e_{il} | 0 \leq e_{il} < 0.25 \text{ or } 0.75 \leq e_{il} < 1\}$. $p^l_{tc-boundary}$ is defined as

$$p^l_{tc-boundary} = \frac{|E^l_{tc-boundary}|}{|E|} \tag{4}$$

Note that the values of $p^l_{tc-boundary}$ for FSCS-ART, and RRT are usually larger than 0.5.

- **The probability of the l^{th} coordinate of a random candidate to be selected from the central subrange of the l^{th} dimension of I** (denoted by $P^l_{can-central}$). For a candidate c randomly generated according to 2, its l^{th} coordinate is denoted by c_l, where $0 \leq c_l < 1$. $P^l_{can-central}$ is defined as the probability of $0.25 \leq c_l < 0.75$. The value of $P^l_{can-central}$ is decided by α in 2, and 3, as follows.

$$P^l_{can-central} = \begin{cases} \dfrac{1}{2(1-\alpha)} & , \quad \text{when} \quad 0 \leq \alpha < 0.25 \\ 1 - \dfrac{1}{16\alpha(1-\alpha)}, & \quad \text{when} \quad 0.25 \leq \alpha \leq 0.5 \end{cases} \tag{5}$$

Appendix B contains the deduction of 5. Note that $P^l_{can-central}$ is within $[0.5, 0.75]$.

Our approach uses the following three steps to decide the value of α for each dimension, successively after each new test case is selected.

1. For each coordinate, measure $p^l_{tc-boundary}$ along the testing process, where $l = 1, 2, \cdots, N$.
2. We propose to offset the tc-biases of FSCS-ART, and RRT by the bias on the center brought by our test profile. The value of $P^l_{can-central}$ (which indicates how likely a candidate will be selected from the central subrange) is set as close to $p^l_{tc-boundary}$ (which indicates the ratio of executed test cases in the boundary subrange) as possible. Considering different value ranges of $P^l_{can-central}$, and $p^l_{tc-boundary}$, we define the s-expected value of $P^l_{can-central}$ as

$$s\text{–expected value of } P^l_{can-central} = \begin{cases} 0.5 & , \text{ when } p^l_{tc-boundary} < 0.5 \\ p^l_{tc-boundary}, & \text{when } 0.5 \le p^l_{tc-boundary} \le 0.75 \\ 0.75 & , \text{ when } p^l_{tc-boundary} > 0.75. \end{cases} \quad (6)$$

3. Use 5 to calculate a value of α such that $P^l_{can-central}$ satisfies 6.

```
1.  Input an integer k, where k > 1.
2.  Set n = 0, E = {}, C = {}, and reveal = false.
3.  Set α_1 = α_2 = ··· = α_N = 0.5, where N denotes the dimension of I.
4.  while (not reveal)
5.     if (n = 0)
6.        Randomly generate a test case t from I, according to a uniform distribution.
7.     else
8.        M = 0.
9.        Randomly generate k candidates c_1, c_2, ··· , c_k from I, where each coordinate c_jl of a candidate
          c_j is generated based on α_l, and according to 2, j = 1,2,··· ,k, and l = 1,2,··· ,N.
10.       Store these candidates into C.
11.       for each candidate c_j ∈ C, where j = 1,2,··· ,k
12.          Calculate m = min_{i=1}^{n} dist(c_j, e_i), where e_i ∈ E.
13.          if (m > M)
14.             Set M = m, and b = j.
15.          end_if
16.       end_for
17.       Set t = c_b.
18.    end_if
19.    Use t to test the target program.
20.    if (t reveals a failure)
21.       Set reveal = true.
22.    else
23.       Store t into E, and increment n by 1.
24.       Calculate p^l_{tc-boundary}, and the s-expected value of P^l_{can-central} for each coordinate according to
          4, and 6, respectively, where l = 1,2,··· ,N.
25.       Calculate the values of α_1, α_2, ··· , α_N according to 5.
26.    end_if
27. end_while
28. Report the failure detected, and exit.
```

Figure 3. The algorithm of FSCS-ART-DNC.

The new approach generates candidates according to separate non-uniform distributions in each dimension, which are dynamically tuned along the testing process. We name the new approach *adaptive random testing with dynamic non-uniform candidate distribution* (ART-DNC). Note that

1. Input an integer $maxTrial$, and a real number $initialR$, where $maxTrial > 0$ and $initialR > 0$.
2. Set $n = 0$, $E = \{\}$, and *reveal* = **false**.
3. Set $\alpha_1 = \alpha_2 = \cdots = \alpha_N = 0.5$, where N denotes the dimension of I.
4. **while** (**not** *reveal*)
5. **if** ($n = 0$)
6. Randomly generate a test case t from I, according to a uniform distribution.
7. **else**
8. Set $noTrial = 0$, $R = initialR$, and *outside* = **false**.
9. **for** each element $e_i \in E$, where $i = 1, 2, \cdots, n$
10. Determine a circular exclusion zone z_i, whose size is set as $\frac{R \cdot |I|}{|E|}$.
11. **end_for**
12. **while** (**not** *outside*)
13. Increment *noTrial* by 1.
14. **if** ($noTrial = maxTrial$)
15. Set $noTrial = 0$ and $R = \max\{0, R - 0.1\}$.
16. **for** each element $e_i \in E$, where $i = 1, 2, \cdots, n$
17. Determine a circular exclusion zone z_i, whose size is set as $\frac{R \cdot |I|}{|E|}$.
18. **end_for**
19. **end_if**
20. Randomly generate a candidate c from I, where each coordinate c_l of c is generated based on α_l; and according to Equation 2, and $l = 1, 2, \cdots, N$.
21. **if** ($c \notin \bigcup_{i=1}^{|E|} z_i$)
22. Set *outside* = **true**, and $t = c$.
23. **end_if**
24. **end_while**
25. **end_if**
26. Use t to test the target program.
27. **if** (t reveals a failure)
28. Set *reveal* = **true**.
29. **else**
30. Store t into E, and increment n by 1.
31. Calculate $p^l_{tc-boundary}$, and the s-expected value of $P^l_{can-central}$ for each coordinate according to 4, and 6, respectively, where $l = 1, 2, \cdots, N$.
32. Calculate the values of $\alpha_1, \alpha_2, \cdots, \alpha_N$ according to 5.
33. **end_if**
34. **end_while**
35. Report the failure detected, and exit.

Figure 4. The algorithm of RRT-DNC.

