Enhancing mirror adaptive random testing through dynamic partitioning

Huang, Rubing; Liu, Huai; Xie, Xiaodong; Chen, Jinfu

Huang, R., Liu, H., Xie, X., & Chen, J. (2015). Enhancing mirror adaptive random testing through dynamic partitioning. Information and Software Technology, 67, 13–29.
https://doi.org/10.1016/j.infsof.2015.06.003
Document Version: Accepted Manuscript

Published Version: https://doi.org/10.1016/j.infsof.2015.06.003

Enhancing Mirror Adaptive Random Testing through Dynamic Partitioning

Rubing Huang[a], Huai Liu[b,*], Xiaodong Xie[c], Jinfu Chen[a]

[a] *School of Computer Science and Telecommunication Engineering, Jiangsu University, Zhenjiang 212013 Jiangsu, China*
[b] *Australia-India Research Centre for Automation Software Engineering, RMIT University, Melbourne 3001 VIC, Australia*
[c] *School of Computer Science and Technology, Huaqiao University, Xiamen 361021 Fujian, China*

Abstract

Context: Adaptive random testing (ART), originally proposed as an enhancement of random testing, is often criticized for the high computation overhead of many ART algorithms. Mirror ART (MART) is a novel approach that can be generally applied to improve the efficiency of various ART algorithms based on the combination of “divide-and-conquer” and “heuristic” strategies.

Objective: The computation overhead of the existing MART methods is actually on the same order of magnitude as that of the original ART algorithms. In this paper, we aim to further decrease the order of computation overhead for MART.

Method: We conjecture that the mirroring scheme in MART should be dynamic instead of static to deliver a higher efficiency. We thus propose a new approach, namely dynamic mirror ART (DMART), which incrementally partitions the input domain and adopts new mirror functions.

Results: Our simulations demonstrate that the new DMART approach delivers comparable failure-detection effectiveness as the original MART and ART algorithms while having much lower computation overhead. The experimental studies further show that the new approach also delivers a better and more reliable performance on programs with failure-unrelated parameters.

Conclusion: In general, DMART is much more cost-effective than MART. Since its mirroring scheme is independent of concrete ART algorithms, DMART can be generally applied to improve the cost-effectiveness of various ART algorithms.

Keywords: software testing; random testing; adaptive random testing; mirror adaptive random testing

1. Introduction

Software testing has been widely acknowledged as a mainstream technique for assessing and improving software quality. One basic approach to testing is to randomly generate test cases from the set of all possible program inputs (namely the *input domain*). Though very simple, *random testing* (RT) is still considered as one of the state-of-the-art testing techniques, along with other more complicated and systematic testing methods [1, 2]. RT may be the unique testing method that can be used for both operational testing (where the software reliability is estimated) and debug testing (where software failures are actively detected with the purpose of removing relevant bugs) [3]. Despite the controversies in the effectiveness of RT as a debug testing method [4], it has been popularly used to test various systems, such as UNIX utility programs [5], Windows NT applications [6], Java Just-In-Time compilers [7], embedded software systems [8], SQL database systems [9].

Besides the applications of RT into different domains, much research has been conducted on how to improve its effectiveness in detecting failures. *Adaptive random testing* (ART) [10] is one major approach to enhancing RT. The basic idea of ART was motivated by the common observation made by researchers from different areas: The *failure-causing inputs* (i.e., program inputs that can reveal failures) tend to be clustered into contiguous *failure regions* [11–14]. Given that the failure regions are contiguous, the non-failure regions should also be contiguous. In other words, adjacent program inputs show a certain degree of similarity in failure-revealing behaviors. According to this intuition, Chen *et al.* [10] conjectured that test cases should be evenly spread across the whole input domain for achieving high failure-detection effectiveness, and proposed ART to implement the notion of “even spread”. Since the inception of ART, many ART algorithms have been proposed, such as *fixed-sized-candidate-set ART* (FSCS-ART) [10], *lattice-based ART* (LART) [15], and *restricted random testing* (RRT) [16]. ART has also been applied to test various programs [17–19].

Previous studies [10, 15–19] have shown that ART can use fewer test cases than RT to detect the first software failure. However, the high computation overhead of many ART algorithms brings severe criticism and limits ART’s adoption in practice [20]. In order to improve the testing efficiency of ART, many overhead reduction strategies have been proposed [21–26]. Among these strategies, a well-studied testing method is *mirror adaptive random testing* (MART) [21], which is a novel approach based on the combination of “divide-and-conquer” and “heuristic” strategies. MART first divides the whole input domain into equal-sized disjoint subdomains. Then, one subdomain is chosen as the *source domain* while others as the *mirror domains*. MART generates test cases in the source do-

[*] Corresponding author
Email addresses: rbhuang@ujs.edu.cn (Rubing Huang), huai.liu@rmit.edu.au (Huai Liu), xiaodongxie@hqu.edu.cn (Xiaodong Xie), jinfuchen@ujs.edu.cn (Jinfu Chen)

main according to one existing ART algorithm, and then maps each test case from the source domain into the so-called mirror test cases in the mirror domains. As shown in previous studies [21, 27], MART can reduce computation overhead of the original ART algorithms while maintaining similar failure-detection effectiveness.

However, the computation overhead of the existing MART methods actually has the same order of magnitude as that of the original ART algorithms. For example, one ART algorithm, FSCS-ART, has the computation overhead of $O(n^2)$ for generating n test cases. According to previous investigations [21], the MART based on the original FSCS-ART algorithm requires about $O(n^2/m^2)$ time to generate n test cases, where m is the number of subdomains. In other words, the computation overhead of MART based on FSCS-ART is also in the quadratic order.

In this paper, we propose an enhanced MART method, namely *dynamic mirror adaptive random testing* (DMART), which divides the input domain incrementally along the testing process. The simulation results indicate that compared with original MART and ART algorithms, DMART requires much less computation overhead while delivering comparable failure-detection effectiveness. Our empirical studies further show that the new method also has a better and more reliable performance than original MART algorithms on real-life programs especially when there exist some input parameters that are not related to failures.

The paper is organized as follows. Section 2 introduces some background information of ART and MART. Section 3 discusses drawbacks of MART, and then proposes our new DMART method. Section 4 reports our experimental studies, which examine the computational overhead and failure-detection effectiveness of the new approach. The experimental results are given in Section 5. Section 6 discusses the threats to validity of our study. Section 7 presents some related work. Section 8 summarizes the paper.

2. Background

2.1. Adaptive Random Testing

Similar to RT, *adaptive random testing* (ART) [10] also randomly generates program inputs from the input domain. However, ART makes use of additional criteria to choose inputs as test cases in order to evenly spread test cases over the input domain. There are many criteria to guide the selection of test cases, one criterion of which is by distance. *Fixed-sized-candidate-set ART* (FSCS-ART) [10] is one typical algorithm of ART by distance. FSCS-ART uses two test case sets, the *executed set* denoted by E and the *candidate set* denoted by $C = \{c_1, c_2, \cdots, c_k\}$. E contains all test cases which were already executed without revealing any failure; while C contains k randomly generated inputs, where k is assigned by testers before testing and keeps unchanged throughout the testing process. An input in C will be chosen as the next test case in E if it has the longest distance to its nearest neighbor in E.

Previous simulations and empirical studies [10, 15, 16] have demonstrated that the failure-detection effectiveness of ART is better than that of RT in terms of detecting the first software failure using fewer test cases. However, there exists a criticism [20] of ART due to the high computation overhead of many ART algorithms. For example, FSCS-ART requires $O(n^2)$ time to generate n test cases.

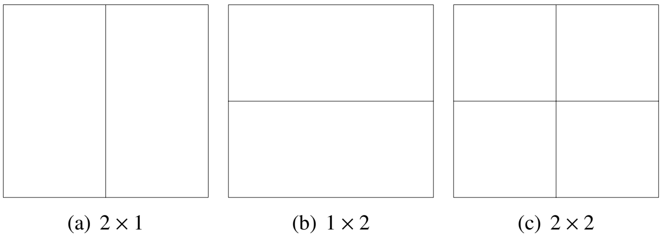


Fig. 1: Some simple ways of mirror partitioning

2.2. Mirror Adaptive Random Testing

As discussed before, many ART algorithms may face the criticism of high computation overhead. To improve the efficiency of ART, Chen *et al.* [21] proposed a novel overhead reduction strategy, namely *mirror adaptive random testing* (MART), which could be generally applied to many existing ART algorithms.

Before testing, MART first divides the input domain into some disjoint and equal-size subdomains, and then assigns one subdomain as the *source domain* while others as the *mirror domains*. After that, MART applies original ART algorithm in the source domain to generate a test case *tc* (namely *source test case*). Then, MART uses a function to map *tc* from the source domain into all mirror domains, to construct other test cases (namely *mirror test cases*).

According to previous studies [21, 27], there are three major components of the *mirroring scheme* in MART, namely *mirror partitioning*, *mirror function*, and *mirror selection order*.

2.2.1. Mirror partitioning

Suppose that the dimension of the input domain is $d \geq 1$. In MART, each coordinate of the input domain is divided into $u_i \geq 1$ $(i = 1, 2, \cdots, d)$ parts of the equal length. Totally, the input domain is partitioned into $u_1 \times u_2 \times \cdots \times u_d$ subdomains. Fig. 1 shows some simple ways of mirror partitioning that could be used for MART with the 2-dimensional input domain. Since the use of a large number of mirror domains "may introduce duplicated test case patterns" (i.e., the distribution of test cases is duplicated in each subdomain) that "may destroy the overall randomness of test case selection", a small number of mirror domains would be more appropriate for MART [21]. In our experimental studies, therefore, we follow the practice adopted in previous studies [21] of choosing mirror partitioning with a small number of mirror domains for MART.

2.2.2. Mirror function

There exist two commonly used mirror functions in MART, namely *Translate* and *Reflect*. Fig. 2 illustrates these two mirror functions in the 2-dimensional input domain. Suppose that $(0, 0)$ and (v_1, v_2) are the minimum and maximum coordinate values of the input domain, respectively. The mirror partitioning is 2×1, where the shaded region D_1 is the source domain, and D_2 is the mirror domain. The *Translate* function will map a test case (x, y) in D_1 into $(x + \frac{v_1}{2}, y)$ in D_2, while the *Reflect* function will map (x, y) in D_1 into $(v_1 - x, y)$ in D_2. Previous simulation results have indicated that there is no significant performance difference between the Translate and Reflect mirror functions [21]. In this study, we use the Translate mirror function for MART.

2.2.3. Mirror selection order

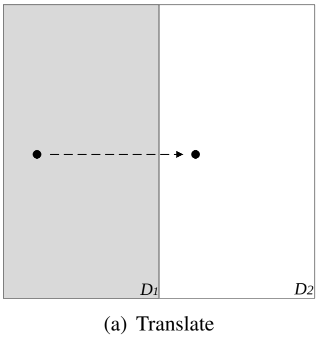
For the mirror selection order, there exist three ways to guide the selection order of mirror domains [27]: (1) sequential order, i.e., mirror domains are chosen according to sequential-ordered sections in each dimension; (2) random order, i.e., mirror domains are selected randomly for generating the next test case; and (3) adaptive-random order, i.e., mirror domains are chosen in a adaptive random manner. As discussed in [27], when the number of mirror domains is small, the F-measure difference among sequential, random, and adaptive-random orders is small as well. With the increase of the number of mirror domains, however, the sequential order generally performs worst; while adaptive-random order has slightly better F-measure performance but more time-consuming than random order. Since a small number of mirror domains will be used in our study, without loss of generality, we use the random order as the mirror selection order for MART in our experiments.

