One-Domain-One-Input: Adaptive Random Testing by Orthogonal Recursive Bisection with Restriction

Hilary Ackah-Arthur, Jinfu Chen, Dave Towey,

Michael Omari, Jiaxiang Xi, and Rubing Huang

One-Domain-One-Input: Adaptive Random Testing by Orthogonal Recursive Bisection with Restriction (ART-ORB)

Hilary Ackah-Arthur, *Member, IEEE CS*, Jinfu Chen*, *Member, IEEE*, Dave Towey, *Member, IEEE*, Michael Omari, Jiaxiang Xi, and Rubing Huang, *Member, IEEE*

***Abstract*—One goal of software testing may be the identification or generation of a series of test cases that can detect a fault with as few test executions as possible. Motivated by insights from research into failure-causing regions of input domains, the even-spreading (even distribution) of tests across the input domain has been identified as a useful heuristic to more quickly find failures. This finding has encouraged a shift in focus from traditional random testing (RT) to its enhancement, adaptive random testing (ART), which retains the randomness of test input selection, but also attempts to maintain a more evenly distributed spread of test inputs across the input domain. Given that there are different ways to achieve the even distribution, several different ART methods and approaches have been proposed. This paper presents a new ART method, called ART-ORB, which explores the advantages of repeated geometric bisection of the input domain, combined with restriction regions, to evenly spread test inputs. Experimental results show a better performance in terms of fewer test executions than RT to find failures. Compared with other ART methods, ART-ORB has comparable performance (in terms of required test executions), but incurs lower test input selection overheads, especially in higher dimensional input space. It is recommended that ART-ORB be used in testing situations involving expensive test input execution.**

***Index Terms*—Random testing, adaptive random testing, partition testing, orthogonal recursive bisection, restricted random testing.**

I. Introduction

In software testing, exhaustive testing (the testing of all possible input combinations) is almost never possible, due to the large and complex nature of most software systems. The selection of appropriate test inputs—ones more likely to reveal failures or problems in the software—is therefore critical for the effective evaluation of the software's quality. Much research has been conducted into diverse testing techniques that could improve the failure detection capability of test inputs [1], [2], [3]. Random testing (RT) [4] is a simple and fundamental technique that generates test inputs by simply randomly selecting them from the entire *input domain* (the set of all possible inputs) [4]. RT has been successfully applied in industry, detecting software failures [5], [6], [7]. However, a criticism of RT has been that, because it does not use any information of the specifications or the program under test in selecting the test inputs, its failure detection effectiveness can be limited [8].

It has been observed that the failure-causing inputs (program inputs that can reveal failures) of most programs form contiguous regions in the input domain [9], [10], [11]. Motivated by this observation, Chen *et al.* [12] proposed the adaptive random testing (ART) approach to enhance the failure detection effectiveness of RT. In addition to selecting test inputs randomly, ART employs a mechanism to evenly spread the inputs over the input domain. Several ART methods have been proposed that employ different strategies to ensure the random and even spread of test inputs [13], [14], [15], [16], [17]. Compared with RT, most of these ART methods provide improved failure detection effectiveness, in terms of F-measure—the number of test inputs executed to find a failure [18], [19]. Many also include mechanisms to reduce the computational overheads incurred due to the additional even-spreading.

Two frequently used strategies employed in ART are partitioning and excluding. Both strategies sample the input domain when performing their testing processes, but differ in the procedures and assumptions they employ: Partitioning considers only the sampling rate of each sub-domain [16], while excluding only considers the sampling rate of the non-excluded regions [14]. However, most ART methods that use partition or exclusion strategies tend to require fewer test input executions before detecting failures.

This paper proposes a new ART method that aims to provide faster failure detection performance (compared with both RT and other ART methods) while maintaining a more acceptable level of computational overheads. The method, called ART by orthogonal recursive bisection (ART-ORB), integrates both partition and exclusion strategies. ART-ORB selects test inputs from outside of restricted regions, and uses pairs of non-failure-revealing tests within a domain to partition the domain geometrically. Section III presents an in-depth description of the method.

