2010 10th International Conference on Quality Software

Adaptive Random Testing by Exclusion through Test Profile

Huai Liu[1], Xiaodong Xie[2], Jing Yang[2], Yansheng Lu[2], and Tsong Yueh Chen[1]

[1]*Centre for Software Analysis and Testing*
Swinburne University of Technology
Hawthorn 3122 VIC, Australia
Email: hliu@swin.edu.au; tychen@swin.edu.au

[2]*College of Computer Science and Technology*
Huazhong University of Science and Technology
Wuhan 430074 Hubei, P.R. China
Email: xiaodongxie@mail.hust.edu.cn; yjhust@smail.hust.edu.cn; lys@mail.hust.edu.cn

***Abstract*—One major objective of software testing is to reveal software failures such that program bugs can be removed. Random testing is a basic and simple software testing technique, but its failure-detection effectiveness is often controversial. Based on the common observation that program inputs causing software failures tend to cluster into contiguous regions, some researchers have proposed that an even spread of test cases should enhance the failure-detection effectiveness of random testing. Adaptive random testing refers to a family of algorithms to evenly spread random test cases based on various notions. Restricted random testing, an algorithm to implement adaptive random testing by the notion of exclusion, defines an exclusion region around each previously executed test case, and selects test cases only from outside all exclusion regions. Although having a high failure-detection effectiveness, restricted random testing has a very high computation overhead, and it rigidly discards all test cases inside any exclusion region, some of which may reveal software failures. In this paper, we propose a new method to implement adaptive random testing by exclusion, where test cases are simply selected based on a well-designed test profile. The new method has a low computation overhead and it does not omit any possible program inputs that can detect failures. Our experimental results show that the new method not only spreads test cases more evenly but also brings a higher failure-detection effectiveness than random testing.**

***Keywords*-Software testing, random testing, adaptive random testing by exclusion, restricted random testing, test profile.**

I. Introduction

Software testing is a main software engineering approach to the software quality assurance. *Random testing* (RT) [19] is a basic and simple software testing technique, which randomly selects test cases from the whole *input domain* (that is, the set of all possible program inputs). When used to detect software failures, RT often selects test cases according to a uniform distribution, that is, all program inputs have the same probability to be chosen as test cases in order that no program bug will be omitted. Although RT has been used to reveal failures in various programs [13], [20], [21], it is controversial whether it is an effective testing method or not [19], as it does not make use of any information of the program under test.

Many researchers [1], [3], [12] have independently observed a common information of faulty programs, that is, program inputs that cause software to exhibit failure behaviours, namely *failure-causing inputs*, tend to be clustered into contiguous *failure regions* [1]. Given that failure regions are contiguous, non-failure regions should also be contiguous. Suppose that a test case t does not reveal a failure. Test cases that are spread away from t may have a higher chance to be failure-causing than t's neighbours. Briefly speaking, an *even spread* of test cases can help improve the failure-detection effectiveness of RT. Based on such an intuition, Chen *et al.* [10] proposed an innovative approach, namely *adaptive random testing* (ART). Various ART algorithms [5], [10], [22] have been developed to achieve an even spread of test cases based on different notions, and ART has been used in various areas [14], [15], [17]. One notion to implement ART is *by exclusion*. *Restricted random testing* (RRT) [5] is one algorithm of ART by exclusion. RRT defines an exclusion region around each previously executed test case. Test case candidates, or simply *candidates*, are continuously generated (normally according to a uniform distribution) until one candidate is picked from outside all the exclusion regions, which is then used as the next test case.

Previous studies [5] have shown that RRT normally has a higher failure-detection effectiveness than RT. However, it is also known that RRT always requires a very high computation overhead. It has been justified that the runtime of RRT is in $O(n^2 \log n)$ for generating n test cases [18]. Moreover, RRT's scheme of discarding all points inside any exclusion region is too rigid, because it is possible that some points inside certain exclusion regions may be failure-causing.

In this paper, we propose a new method that implements ART by exclusion through simply generating test cases based on a well-designed test profile. In our method, all

1550-6002/10 $26.00 © 2010 IEEE
DOI 10.1109/QSIC.2010.61

already executed test cases will be considered to construct a probability distribution, which guides the random selection of the next test case. The paper is organised as follows. In Section II, we introduce some background information. In Section III, we design a test profile and propose an algorithm for ART by exclusion through the designed test profile. In Section IV, we report some simulation studies, which examine the computation overhead, test case distribution, and failure-detection effectiveness of our new method. Section V summarises the paper.

II. Background

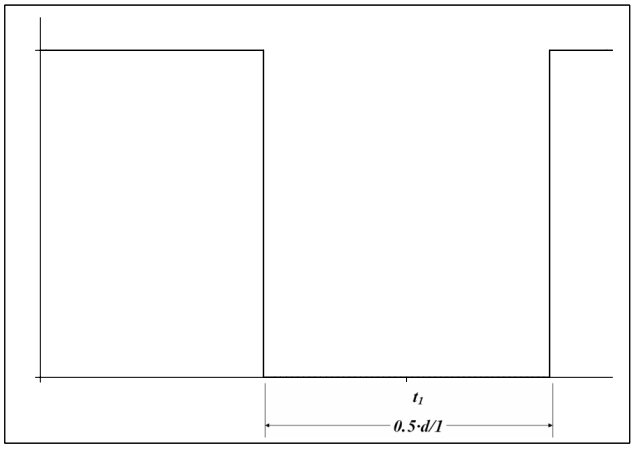
A. RRT: ART by exclusion

RRT [5] implements the notion of exclusion as follows. The first test case is randomly selected from the whole input domain according to a uniform distribution. When there are n ($n \geq 1$) executed test cases, RRT defines an exclusion region around each executed test case. All exclusion regions have the same size $R \cdot d/n$, where d denotes the size of the input domain, and R is referred to as the *target exclusion ratio* that should be preset by testers. Random candidates are generated from the input domain according to a uniform distribution one by one until one candidate is outside all exclusion regions, which will then be selected as the next test case. Figure 1 illustrates RRT in an one-dimensional input domain, where R is set as 0.5. Given the first test case t_1, the second test case can only be selected from outside the region $(t_1 - 0.25 \cdot d/1, t_1 + 0.25 \cdot d/1)$ (Figure 1(a)), and then the third test case from outside the regions $(t_1 - 0.25 \cdot d/2, t_1 + 0.25 \cdot d/2)$ and $(t_2 - 0.25 \cdot d/2, t_2 + 0.25 \cdot d/2)$ (Figure 1(b)).