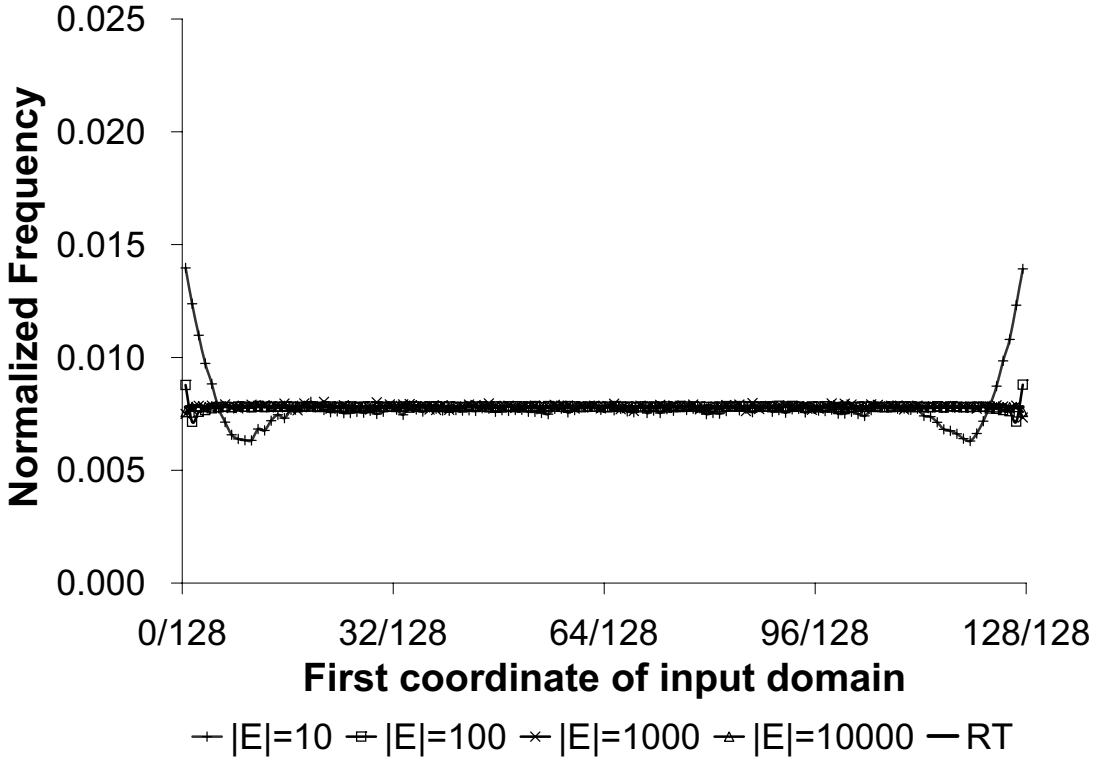
there is a family of software testing methods, namely *statistical testing* [11],[24], which also randomly generates program inputs according to some non-uniform distributions, which are in turn based on some criteria of either the program structure or the functionality. Although our approach also generates random inputs non-uniformly, our purpose is to distribute test cases more evenly, and hence to detect software failures more effectively, without aiming at achieving other criteria (such as program function or structure).

We propose two ART-DNC algorithms, namely *FSCS-ART-DNC*, and *RRT-DNC*, as shown in Figs. 3, and 4, respectively. As a reminder, the test case identification processes of the new ART-DNC algorithms remain the same as those of their counterparts. The aim of using the non-uniform distribution in the candidate generation processes is to improve the failure detection capability of ART, so we call this non-uniform profile a failure driven test profile.

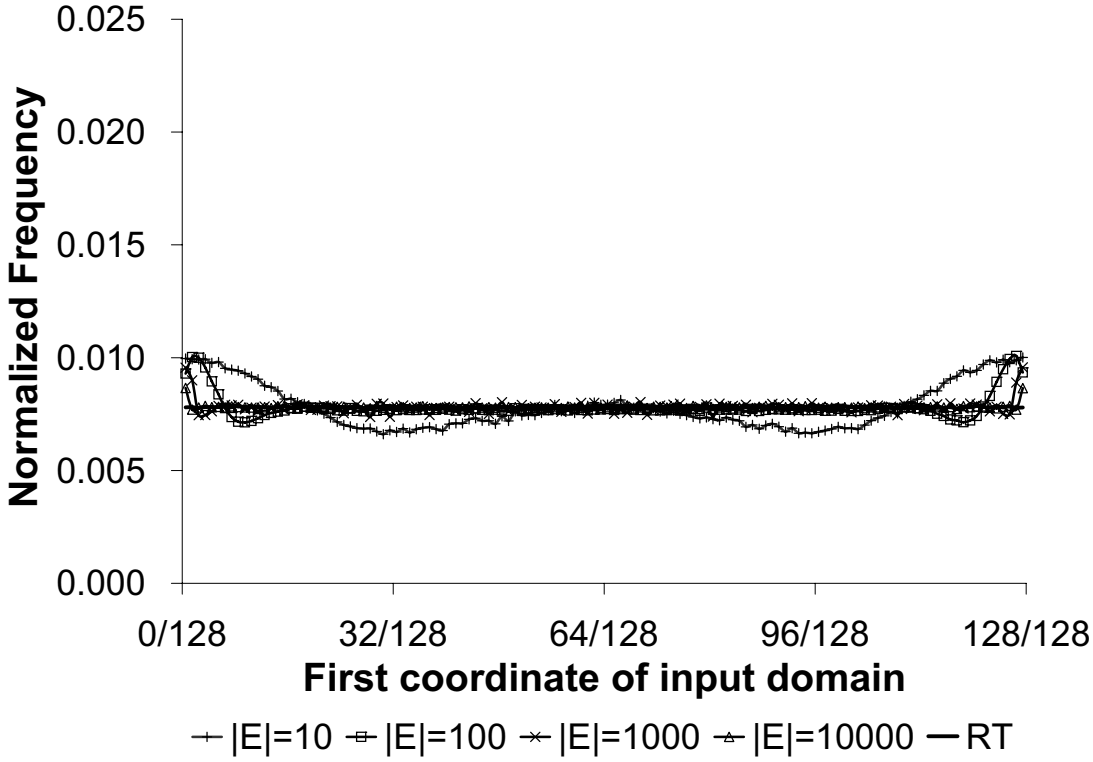
3.4. Test case distributions of new algorithms

Prior to further research, we must check whether using the non-uniform test profile for candidate generation results in a more even spread of test cases. Only then will it make sense to evaluate the failure detection capabilities of ART-DNC. We therefore repeated the simulations in Section 3.1, but instead using FSCS-ART-DNC, and RRT-DNC. The simulations showed similar frequency distributions of test cases for FSCS-ART-DNC, and RRT-DNC. Therefore, we only plot the frequency distribution graph of FSCS-ART-DNC in Fig. 5. For ease of comparison, we replot the test case distribution of pure RT (the same as that in Fig. 1), which is expected to be uniform. The values of *max–min*, and $stdev$ for FSCS-ART-DNC,and RRT-DNC are summarized in Table 2.

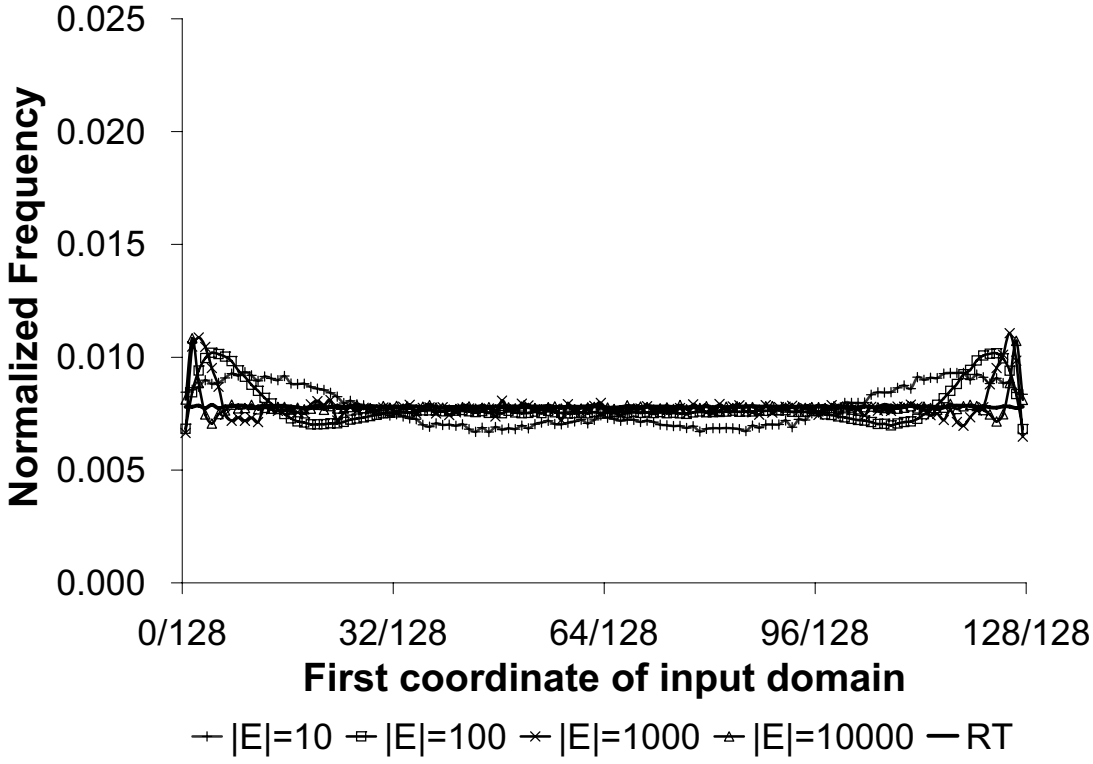
We compared the normalized frequencies (Figs. 1, and 5); and their related values of *max–min*, and $stdev$ (Tables 1, and 2), and observed that the test profile used in ART-DNC does offset the