This paper makes the following contributions:

This work is partly supported by National Natural Science Foundation of China (NSFC grant numbers: U1836116, 61502205, 61762040 and 61872167), the project of Jiangsu provincial Six Talent Peaks (Grant number: XYDXXJS-016), and the Graduate Research Innovation Project of Jiangsu Province (Grant number: KYLX16_0900).

H. Ackah-Arthur, J. Chen, M. Omari, J. Xi, and R. Huang are with the School of Computer Science and Communication Engineering, Jiangsu University, Zhenjiang, 212013, China (e-mail: {hilaryaa, jinfuchen, michael, xijiaxiang, rbhuang} @ujs.edu.cn).

D. Towey is with the School of Computer Science, University of Nottingham Ningbo China, Ningbo, Zhejiang 315100, China (e-mail: dave.towey@nottingham.edu.cn).

*All correspondence should be addressed to Jinfu Chen.

- We propose a new ART method that uses repeated geometric bisection of the input domain combined with use of restricted regions, to better spread test inputs.
- We present an algorithm and a binary tree data structure to provide a detailed process description of the proposed ART method.
- We conduct an investigation of the test input distributions of the proposed ART method. Using two commonly used metrics, we compute the distribution of generated test inputs and compare them with those of other testing methods.
- Using simulations and experiments with 16 error-seeded programs, we validate the failure finding performance of the proposed method, and compare its performance with RT and other, similar ART approaches.

The rest of this paper is structured as follows: Section II describes the background of the study and the related ART methods. Section III presents a detailed description of the proposed ART method, including the core algorithm. The simulation and experimental results are presented in Section IV. These results are discussed in Section V. Some of the potential threats to the validity of this study are examined in Section VI. Finally, Section VII concludes the paper.

II. Background and Related Studies

A. Background

1) Random Testing

Random Testing (RT) [4] is a fundamental and useful technique for testing software. RT involves selecting inputs in a random manner from the input domain until a stopping condition—such as detection of a failure, complete execution of a test suite, or the passage of a specified amount of time—is reached. It can efficiently generate large numbers of candidate tests, and need not have human influence or bias in the test case generation process [4]. This random generation may have the advantage of revealing failures that cannot be detected by deterministic approaches like branch testing [8] or domain testing [9]. The relative ease with which RT can usually be implemented, combined with the ability to calculate reliability estimates [4], make RT an attractive testing option that has been successfully applied in many real-world applications [5], [6], [7]. However, because it does not make use of additional available information from the program being tested [8], RT's failure detection effectiveness may be limited.

Empirical studies have shown that failure regions (portions of the input domain which, when selected as program input, reveal failures), tend to cluster into contiguous regions, especially for programs with numerical input domains [9], [10], [11]. Based on this observation, it is possible to make a simple improvement to RT, using generic information about the typical failure patterns seen in many programs.

2) Failure Pattern

Chan *et al.* [20] identified three categories of failure patterns: *point*, *strip* and *block*. An illustration of these failure patterns in a two-dimensional input domain is shown in Fig. 1.

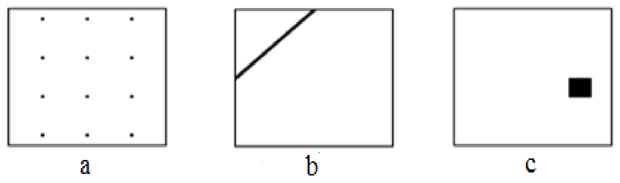


Fig. 1. Classifications of patterns of failure-causing inputs: (a) point pattern, (b) strip pattern, and (c) block pattern. The outer boundaries of each subfigure represent the borders of the two-dimensional input domain, and the filled regions represent the failure patterns (clusters of failure-causing inputs).

The point pattern may be characterized by multiple stand-alone points or small-sized regions scattered across the input domain. Strip patterns are narrow and elongated. A typical example of this failure pattern is White and Cohen's [9] domain errors. The main characteristic of the block pattern is that the failure-causing inputs are localized in either a single or a few contiguous, compact regions of the program's input space, with no obvious elongation in any dimension. Chan *et al.* [20] noted that point pattern may sometimes be spread in a regular manner throughout the input domain. They also explained that strip and block failure patterns were likely to be more common than point patterns. Examples 1, 2, and 3 show sample pseudo-code program snippets containing specific errors that lead to the three failure pattern types.