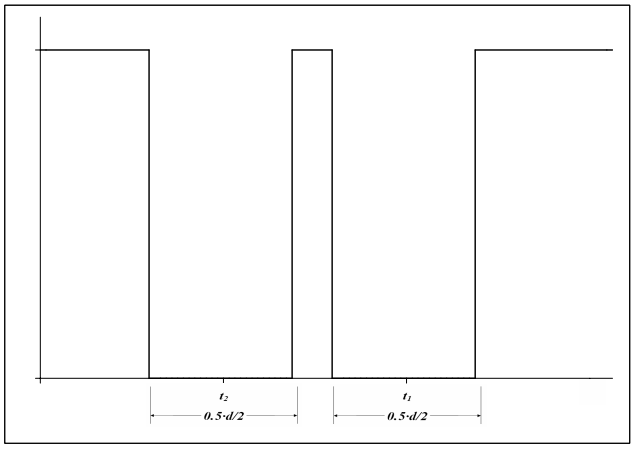
Simulations and empirical studies [5] have shown that RRT is more effective than RT in detecting software failures, and its failure-detection effectiveness becomes higher with the increase of R. Some guidelines have been identified for the setting of R.

Mayer and Schneckenburger [18] conducted an experiment to examine the computation overhead of RRT. They found that for selecting the n^{th} test case, RRT requires to generate $O(\log n)$ candidates. RRT measures the distance between each candidate and each executed test case to check whether the candidate is inside the exclusion region or not. As a result, RRT requires $O(n \log n)$ time to generate the n^{th} test case, and thus $O(n^2 \log n)$ time to select n test cases.

RRT can be considered from another perspective, that is, it selects test cases according to a *special* test profile. In this profile, the points outside the exclusion regions follow a uniform distribution, while all the points inside any exclusion region have no chance to be selected, as shown in Figure 1. However, such a scheme is too rigid, and may omit some failure-causing test cases. Since some program inputs inside exclusion regions may be failure-causing, it is necessary to allocate a non-zero probability to them.



(a) $n = 1$



(b) $n = 2$

Figure 1. Illustration of RRT with $R = 0.5$ in one-dimensional input domain

B. F-measure: measurement of failure-detection effectiveness

The failure-detection effectiveness of ART algorithms has been evaluated by *F-measure*, which refers to the expected number of test cases to detect the first failure. Previous studies [8] have demonstrated that the F-measure is particularly suitable for adaptive testing strategy (such as ART). The F-measure of an ART algorithm can be measured through simulation studies [5], [7], [9], as introduced in the following.

Failure-causing inputs determine two basic features of a faulty program. One feature is *failure pattern*, which refers to the geometry and distribution of failure regions. The other feature is *failure rate*, denoted by θ, which refers to the ratio between the number of failure-causing inputs and the total number of all program inputs. To simulate a faulty program, the failure pattern and θ are pre-defined, based on which, the size and shape of failure region will be decided.

Failure region is then randomly placed in the input domain. A testing strategy is implemented to generate test cases until a point inside the failure region is picked (that is, a failure is detected). The number of test cases to detect the first failure, referred to as the *F-count*, is recorded. Such a process is repeatedly run for a sufficient number of times to ensure that the mean value of F-counts can be regarded as a statistically reliable estimate of F-measure. Chen *et al.* [9] have conducted extensive simulations to study the F-measure of ART under various failure patterns. It was found that ART has a smaller F-measure when the dimension of input domain is lower, the failure region is more compact, or the number of failure regions is smaller.

C. Discrepancy and dispersion: measurement of test case distribution

Discrepancy and *dispersion* are two commonly used metrics to measure the equidistribution of sample points. Discrepancy intuitively indicates whether different regions inside the input domain have similar points' densities. One popular definition of discrepancy is the maximal difference of point densities for various regions. Dispersion intuitively indicates whether there is a large empty spherical region (containing no point) inside the input domain. Dispersion is often measured by the maximal distance that any point has from its nearest neighbour. Lower discrepancy and lower dispersion imply a better equidistribution of sample points. Chen *et al.* [6] have used discrepancy and dispersion to measure the test case distribution of various ART algorithms, and found that there is a strong correlation between the even spread of test cases and the failure-detection effectiveness.

III. A New Method for ART by Exclusion

In this paper, we propose a new method to implement ART by exclusion. In the method, test cases are simply selected according to a test profile which is different from the uniform distribution. The design of such a profile is introduced in the following section.

A. Design of a test profile for ART by exclusion

In order to implement the notion of exclusion, a test profile should have the following features.

- The more adjacent a point is to the already executed test cases, the less likely it should be selected as a test case.
- The farther away a point is from the already executed test cases, the more likely it should be selected as a test case.
- The probability distribution should be dynamically adjusted to preserve the above two characteristics.

To illustrate our method, we design a test profile as follows.

Suppose that inside a one-dimensional input domain $[0, 1)$, there are n executed test cases, $t_1, t_2, \cdots, t_n$. We reorder these test cases in ascending order, and then get $e_1, e_2, \cdots, e_n$, where $0 \leq e_1 \leq e_2 \leq \cdots \leq e_n < 1$. To select the next test case t_{n+1}, we construct the probability distribution $f_{n+1}(x)$ (where $x \in [0, 1)$ is a random variable) as follows.

- When $e_i \leq x < e_{i+1}$ $(i = 1, 2, \cdots, n-1)$, $f_{n+1}(x) = C_{n+1}(x - e_i)(e_{i+1} - x)$, where C_{n+1} is a constant whose value will be given later. As shown in Figure 2, the probability of $x = e_i$ or $x = e_{i+1}$ is 0, and the probability increases as x is approaching $(e_i + e_{i+1})/2$. In other words, the points that are close to e_i or e_{i+1} have low probability to be selected as test cases; while the points that are far from e_i and e_{i+1} have high probability to be selected.