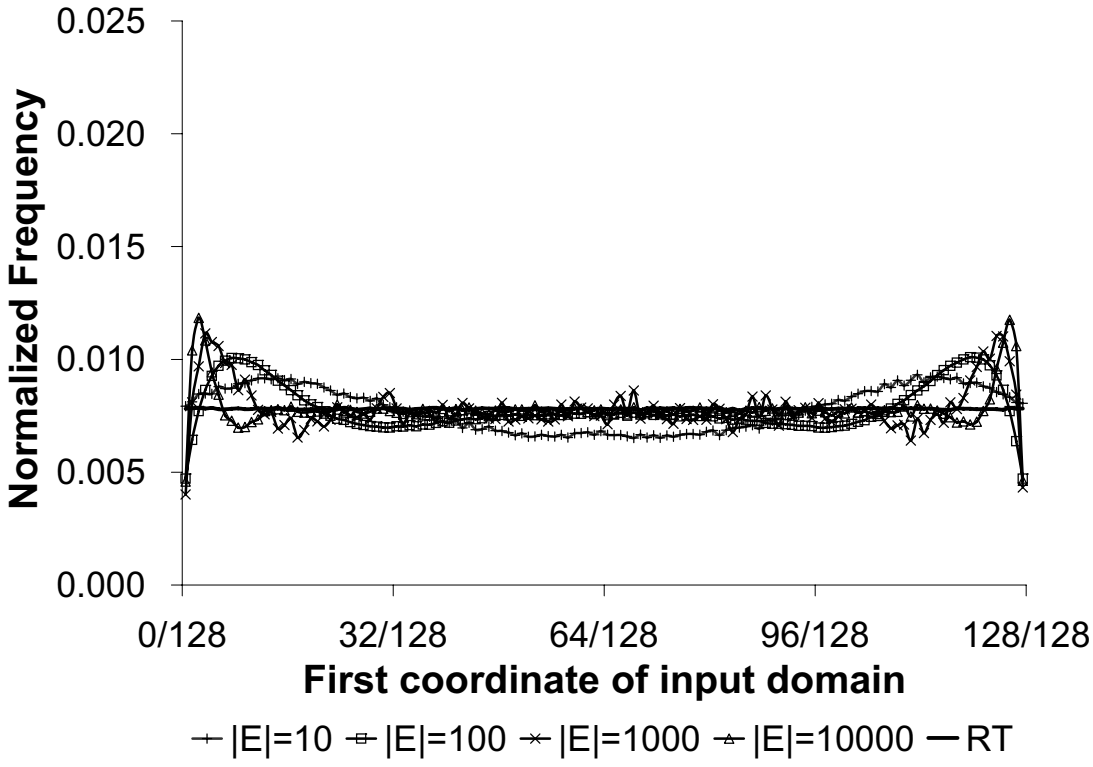
5.a Frequency distribution for 1D FSCS-ART-DNC



5.b Frequency distribution for 2D FSCS-ART-DNC



5.c Frequency distribution for 3D FSCS-ART-DNC



5.d Frequency distribution for 4D FSCS-ART-DNC

Figure 5. Frequency distribution of one coordinate of test cases generated by FSCS-ART-DNC

tc-biases of FSCS-ART, and RRT.

Table 2. Values of *max–min*, and *stdev* for FSCS-ART-DNC, and RRT-DNC

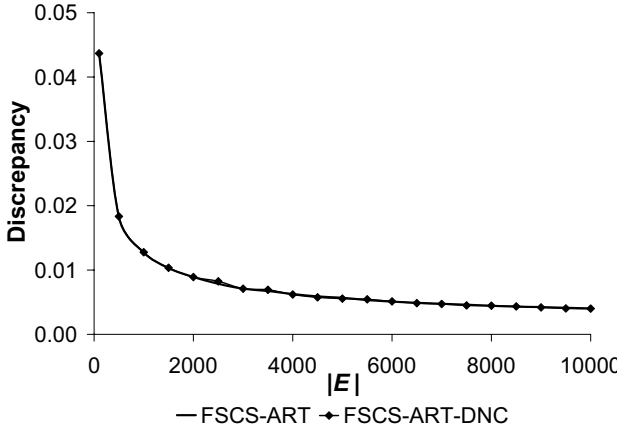
N	testing strategy	$\|E\| = 10$		$\|E\| = 100$		$\|E\| = 1000$		$\|E\| = 10000$	
		max–min	*stdev*	*max–min*	*stdev*	*max–min*	*stdev*	*max–min*	*stdev*
1	FSCS-ART-DNC	7.63E-03	1.16E-03	1.64E-03	1.52E-04	7.00E-04	9.52E-05	3.03E-04	3.71E-05
	RRT-DNC	5.50E-03	9.03E-04	1.42E-03	1.30E-04	5.70E-04	8.19E-05	2.29E-04	3.03E-05
2	FSCS-ART-DNC	3.40E-03	1.00E-03	2.94E-03	5.48E-04	2.19E-03	3.07E-04	9.63E-04	1.12E-04
	RRT-DNC	4.60E-03	1.42E-03	5.53E-03	9.68E-04	5.11E-03	5.62E-04	1.37E-03	1.60E-04
3	FSCS-ART-DNC	2.65E-03	8.88E-04	3.38E-03	8.20E-04	4.58E-03	7.00E-04	3.77E-03	4.18E-04
	RRT-DNC	4.76E-03	1.63E-03	7.74E-03	1.80E-03	7.92E-03	1.27E-03	6.51E-03	7.37E-04
4	FSCS-ART-DNC	2.80E-03	9.43E-04	5.38E-03	1.02E-03	7.14E-03	1.03E-03	7.25E-03	8.75E-04
	RRT-DNC	6.49E-03	2.27E-03	9.08E-03	2.06E-03	9.80E-03	1.73E-03	7.76E-03	1.25E-03

We further investigate the test case distribution of ART-DNC algorithms by repeating the simulations in [4] on FSCS-ART-DNC, and RRT-DNC. They distribute their test cases similarly. Therefore, we only plot discrepancy, and dispersion for FSCS-ART-DNC, with previous FSCS-ART's data for ease of comparison, in Figs. 6, and 7, respectively. The simulations results show that ART-DNC algorithms usually have smaller values of discrepancy than the original ART algorithms, and they have similar values of dispersion.

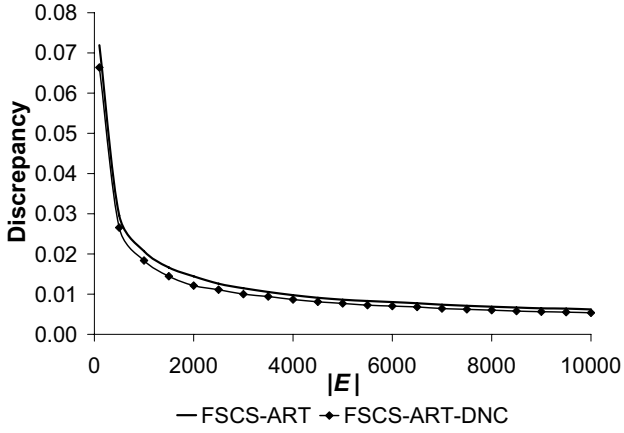
In summary, the experimental results have demonstrated that our approach achieves a more even spread of test cases. By using such a simple failure driven test profile (linear combination of two uniform-distributed variables) for candidate generation, we are able to achieve a more even test case distribution of ART.