Example 1: A program fault that results in block failure pattern.
```
INTEGER X, Y, Z
INPUT X, Y
IF (X > 0 AND X < 10 AND Y > 0 AND Y < 10)
        Z = X  /* correct statement: Z = 2 * X   */
ELSE
        Z = 2 * Y
OUTPUT Z
```

Example 2: A program fault that results in strip failure pattern.
```
INTEGER X, Y, Z
INPUT X; Y
IF (Y <= 0)  /* correct statement: IF (Y <= 1) */
        Z = X – 2 * Y
ELSE
        Z = X + 2 * Y
OUTPUT Z
```

Example 3: A program fault that results in point failure pattern.
```
INTEGER X, Y, Z
INPUT X, Y;
IF (X mod 4 = 0 AND Y mod 4 = 0)
        Z = X – Y  /* correct statement: Z = X + Y */
ELSE
        Z = X * Y
OUTPUT Z
```

Intuitively, if the failure pattern is of block or strip type, then selection of test inputs close to each other would be less likely to quickly find the failure region quickly, in terms of the F-measure: A wide-spread and even distribution of test input execution should be faster. Consequently, after execution of a test that has not revealed a failure, choosing a next test input that is farther away from all the previously executed tests should be more likely to reveal failure. Inspired by these failure patterns observations, Chen *et al.* [12] proposed Adaptive Random Testing (ART), an improvement on the failure-detection capability of RT.

3) *Adaptive Random Testing*

ART is essentially a random testing method, but with a mechanism that uses information about the location of previously executed tests to widely spread test inputs over the input domain. Previous empirical studies [21], [15], [22] and experimental analyses [12], [23] have shown that ART can significantly outperform RT in terms of the F-measure, especially when the failure patterns are of block type. In some studies, approximately 50% fewer tests have been required to detect the first failure than RT. However, the overheads associated with the ART test generation process can be substantial, and may outweigh the advantages of executing fewer tests.

Research into ART methods that can maintain reductions in required test executions to find failures, but also minimize associated overheads, has yielded a number of ART implementations. These can be grouped according to several strategies, including: ‘distance strategy’ spreads test inputs by ensuring that each next test is far from all executed tests; 'exclusion strategy' uses exclusion regions around executed inputs to restrict test selection to other parts of the input domain; and 'partition strategy' divides the input domain into several sub-domains and distributes the selection of test inputs among them.

This paper proposes a new ART method that is based on partitioning, but that also employs exclusion to achieve the even spread of test inputs across the input domain.

4) *Orthogonal Recursive Bisection (ORB) Strategy*

Two traditional objectives of a good partitioning scheme may be: splitting data evenly among partitions; and ensuring efficient access to non-local data. The Orthogonal Recursive Bisection ORB strategy [24] is a domain decomposition approach that has been used to define mutual interactions among discrete entities in scientific simulations (such as molecules, charges, astrophysical bodies, etc.). It has also been used to distribute a large cardiac model data set to a distributed memory supercomputer [25]. ORB recursively subdivides a computational space into two domains with the equal numbers of particles, or the same calculation costs. It forms a balanced binary tree by geometrically splitting the domains (uniformly or non-uniformly) each time the process splits the tree. The direction of the division is chosen arbitrarily or alternates orthogonally (*x*, *y*, *z*, *x*,..., for a 3-dimensional, 3D, space), to form a cascade of sub-domains. In the example in Fig. 2, a 3D input space is split on the *x*-axis into two sub-domains. For each sub-domain, the split is next performed on the *y*-axis, yielding four sub-domains. The third split is applied to each of these four sub-domains on the *z*-axis, resulting in eight sub-domains. This iterative splitting along the *x*, *y*, and *z* axes produces a binary tree whose number of leaves equals the current number of partitions. The eight sub-domains represent the leaves of the binary tree.

ORB is relatively simple to implement and can be quite efficient. It requires an algorithm to determine the bisection point at each level. For incremental and non-uniform distributions of sub-domains, the approach picks the largest sub-domain each time and divides again. However, the aspect ratio of a sub-domain could become large, due to the direction of splitting (alternating among the axes), and can result in a sub-optimal interaction list in some applications [24], [25]. This problem can be solved by ensuring that the geometric split is always along the longest dimension. The bisection process is repeated until the desired number of (sub-)domains is obtained. ORB results in compact and localized sub-domains [26].