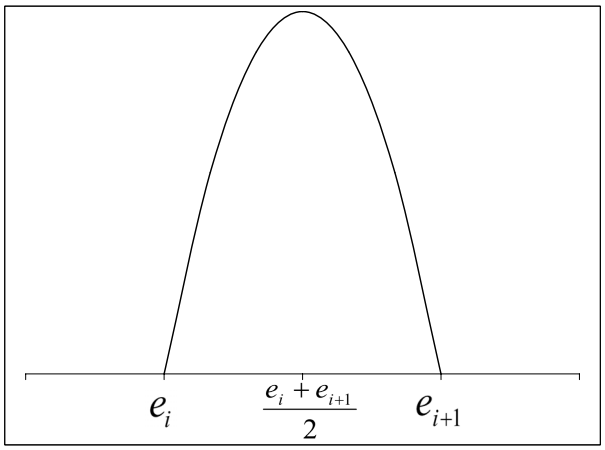
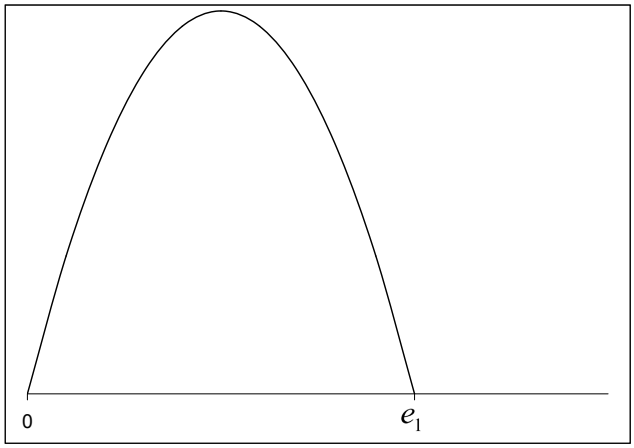
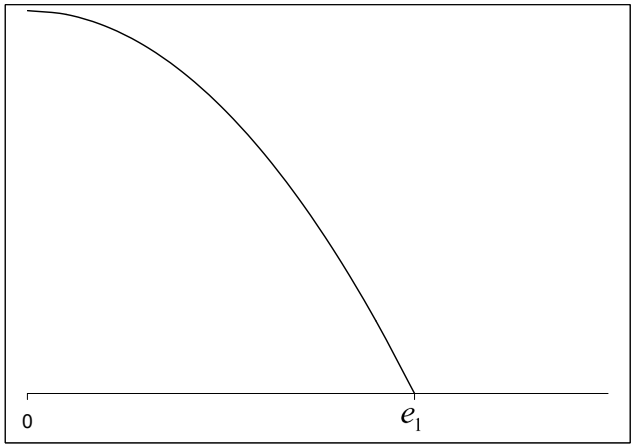


Figure 2. $f_{n+1}(x)$ when $e_i \leq x < e_{i+1}$ $(i = 1, 2, \cdots, n-1)$

- When $0 \leq x < e_1$, $f_{n+1}(x) = C_{n+1}(x - a_{n+1})(e_1 - x)$, where a_{n+1} is a random variable uniformly distributed in $[0 - e_1, 0]$. The reason behind the setting of a_{n+1} is explained as follows. If a_{n+1} is fixed rather than randomly selected, there would be a certain degree of bias on the points inside $[0, e_1)$. For example, if $a_{n+1} \equiv 0$ (Figure 3(a)), the probability of $x = 0$ is always 0, that is, there would be a bias of not selecting 0 as test case; while if $a_{n+1} \equiv 0 - e_1$ (Figure 3(b)), the probability of $x = 0$ would be very high, that is, there would be a bias of selecting 0 as test case. Therefore, in order to avoid any possible bias on test case selection, the value of a_{n+1} is randomly decided.
- When $e_n \leq x < 1$, $f_{n+1}(x) = C_{n+1}(x - e_n)(b_{n+1} - x)$, where b_{n+1} is a random variable uniformly distributed in $[1, 2 - e_n]$. The reason behind the setting of b_{n+1} is similar to that for a_{n+1}.
- According to the definition of probability, we should have $\int_0^1 f_{n+1}(x)dx = 1$, that is, $\int_0^{e_1} C_{n+1}(x -$



(a) $a_{n+1} \equiv 0$



(b) $a_{n+1} \equiv 0 - e_1$

Figure 3. $f_{n+1}(x)$ when $0 \leq x < e_1$

$a_{n+1})(e_1 - x)dx + \sum_{i=1}^{n-1} \int_{e_i}^{e_{i+1}} C_{n+1}(x - e_i)(e_{i+1} - x)dx + \int_{e_n}^{1} C_{n+1}(x - e_n)(b_{n+1} - x)dx = 1$. Therefore, we can calculate $C_{n+1} = 1/(\int_0^{e_1} (x - a_{n+1})(e_1 - x)dx + \sum_{i=1}^{n-1} \int_{e_i}^{e_{i+1}} (x - e_i)(e_{i+1} - x)dx + \int_{e_n}^{1} (x - e_n)(b_{n+1} - x)dx)$.

It should be noted that in ART algorithms, the first test case is always generated according to a uniform distribution, that is, $f_1(x) = 1$ when $0 \leq x < 1$. Some examples of $f_{n+1}(x)$ are illustrated in Figure 4.

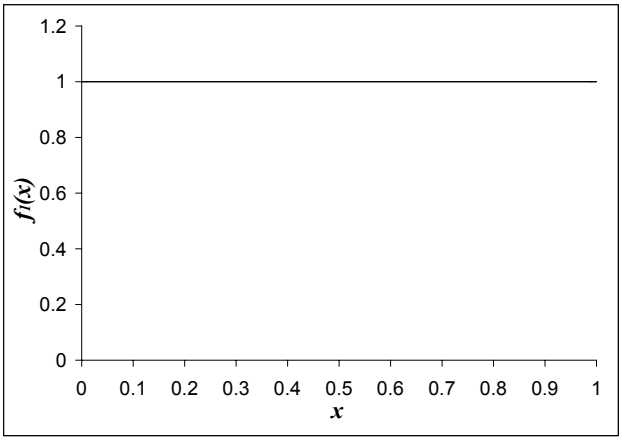
B. Generation of multidimensional test cases

In the previous section, we design a test profile that is suitable for one-dimensional input domain. In many situations, a program under test can have multiple input parameters, and thus the input domain can be multidimensional. One straightforward method for generating one m-dimensional ($m > 1$) test case is to simply combine m one-dimensional points, each of which is independently selected based on the test profile designed in Section III-A. However, an even spread on each coordinate does not necessarily imply the even spread over the whole input domain. Figure 5 gives an example of such a combination method. $\{x_1, x_2, \cdots, x_{10}\}$ (Figure 5(a)) and $\{y_1, y_2, \cdots, y_{10}\}$ (Figure 5(b)) are two sequences, each of which contains 10 test cases evenly spread inside the one-dimensional input domain $[0, 1)$. We simply combine them to a sequence of two-dimensional test cases, that is, $\{(x_1, y_1), (x_2, y_2), \cdots, (x_{10}, y_{10})\}$. As shown in Figure 5(c), the generated test cases are not evenly spread in the two-dimensional input domain. Therefore, we do not adopt such a combination method.