3.5. Failure detection capabilities of new algorithms

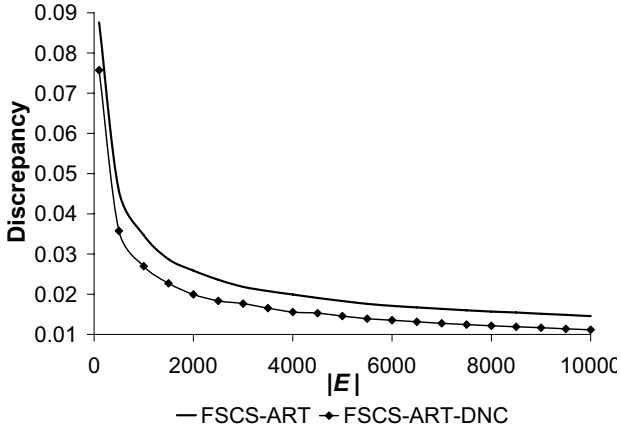
We conducted a series of simulations to investigate whether the more even test case distributions would improve the failure detection capability. We follow the simulation procedure as described in Section 2.3. We first investigated the failure detection capabilities of the new algorithms when failure-causing inputs are well clustered into one region. The experimental settings are as follows.



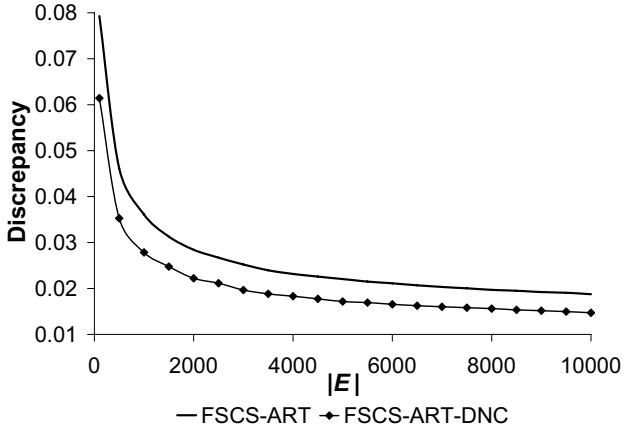
6.a Discrepancy in 1D space



6.b Discrepancy in 2D space

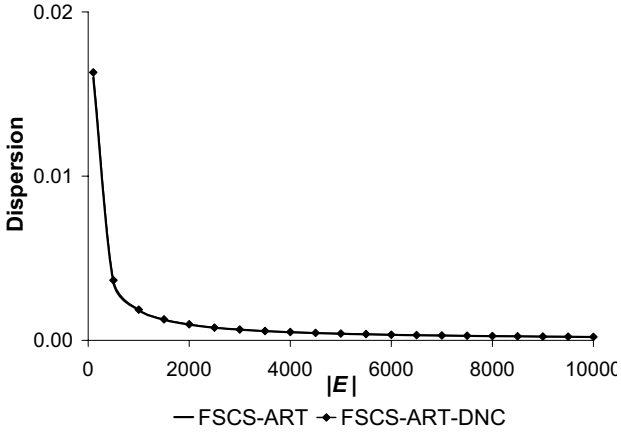


6.c Discrepancy in 3D space

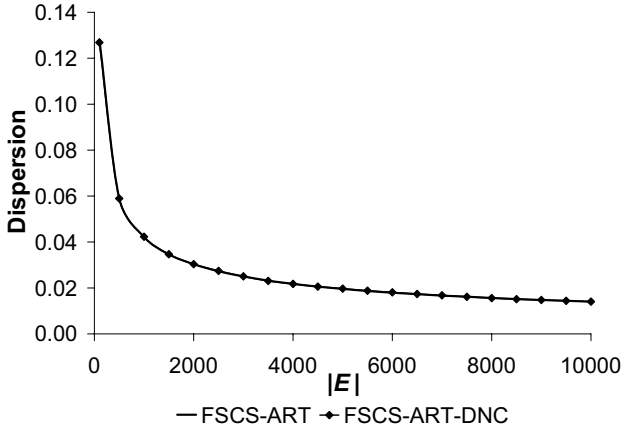


6.d Discrepancy in 4D space

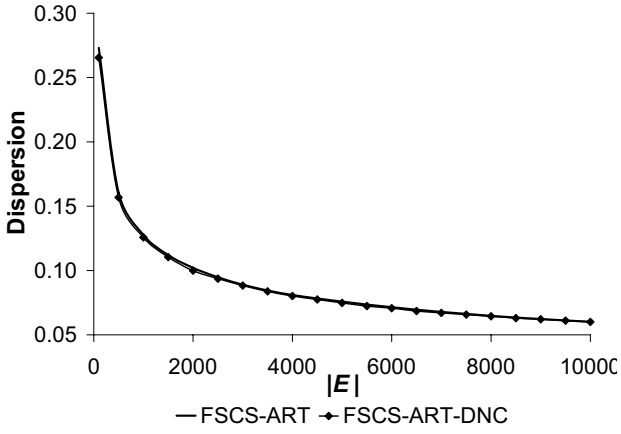
Figure 6. Comparison of discrepancy between FSCS-ART-DNC, and FSCS-ART.



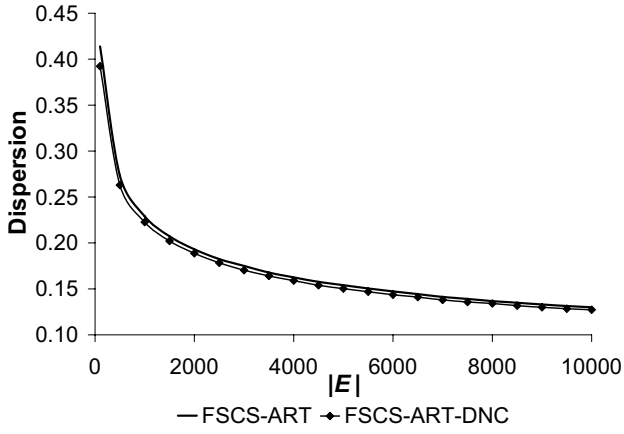
7.a Dispersion in 1D space



7.b Dispersion in 2D space



7.c Dispersion in 3D space



7.d Dispersion in 4D space

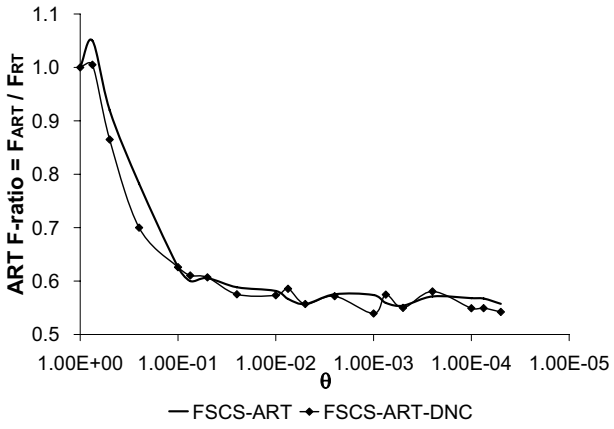
Figure 7. Comparison of dispersion between FSCS-ART-DNC, and FSCS-ART.

- N: 1, 2, 3, and 4.
- θ: 0.75, 0.5, 0.25, 0.1, 0.075, 0.05, 0.025, 0.01, 0.0075, 0.005, 0.0025, 0.001, 0.00075, 0.0005, 0.00025, 0.0001, 0.000075, and 0.00005.
- Failure pattern: a single square/cubic failure region is randomly placed inside I.