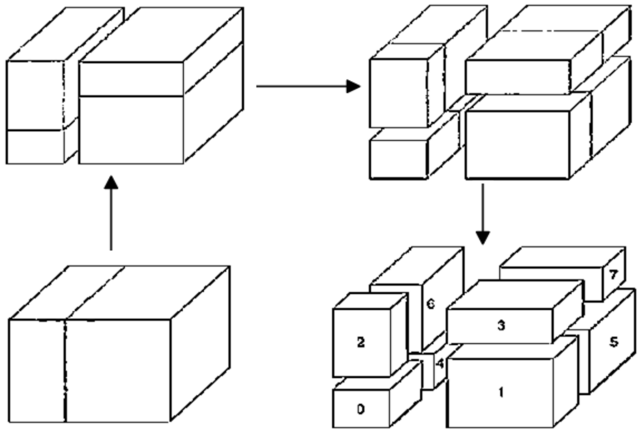


Fig. 2. Decomposition of a 3D domain using Orthogonal Recursive Bisection.

In this study, we modify the basic ORB algorithm and apply it in an ART implementation called ART-ORB to enhance the even spread of tests within the input domain. ART-ORB non-uniformly and incrementally partitions a given input domain by splitting the longest dimension of the largest domain each time. The domain-splitting mechanism ensures some distance between selected test inputs in different sub-domains, thereby enhancing their even spread. To further enhance the spread, and reduce the possibility of selecting tests close to each other within any sub-domain, ART-ORB also incorporates an exclusion strategy.

B. *Related Studies*

Adaptive Random Testing (ART) is a family of RT-based testing methods that aim to find failures faster by evenly spreading test inputs over the input domain. Several ART methods have been developed based on various strategies, using, for example, distance, exclusion, or partitioning. An early ART algorithm using the distance strategy is *Fixed-Size-Candidate-Set ART* (FSCS-ART) [12]. Using a uniform distribution, FSCS-ART generates a set of random test inputs (referred to as the *candidate set*), and computes their distances to all previous tests (the *executed set*). The element from the candidate set that is furthest from the executed tests is then chosen as the next test input. The underlying distance strategy of FSCS-ART has been used in several ART implementations [27], [28], [29].

An exclusion strategy restricts test case selection to certain areas of the input domain. Several exclusion strategies exist, with the original being Restriction-based ART (RART), also known as *Restricted Random Testing* (*RRT*) [30]. RRT makes use of exclusion regions drawn around previously executed tests, and restricts generation of the next test input to being from outside of these exclusion regions. For two-dimensional (2D) input domains, the exclusion regions are typically circles which ensure that a minimum distance exists between all generated tests (equal to the radius of the exclusion region). The size of each exclusion region is related to both the size of the entire input domain, and the number of previously executed tests [30]. For a 2D input domain with target exclusion area A_t and N previously executed tests, each exclusion region has a radius *(r)* of:

$$Radius(r) = \sqrt{(A_t/N\pi)} \qquad (1)$$

The *target exclusion area* A_t is the portion of the input domain area A that we attempt to exclude from test generation. It is determined by the *target exclusion ratio*, R [31]: $A_t = AR$.

ART methods that use a partitioning strategy are inspired by partition testing [32], which involves test case generation methods that divide the input domain into a number of partitions and select test inputs from within each partition. These ART methods can be categorized according to *how* they select tests from the partitions.