3. Mirror Adaptive Random Testing with Dynamic Partitioning

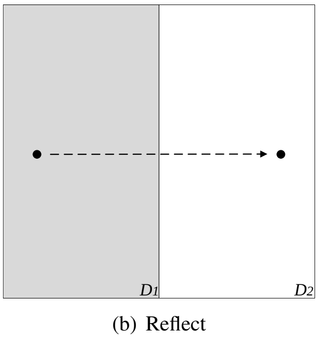
In this section, we first discuss some challenges testers are confronted with when using MART as the testing method. Then, we propose a new testing method, namely *dynamic mirror adaptive random testing* (DMART), which address these challenges.

3.1. Outstanding issues of MART

Previous simulation results have indicated that MART can achieve less computation overhead than original ART algorithms while maintaining the similar failure-detection effectiveness [21, 27]. Previous studies [21] have shown that when generating n test cases, the time complexity of FSCS-ART is in $O(n^2)$; while MART using FSCS-ART algorithm requires the $O(n^2/m^2)$ time, where m is the number of subdomains. However, we can also observe that the computation overhead of MART actually has the same order of magnitude as that of the original ART algorithm. What is worse, as mentioned in Section 2.2, it is better to use a small number of mirror domains in order to avoid "duplicated test case patterns" [21]. Since the number of subdomains (m) should be constrained, the computation overhead of MART cannot be significantly reduced.

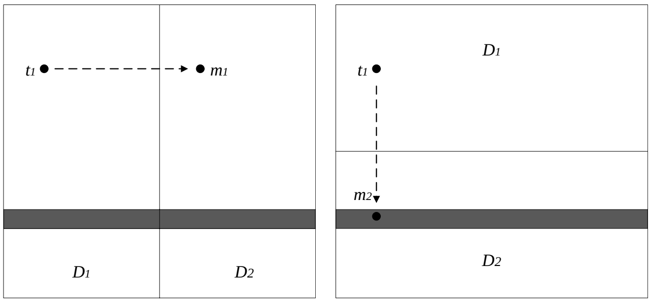


(a) Translate



(b) Reflect

Fig. 2: Two simple mirror functions



(a) 2×1

(b) 1×2

Fig. 3: Another challenge of MART: Failure-unrelated parameter

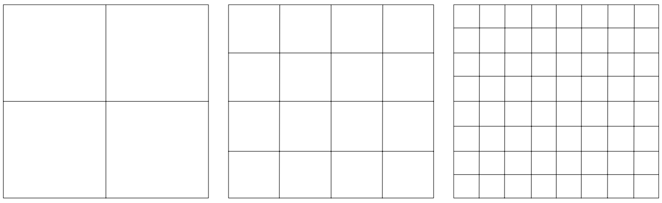
In addition, different mirroring schemes of MART has different failure-detection capabilities, especially when part of input parameters of the program under test is not related to software failures. As illustrated in Fig. 3, in the 2-dimensional input domain $(\{(0, 0), (v_1, v_2)\})$, the failure (failure region is denoted by the grey rectangle) is only sensitive to the vertical coordinate. We denote the vertical coordinate as *failure-related parameter* while the horizontal coordinate as *failure-unrelated parameter*. Consider two ways of mirror partitioning 2×1 and 1×2 for MART. D_1 is designated as the source domain while D_2 as the mirror domain. Suppose that $t_1 = (x, y)$ is a source test case generated by ART in D_1, the mirror test case $m_1 = (x+(v_1/2), y)$ in Fig. 3(a) or $m_2 = (x, y - (v_2/2))$ in Fig. 3(b) is constructed by translating t_1 from D_1 to D_2. Since the vertical coordinate is failure-related, the mirroring process in MART with 2×1 plays no role in revealing this failure; while MART with 1×2 could have higher chance to detect this failure. Unfortunately, prior to testing, it is impossible to know which parameters in the software under test are failure-related or failure-unrelated, so testers may face some difficulties to choose the suitable mirroring scheme for MART.

3.2. New Mirroring Scheme

In this section, we will present a new mirroring scheme to address the outstanding issues of the original MART. *Dynamic mirror partitioning* will be used to reduce the computation overhead; while *all-coordinated mirror function* will be applied to improve the failure-detection effectiveness under the situation of failure-unrelated parameters.

3.2.1. Dynamic Mirror Partitioning

The current ways of mirror partitioning of MART are static, which indicates that the mirror partitioning of MART is fixed before testing and will not be changed during the whole testing



(a) First partitioning (b) Second partitioning (c) Third partitioning

Fig. 4: An example of dynamic mirror partitioning

process [21, 27]. They are simple and easy to be implemented, but the number of test cases in the source domain increases as the number of generated test cases increases, and thus results in high computation overhead. In this study, we design a new way of mirror partitioning, namely *dynamic mirror partitioning* (abbreviated as DMP), to reduce the high computation overhead.

Consider the d-dimensional input domain D. DMP first bisects each dimension of D, i.e., D is partitioned into 2^d subdomains $D_1, D_2, \cdots, D_{2^d}$, which satisfy the following properties: (1) $D = \bigcup_{i=1}^{2^d} D_i$; (2) $|D_i| = \frac{|D|}{2^d}$, where $i, j = 1, 2, \cdots, 2^d$ and $|\cdot|$ denotes the size of a (sub)domain; and (3) $D_i \bigcap D_j = \emptyset$, where $i, j = 1, 2, \cdots, 2^d$ and $i \neq j$. Once the number of test cases in each subdomain reaches the largest value, namely *cut-off* denoted by δ (which will be determined by testers before testing), DMP does the next partitioning by applying the above bisectional division process on each subdomain recursively. We further define a *depth* value μ to represent how many times such bisection process has been conducted. For a given value of μ, there will be a total of $2^{\mu * d}$ subdomains for the d-dimensional input domain.

Fig. 4 depicts an illustrative example of the DMP process with the 2-dimensional input domain. The first partitioning divides the input domain into $2^{1*2} = 4$ (where $\mu = 1$ and $d = 2$) equal-sized subdomains (as shown in Fig. 4(a)), and the second partitioning divides each subdomain previously shown in Fig. 4(a) into four same-sized parts, so as to obtain $2^{2*2} = 16$ ($\mu = 2$) subdomains (Fig. 4(b)). Similarly, as shown in Fig. 4(c), the third partitioning could partition the input domain into $2^{3*2} = 64$ ($\mu = 3$) equal-sized subdomains.

Compared with static mirror partitioning in previous MART [21, 27], the number of subdomains used in DMART is dynamically changed along the testing process. DMP intuitively has at least two advantages:

- In DMP, only up to δ (a constant) test cases can be generated in each subdomain, which can help reduce the computation overhead as low as in the linear order. On the contrary, in previous MART, there is no such constant limitation on the number of test cases in each subdomain, and thus the computation overhead can be very high.

- DMP has only one mirror partitioning scheme; in other words, unlike in previous MART, testers do not need to select the mirror partitioning scheme prior to testing.

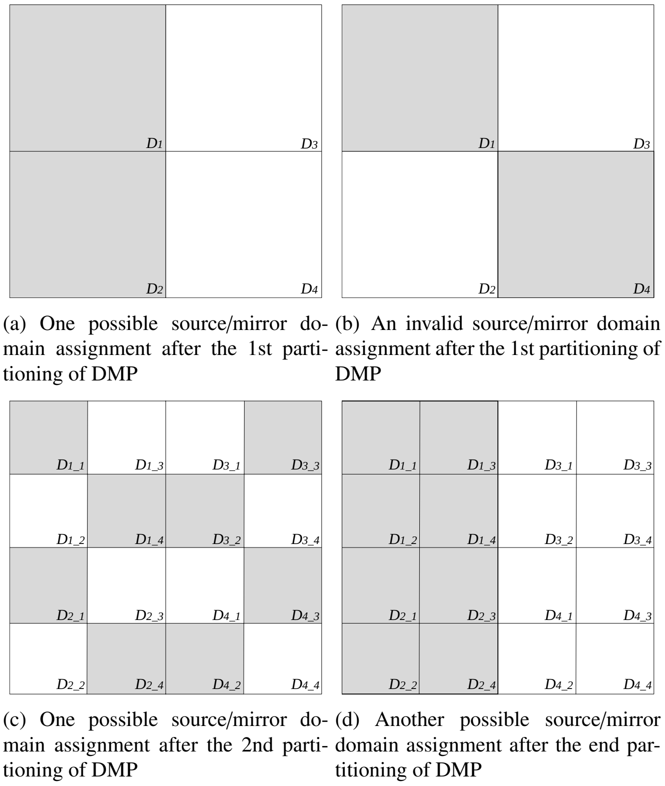
3.2.2. Assignment of source/mirror domains

In previous MART, only one subdomain will be allocated as the source domain, while all other subdomains are the mirror domains. On the contrary, in our DMART approach, we need to assign half of the subdomains as the source domains, while the other half as the mirror domains. More specifically, among all $2^{d*\mu}$ subdomains after each μth partitioning of DMP ($\mu = 1, 2, \cdots$), there will be $2^{d*(\mu-1)}$ pairs of the source/mirror domains. Suppose that the d-dimensional input domain is represented by $D = \{(u_1, u_2, \cdots, u_d), (v_1, v_2, \cdots, v_d)\}$, where $u_l < v_l (l = 1, 2, \cdots, d)$. We allocate the source/mirror domains through the following procedure.

1. Randomly select a subdomain D_i from all the subdomains that are neither source nor mirror domains, where the value range of D_i is from $(p_1, p_2, \cdots, p_d)$ to $(p_1 + r_1, p_2 + r_2, \cdots, p_d + r_d)$, where $r_l = (v_l - u_l)/\mu$ and $u_l \leq p_l \leq v_l - r_l$ $(l = 1, 2, \cdots, d)$.
2. Assign D_i as a source domain.
3. Find a subdomain D_j such that the value range of D_j is from $(q_1, q_2, \cdots, q_d)$ to $(q_1 + r_1, q_2 + r_2, \cdots, q_d + r_d)$, and each pair of p_l and $q_l (l = 1, 2, \cdots, d)$ satisfies the following equation:
$$|p_l - q_l| = \frac{v_l - u_l}{2}. \tag{1}$$
4. Assign D_j as the mirror domain corresponding to D_i.
5. Repeat Steps 1-4 until every subdomain is either source or mirror domain.

As an illustrative example, consider a two-dimensional input domain. Fig. 5 shows some examples of assigning the source/mirror domain. In Fig. 5(a), the DMP first divides the input domain into four equal-sized subdomains, D_1, D_2, D_3, and D_4, and then two mapping pairs of the source/mirror domain are obtained, i.e., (D_1, D_4) and (D_2, D_3). Fig. 5(a) shows that D_1 and D_2 can be assigned as the source domains (marked by the shaded regions). It should be noted that D_1 and D_3 can also be the source domains. However, D_1 and D_4 cannot be designated as the source domains simultaneously, as shown in Fig. 5(b), because they satisfy Eq. (1) and thus should be a pair of source/mirror domains. Test cases can be generated in each source domain (D_1 and D_2 in Fig. 5(a)), and then mapped into each mirror domain (D_4 and D_3, respectively), as to be discussed in the next section. When δ test cases are generated from each subdomain, each D_i will be further bisectionally divided into four equal-sized parts (i.e., the second partitioning of DMP), D_{i_1}, D_{i_2}, D_{i_3}, and D_{i_4}. According to above allocation procedure, when D_{l_i} is selected as a source domain, $D_{(5-l)\_i}$ will be its corresponding mirror domain, or vice versa, where $i, l = 1, 2, 3, 4$. Figs. 5(c) and 5(d) give two possible ways of assigning source and mirror domains, where the shaded regions denote source domains.