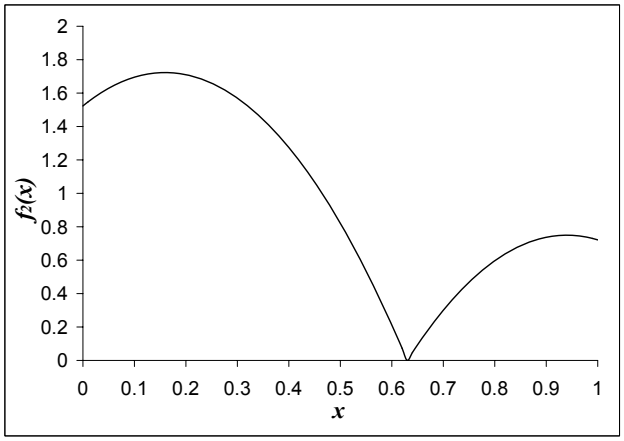
In this study, we construct multidimensional test cases using *Hilbert space-filling curve* function [16]. Hilbert space-filling curve can map the points on a one-dimensional line to points on a multidimensional curve. Figure 6 illustrates how to map one-dimensional points to two-dimensional points. A first-order Hilbert curve is plotted in Figure 6(a), where four one-dimensional points 0, 0.25, 0.5, and 0.75 are mapped to four two-dimensional points $(0.25, 0.25)$, $(0.25, 0.75)$, $(0.75, 0.75)$, and $(0.75, 0.25)$, respectively. The accuracy of such mapping will be improved as the order of the curve increases. Figure 6(b) depicts a second-order Hilbert curve, which is more precise than the first-order curve.

The Hilbert spacing-filling curve function used in this study is defined as $H_k^m : I \rightarrow I^m$, where $I = [0, 1)$, $m > 1$ is the dimension, and k is the order of Hilbert curve. Given a binary representation of one-dimensional point $P^1 = 0.u_1^1 u_2^1 \cdots u_m^1 u_1^2 u_2^2 \cdots u_m^2 \cdots u_1^k u_2^k \cdots u_m^k$, the m-dimensional point P^m is derived as follows. Each m bits $u_1^j u_2^j \cdots u_m^j (j = 1, 2, \cdots, k)$ is converted to $v_1^j v_2^j \cdots v_m^j$. Such conversion is run for k times. Finally, $P^m = (0.v_1^1 v_1^2 \cdots v_1^k, 0.v_2^1 v_2^2 \cdots v_2^k, \cdots, 0.v_m^1 v_m^2 \cdots v_m^k)$. For example, given $m = 3$, $k = 4$, and a one-dimensional point $P^1 = (0.247314453125)_{10} = (0.001111110101)_2$, where $(\cdot)_{10}$ and $(\cdot)_2$ denote the decimal and the binary representations of the point, respectively. We get $u_1^1 u_2^1 u_3^1 = 001$, $u_1^2 u_2^2 u_3^2 = 111$, $u_1^3 u_2^3 u_3^3 = 110$, and $u_1^4 u_2^4 u_3^4 = 101$. According to the transformation rule of the Hilbert curve [16], $u_1^1 u_2^1 u_3^1$, $u_1^2 u_2^2 u_3^2$, $u_1^3 u_2^3 u_3^3$, and $u_1^4 u_2^4 u_3^4$ are converted to $v_1^1 v_2^1 v_3^1 = 001$, $v_1^2 v_2^2 v_3^2 = 010$, $v_1^3 v_2^3 v_3^3 = 011$, and $v_1^4 v_2^4 v_3^4 = 111$, respectively. Therefore, the three-dimensional point $P^3 = (0.v_1^1 v_1^2 v_1^3 v_1^4, 0.v_2^1 v_2^2 v_2^3 v_2^4, 0.v_3^1 v_3^2 v_3^3 v_3^4) = (0.0001, 0.0111, 0.1011)_2 = (0.0625, 0.4375, 0.6875)_{10}$. Details of the conversion can be found in [16].

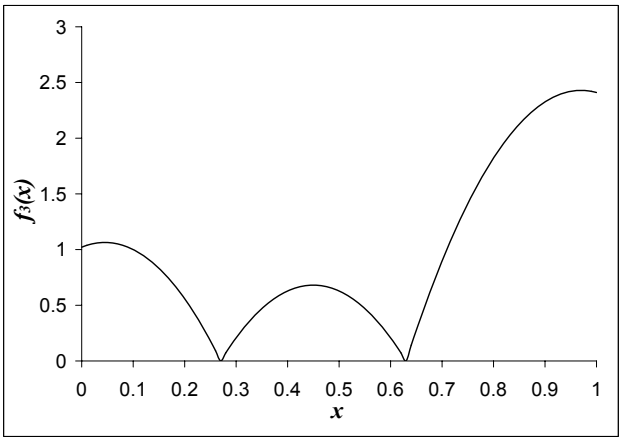
In this study, we first generate one-dimensional points according to the test profile designed in Section III-A, and then use the Hilbert space-filling curve function [16] to covert them into m-dimensional test cases.