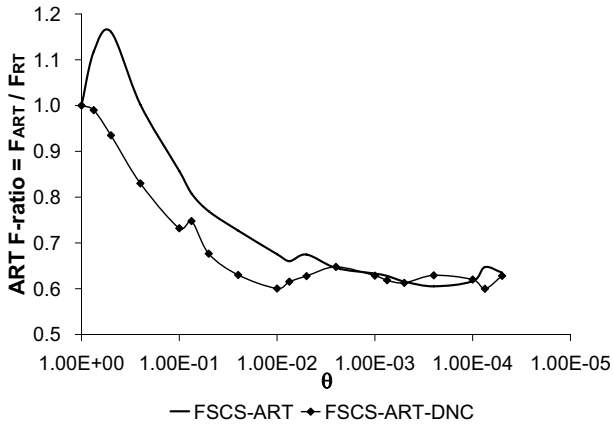
The results of the simulations are reported in Figs. 8, and 9. For ease of comparison, the simulation results of FSCS-ART, and RRT under the same experimental settings are also plotted.

Based on the experimental data, we have the following observations.

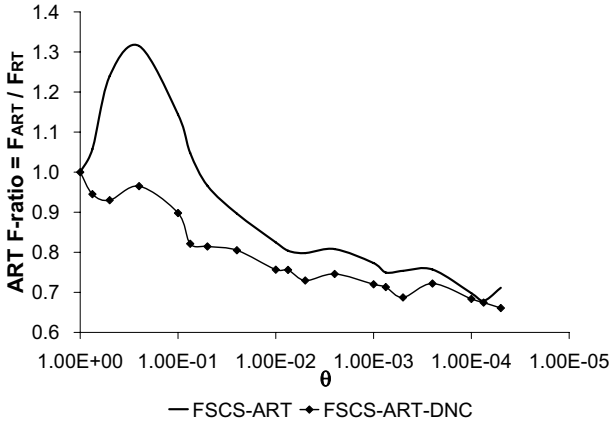
- Compared with the original FSCS-ART, and RRT algorithms, both FSCS-ART-DNC, and RRT-DNC algorithms have better or similar failure detection capabilities. On average, in 1D, 2D, 3D, and 4D spaces, FSCS-ART-DNC improves the failure detection capability of FSCS-ART by 1.65%, 6.66%, 10.35%, and 15.00%, respectively; and the performance improvements of RRT-DNC over RRT are 0.73%, 4.00%, 10.84%, and 17.38%, respectively.
- The higher N, or higher θ, the better the performance improvement of ART-DNC algorithms over their counterparts. In 1D space, FSCS-ART-DNC, and FSCS-ART have similar failure



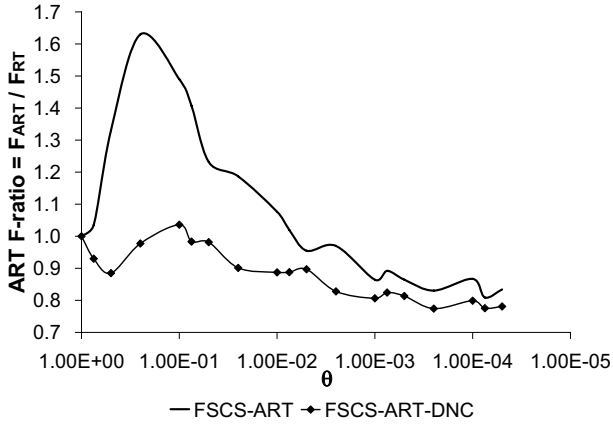
8.a FSCS-ART-DNC vs. FSCS-ART in 1D space



8.b FSCS-ART-DNC vs. FSCS-ART in 2D space

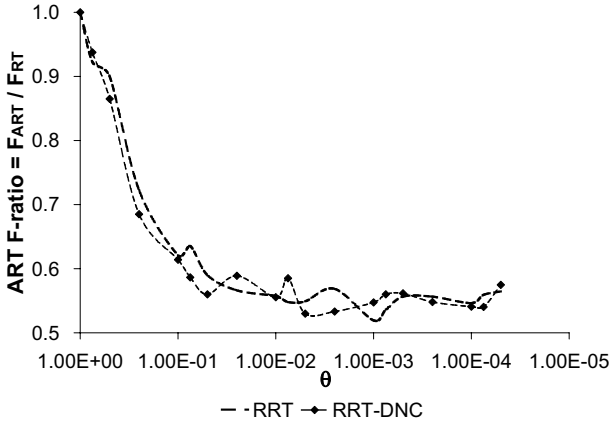


8.c FSCS-ART-DNC vs. FSCS-ART in 3D space

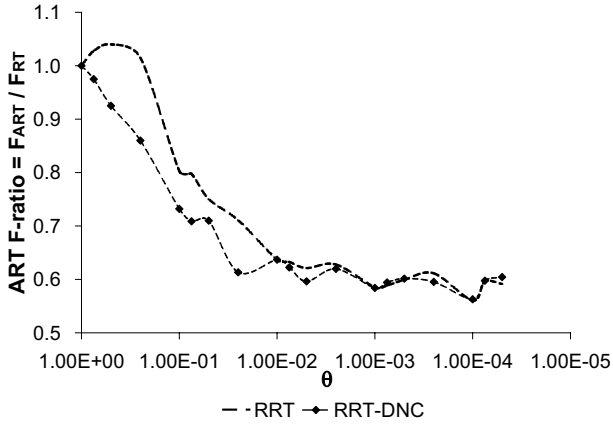


8.d FSCS-ART-DNC vs. FSCS-ART in 4D space

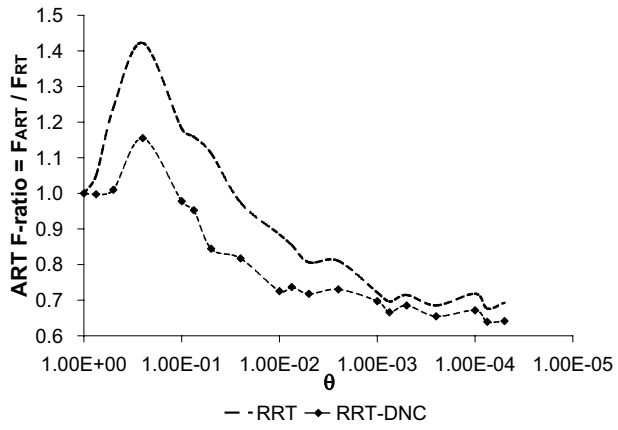
Figure 8. Comparison of failure detection capability between FSCS-ART-DNC, and FSCS-ART



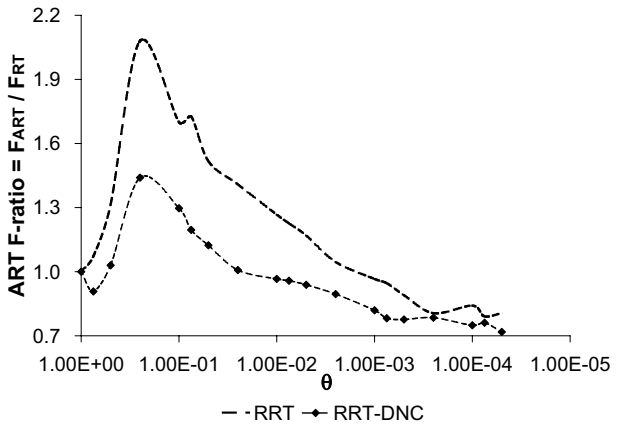
9.a RRT-DNC vs. RRT in 1D space



9.b RRT-DNC vs. RRT in 2D space



9.c RRT-DNC vs. RRT in 3D space



9.d RRT-DNC vs. RRT in 4D space

Figure 9. Comparison of failure detection capability between RRT-DNC, and RRT

detection capabilities; but in 4D space, the performance of FSCS-ART-DNC is 14.64%, or 40.03% better than FSCS-ART when θ is 0.0025, or 0.25, respectively.

Both ART-DNC algorithms can distribute their test cases more evenly than the original ART algorithms, so it is intuitively expected that the former have better failure detection capabilities than the latter. Thus, the first observation is consistent with our expectation. It has been shown in [4] that the test case distributions of the original algorithms are less even when either θ or N is higher. Therefore, it is understandable to have the second observation that the new ART-DNC algorithms outperform their counterparts more under the situations of higher θ, and higher N.