As discussed before, DMART and the previous MART [21, 27] differ in how source/mirror domains are allocated: DMART assigns half of subdomains as the source domains and the other half as the mirror domains, while MART only allocates one subdomain as the source domain and all the remaining subdomains as the mirror domains. Since an original ART algorithm



(a) One possible source/mirror domain assignment after the 1st partitioning of DMP

(b) An invalid source/mirror domain assignment after the 1st partitioning of DMP

(c) One possible source/mirror domain assignment after the 2nd partitioning of DMP

(d) Another possible source/mirror domain assignment after the end partitioning of DMP

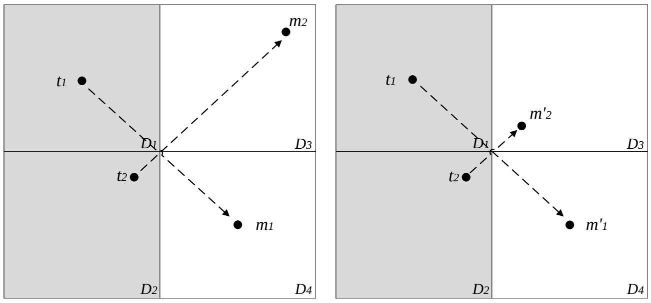
Fig. 5: Illustration of assigning the source/mirror domain

is applied to generate test cases independently in each source domain, the test case patterns will be different for all the source domains in DMART, and each test case pattern will be duplicated at most once. In other words, the source/mirror domains allocation in DMART will also alleviate the problem of "duplicated test case patterns" in the previous MART studies [21].

3.2.3. *All-coordinated Mirror Function*

As discussed in Section 3.1, the mirror function in MART may generate some mirror test cases that have the same value as the source test case at one (or some) coordinate(s). In this section, we propose a new mirror function, namely *all-coordinated mirror function* (AMF), which generates mirror test cases that have different values from the source test cases at all coordinates.

Similar to mirror functions in MART [21, 27], AMF also has two types: (1) *All-coordinated Translate*; and (2) *All-coordinated Reflect*. They have similar meanings to Translate and Reflect in MART, respectively. However, our proposed mirror functions translate (or reflect) source test cases into mirror test cases by linearly displacing each coordinate.



(a) All-coordinated Translate

(b) All-coordinated Reflect

Fig. 6: An example of two all-coordinated mirror functions

Suppose that in the d-dimensional input domain $D = \{(u_1, u_2, \cdots, u_d), (v_1, v_2, \cdots, v_d)\}$, the source domain is represented by $D_i = \{(p_1, p_2, \cdots, p_d), (p_1 + r_1, p_2 + r_2, \cdots, p_d + r_d)\}$ and the corresponding mirror domain by $D_j = \{(q_1, q_2, \cdots, q_d), (q_1 + r_1, q_2 + r_2, \cdots, q_d + r_d)\}$. Let a source test case t_x from D_i be $(x_1, x_2, \cdots, x_d)$, and its corresponding mirror test case m_y in D_j be $(y_1, y_2, \cdots, y_d)$. When using the All-coordinated Translate, t_x and m_y should satisfy the following equation on each coordinate:

$$|x_l - y_l| = \frac{v_l - u_l}{2}. \tag{2}$$

When using the All-coordinated Reflect, t_x and m_y should satisfy the following equation on each coordinate:

$$x_l + y_l = v_l + u_l. \tag{3}$$

Fig. 6 gives an example of these two types of AMF in the 2-dimensional input domain ($\{(0, 0), (v_1, v_2)\}$). In Fig. 6(a), the All-coordinated Translate directly translates source test case $t_1 = (x_1, y_1)$ in D_1 into D_4 to construct mirror test case $m_1 = (x_1 + (v_1/2), y_1 - (v_2/2))$, and translates $t_2 = (x_2, y_2)$ in D_2 into D_3 for generating $m_2 = (x_2 + (v_1/2), y_2 + (v_2/2))$. In Fig. 6(b), the All-coordinated Reflect directly reflects t_1 to $m'_1 = (v_1 - x_1, v_2 - y_1)$ while t_2 to $m'_2 = (v_1 - x_2, v_2 - y_2)$. Intuitively speaking, the All-coordinated Reflect may generate mirror test cases that are close to the corresponding source test cases (for example, as shown in Fig. 6(b), m_2 is close to t_2), while All-coordinated Translate does not have such a problem. In this paper, therefore, we use the All-coordinated Translate as the mirror function.

As discussed in Section 3.1, in original MART, no matter how many mirrors or mirror domains it has, the mirroring function cannot guarantee the diversity on all coordinates; instead, some test cases may have exactly the same values on certain coordinates. On the contrary, since the mirror test case is constructed by mapping all coordinates of the source test case, DMART diversifies test cases on each coordinate, which would improve the failure-detection effectiveness under the situation of failure-unrelated parameters.

3.2.4. *Selection of source domains*

Since there exist only one source domain and many mirror domains in MART [21, 27], testers require to choose the selection order of mirror domains to generate mirror test cases. However, in DMART, the number of source domains is equal to that of mirror domains, and each source domain is corresponding to one and only one mirror domain. Therefore, in the new DMART approach we need to consider the selection order of source domains instead of mirror domains.

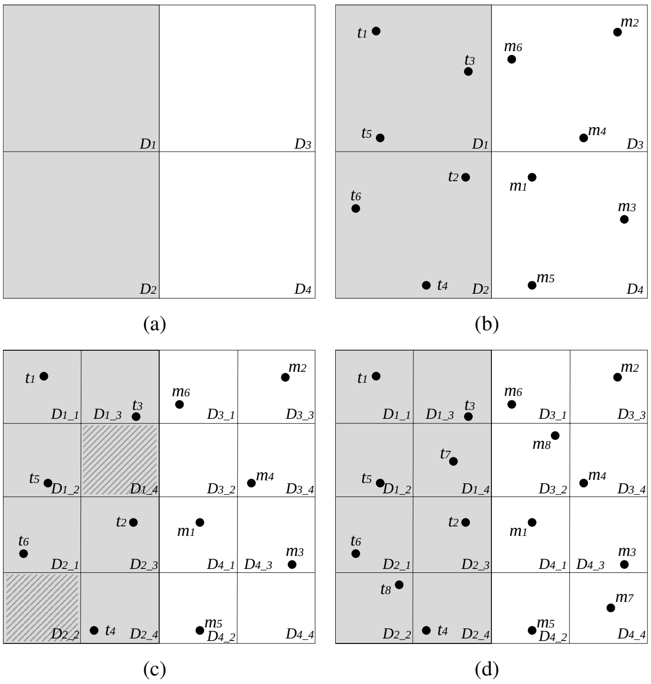
Based on the "even spread" notion, we always select the source domain that contains the smallest number of test cases among all source domains. When more than one source domains have the fewest test cases, we randomly select one of them as the domain where the next test case will be generated from.

3.3. Algorithm

In this section, we present the detailed algorithm of DMART, which adopts the proposed new mirroring scheme given in the previous Section 3.2.

The DMART algorithm first defines a specified integer denoted as *cutoff*, δ, and then uses the DMP process over the whole input domain to obtain $s = 2^{d*\mu}$ subdomains, where μ is the depth (or partitioning rounds) of the DMP. The next step is to assign previously executed test cases (that form the executed set E) into their relevant subdomains, which obtains $E_1, E_2, \cdots, E_s$ that denote the sets of executed test cases in $D_1, D_2, \cdots, D_s$ respectively. Suppose that D_i $(1 \leq i \leq s/2)$ is a new source domain and D_j $(s/2 < j \leq s)$ is a new mirror domain to D_i after each round of DMP. We can observe that already executed test cases populated in D_j (i.e., E_j) are also mirror test cases of those populated in D_i (i.e., E_i). This observation would guarantee the even spreading of test cases over each subdomain. With regard to test case generation, DMART repeatedly selects the least populated source domain D_i $(1 \leq i \leq s/2)$ such that $|E_i| = \min_{h \in \{1,2,\cdots,s/2\}}\{|E_h|\}$, and then takes the following steps: (1) generate a source test case *tc* in the source domain D_i $(1 \leq i \leq s/2)$, and (2) use the AMF to construct the mirror test case *mc* based on *tc* in the corresponding mirror domain D_j $(s/2 < j \leq s)$. When the number of total test cases reaches $s * \delta$, DMART will take another round of DMP. Such a procedure will be repeated until certain stopping criteria (such as "a failure has been detected", "a certain number of test cases have been executed", "testing resources have been exhausted", etc.) are reached. Algorithm 1 shows the detailed information of DMART.

Fig. 7 gives an example to illustrate the operation process of DMART in the 2-dimensional input domain with $\delta = 3$. For ease of description, we assume that point t_i is the i-th source test case while point m_i is the i-th mirror test case constructed based on t_i. In Fig. 7(a), DMART uses the DMP to divide the input domain into four subdomains, and then designates D_1 and D_2 as source domains while D_3 and D_4 as mirror domains. In Fig. 7(b), DMART randomly chooses a source domain to generate the first test case t_1 in D_1 using original ART algorithms, and then mirrors t_1 into D_4 to generate the next test case m_1. Similarly, DMART selects D_2 to generate source test case t_2 and mirror test case m_2 in D_3. When the number of total test cases reaches $2^2 * \delta = 12$, as shown in Fig. 7(c), DMP is re-triggered. Since the number of subdomains increases (i.e., $2^{2*\mu} = 16$ where $\mu = 2$ is the depth of DMP), DMART requires to assign previously executed test cases in new subdomains, and then also assigns new source and mirror domains. After that, the distribution of test cases in source domains is $|T_{1\_1}| = 1$, $|T_{1\_2}| = 1$, $|T_{1\_3}| = 1$, $|T_{1\_4}| = 0$, $|T_{2\_1}| = 1$, $|T_{2\_2}| = 0$, $|T_{2\_3}| = 1$, and $|T_{2\_4}| = 1$ respectively. According to the algorithm, D_{1_4} or D_{2_2} (the region with left diagonals) should be chosen, of which the next test case will be generated. In Fig. 7(d), when each subdomain has the same number of test cases, DMART will randomly selects a source domain to generate the following test cases.



(a) (b) (c) (d)

Fig. 7: Illustration of the DMART algorithm

3.4. Complexity Analysis

In this section, we briefly investigate the time and space complexity of DMART by a formal mathematical analysis.

3.4.1. Time Complexity

The time complexity of DMART algorithm mainly depends on the process of generating new test cases.

If the next test case is generated from a source domain, all executed test cases contained in the source domain will be used in the generation process. In other words, the test case generation time in the source domain is in $O(|E_i|)$, where E_i refers to the set of executed test cases in the subdomain D_i. Since $|E_i|$ has a maximum value of δ (which is a constant), it only requires a constant time to generate the next test case from a source domain. Moreover, if the next test case is generated from a mirror domain, the process is implemented by simply executing the AMF function, the time complexity of which is constant. In summary, the generation of the next test case always requires the constant time.

Considering the constant time complexity in each step, it can be concluded that DMART requires $O(n)$ time to generate n test cases, where $n = |E|$, the size of execute set E.