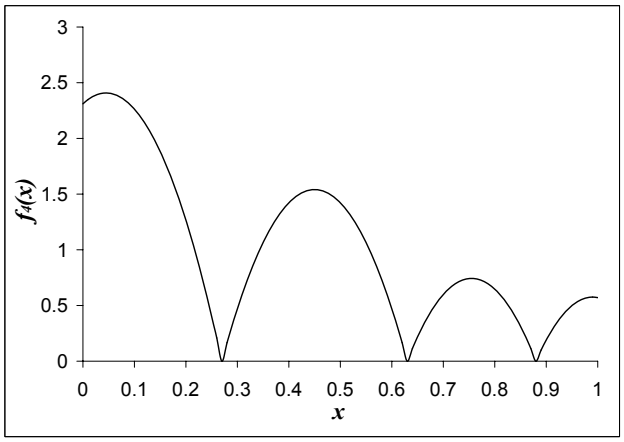
(a) $n = 0$



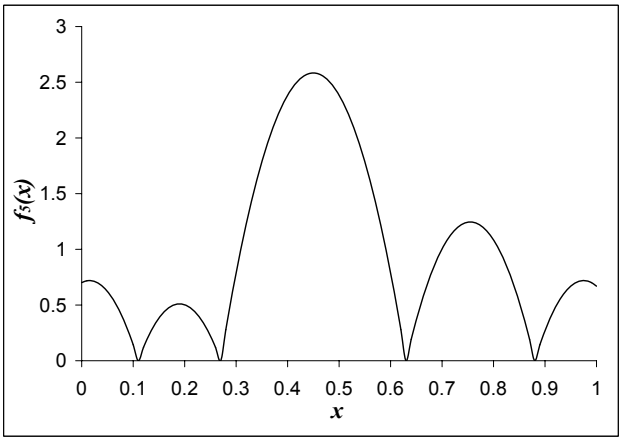
(b) $n = 1$



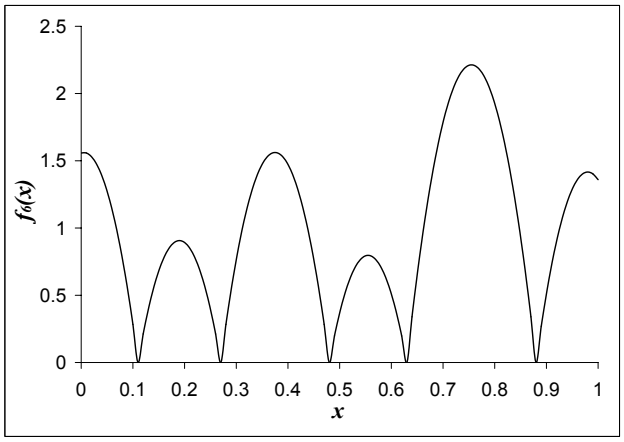
(c) $n = 2$



(d) $n = 3$

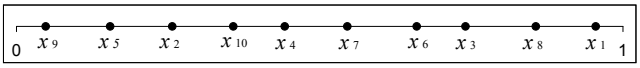


(e) $n = 4$

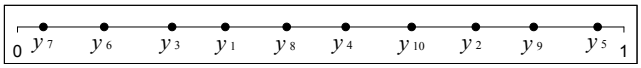


(f) $n = 5$

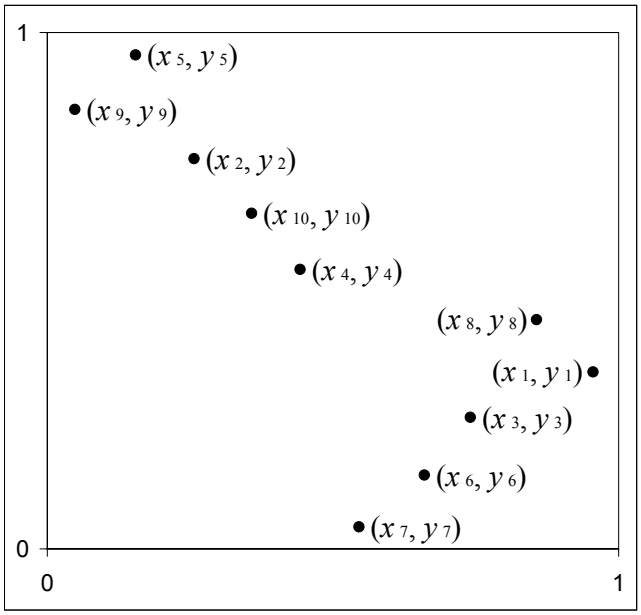
Figure 4. $f_{n+1}(x)$: the probability distribution to generate the $(n+1)^{\text{th}}$ test case



(a) $\{x_1, x_2, \cdots, x_{10}\}$



(b) $\{y_1, y_2, \cdots, y_{10}\}$



(c) $\{(x_1, y_1), (x_2, y_2), \cdots, (x_{10}, y_{10})\}$

Figure 5. An example of generating two-dimensional test cases by simply combining one-dimensional test cases

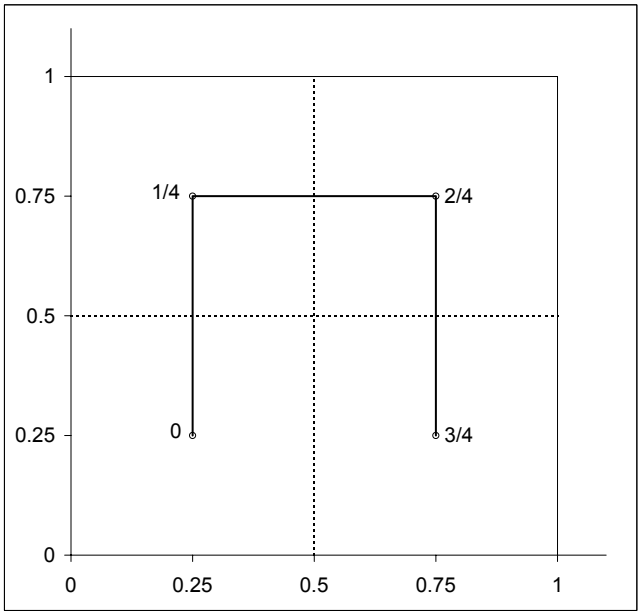
C. Algorithm

Detailed algorithm of ART by exclusion through test profile (abbreviated as ART-E_{TP} in this paper) is shown in Figure 7, where the termination condition can be "a certain number of test cases have been executed", or "the first software failure is detected", etc. For convenience of illustration, it is assumed in Figure 7 that the program under test only accepts numeric inputs. Readers can refer to [2], [11] for the application of ART into non-numeric programs.