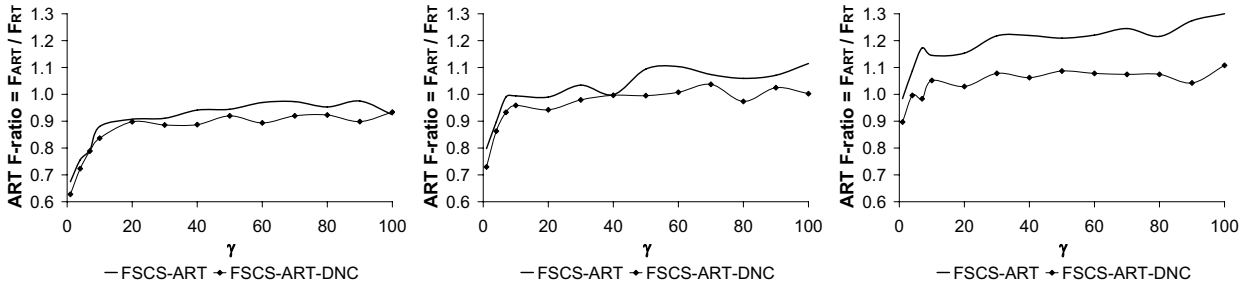
Besides N and θ, the performance of ART algorithms also depends on: a) the compactness of the failure region, b) the number of failure regions, and c) the size of any existing predominant failure region [6]. We conducted further simulations to investigate the performance of ART-DNC algorithms under various versions of such situations. As shown in Figs. 8, and 9, FSCS-ART-DNC, and RRT-DNC have similar trends of failure detection capabilities, so we only conducted the simulations on FSCS-ART-DNC with the following experimental settings.

- N: 2, 3, and 4.
- θ: 0.005, and 0.001.
- Experiment to investigate the impact of the compactness of failure region on the failure detection capability.
  - Failure pattern: a single rectangular/cuboid region is randomly placed inside I. The ratios among edge lengths of the rectangular/cuboid region are $1:\gamma$, $1:\gamma:\gamma$, and $1:\gamma:\gamma:\gamma$ in 2D, 3D, and 4D spaces, respectively, where $\gamma \geq 1$.
  - γ: 1, 4, 7, 10, 20, 30, 40, 50, 60, 70, 80, 90, and 100. As explained in [6], the larger γ is, the less compact the failure region is.
- Experiment to investigate the impact of the number of failure regions on the failure detection capability.
  - Failure pattern: a number of square/cubic regions are randomly placed inside I. Suppose that there are n failure regions, denoted by $R_1, R_2, \cdots, R_n$, respectively. For all regions, $|R_i| = \frac{\rho_i}{\sum_{j=1}^{n} \rho_j} \cdot \theta \cdot |I|$, where ρ_i is a random number uniformly distributed in $[0,1)$, and $i = 1, 2, \cdots, n$.
  - The number of failure regions: 1, 4, 7, 10, 20, 30, 40, 50, 60, 70, 80, 90, and 100.
- Experiment to investigate the impact of the existence, and the size of a predominant failure region on the failure detection capability.
  - Failure pattern: a number of square/cubic regions are randomly placed inside I. Suppose that there are n failure regions, denoted by $R_1, R_2, \cdots, R_n$, respectively. For one region R_n, set $|R_n| = \nu \cdot \theta \cdot |I|$, where $\nu = 0.3, 0.5$ and 0.8. For all the other regions, $|R_i| =$

$\frac{\rho_i}{\sum_{j=1}^{n-1}\rho_j} \cdot (1-\nu) \cdot \theta \cdot |I|$, where ρ_i is a random number uniformly distributed in $[0,1)$, and $i = 1, 2, \cdots, n-1$.

– The number of failure regions: 1, 4, 7, 10, 20, 30, 40, 50, 60, 70, 80, 90, and 100.

The simulations showed that, similar to FSCS-ART, FSCS-ART-DNC has a poorer failure detection capability when: a) the failure region is less compact, b) the number of failure regions is larger, or c) the size of the predominant failure region is smaller. However, FSCS-ART-DNC outperforms FSCS-ART in most scenarios, and the performance improvement becomes more significant with the increase of N or θ. Because the performance improvements of FSCS-ART-DNC over FSCS-ART for the cases of $\theta = 0.005$, and $\theta = 0.001$ are similar to each other, we only report the experimental results under the situation of $\theta = 0.005$ in Figs. 10, 11, and 12.

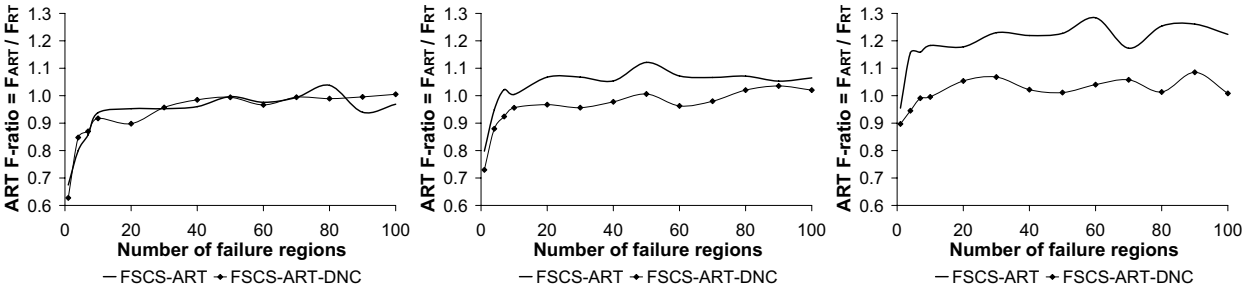


10.a 2D 10.b 3D 10.c 4D

Figure 10. Failure detection capabilities of FSCS-ART-DNC on a rectangular/cuboid failure region when $\theta = 0.005$.

4. Conclusions

RT, a fundamental software testing method, usually selects test cases according to a uniform distribution (for debug testing), or an operational profile (for reliability assessment). Failure detection capability is an important attribute of any testing method. Generally speaking, the better failure detection capability a testing method has, the more effectively program bugs can be removed, and