Note that DMART has an additional process of partitioning subdomains and allocating the executed test cases into subdomains. In DMART, when the number of executed test cases in each subdomain reaches the cutoff value δ, every subdomain

Algorithm 1 Dynamic mirror adaptive random testing (DMART)

```
Initialize the cutoff $\delta$
Set $E = \{\}$ /* the executed set to store executed test cases in the $d$-dimensional input domain $D$ */
Set $\mu = 1$ /* the depth (or partitioning times) of the DMP */
Set $d\_queue = \{\}$; /* a list to store subdomains */
Set $e\_queue = \{\}$; /* a list to store executed subsets distributed in subdomains */
while (Stopping criteria not reached)
    Bisectionally divide $D$ into $s = 2^{d*\mu}$ subdomains;
    Assign source and mirror domains;
    Annotate source domains as $D_1, D_2, \cdots, D_{s/2}$ and mirror domains as $D_{s/2+1}, D_{s/2+2}, \cdots, D_s$;
    Set $d\_queue = \{D_1, D_2, \cdots, D_s\}$;
    Assign each test case in $E$ into $E_1, E_2, \cdots, E_s$ according to $D_1, D_2, \cdots, D_s$, and then set $e\_queue = \{E_1, E_2, \cdots, E_s\}$;
    while ($|E| < s * \delta$) or (Stopping criteria not reached)
        Select the least populated source domain $D_i$ from $d\_queue$; /* i.e., $D_i$ such that $|E_i| = \min_{h\in\{1,2,\cdots,s/2\}} |E_h|$ */
        Set $tc = \text{ART}(D_i, E_i)$; /* i.e., apply an ART algorithm in $D_i$ based on $E_i$ to generate a test case $tc$ */
        Add $tc$ into $E_i$ and $E$, and execute $tc$.
        Construct the mirror test case $mc$ from $tc$ using AMF in the corresponding mirror domain $D_j$ ($s/2 < j \leq s$);
        Add $mc$ into $E$, and execute $mc$.
    end_while
    Set $d\_queue = \{\}$;
    Set $e\_queue = \{\}$;
    Increment $\mu$ by 1;
end_while
```

will be partitioned into 2^d new subdomains by bisecting each dimension, where d is the dimension of the input domain. Apparently, such partitioning requires the time in $O(2^d)$. In addition, all δ executed test case in every previous subdomain will be re-allocated into the corresponding new 2^d subdomains, which obviously requires the time in $O(\delta \times 2^d)$. Since there are $\frac{n}{\delta}$ subdomains ($n = |E|$), the total time complexity for the whole process is $O(n)$ (considering that both δ and d are constants). Nevertheless, such a process is not implemented every time a new test case is generated; instead, it is only implemented when $n = \delta \times 2^{d*\mu}$. Therefore, the time spent for this process is negligible compared to that for generating test cases, and thus will not affect the overall time complexity of DMART. As demonstrated later in Section 5.3, DMART only requires linear time (that is, $O(n)$) for generating test cases.

3.4.2. Space Complexity

During the procedure of DMART, memory space is required to store all n executed test cases and all $2^{d*\mu}$ subdomains. According to the DMART algorithm, we can have $\delta \times 2^{d*(\mu-1)} \leq n \leq \delta \times 2^{d*\mu}$; in other words, $2^{d*\mu}$ is at most in $O(n)$. Therefore, the space complexity for DMART is $O(n)$.

4. Experimental Studies

We conducted a series of simulations and empirical studies to evaluate the performance of DMART. The design and settings of the experiments are described in this section.

4.1. Research Questions

We propose DMART to further reduce the computation overhead of MART as well as ART algorithms. It is therefore required to examine the test case generation time of the new MART method. In addition, while decreasing the computation overhead, we hope that DMART also has comparable failure-detection effectiveness to MART and the original ART algorithms in order to deliver a high cost-effectiveness in testing. Thus, it is also important to evaluate the failure-detection effectiveness of DMART under different scenarios. Our experimental studies help us answer the following two research questions.

RQ1 How effective is DMART at revealing failures?

RQ2 To what extent DMART can reduce the computation overhead as compared to MART and ART algorithms?

4.2. Variables and Measures

4.2.1. Independent variables

The independent variable in the experimental studies is the test case selection method. DMART, the new approach proposed in this paper, is definitely chosen for this variable. In addition, we selected two methods, FSCS-ART and MART. as the baselines for comparison. FSCS-ART was selected as the representation of ART following previous studies on MART [21, 27]. Accordingly, we also adopted the MART and DMART algorithms based on the original FSCS-ART. In addition, we decided to use the "translate" mirroring function and the random mirror selection order, following previous investigations of MART [21, 27].

4.2.2. Dependent variables

The dependent variable for RQ1 is actually the metric to evaluate and compare the failure-detection effectiveness of testing techniques. We follow previous studies on ART [10, 15, 16,

Table 1: Program name, dimension (d), input domain, seeded fault type, total number of faults, and failure rate of each fault-seeded program

Program	d	Input Domain		Seeded Fault Type				Total	Failure
Name		From	To	AOR	ROR	SVR	CR	Faults	Rate
`airy`	1	(-5000.0)	(5000.0)	0	0	0	1	1	0.000716
`bessj0`	1	(-300000.0)	(300000.00)	2	1	1	1	5	0.001373
`erfcc`	1	(-30000.0)	(30000.00)	1	1	1	1	4	0.000574
`probks`	1	(-50000.0)	(50000.00)	1	1	1	1	4	0.000387
`tanh`	1	(-500.0)	(500.00)	1	1	1	1	4	0.001817
`bessj`	2	(2.0, -1000.0)	(300.0, 15000.0)	2	1	0	1	4	0.001298
`gammq`	2	(0.0, 0.0)	(1700.0, 40.0)	0	3	1	1	5	0.000830
`sncndn`	2	(-5000.0, -5000.0)	(5000.0, 5000.0)	0	0	4	1	5	0.001623
`golden`	3	(-100.0, -100.0, -100.0)	(60.0, 60.0, 60.0)	0	3	1	1	5	0.000550
`plgndr`	3	(10.0, 0.0, 0.0)	(500.0, 11.0, 1.0)	1	2	0	2	5	0.000368
`cel`	4	(0.001, 0.001, 0.001, 0.001)	(1.0, 300.0, 10000.0, 1000.0)	1	1	0	1	3	0.000332
`el2`	4	(0.0, 0.0, 0.0, 0.0)	(250.0, 250.0, 250.0, 250.0)	1	3	2	3	9	0.000690

AOR: arithmetic operator replacement, ROR: relational operator replacement, SVR: scalar variable replacement, and CR: contant replacement.

25, 26, 28] to use the *F-measure*, which refers to the expected number of test cases required to detect the first software failure. In our study, F_{RT} denotes the F-measure of RT. According to uniform distribution, theoretically, $F_{RT} = 1/\theta$ when test cases are selected with replacement. F_{ART} denotes the F-measure of ART. *ART F-ratio* denotes the ratio of F_{ART} to F_{RT}, which measures the F-measure improvement of ART over RT.

The dependent variable for RQ2 is mainly related to the execution time of the testing techniques under our study. As reported in the next section, in our simulation, we measured the time required for generating a certain number of test cases; while in our empirical studies, we measured the average time to detect the first failure for each real-life program.

4.3. Simulations and Object Programs

To address RQ1, we attempted to examine and compare the values of F_{ART} for FSCS-ART, MART, and DMART. As discussed in [28], F_{ART} has been influenced by many factors, such as dimension, the number and the compactness of failure regions, as well as the existence and the size of a predominant failure region, so theoretical analysis of F_{ART} is of great difficulty. In previous studies [10, 15, 16, 25, 26, 28], therefore, researchers commonly used simulations or empirical studies to investigate F_{ART}. In our study, we conducted both simulations and empirical studies to evaluate the failure-detection effectiveness.

To address RQ2, we conducted simulations to evaluate the test case generation time of DMART as compared to the original ART and MART algorithms. In the simulations, we used each of FSCS-ART, MART, and DMART to generate totally 20,000 test cases in two- and three-dimensional input domains. The execution time taken to generate n test cases ($n = 500, 1000, 1500, \ldots, 20000$) was recorded and thus compared. In addition, in the empirical studies, we compared DMART and ART/MART based on the average time taken to detect the first failure.

In simulations, we attempted to mimic faulty programs under different situations. For a faulty program, there are two basic features. One feature is *failure rate*, normally denoted by θ, which is defined as the ratio of the number of failure-causing inputs to the number of all possible program inputs. Another feature is *failure pattern*, which refers to the shapes of failure regions together with their distributions over the input domain. Both θ and failure pattern are fixed after coding but unknown before testing. In our simulations, θ and failure pattern were defined in advance, and the failure regions were randomly placed in the input domain. When a point generated by an ART algorithm is inside a failure region, a failure is said to be detected. One major advantage of simulations is that they can provide a full picture of F_{ART} under various scenarios. In our study, we simulated the following four different failure patterns (FPs).

FP-I The first failure pattern to be simulated is that the failure-causing inputs are well clustered into a single square/hypercubic region. We mainly focused on the situation where a single square failure region was randomly placed inside a two-dimensional square input domain, and θ was set as 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, and 0.0005. In addition, in order to check the consistency of our data, we also considered one-, three-, and ten-dimensional input domain while θ was set as 0.005 and 0.001.

FP-II The second failure pattern was set as a single rectangular failure region randomly placed inside a two-dimensional input domain. The ratio among edge lengths of the rectangular region is $1 : \alpha$, where α is set as 1, 4, 7, 10, 20, 30, 40, 50, 60, 70, 80, 90, and 100. Intuitively speaking, with the increase of α, the rectangular region becomes less compact. In addition, θ was set as 0.005, and 0.001.

This setting is to investigate the impact of the compactness of failure region on the failure-detection effectiveness of DMART.

FP-III The third failure pattern was set as a number of square regions randomly placed inside a two-dimensional input domain. Suppose there are β failure regions, denoted by $R_1, R_2, \cdots, R_\beta$. For each failure region R_i, we set $|R_i| = (\rho_i / \sum_{j=1}^{\beta} \rho_j) \cdot \theta \cdot |D|$, where ρ_i is randomly chosen from $[0, 1)$ according to uniform distribution, $i = 1, 2, \cdots, \beta$, and $|D|$ denotes the size of the input domain. The number of failure regions β is set as 1, 4, 7, 10, 20, 30, 40, 50, 60, 70, 80, 90, and 100, respectively. In addition, θ was set as 0.005 and 0.001. This setting is to investigate the impact of the number of failure regions on the failure-detection effectiveness of DMART.

FP-IV In the fourth failure pattern, there exists a predominant region among a number of square failure regions randomly placed inside a two-dimensional input domain. We assume there are β failure regions, denoted by $R_1, R_2, \cdots, R_\beta$, respectively. For one failure region R_1, we set $|R_1| = w \cdot \theta \cdot |D|$, where $w = 0.3, 0.5$, and 0.8. For all the other failure regions, we set $|R_i| = (\rho_i / \sum_{j=2}^{\beta} \rho_j) \cdot (1 - w) \cdot \theta \cdot |D|$, where ρ_i is randomly selected from $[0, 1)$ according to uniform distribution, $i = 2, 3, \cdots, \beta$, and $|D|$ denotes the size of the input domain. The number of failure regions β is set as 1, 4, 7, 10, 20, 30, 40, 50, 60, 70, 80, 90, and 100, respectively. Similar to FP-III, θ was set as 0.005 and 0.001. This experiment is to investigate the impact of the existence, and the size of a predominant failure region on the failure-detection effectiveness of DMART.