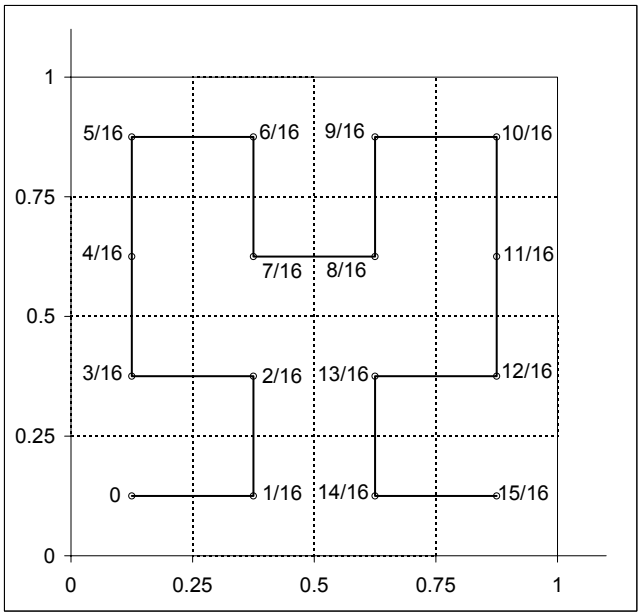
From Figure 4, we can see that the test profile in ART-E_{TP} only allocates zero probability to executed test cases, while all other points have certain probabilities to be selected as test cases. In other words, ART-E_{TP} does not omit any program input that may be failure-causing.

IV. Experiments

We conducted some simulations to study the execution time, test case distribution, and failure-detection effectiveness of ART-E_{TP}. The experimental results are reported as follows.



(a) First-order Hilbert curve



(b) Second-order Hilbert curve

Figure 6. Illustration of Hilbert space-filling curve

A. Execution time of ART-E_{TP}

In ART-E_{TP} (Figure 7), test cases are generated through three steps, (i) generating random number x according to the probability distribution $f_{n+1}(x)$ (Figure 4), (ii) converting x to an m-dimensional point p, and (iii) mapping p to a test case tc. Obviously, the last two steps require constant time. In this study, we implemented the first step in a naive way. First, we calculate the cumulative distribution function

/* m is the dimension of input domain.*/
/* The value range of i^{th} ($i = 1, 2, \cdots, m$) coordinate of the input domain is from O^i_{start} to O^i_{end}. */
1. Set $n = 0$ and $T = \{\}$.
2. **while** (termination condition does not meet)
3. Randomly generate a real number x from input domain, according to probability distribution $f_{n+1}(x)$ (Figure 4).
4. **if** ($m = 1$)
5. Set a one-dimensional point $p = (p_1)$, where $p_1 = x$.
6. **else**
7. Convert x to an m-dimensional point p = (p_1, p_2, $\cdots$, p_m) using Hilbert space-filling curve function.
8. **end_if**
9. Map p to a test case $tc = (tc_1, tc_2, \cdots, tc_m)$, where $tc_i = p_i \times (O^i_{end} - O^i_{start}) + O^i_{start}$.
10. Execute the test case tc.
11. Set $t_{n+1} = x$, and add t_{n+1} into T.
12. Increment n by 1.
13. Update the probability distribution $f_{n+1}(x)$.
14. **end_while**
15. Exit.

Figure 7. The algorithm of ART-E_{TP}

$F_{n+1}(x) = \int_{-\infty}^{x} f_{n+1}(\tau)d\tau$. Since $f_{n+1}(x)$ is a piecewise function composed of $n + 1$ segments, the calculation of $F_{n+1}(x)$ requires $O(n)$ time at the worst case. Second, a random number ρ is generated according to the uniform distribution, which only needs constant time. Finally, we calculate the solution of $F_{n+1}(x) = \rho$. $F_{n+1}(x)$ is also a piecewise function composed of $n + 1$ segments, so it also requires $O(n)$ time to find the solution. In summary, the execution time for generating the n^{th} test case is $O(n)$, and ART-E_{TP} requires $O(n^2)$ to generate n test cases. Obviously, ART-E_{TP} has a lower computation overhead than RRT (whose execution time is $O(n^2 \log n)$).

We have conducted some simulations to evaluate the execution time of ART-E_{TP}. The simulations were conducted on a machine with an Intel Pentium(R) Dual-Core CPU T4300 running at 2.10 GHz and 2048 megabytes of RAM. We recorded the time taken to generate n two-dimensional test cases, where n = 500, 1000, 1500, 2000, 2500, and 3000. The simulation results are given in Figure 8, in which, x- and y-axes denote n and the time required to generate n test cases, respectively. It is clearly shown that the execution time of ART-E_{TP} and RRT is $O(n^2)$ and $O(n^2 \log n)$, respectively, that is, the experimental data is consistent with the above analysis.

B. Test case distribution of ART-E_{TP}

A series of simulations were conducted to examine the discrepancy and dispersion of ART-E_{TP}. The dimension of input domain is 1, 2, 3, 4, or 5, the order of Hilbert curve (k) is set as 32, and the number of test cases are set as 100, 1000, or 10000. Discrepancy is calculated

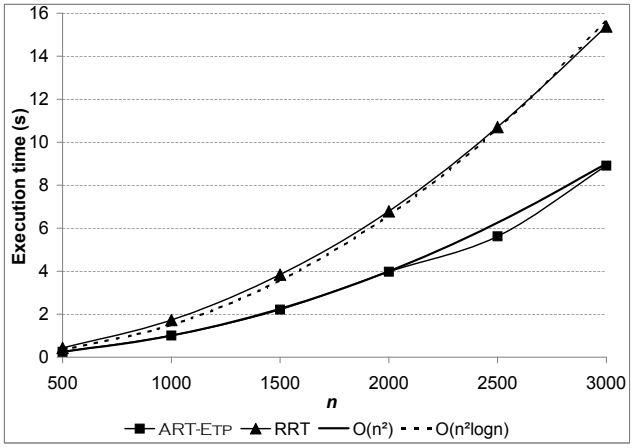


Figure 8. comparison of execution time between ART-E_{TP} and RRT

as $\max_{i=1,2,\cdots,1000} \left| \frac{n_i}{n} - \frac{|D_i|}{d} \right|$, where d denotes the size of the input domain, $D_1, D_2, \cdots, D_{1000}$ are 1000 randomly defined subdomains of the input domain, $|D_i|$ is the size of D_i, n denotes the total number of test cases, and n_i is the number of test cases in D_i. Dispersion is calculated as $\max_{i=1,2,\cdots,n} dist(t_i, \eta(t_i, T \backslash \{t_i\}))$, where $dist(p, q)$ denotes the Euclidean distance between two points p and q, $\eta(p, S)$ refers to p's nearest neighbour in set S, and $T = \{t_1, t_2, \cdots, t_n\}$ is the set of all test cases. The experimental results are shown in Tables I and II. For ease of comparison, Tables I and II also include the previous results on RT.