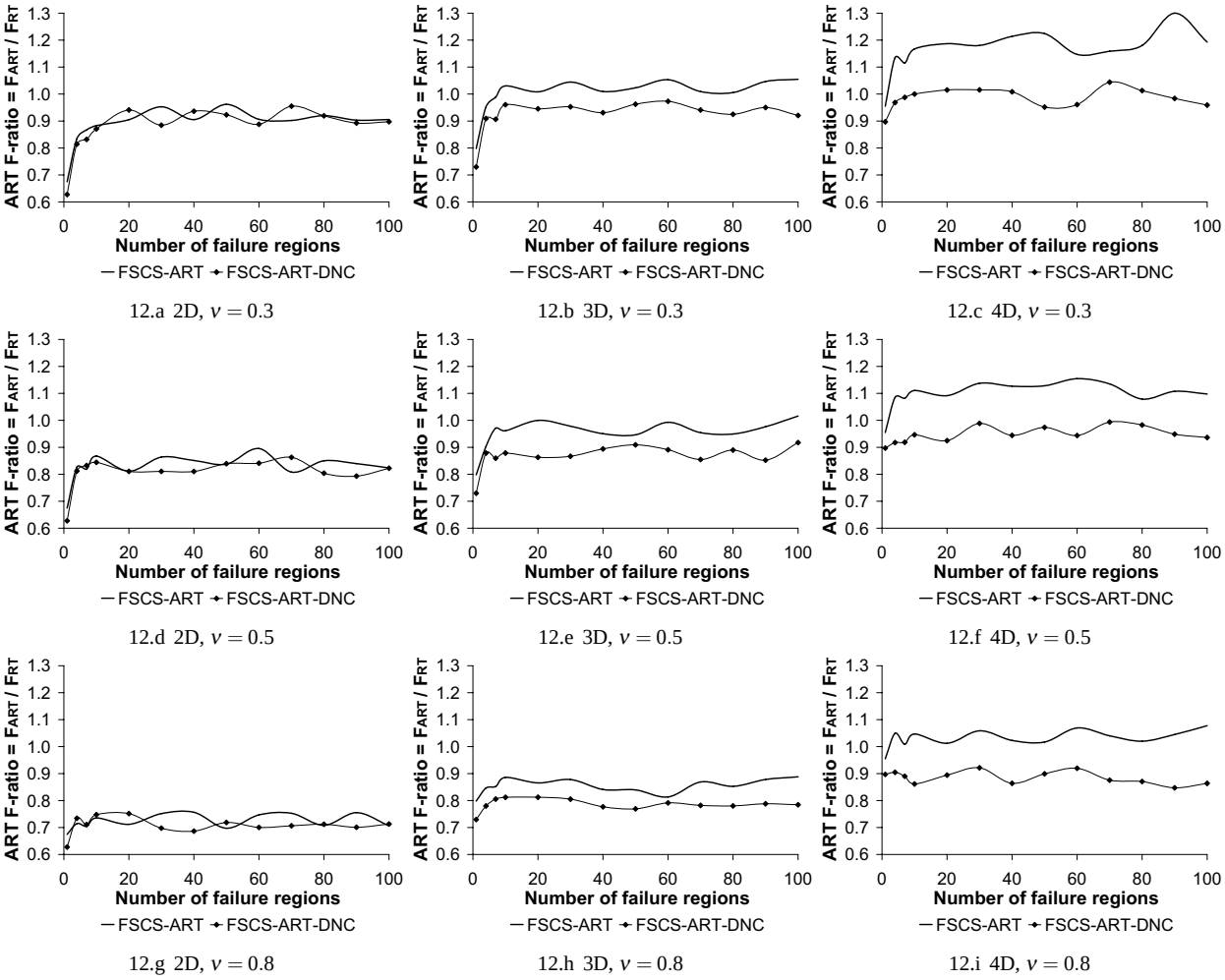


11.a 2D 11.b 3D 11.c 4D

Figure 11. Failure detection capabilities of FSCS-ART-DNC on multiple failure regions when $\theta = 0.005$.

thus the more significantly the software reliability may be improved. Neither the operational profile nor the uniform distribution makes use of any information about the probability distribution of failure-causing inputs. Therefore, RT has often been criticized to be likely to have a poor failure detection capability. Recently, motivated by the observation that failure-causing inputs are clustered into contiguous failure regions, Chen *et al.* proposed adaptive random testing (ART) to enhance the failure detection capability of RT. The basic principle of ART is to evenly spread random test cases over the input domain. Many ART algorithms randomly generate test case candidates according to uniform distribution, like RT in the context of debug testing. But they further use some criteria to identify test cases among candidates so as to ensure an even spread of executed test cases. There have been studies to enhance ART by distributing test cases more evenly, but all of them have adopted the approach of enhancing the test case identification process.

In this paper, motivated by the argument that the uniform-distributed test profile is not designed to ensure a good failure detection capability of RT, we proposed to use a different test profile at the candidate generation process. As an example of illustration, we selected a dynamic non-uniform distribution as the failure driven test profile to guide the random selection of candidates. We integrated this new test profile with the test case identification criteria of some existing ART algorithms, and developed a family of new ART algorithms, namely adaptive random testing with



12.a 2D, $\nu = 0.3$ 12.b 3D, $\nu = 0.3$ 12.c 4D, $\nu = 0.3$

12.d 2D, $\nu = 0.5$ 12.e 3D, $\nu = 0.5$ 12.f 4D, $\nu = 0.5$

12.g 2D, $\nu = 0.8$ 12.h 3D, $\nu = 0.8$ 12.i 4D, $\nu = 0.8$

Figure 12. Failure detection capabilities of FSCS-ART-DNC on multiple failure regions with one predominant region when $\theta = 0.005$.

dynamic non-uniform candidate distribution (ART-DNC). Our experimental results have shown that test cases selected by the ART-DNC algorithms are more evenly distributed than those selected by the corresponding ART algorithms, and that ART-DNC algorithms have better failure detection capabilities than their counterparts.

Note that the basic idea of our approach is not restricted to improve FSCS-ART, and RRT, but shall be applicable to enhance other ART algorithms. As a pilot study, we have only tried one non-uniform distribution as the new test profile for ART/RT. It is worthwhile to further investigate whether, and to what extent, other distributions can be used to enhance the failure detection capability of ART/RT.

A. Derivation of the probability density function of Y

The probability density functions (pdf) of X_1, and X_2 are

$$f_{X_1}(x_1) = \begin{cases} 0, & \text{when} \quad x_1 < 0 \text{ or } x_1 \geq 1 \\ 1, & \text{when} \quad 0 \leq x_1 < 1 \end{cases}, \quad f_{X_2}(x_2) = \begin{cases} 0, & \text{when} \quad x_2 < 0 \text{ or } x_2 \geq 1 \\ 1, & \text{when} \quad 0 \leq x_2 < 1 \end{cases} \tag{7}$$

Then, the probability distribution function (PDF) of Y, denoted by $F_Y(y)$, can be calculated as

$$\begin{aligned} F_Y(y) &= Pr(Y \leq y) = Pr(\alpha X_1 + (1-\alpha)X_2 \leq y) \\ &= \iint\limits_{\alpha x_1 + (1-\alpha)x_2 \leq y} f_{X_1}(x_1) f_{X_2}(x_2) \mathrm{d}x_1 \mathrm{d}x_2 = \int_{-\infty}^{\infty} \left[\int_{-\infty}^{\frac{y-(1-\alpha)x_2}{\alpha}} f_{X_1}(x_1) \mathrm{d}x_1 \right] f_{X_2}(x_2) \mathrm{d}x_2 \end{aligned} \tag{8}$$

Then, the pdf of Y, denoted by $f_Y(y)$, can be calculated as

$$f_Y(y) = \frac{\mathrm{d}}{\mathrm{d}y} F_Y(y) = \frac{1}{\alpha} \int_{-\infty}^{\infty} f_{X_1} \left[\frac{y-(1-\alpha)x_2}{\alpha} \right] f_{X_2}(x_2) \mathrm{d}x_2 \tag{9}$$

Obviously, when $0 \leq \dfrac{y-(1-\alpha)x_2}{\alpha} < 1$, i.e., $\dfrac{y-\alpha}{1-\alpha} < x_2 \leq \dfrac{y}{1-\alpha}$, $f_{X_1}\left[\dfrac{y-(1-\alpha)x_2}{\alpha}\right] = 1$; otherwise, $f_{X_1}\left[\dfrac{y-(1-\alpha)x_2}{\alpha}\right] = 0$. Therefore, $f_Y(y)$ can be calculated by the following steps.

- When $y < 0$, $f_Y(y) = 0$.
- When $y \geq 1$, $f_Y(y) = 0$.
- When $0 \leq y < \alpha$, $f_Y(y) = \dfrac{1}{\alpha} \displaystyle\int_0^{\frac{y}{1-\alpha}} \mathrm{d}x_2 = \frac{y}{\alpha(1-\alpha)}$.
- When $\alpha \leq y < 1-\alpha$, $f_Y(y) = \dfrac{1}{\alpha} \displaystyle\int_{\frac{y-\alpha}{1-\alpha}}^{\frac{y}{1-\alpha}} \mathrm{d}x_2 = \frac{1}{1-\alpha}$.
- When $1-\alpha \leq y < 1$, $f_Y(y) = \dfrac{1}{\alpha} \displaystyle\int_{\frac{y-\alpha}{1-\alpha}}^{1} \mathrm{d}x_2 = \frac{1-y}{\alpha(1-\alpha)}$.