Each of these FPs represents a possible failure pattern in practice, and they are used to evaluate how different factors affect the performance of ART algorithms. According to previous investigations [28], ART normally performs the best under FP-I. Its failure-detection effectiveness becomes lower when the failure region is less compact (FP-II), there is more distinct failure regions (FP-III), or the size of the predominant region is smaller (FP-IV).

In empirical studies, we used real-life programs as our object programs. Previous ART studies [10, 16, 28] have adopted 12 fault-seeded programs to evaluate the effectiveness of ART algorithms in practical situations. These published programs were taken from ACM's collected algorithms [29] and the Numerical Recipes [30], and were converted into C++ languages [10]. Some faults were seeded into each programs using mutation technique [31]. For ease of comparison with previous experimental results, we selected all these 12 programs as the objects of our empirical studies. Table 1 summarizes the detailed information of these 12 real-life programs. After analyzing above 12 object programs, we have found that there exist three programs which involve failure-unrelated parameters: (1) in the program `plgndr`, the third parameter is not related to the failure; (2) in the program `cel`, the first, third, and forth parameters are failure-unrelated; and (3) in the program `el2`, the third and forth parameters are not related to the failure.

4.4. Experiment Environment

The simulations were conducted on a machine serving Windows XP, equipped with an Intel(R) Core(TM) i3-3240 CPU (3.40 GHz, 4 core) with 4GB of RAM. The used programming language was Microsoft Visual Studio 2005 (Visual C++).

4.5. Data Collection

In the experiments for evaluating F-measures, test cases were generated (using a testing strategy) and executed until a failure was detected. For simulations, a failure is said to be detected when a point is picked up from the simulated failure region; while for empirical studies, a failure refers to the different behaviors between the object program and the faulty version. In each experiment run, the number of test cases required to detect a failure is referred to as the F-count. Such a process was repeated for a sufficient number (S) of times to guarantee that the mean value of F-counts can be used as an approximate of F_{ART} within a certain confidence level (95% used in this paper) and a certain accuracy range (±5% used in this paper). As for the detailed calculation of S, readers can refer to [21].

5. Experimental Results

5.1. Answer to RQ1–Part 1: Simulations

5.1.1. FP-I

The simulation results for FP-I are reported in Fig. 8 and Table 2. We use DMART-50 and DMART-100 to represent DMART with $\delta = 50$ and $\delta = 100$, respectively. In Fig. 8, x-axis represents θ in the logarithmic scale, while y-axis stands for the ART F-ratio. For convenience of comparison, the simulation results of FSCS-ART and MART have also been given. Similar to previous simulation design, the failure pattern is a single square/hypercubic region, which means that each permutation of the mirroring partitioning $u_1 \times u_2 \times \cdots \times u_d$ has no impact on the failure-detection capability of MART. Therefore, we use MART-$u_1 \times u_2 \times \cdots \times u_d$ ($1 \leq u_1 \leq u_2 \leq \cdots \leq u_d \leq 2$) to represent MART in the simulations for FP-I. In Table 2, for a clearer presentation, we use MART-$1^i \times 2^j$ ($i + j = 10$) to denote the MART with $d = 10$, which implies that j coordinates are bisected, while the remaining i coordinates not. For example, MART-$1^5 \times 2^5$ means MART-1×1×1×1×1×2×2×2×2×2. In the simulations for MART with $d = 10$, we did not use MART with a very small number of subdomains (such as MART-$1^9 \times 2^1$), because the computation overhead becomes much higher with the increase of dimension and using a small number subdomains cannot reduce the overhead very well (which jeopardizes the very benefit of MART). We did not use MART with a very large number of subdomains (such as MART-2^{10}), because it is not advisable to have too many subdomains in MART (to prevent too many "duplicated test case patterns", as discussed in Section 3.1).

Based on experimental data, we can observe the followings.

Table 2: Failure-detection effectiveness (ART F-ratio) of DMART on single square failure region with $d = 1, 3, 10$ and $\theta = 0.005, 0.001$

	θ	FSCS-ART	MART-2	DMART-50	DMART-100		
$d = 1$	0.005	0.5570	0.5684	0.5590	0.5634		
	0.001	0.5739	0.5672	0.5620	0.5569		
$d = 3$	θ	FSCS-ART	MART- $1 \times 1 \times 2$	MART- $1 \times 2 \times 2$	MART- $2 \times 2 \times 2$	DMART-50	DMART-100
	0.005	0.7980	0.8171	0.8332	0.8275	0.8350	0.8209
	0.001	0.7731	0.9441	0.9757	0.9566	0.8136	0.7670
$d = 10$	θ	FSCS-ART	MART- $1^7 \times 2^3$	MART- $1^5 \times 2^5$	MART- $1^3 \times 2^7$	DMART-50	DMART-100
	0.005	3.6177	3.9719	4.1365	4.0294	1.0185	0.9733
	0.001	2.7480	2.1017	1.7108	1.3051	1.0218	0.9945

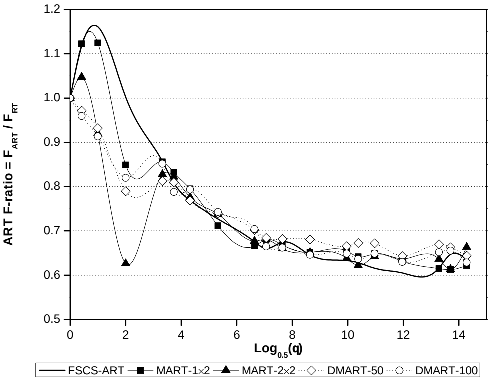


Fig. 8: Failure-detection effectiveness of DMART on single square failure region with $d = 2$

- For small dimensionality (such as $d = 1, 2, 3$), DMART, MART, and FSCS-ART have similar failure-detection capabilities, and generally outperform RT. However, for high dimensionality (for example $d = 10$), DMART normally performs much better than FSCS-ART and MART.
- DMART with high δ seems to have better failure-detection effectiveness than that with low δ, but the difference in effectiveness is not so obvious.
- The mirror scheme has no influence on the effectiveness of MART, which means that MART with different mirror schemes performs similarly on FP-I.

As observed before, DMART-100 seems to have slightly better performance than that with DMART-50, though the difference in performance is not so significant. In the following sections, therefore, we only adopt DMART-100 to represent the DMART algorithm.

5.1.2. FP-II

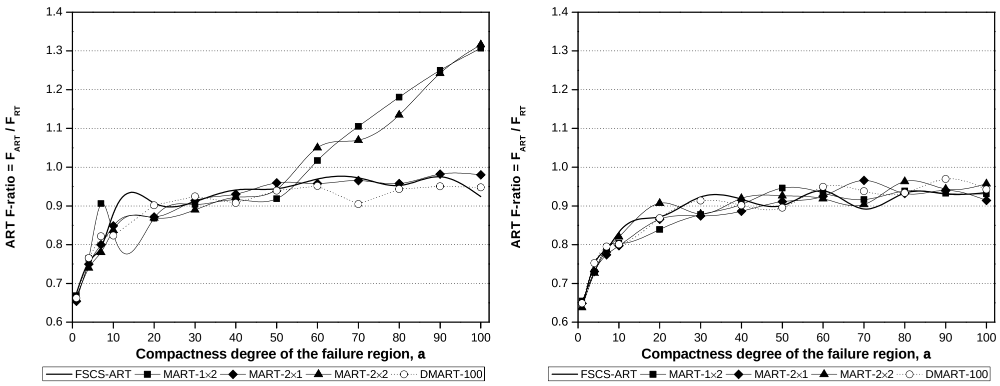
Fig. 9 shows the simulation results for FP-II. In this experiment, we distinguish MART-1×2 from MART-2×1. The reason is that as discussed in Section 3.1, different mirror schemes have strong impacts on the failure-detection effectiveness of MART when some input parameters are more sensitive to failures than other parameters.

As shown in Fig. 9, the following observations can be obtained from the simulation results.

- Similar to FSCS-ART and MART, the failure-detection effectiveness of DMART depends on the compactness of the failure region, i.e., DMART has poorer performance when the failure region is less compact.
- DMART has similar failure-detection capabilities to FSCS-ART, irrespective of α or θ.
- For $\theta = 0.001$, DMART has similar performance to MART with all mirror schemes, regardless of α. However, for $\theta = 0.005$,
  - When $1 \leq \alpha \leq 50$, DMART has similar failure-detection capability to all MART versions.
  - When $50 < \alpha \leq 100$, DMART performs similarly to MART-2 × 1. However, DMART has better performance than MART-1 × 2 and MART-2 × 2, and the improvements become larger with increase of α. In other words, MART has different performances on different mirror schemes.

We briefly explain why MART has different failure-detection capabilities when choosing different mirror schemes when $\theta = 0.005$ and $50 < \alpha \leq 100$. According to experimental setup, the edge length ratio of the rectangular region is $1 : \alpha$, which means that the x-axis edge length of the rectangular region is $\sqrt{\theta/\alpha}$ while the y-axis edge length $\sqrt{\theta\alpha}$, assuming that the input domain is a unit square. With the increase of α, the y-axis edge length becomes large while the x-axis edge length becomes small. For example, when $\alpha > 50$, the y-axis edge length is larger than 0.5 while the x-axis edge length is smaller than 0.01, which indicates that y-axis increasingly becomes less sensitive to software failures while the x-axis becomes more sensitive to failures.

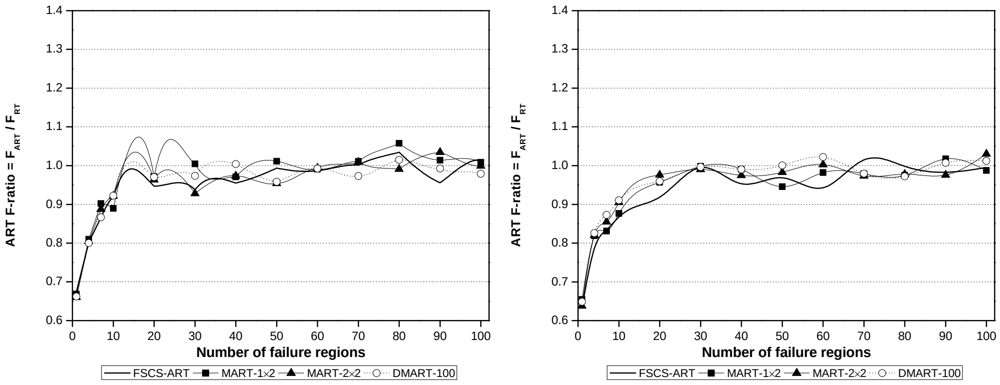
In Section 3.1, we have shown by Fig. 3 that t_1 and m_1 have the same value at the vertical coordinate; while t_1 and m_2 have the same value at the horizontal coordinate. In other words, the mirroring process (actually due to mirror function) in MART with whether 2 × 1 or 1 × 2 is conducted for only one coordinate, not for all dimensions, which implies that MART does not achieve the even spread of test cases on each coordinate. Both mirror functions, Translate and Reflect, have such drawback. By using all-coordinated mirror function, DMART solves this problem.



(a) $\theta = 0.005$

(b) $\theta = 0.001$

Fig. 9: Failure-detection effectiveness of DMART on single rectangular failure region



(a) $\theta = 0.005$

(b) $\theta = 0.001$

Fig. 10: Failure-detection effectiveness of DMART on multiple square failure regions

5.1.3. FP-III

Similar to Section 5.1.1, we use MART-1 × 2 only as one version of MART, because MART-1 × 2 and MART-2 × 1 are equivalent to each other for FP-III, where failure regions are squares. The simulation results for FP-III are given in Fig. 10, from which we can observe that the F-measures of DMART becomes larger with the increase of the number failure regions, regardless of θ. It can also be observed that DMART has similar failure-detection capabilities to FSCS-ART and MART, irrespective of θ and the number of failure regions.