Table I
COMPARISON OF ART-E_{TP} AND RT BASED ON DISCREPANCY

Dimension	Number of test cases	RT	ART-E_{TP}
1	100	0.105615	0.054880
	1000	0.033989	0.016212
	10000	0.010514	0.004686
2	100	0.109297	0.082950
	1000	0.034987	0.021528
	10000	0.010599	0.005744
3	100	0.092499	0.104634
	1000	0.029106	0.028244
	10000	0.009178	0.008254
4	100	0.078544	0.118603
	1000	0.024617	0.034776
	10000	0.007688	0.015676
5	100	0.054966	0.121893
	1000	0.016942	0.049458
	10000	0.005150	0.047343

Based on the experimental data, we have the following observations. First, ART-E_{TP} has lower dispersion than RT.

Table II
COMPARISON OF ART-E_{TP} AND RT BASED ON DISPERSION

Dimension	Number of test cases	RT	ART-E_{TP}
1	100	0.027164	0.017770
	1000	0.003754	0.002100
	10000	0.000489	0.000217
2	100	0.148775	0.134675
	1000	0.053355	0.046886
	10000	0.018926	0.015734
3	100	0.285127	0.277940
	1000	0.144302	0.136340
	10000	0.071440	0.069181
4	100	0.413756	0.409610
	1000	0.244758	0.239933
	10000	0.146141	0.141043
5	100	0.526723	0.523930
	1000	0.346172	0.345200
	10000	0.226781	0.223155

Second, ART-E_{TP} has lower discrepancy than RT in one- and two-dimensional input domains. Third, the discrepancy of ART-E_{TP} increases with the increase of dimension, and becomes higher than that of RT in four- and five-dimensional input domains. Similar observation was also made for many other ART algorithms [6]. In summary, in terms of dispersion, ART-E_{TP} spreads test cases more evenly than RT, but with respect to discrepancy, ART-E_{TP} spreads test cases less evenly when the dimension of input domain is higher.

C. Failure-detection effectiveness of ART-E_{TP}

We also conducted some simulations to evaluate the F-measures of ART-E_{TP}, as reported in the following.

First, we set the failure pattern as a single hypercube failure region randomly placed inside the input domain, θ was set as 0.01, 0.005, 0.002, 0.001, or 0.0005, and the dimension of the input domain is 1, 2, 3, 4, or 5. The experimental results are summarised in Table III. Theoretically, the F-measure of RT with replacement is equal to $1/\theta$, so Table III does not include the F-measure of RT, but gives the improvement percentage of ART-E_{TP} over RT.

From Table III, we can observe that ART-E_{TP} always outperforms RT, but the failure-detection effectiveness of ART-E_{TP} differs for different dimensions of input domain. ART-E_{TP} can save almost 40% test cases than RT to detect the first failure for one-dimensional input domain, while the improvement become around 23%, 15%, 9%, and 6% for two-, three-, four-, and five-dimension, respectively. Briefly speaking, similar to other ART algorithms, the failure-detection effectiveness of ART-E_{TP} also depends on the dimension.

Based on the experimental results in Tables I, II, and III, we can observe that there is a correlation between the even spread of test cases and the failure-detection effectiveness. When the dimension of input domain is low, ART-E_{TP} not only achieves an even spread of test cases but also has a very small F-measure. With the increase of dimension, test cases selected by ART-E_{TP} show a certain degree of uneven distribution (that is, a high discrepancy), while the F-measure of ART-E_{TP} becomes larger. Such an observation is not surprising, as it is consistent with the basic notion of ART-E_{TP}, that is, an even spread of test cases can bring a high failure-detection effectiveness.

Table III
F-MEASURES OF ART-E_{TP} ON SINGLE HYPERCUBE FAILURE REGION

Dimension	θ	F-measure	Improvement over RT
1	0.01	62.55	37.45%
	0.005	122.61	38.70%
	0.002	311.12	37.78%
	0.001	620.98	37.90%
	0.0005	1261.35	36.93%
2	0.01	76.29	23.71%
	0.005	154.36	22.82%
	0.002	381.66	23.67%
	0.001	754.17	24.58%
	0.0005	1523.81	23.81%
3	0.01	86.64	13.36%
	0.005	169.31	15.34%
	0.002	422.93	15.41%
	0.001	830.44	16.96%
	0.0005	1705.47	14.73%
4	0.01	93.20	6.80%
	0.005	182.89	8.56%
	0.002	449.26	10.15%
	0.001	908.16	9.18%
	0.0005	1815.57	9.22%
5	0.01	94.58	5.42%
	0.005	186.11	6.95%
	0.002	470.57	5.89%
	0.001	959.09	4.09%
	0.0005	1845.51	7.72%

Second, we set the failure pattern as a single rectangular failure region randomly placed inside a two-dimensional input domain. The edge length ratio of the rectangular region is $1 : \alpha$, where $\alpha = 1, 4, 7, 10, 40, 70, 100$. Obviously, the rectangular region becomes less compact with the increase of α. θ was set as 0.005. Table IV reports the experimental results.

It can be observed from Table IV that similar to other ART algorithms, ART-E_{TP} has larger F-measures when the failure region is less compact.

Third, the failure pattern was set as multiple equal-sized square regions randomly placed inside a two-dimensional

Table IV
F-MEASURES OF ART-E_{TP} ON SINGLE RECTANGULAR FAILURE REGION

α	F-measure	Improvement over RT
1	154.36	22.82%
4	162.12	18.94%
7	166.10	16.95%
10	182.49	8.75%
40	183.18	8.41%
70	186.41	6.80%
100	192.43	3.78%

input domain. The number of failure regions β is set as 1, 4, 7, 10, 40, 70, 100, and $\theta = 0.005$. The experimental results are given in Table V.

Table V
F-MEASURES OF ART-E_{TP} ON MULTIPLE SQUARE FAILURE REGIONS

β	F-measure	Improvement over RT
1	154.36	22.82%
4	174.58	12.71%
7	184.11	7.94%
10	189.21	5.39%
40	193.32	3.34%
70	194.02	2.99%
100	195.86	2.07%

Based on the data in Table V, we observe that the F-measure of ART-E_{TP} becomes larger with the increase of the number of failure regions.

Briefly speaking, similar to other ART algorithms, ART-E_{TP} performs best when the failure-causing inputs are clustered into a compact region, but the failure-detection effectiveness of ART-E_{TP} depends on many factors, such as the dimension of input domain, the compactness of failure region, and the number of failure regions.