B. Derivation of $P^l_{can-central}$

As defined in Section 3.3, $P^l_{can-central}$ refers to the probability of a point being within $[0.25, 0.75)$.

$$P^l_{can-central} = \int_{0.25}^{0.75} f_Y(y) \mathrm{d}y = 2 \int_{0.25}^{0.5} f_Y(y) \mathrm{d}y \tag{10}$$

- When $0 \leq \alpha < 0.25$, $P^l_{can-central}$ can be calculated as

$$P^l_{can-central} = 2\int_{0.25}^{0.5} \frac{1}{1-\alpha} \mathrm{d}y = \frac{1}{2(1-\alpha)} \tag{11}$$

- When $0.25 \leq \alpha \leq 0.5$, $P^l_{can-central}$ can be calculated as

$$\begin{aligned} P^l_{can-central} &= 2\int_{0.25}^{\alpha} \frac{y}{\alpha(1-\alpha)} \mathrm{d}y + 2\int_{\alpha}^{0.5} \frac{1}{1-\alpha} \mathrm{d}y = \frac{y^2}{\alpha(1-\alpha)}\Big|_{0.25}^{\alpha} + \frac{2y}{1-\alpha}\Big|_{\alpha}^{0.5} \\ &= \frac{\alpha^2 - 0.25^2}{\alpha(1-\alpha)} + \frac{1-2\alpha}{1-\alpha} = \frac{-\alpha^2 + \alpha - 0.0625}{\alpha(1-\alpha)} \\ &= 1 - \frac{1}{16\alpha(1-\alpha)} \end{aligned} \tag{12}$$

Acknowledgment

This research project is supported by an Australian Research Council Discovery Grant (DP0880295). We are grateful to Doug Grant, and Robert Merkel for their invaluable discussion.

References

[1] P. E. Ammann, and J. C. Knight, "Data diversity: an approach to software fault tolerance," *IEEE Transactions on Computers*, Vol. 37, No. 4, pp. 418–425, 1988.

[2] P. G. Bishop, "The variation of software survival times for different operational input profiles," *Proceedings of the 23rd International Symposium on Fault-Tolerant Computing (FTCS-23)*, pp. 98–107, 1993.

[3] K. P. Chan, T. Y. Chen, and D. Towey, "Restricted random testing: adaptive random testing by exclusion," *International Journal of Software Engineering and Knowledge Engineering*, vol. 16, No. 4, pp. 553–584, 2006.

[4] T. Y. Chen, F.-C. Kuo, and H. Liu, "On test case distributions of adaptive random testing," *Proceedings of the 19th International Conference on Software Engineering and Knowledge Engineering (SEKE 2007)*, pp. 141–144, 2007.

[5] T. Y. Chen, F.-C. Kuo, and H. Liu, "Distributing test cases more evenly in adaptive random testing," *The Journal of Systems and Software*, vol. 81 No. 12 pp.2146–2162, 2008.

[6] T. Y. Chen, F.-C. Kuo, and Z. Q. Zhou, "On favorable conditions for adaptive random testing," *International Journal of Software Engineering and Knowledge Engineering*, vol. 17 No. 6, pp. 805–825, 2007.

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

[8] T. Y. Chen, and R. Merkel, "An upper bound on software testing effectiveness," *ACM Transactions on Software Engineering and Methodology*, vol. 17, No. 3, pp. 16:1–16:27, 2008.

[9] I. Ciupa, A. Leitner, M. Oriol, and B. Meyer, "Object distance and its application to adaptive random testing of object-oriented programs," *Proceedings of the First International Workshop on Random Testing (RT'06)*, pp. 55–63, 2006.

[10] I. Ciupa, A. Leitner, M. Oriol, and B. Meyer, "ARTOO: adaptive random testing for object-oriented software," *Proceedings of the 30th International Conference on Software Engineering (ICSE'08)*, pp. 71–80, 2008.

[11] A. Denise, M.-C. Gaudel, and S.-D. Gouraud, "A generic method for stastical testing," *Proceedings of the 15th International Symposium on Software Reliability Engineering (ISSRE04)*, pp. 25–34, 2004.

[12] G. B. Finelli, "NASA software failure characterization experiments," *Reliability Engineering and System Safety*, Vol. 32, No. 1-2, pp. 155–169, 1991.

[13] P. G. Frankl, R. G. Hamlet, B. Littlewood, and L. Strigini, "Evaluating testing methods by delivered reliability," *IEEE Transactions on Software Engineering*, Vol. 24, No. 8, pp. 586–601, 1998.

[14] E. Girard, and J. Rault, "A programming technique for software reliability," *Proceedings of 1973 IEEE Symposium on Computer Software Reliability*, pp. 44–50, 1973.

[15] R. Hamlet, "Random testing," J. Marciniak, editor, *Encyclopedia of Software Engineering*. John Wiley & Sons, second edition, 2002.

[16] F.-C. Kuo, *On adaptive random testing*, PhD thesis, Faculty of Information and Communication Technologies, Swinburne University of Technology, 2006.

[17] J. Mayer, "Lattice-based adaptive random testing," *Proceedings of the 20th IEEE/ACM International Conference on Automated Software Engineering (ASE 2005)*, pp. 333–336, 2005.

[18] J. Mayer, and C. Schneckenburger, "Adaptive random testing with enlarged input domain," *Proceedings of the 6th International Conference on Quality Software (QSIC 2006)*, pp. 251–258, 2006.

[19] R. Merkel, *Analysis and Enhancements of Adaptive Random Testing*, PhD thesis, School of Information Technology, Swinburne University of Technology, 2005.

[20] J. D. Musa, "Operational profiles in software-reliability engineering," *IEEE Software*, Vol. 10, No. 2, pp. 14–32, 1993.

[21] G. J. Myers, *The Art of Software Testing*. John Wiley and Sons, second edition, 2004. Revised and updated by T. Badgett, and T. M. Thomas, with C. Sandler.

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

[23] T. A. Thayer, M. Lipow, and E. C. Nelson, *Software Reliability*, North-Holland Publishing Company, 1978.

[24] P. Thévenod-Fosse, and H. Waseselynck, "An investigation of software statistical testing," *The Journal of Software Testing, Verification and Reliability*, Vol. 1, No. 2, pp. 5–26, 1991.

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

[26] T. Yoshikawa, K. Shimura, and T. Ozawa, "Random program generator for Java JIT compiler test system," *Proceedings of the 3rd International Conference on Quality Software (QSIC 2003)*, pp. 20–24, 2003.

Tsong Yueh Chen is a Chair Professor of Software Engineering at the Faculty of Information and Communication Technologies in Swinburne University of Technology. He received his PhD degree in Computer Science from the University of Melbourne; MSc, and DIC in Computer Science from Imperial College of Science and Technology; and BSc., and MPhil. from The University of Hong Kong. His current research interests include software testing and debugging, software Maintenance, and software design.

T. Y. Chen, Professor
Faculty of Information and Communication Technologies
Swinburne University of Technology
John Street, Hawthorn, Victoria 3122, Australia
e-mail: tchen@ict.swin.edu.au

F.-C. Kuo is a Lecturer at the Faculty of Information and Communication Technologies in Swinburne University of Technology. She received her PhD degree in Software Engineering, and BSc. (Honors) in Computer Science, both from the Swinburne University of Technology. Her current research interests include software testing, debugging, and project management.

Fei-Ching Kuo, Lecturer
Faculty of Information and Communication Technologies
Swinburne University of Technology
John Street, Hawthorn, Victoria 3122, Australia
e-mail: dkuo@ict.swin.edu.au

Huai Liu is a PhD candidate, and a Research Associate at the Faculty of Information and Communication Technologies in Swinburne University of Technology. He received his MEng. in Communications and Information System, and BEng. in Physioelectronic Technology, both from Nankai University. His current research interests include software testing, and telecommunications.

Huai Liu
Faculty of Information and Communication Technologies
Swinburne University of Technology
John Street, Hawthorn, Victoria 3122, Australia
e-mail: hliu@ict.swin.edu.au