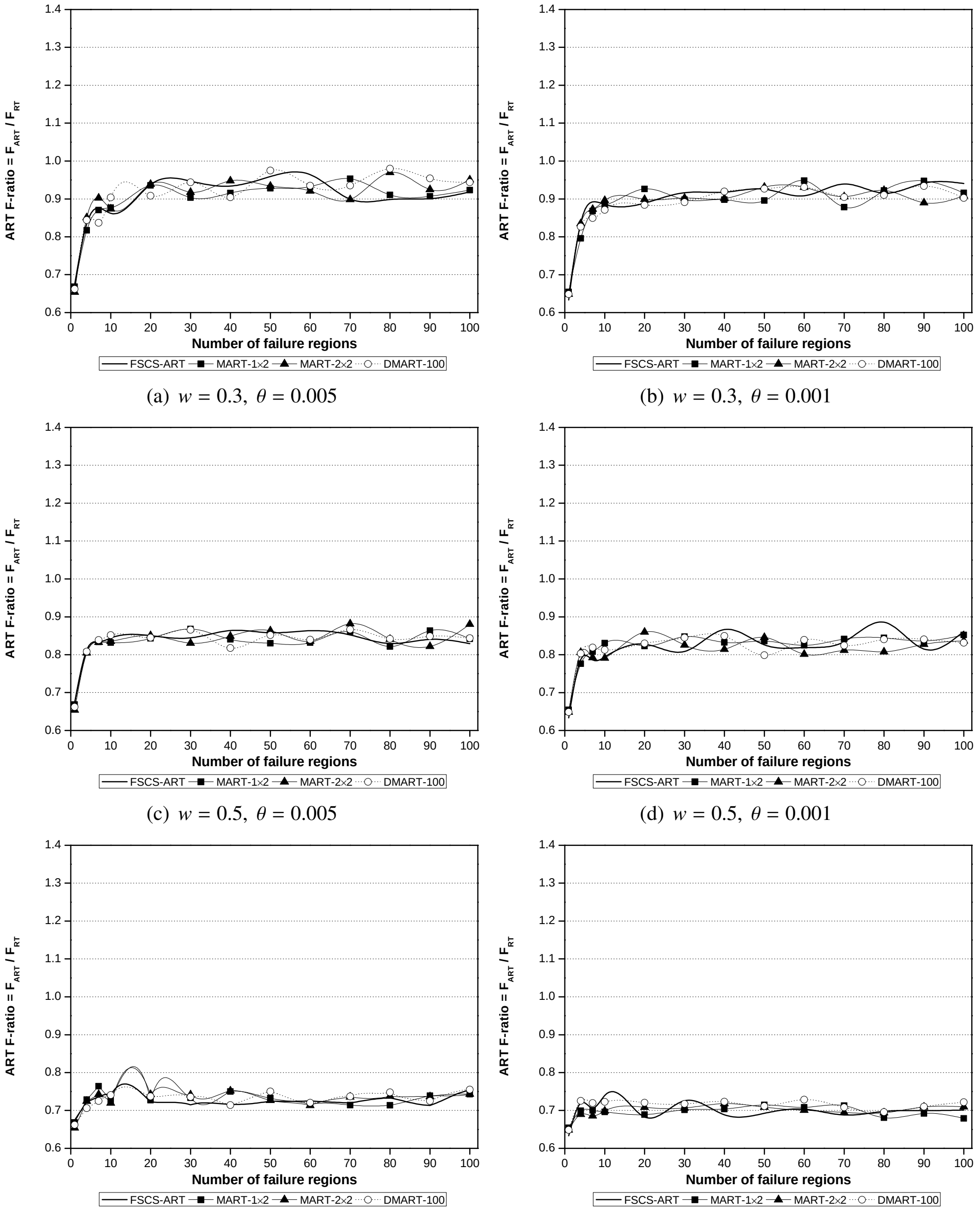
5.1.4. FP-IV

Similar to Sections 5.1.1 and 5.1.3, we use MART-1×2 and MART-2×2 as the representative versions of MART. Fig. 11 reports the simulation results of DMART against FSCS-ART and MART on FP-IV, which show that the F-measures of DMART depend on the number of failure regions and the size of the predominant failure region. DMART could have better failure-detection capabilities, especially when the number of failure regions is smaller and the size of the predominant failure regions is larger, regardless of θ. Additionally, it can be also observed from the figures that the F-measures of DMART are very similar to those of FSCS-ART and MART.

Briefly speaking, similar to other ART algorithms, the failure-detection effectiveness of DMART also depends on many factors such as the dimension of input domain, the compactness of failure region, the number of failure regions, and the size of the predominant failure region. Compared with FSCS-ART and MART, DMART achieves at least the similar failure-detection effectiveness in most cases.

5.2. Answer to RQ1–Part 2: Empirical Studies

Fig. 12 shows the F-measure results of empirical studies, and gives the 95% confidence interval of the F-measure for each method, as represented by two error bars. For ease of comparison, we also include the previous results of FSCS-ART and MART. As discussed in Section 3.1, MART using different mirroring schemes has different failure-detection capabilities especially when testing programs with failure-unrelated parameters. Since the failure pattern in each fault-seeded program is unknown for testers before testing (more specifically, in reality, testers do not have the prior knowledge of which parameters in the program are failure-related or failure-unrelated), in this empirical study it is necessary to consider all possible mirror schemes for MART. For example, for object programs with $d = 3$, we adopt the following 7 mirror schemes for MART: $1\times1\times2$, $1\times2\times1$, $2\times1\times1$, $1\times2\times2$, $2\times1\times2$, $2\times2\times1$, and $2\times2\times2$. In other words, for d-dimensional programs, there are



(a) $w = 0.3,\ \theta = 0.005$

(b) $w = 0.3,\ \theta = 0.001$

(c) $w = 0.5,\ \theta = 0.005$

(d) $w = 0.5,\ \theta = 0.001$

(e) $w = 0.8,\ \theta = 0.005$

(f) $w = 0.8,\ \theta = 0.001$

Fig. 11: Failure-detection effectiveness of DMART on multiple square failure regions with one predominant region

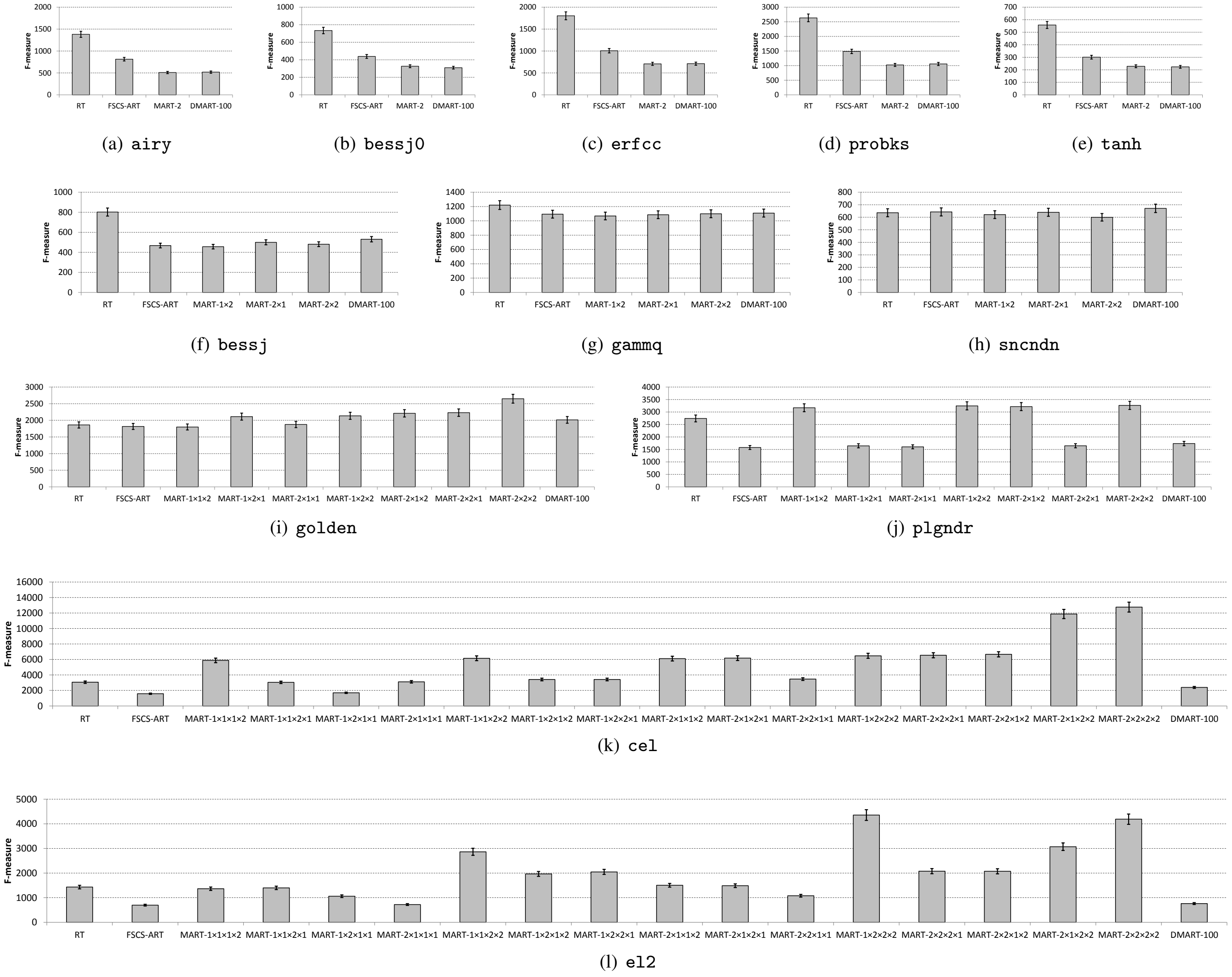
$(2^d - 1)$ mirror schemes for MART, which will be all considered in this study.

Based on experimental results, we can have the following observations.

- For real-life programs with $d = 1$ (airy, bessj0, erfcc, probks, and tanh), DMART has much smaller F-measures than FSCS-ART; for programs with $d \geq 2$, in most cases DMART and FSCS-ART perform similarly.
- Compared with MART, when testing object programs without failure-unrelated parameters (including airy, bessj0, erfcc, probks, tanh, bessj, gammq, sncndn, and golden), DMART has similar F-measures in most cases. However, when testing programs with failure-unrelated parameters (including plgndr, cel, and el2), DMART generally has similar failure-detection effectiveness to the best case of MART. In other words, MART with different mirroring schemes has different performances. It can be also observed that when the mirror partitioning is conducted on at least one failure-unrelated parameter, the F-measure of MART will become larger than that of RT.