V. DISCUSSION AND CONCLUSION

Adaptive random testing (ART) [10] enhances the failure-detection effectiveness of random testing (RT) by evenly spreading test cases all over the input domain. One notion to achieve an even spread of test cases is by exclusion. The existing algorithm to implement ART by exclusion, namely restricted random testing (RRT) [5], has a very high computation overhead and has a very rigid exclusion scheme that may prevent some failure-causing inputs from being selected. In this paper, we proposed a new and simple method to implement ART by exclusion, which directly selects test cases according to a well-designed test profile. Compared with RRT, ART by exclusion through test profile (ART-E_{TP}) does not require a large number of random candidates for selecting each test case, and thus has a low computation overhead. The test profile used in ART-E_{TP} allocates certain probabilities to all possible program inputs except executed test cases, and thus does not omit any failure-causing input. Our experimental results showed that ART-E_{TP} not only brings lower dispersion, but also has a higher failure-detection effectiveness than RT.

ART with dynamic non-uniform candidate distribution (ART-DNC) [7] also uses a test profile that is dynamically adjusted during the testing process, but the profile is only used for the generation of candidates from which additional criteria must be applied to select one as the test case. However, the test profile in ART-E_{TP} is used to directly generate test cases from the input domain, without a preprocessing procedure of constructing test case candidates.

Cai *et al.* [4] recently proposed a technique namely *dynamic random testing* (DRT), where a dynamically updated test profile is used to randomly select test cases. In DRT, the input domain is first partitioned into k subdomains $(D_1, D_2, \cdots, D_k)$, each of which has a probability $(p_1, p_2, \cdots, p_k)$ to be selected. Program inputs in the same subdomain have the same chance to be chosen as test cases (that is, uniform distribution). When a test case from one subdomain D_i detects a failure, p_i will be increased; otherwise, p_i will be decreased. A uniform profile is still used for the program inputs in the same subdomain. Our ART-E_{TP} method does not partition the input domain; and the test profile in ART-E_{TP} was designed according to the intuition of evenly spreading test cases, which is not the principle of DRT.

The designs of ART-E_{TP} and ART-DNC [7] have the same objective, that is, to reveal the first failure as quickly as possible, while DRT aims at detecting software failures as many as possible. However, all these studies make use of dynamic adjustment of test profiles during the testing process to achieve their goals.

As a pilot study, this paper only demonstrated the new method based on one test profile. Apparently, various test profiles can be designed for ART by exclusion. It is worthwhile to further study the applicability of different adaptive test profiles, and examine to what extent these profiles can improve the failure-detection effectiveness. The F-measure of the algorithm demonstrated in the paper is not as small as those of some ART algorithms [7], [10]. There should be many rooms to further improve the effectiveness of ART-E_{TP}. Similar to other ART algorithms, ART-E_{TP} shows a certain degree of uneven test case distribution especially when the dimension of input domain is high. Spreading test cases more evenly may significantly improve the failure-detection effectiveness of ART-E_{TP}. In this paper, we generated multidimensional test cases by converting one-dimensional test cases by Hilbert space-filling curve function. It is interesting to study the effectiveness of other

multidimensional space-filling curve functions. It is also quite promising to construct some multidimensional test profiles and directly use these profiles into ART-E_{TP}.

ACKNOWLEDGMENT

This research project is supported by an Australian Research Council Discovery Grant (DP0880295).

REFERENCES

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

[2] A. C. Barus, T. Y. Chen, F.-C. Kuo, R. Merkel, and G. Rothermel. Adaptive random testing of programs with arbitrary input types: A methodology and an empirical study. Submitted for publication.

[3] 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*, pages 98–107, 1993.

[4] K.-Y. Cai, H. Hu, C.-H. Jiang, and F. Ye. Random testing with dynamically updated test profile. In *Proceedings of the 20th International Symposium On Software Reliability Engineering (ISSRE 2009)*, 2009. Fast abstract 198.

[5] 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*, 16(4):553–584, 2006.

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

[7] T. Y. Chen, F.-C. Kuo, and H. Liu. Application of a failure driven test profile in random testing. *IEEE Transactions on Reliability*, 58(1):179–192, 2009.

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

[9] 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*, 17(6):805–825, 2007.

[10] T. Y. Chen, T. H. Tse, and Y. T. Yu. Proportional sampling strategy: A compendium and some insights. *The Journal of Systems and Software*, 58(1):65–81, 2001.

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

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

[13] 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.

[14] H. Jaygarl, C. K. Chang, and S. Kim. Practical extensions of a randomized testing tool. In *Proceedings of the 33rd Annual International Computer Software and Applications Conference (COMPSAC 2009), Vol. I*, pages 148–153, 2009.

[15] B. Jiang, Z. Zhang, W. Chan, and T. Tse. Adaptive random test case prioritization. In *Proceedings of the 24th IEEE/ACM International Conference on Automated Software Engineering (ASE 2009)*, pages 233–244, 2009.

[16] J. Lawder. Calculation of mappings between one and n-dimensional values using the Hilbert space-filling curves. Technical Report JL1/00, School of Computer Science and Information Systems, Birkbeck College, University of London, 2000.

[17] Y. Lin, X. Tang, Y. Chen, and J. Zhao. A divergence-oriented approach to adaptive random testing of Java programs. In *Proceedings of the 24th IEEE/ACM International Conference on Automated Software Engineering (ASE 2009)*, pages 221–232, 2009.

[18] J. Mayer and C. Schneckenburger. An empirical analysis and comparison of random testing techniques. In *Proceedings of the 2006 ACM/IEEE International Symposium on Empirical Software Engineering (ISESE 2006)*, pages 105–114, 2006.

[19] 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.

[20] J. Regehr. Random testing of interrupt-driven software. In *Proceedings of the 5th ACM International Conference on Embedded Software (EMSOFT 2005)*, pages 290–298, 2005.

[21] K. Sen, D. Marinov, and G. Agha. CUTE: A concolic unit testing engine for C. In *Proceedings of the 10th European Software Engineering Conference held jointly with 13th ACM SIGSOFT International Symposium on Foundations of Software Engineering (ESEC/FSE 2005)*, pages 263–272, 2005.

[22] A. F. Tappenden and J. Miller. A novel evolutionary approach for adaptive random testing. *IEEE Transactions on Reliability*, 58(4):619–633, 2009.