5.3. *Answer to RQ2: Execution Time*

Fig. 13 shows the test case generation time of FSCS-ART, MART and DMART, in which *x*-axis represents *n* while *y*-axis represents the execution time required to generate *n* test



(a) `airy` (b) `bessj0` (c) `erfcc` (d) `probks` (e) `tanh`

(f) `bessj` (g) `gammq` (h) `sncndn`

(i) `golden` (j) `plgndr`

(k) `cel`

(l) `el2`

Fig. 12: F-measures on each object program

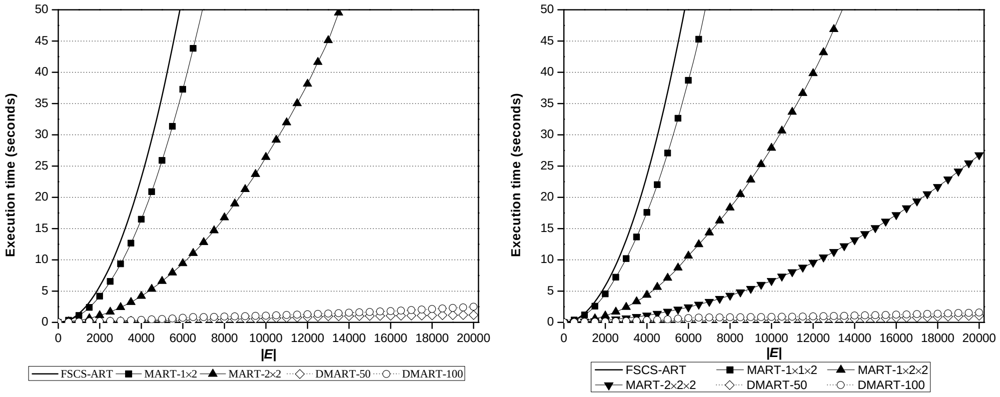
cases. It should be noted that MART with mirroring partitioning $u_1 \times u_2 \times \cdots \times u_d$ ($1 \leq u_i \leq 2, i = 1, 2, \cdots, d$) actually has the same test case generation time as that with each mirroring partitioning $u_{j_1} \times u_{j_2} \times \cdots \times u_{j_d}$, where $(u_{j_1}, u_{j_2}, \cdots, u_{j_d})$ is a permutation of $(u_1, u_2, \cdots, u_d)$, because the number of subdomains for the mirroring partitioning is fixed (i.e., $\prod_{i=1}^{d} u_i$). Therefore, we use MART-$u_{j_1} \times u_{j_2} \times \cdots \times u_{j_d}$, where $u_{j_1} \leq u_{j_2} \leq \cdots \leq u_{j_d}$, to represent the MART algorithm.

Based on the experimental results, we can have the following observations.

- DMART has much lower execution time than FSCS-ART and MART with different ways of mirroring partitioning for generating the same number of test cases, regardless of the dimension of the input domain.
- DMART with higher cutoff value δ generally has higher computational overhead than that with lower δ, but the difference in overhead within different DMART methods is ignorable compared with the difference between DMART and MART/FSCS-ART.
- Consistent with previous MART studies [21, 27], the test case generation time of MART is less than that of FSCS-ART, and MART with the larger number of mirror domains needs less test case generation time than that with the smaller number of mirror domains.
- Similar to FSCS-ART, the computational overhead of MART is also in $O(n^2)$. However, the computation overhead of DMART is in linear order, i.e., $O(n)$.

Fig. 14 reports the average time rather than generation time to detect the first failure for each object program. Based on experimental data, we can observe that DMART needs much less average time to detect the first failure than FSCS-ART and MART with any mirror partitioning, irrespective of object program.

In summary, our DMART method can deliver similar failure-detection effectiveness as FSCS-ART but with much lower computation overhead (execution time and average time



(a) $d = 2$

(b) $d = 3$

Fig. 13: Test case generation time of DAMRT algorithm

to detect the first failure). Compared with MART, our DMART method not only requires much less time, but also delivers more reliable failure-detection effectiveness especially when there are failure-unrelated input parameters. Therefore, we can suggest that DMART should be considerably more cost-effective than both DMART and FSCS-ART.

6. Threat to Validity

Despite our best efforts, our experiments may face some threats to validity. In this section, we briefly discuss them, which are classified into the following categories.

6.1. Threats to Construct Validity

Construct validity refers to whether or not we have conducted the studies fairly. In this paper, we emphasize the failure-detection effectiveness of each ART method, measured with the F-measure (or ART F-ratio). As we know, there are another two most popular effectiveness of metrics: P-measure – the probability of detecting at least one failure; and E-measure – the expected number of failures detected [10]. Generally, a set of test cases with a given size is assumed when using P-measure or E-measure. Since the size of the test case set has significant impact on the P-measure or E-measure values, F-measure is more suitable than P-measure and E-measure to be used to compare adaptive methods, such as ART [32]. Additionally, previous studies have consistently demonstrated that whenever ART has smaller F-measures than RT, ART also performs better than RT in terms of P-measure [32]; while E-measure is even less appropriate than P-measure as it has been observed [25] that multiple failures may be associated with single fault. As a consequence, F-measure rather than P-measure or E-measure is used as the evaluation metric in this study.

6.2. Threats to External Validity

External validity is specifically about to what extent our experimental results can be generalized. The first threat to external validity of the experiment is the choice of the representative ART algorithms. The main purpose of this paper is to propose an enhanced *mirror adaptive random testing* (MART), i.e., *dynamic mirror adaptive random testing* (DMART). Both MART and DMART can be applied to most of ART algorithms. Since previous MART studies [21, 27] have used one version of ART by distance to represent the ART algorithm, namely *fixed size candidate set ART* (FSCS-ART) [10], it is reasonable to use FSCS-ART for DMART in our study.

The second threat to external validity is the cutoff value δ used in the DMART algorithm. We choose 50 and 100 for δ in our experiments, but we do not discuss in depth the impact of δ on the effectiveness of DMART.

The third threat to external validity is that the selection of object programs is based on the previous study and for ease of comparison, and the dimension of each program is not very high. However, there is no reason why the results on higher dimension will be quite different from the current ones.

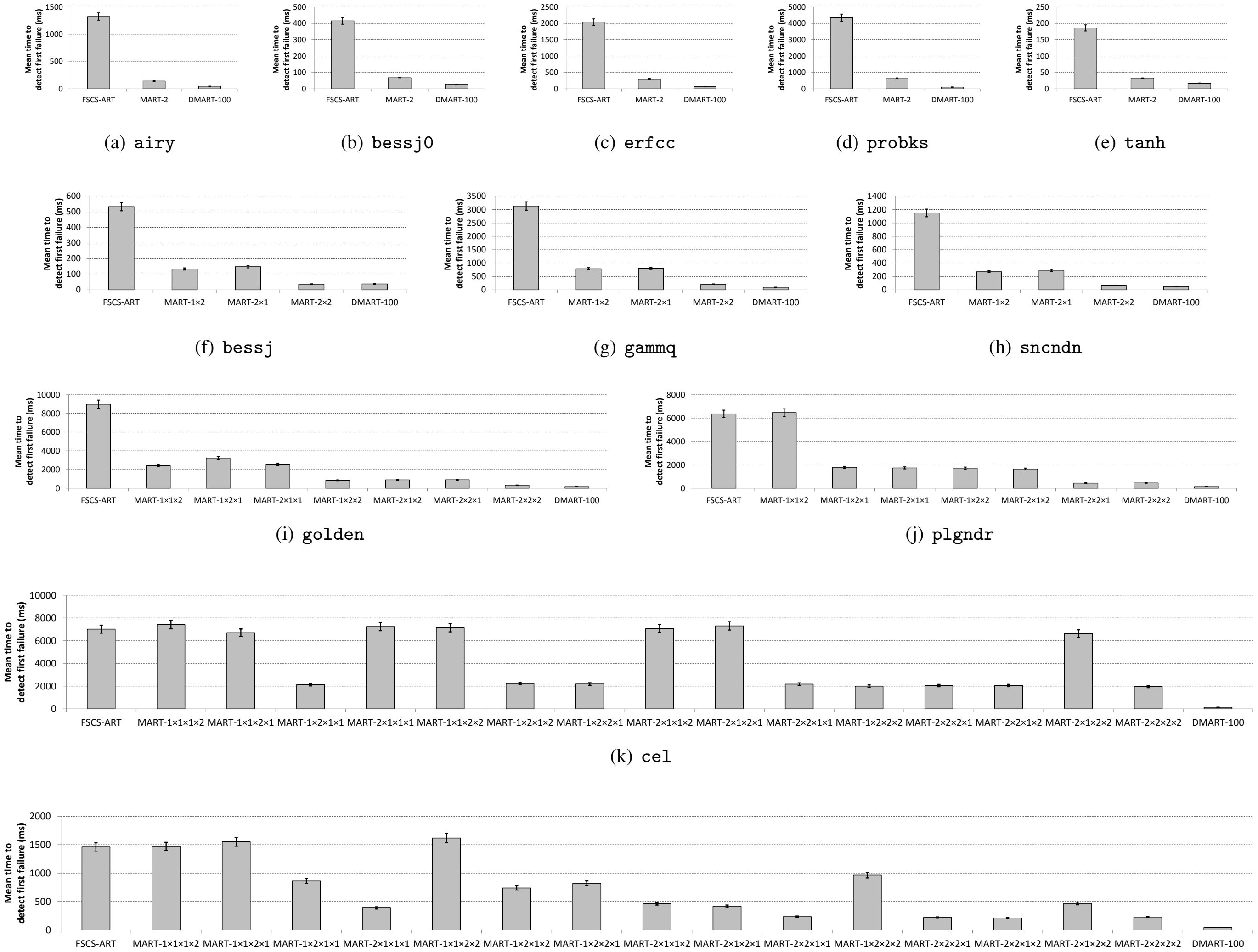
To address these potential threats, additional studies, using other ART algorithms such as *lattice-based ART* (LART) [15] and *restricted random testing* (RRT) [16], more δ values, and more object programs, will be conducted in the future.

6.3. Threats to Internal Validity

Internal validity refers to whether or not there were mistakes in the experiments. We have manually cross-validated our analyzed programs on various examples, and we have confidence on the correctness of the simulation and empirical setups.

7. Related Work

It is worthwhile to note that there are some other overhead reduction techniques for ART. In this section, we briefly review five techniques, namely *ART through dynamic partitioning* (ART-DP) [22], *ART through iterative partitioning* (ART-IP) [23], *forgetting* [24], *fast random border centroidal voronoi tessellations* (RBCVT-Fast) [25], and *ART with divide-and-conquer* (ART-DC) [26].



(a) airy (b) bessj0 (c) erfcc (d) probks (e) tanh

(f) bessj (g) gammq (h) sncndn

(i) golden (j) plgndr

(k) cel

(l) el2

Fig. 14: Average time to detect the first failure for each object program

7.1. ART-DP

Chen et al. [22] have proposed ART-DP, which partitions the input domain dynamically. There are two main types of ART-DP, namely *ART by bisection* (ART-B) and *ART by random partitioning* (ART-RP). ART-B divides the input domain into equally-sized partitions by bisecting the longest coordinate. The next test case is randomly generated from one empty partition. Once each partition contains a test case, another partitioning process will be conducted. ART-RP always selects the next test case randomly from the largest subdomain in the input domain, and uses this test case to further divide the subdomain on all coordinates. The time complexities of ART-B and ART-DP are $O(n)$ and $O(n \log n)$, respectively, for generating n test cases [33]. Though our DMART approach also involves dynamic partitioning, it is not a standalone ART algorithm but used to improve the efficiency of existing ART algorithms.

7.2. ART-IP

ART-IP [23] provides a new partitioning approach. When all the partitions are either non-empty (that is, containing executed test cases) or adjacent to the non-empty partitions, ART-IP applies an "iterative" partitioning scheme to generate totally new partitions from scratch. On the contrary, ART-DP always generates new partitions by dividing the previous partitions. For the generation of the next test case, ART-IP not only eliminates the non-empty partition, but also disregards their neighbors. In other words, ART-IP achieves an even spread of test cases by selecting test cases only from the partitions that are far away from the non-empty partitions. Similar to ART-DP, ART-IP is a standalone ART algorithm, whereas our DMART approach can be applied to reduce the computation overhead of many existing ART algorithms.

7.3. Forgetting

Generally speaking, generating the next test case is very time-consuming, because with the increase of the number of previously executed test cases (i.e., $|E|$), their computation overhead become higher. The forgetting overhead reduction technique [24] generates the next test case by utilizing a constant number of previously executed test cases rather than all of them. Chan's studies [24] showed that if a constant number of previously executed test cases is used, the forgetting makes the new test case generation independent of $|E|$, which leads to the time complexity in linear order. However, the simulation results also show that the F-measure of the ART algorithm with forgetting depends on the number of test cases used in generating the next test case: The fewer executed test cases were used, the larger the F-measure was. Compared with the forgetting technique, our DMART approach delivers a more consistent performance improvement over RT.

7.4. RBCVT-Fast

Recently, Shahbazi *et al.* [25] have proposed a novel testing approach by using the Centroidal Voronoi Tessellations, namely *random border centroidal voronoi tessellations* (RBCVT), in order to make random test cases evenly spread over the input domain. Since the order of time complexity of RBCVT is quadratic to $|E|$ (i.e., $O(|E|^2)$), they have used a novel search algorithm to develop an alternative RBCVT, namely RBCVT-Fast, which can generate test cases with linear runtime ($O(|E|)$). RBCVT-Fast was a "brand-new" ART algorithm, different from all other ART algorithms.

7.5. ART-DC

Chow et al. [26] have proposed ART-DC to improve the efficiency of ART algorithms. ART-DC first defines a constant number, namely *threshold* denoted by λ, and then uses the ART algorithm to generate test cases over the whole input domain. When the number of already executed test cases reaches λ, ART-DC bisectionally divides the input domain into some subdomains, and then generates test cases within each subdomain. The simulation results show that ART-DC can significantly improve efficiency of ART algorithms while maintaining the similar failure-detection effectiveness.

Though ART-DC and our DMART approach show a certain degree of similarity especially with respect to the bisectional partitioning process, they are different from basic intuition. ART-DC makes use of the concept of "divide-and-conquer" and divides a large problem into some sub-problems. DMART uses the bisectional partitioning process to dynamically construct source and mirror domains, with the specific purpose of overcoming the drawbacks of the static mirroring scheme in the original MART method.

In ART-DC, the original ART algorithm is implemented in each subdomain, while DMART only requires the implementation in half of the subdomains. In other words, as compared to ART-DC, DMART virtually incurs half of the execution cost for generating the same number of test cases. With regard to the failure-detection effectiveness, by examining the simulations results in previous study of ART-DC [26], we can observe that DMART performs at least similarly to ART-DC.

8. Conclusion

Adaptive random testing (ART) was proposed to enhance the failure-detection effectiveness of random testing (RT). Previous studies have demonstrated that ART can generate more evenly distributed test cases than RT over the input domain, and requires fewer test cases to detect the first failure. However, it was also noted that ART needs higher computation overhead to achieve the even spread of test cases. Mirror adaptive random testing (MART) was proposed as a light-weight ART technique to reduce computation overhead, which applied original ART algorithms to the source domain and then used a simple mirror function to generate test cases in the mirror domains. MART can reduce computational overhead compared with original ART algorithms while maintaining similar failure-detection effectiveness. However, MART actually cannot decrease the order of magnitude for computation overhead of original ART algorithms. Additionally, before testing testers may face a challenge of choosing the mirroring scheme (including mirror partitioning, mirror function, and mirror selection order) for MART, because different mirroring schemes may result in significantly different performances. In this paper, we proposed an enhancement of MART, namely dynamic mirror adaptive random testing (DMART), in order to overcome the above drawbacks of MART by using a new mirroring scheme. The simulation results indicate that DMART needs much less time than the original ART and MART algorithms for generating the same number of test cases, while achieving at least the similar failure-detection capabilities. Additionally, the empirical studies show that DMART has a more reliable performance than MART. In particular, DMART performs better than MART when testing programs with failure-unrelated parameters.

One important research direction for future work is on the cutoff value δ used in DMART. Intuitively speaking, DMART with larger δ generally has better failure-detection capability but requires higher execution costs, because the number of test cases used in the generation of the next test cases becomes larger. On the other hand, if δ is assigned a small value, the testing efficiency of DMART can be improved, but its testing effectiveness would be constrained. In other words, there exists a trade-off between testing effectiveness and efficiency of DMART. In practice, when testers want to have better testing efficiency, it is reasonable to choose smaller δ values, such as 5 and 10; on the contrary, when testers want to deliver higher testing effectiveness, larger δ values should be selected, such as 100 and 200. It is quite interesting to investigate the impacts of different values of δ and see how to balance the trade-off between the effectiveness and efficiency.

Since the mirroring scheme of DMART is independent of concrete ART algorithms, DMART can be generally applied to improve the cost-effectiveness of various ART algorithms. Nevertheless, it is still desirable to have more investigations on the integration of DMART and other ART algorithms. The current studies of both MART and DMART have focused on numeric input domains. It is also worthwhile to extend MART and DMART to other types of input domains, especially those with non-numeric inputs [34].

Acknowledgement

This work is in part supported by the National Natural Science Foundation of China (Grant No. 61202110 and 61103053), the Natural Science Foundation of Jiangsu Province (Grant No. BK2012284), and the Senior Personnel Scientific Research Foundation of Jiangsu University (Grant No. 14JDG039).

References

[1] P. Bourque, R. E. D. Farley (Eds.), SWEBOK 3.0: Guide to the Software Engineering Body of Knowledge, 3rd Edition, IEEE, 2014.

[2] A. Orso, G. Rothermel, Software testing: A research travelogue (2000-2014), in: Proceedings of Future of Software Engineering, FOSE '14, 2014, pp. 117–132.

[3] P. G. Frankl, R. G. Hamlet, B. Littlewood, L. Strigini, Evaluating testing methods by delivered reliability, IEEE Transactions on Software Engineering 24 (8) (1998) 586–601.

[4] G. J. Myers, The Art of Software Testing, John Wiley & Sons: New York, 2004.

[5] B. P. Miller, L. Fredriksen, B. So, An empirical study of the reliability of unix utilities, Communications of the ACM 33 (12) (1990) 32–44.

[6] J. Forrester, B. Miller, An empirical study of the robustness of windows nt applications using random testing, in: Proceedings of the 4th USENIX Windows Systems Symposium (WSS '00), 2000, pp. 59–68.

[7] T. Yoshikawa, K. Shimura, T. Ozawa, Random program generator for java jit compiler test system, in: proceedings of the 3rd International Conference on Quality Software (QSIC '03), 2003, pp. 20–24.

[8] J. Regehr, Random testing of interrupt-driven software, in: Proceedings of the 5th ACM International Conference on Embedded Software (EMSOFT '05), 2005, pp. 290–298.

[9] H. Bati, L. Giakoumakis, S. Herbert, A. Surna, A genetic approach for random testing of database systems, in: Proceedings of the 33rd International Conference on Very Large Data Bases (VLDB '07), 2007, pp. 1243–1251.

[10] T. Y. Chen, H. Leung, I. K. Mak, Adaptive random testing, in: Proceedings of the 9th Asian Computing Science Conference (ASIAN '04), 2004, pp. 320–329.

[11] L. White, E. Cohen, A domain strategy for computer program testing, IEEE Transactions on Software Engineering 6 (3) (1980) 247–257.

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

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

[14] P. G. Bishop, The variation of software survival times for different operational input profiles, in: Prceedings of the 23rd International Symposium on Fault-Tolerant Computing (FTCS '93), 1993, pp. 98–107.

[15] J. Mayer, Lattice-based adaptive random testing, in: Proceedings of the 20th IEEE/ACM International Conference on Automated Software Engineering (ASE '05), 2005, pp. 333–336.

[16] K. P. Chan, T. Y. Chen, D. Towey, Restricted random testing: Adaptive random testing by exclusion, International Journal of Software Engineering and Knowledge Engineering 16 (4) (2006) 553–584.

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

[18] Y. Lin, X. Tang, Y. Chen, 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 '09), 2009, pp. 221–232.

[19] H. Hemmati, A. Arcuri, L. Briand, Achieving scalable model-based testing through test case diversity, ACM Transactions on Software Engineering and Methodology 22 (1) (2012) 6:1–6:42.

[20] A. Arcuri, L. Briand, Adaptive random testing: An illusion of effectiveness?, in: Proceedings of the 20th International Symposium on Software Testing and Analysis (ISSTA '11), 2011, pp. 265–275.

[21] T. Y. Chen, F.-C. Kuo, R. Merkel, S. P. Ng, Mirror adaptive random testing, Information & Software Technology 46 (15) (2004) 1001–1010.

[22] T. Y. Chen, R. Merkel, P. K. Wong, G. Eddy, Adaptive random testing through dynamic partitioning, in: Proceedings of the 4th International Conference on Quality Software (QSIC '04), 2004, pp. 79–86.

[23] T. Y. Chen, D. H. Huang, Z. Q. Zhou, On adaptive random testing through iterative partitioning, Journal of Information Science and Engineering 27 (4) (2011) 1449–1472.

[24] K.-P. Chan, T. Y. Chen, D. Towey, Forgetting test cases, in: Proceedings of the 30th Annual International Computer Software and Applications Conference (COMPSAC '06), 2006, pp. 485–494.

[25] A. Shahbazi, A. F. Tappenden, J. Miller, Centroidal voronoi tessellations — a new approach to random testing, IEEE Transactions on Software Engineering 39 (2) (2013) 163–183.

[26] C. Chow, T. Y. Chen, T. Tse, The art of divide and conquer: An innovative approach to improving the efficiency of adaptive random testing, in: Proceedings of the 13th International Conference on Quality Software (QSIC '13), 2013, pp. 268–275.

[27] F.-C. Kuo, An indepth study of mirror adaptive random testing, in: Proceedings of the 9th International Conference on Quality Software (QSIC '09), 2009, pp. 51–58.

[28] T. Y. Chen, F.-C. Kuo, Z. Zhou, On favourable conditions for adaptive random testing, International Journal of Software Engineering and Knowledge Engineering 17 (6) (2007) 805–825.

[29] ACM, Collected Algorithms from ACM, Association for Computer Machinery: New York, 1980.

[30] W. Press, B. P. Flannery, S. A. Teulolsky, W. T. Vetterling, Numerical Recipes, Cambridge University Press: Cambridge, 1986.

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

[32] S. Anand, E. K. Burke, T. Y. Chen, J. Clark, M. B. Cohen, W. Grieskamp, M. Harman, M. J. Harrold, P. Mcminn, An orchestrated survey of methodologies for automated software test case generation, Journal of Systems and Software 86 (8) (2013) 1978–2001.

[33] J. Mayer, C. Schneckenburger, An empirical analysis and comparison of random testing techniques, in: Proceedings of the 2006 ACM/IEEE International Symposium on Empirical Software Engineering (ISESE2006), 2006, pp. 105–114.

[34] A. Barus, T. Y. Chen, F.-C. Kuo, H. Liu, R. Merkel, G. Rothermel, A novel linear-order algorithm for adaptive random testing on programs with non-numeric inputs, Tech. rep., University of Nebraska – Lincoln.