Two partition-based ART methods that draw tests randomly from within partitions are *ART by Random Partition* (ART-RP), and *ART by Bisection* (ART-B) [13]. Neither of these methods requires distance computations for the test selection. ART-RP uses the executed tests to iteratively partition the input domain, generating the next test from the largest of the partitions. Although random selection of test inputs from the largest partition enhances the even spread of the tests, there is a chance that the selected input may be close to a previously executed one. Similar to ART-RP, our proposed ART-ORB also randomly selects the next test from the largest sub-domain, using the previously executed tests to partition the domain. However, while ART-RP partitions a 2D domain into four sub-domains, ART-ORB partitions it into two. Also, ART-B iteratively bisects the largest dimension of the input domain to create equally-sized partitions: It randomly generates a test from each partition, and bisects all partitions as soon as they all contain tests. The failure detection effectiveness of ART-B reduces over time: Because there is no mechanism in place to generate tests from empty partitions, eventually, test inputs will be next to each other. As the number of partitions increases, test inputs subsequently selected from two empty regions next to each other may have similar characteristics. ART-ORB incrementally partitions each domain into two non-uniform sub-domains, only requiring reassignment of two test inputs. ART-B, in contrast, periodically partitions all domains into uniform sub-domains, incurring the overhead of multiple test input reassignments. Because neither ART-RP nor ART-B involves distance computations or comparisons, they have lower time overheads compared to other ART approaches. However, they also have lower failure-detection capability. Although both methods have some similarity to ART-ORB, they differ in their test selection, and partitioning method.

A second category of partition-based ART approaches involves applying a basic ART method (FSCS-ART [12] or RRT [30], for example) to select test inputs from within the partitions. *ART with divide-and-conquer* (ART-DC) [27] is one such method. It divides each dimension of the input domain into smaller, equal-sized, sub-domains when a predefined number (the *threshold*) of tests have been executed. Test inputs are then selected from these sub-domains using the original ART algorithm [12]. ART-DC has similar failure detection effectiveness to both FSCS-ART and RRT, and its computational complexity depends on the value of the threshold and the ART algorithm applied. Because of the overheads from fully applying the original ART algorithm in each of the sub-domains, the computational complexity can be comparatively high, especially for high threshold values. Although it also incrementally partitions the input domain, ART-DC is quite different from ART-ORB, both in terms of its partitioning approach, and test case selection in the partitions.

Another partition-based ART approach is *two-point partitioning ART* (ART-TPP) [28], which augments the *ART by balancing* approach [29] by applying a different test case selection criterion and using the midpoint of the test inputs to further partition the domain. ART-TPP selects the largest partition and, if there is no executed test there already, it randomly generates a test input. If the partition already contains an executed test, then a candidate set of random test inputs is generated, and the one furthest from the executed test is selected as the next test input. The partition is then divided through the midpoint of these last two test inputs. ART-TPP has some similarity to ART-ORB, as it also bisects through the midpoint of two tests within a partition and selects the next test from within the largest partition. However, it differs in its test case selection process and the number of partitions generated. While ART-ORB restricts regions as part of the selection strategy, ART-TPP computes distances to candidate test cases. Also, ART-TPP divides all dimensions of a domain while

TABLE I
COMMON CHARACTERISTICS OF ART METHODS

No.	Characteristics	ART-RP	ART-B	ART-DC	ART-TPP	ART-ORB
1	Random selection of test inputs within sub-regions	✓	✓			
2	Employs FSCS strategy to select test inputs within sub-regions			✓	✓	
3	Employs RRT strategy to select test inputs within sub-regions			✓		✓
4	Distance computation			✓	✓	✓
5	Generates test inputs from largest sub-region	✓			✓	✓
6	Employs location of test inputs to partition sub-regions	✓			✓	✓
7	Bisects using the longest dimension of a sub-region		✓			✓
8	Partitions sub-regions into equal sizes		✓	✓		
9	Partitions each sub-region at a time	✓			✓	✓
10	Partitions domains after a predefined number of test input selections		✓	✓		
11	Partitions a sub-region into two in each partitioning process		✓			✓

ART-ORB divides only one dimension for each test selection iteration. Although ART-TPP has comparable stability and failure detection capability to other partition-based ART methods [13], its associated candidate selection and distance calculations can become computationally expensive. The partitioning of each region into four sub-regions, and the checking and reassigning of previous test inputs to their respective sub-regions within the sub-domains increases the ART-TPP overheads.

Table I summarizes the similarities and differences among the described partition-based ART methods and ART-ORB. Since ART-DC uses either FSCS or RRT when selecting tests within sub-regions of the input domain, Characteristics 2 and 3 are both selected for ART-DC in the table.

The partitioning strategy has been used in many variations of ART [33], [22], [21]. A possible drawback of this strategy, however, is that it can incur high overheads, which can be wasteful, especially for strategies that divide the input domain from the very start of the testing (when faults may be less likely to be detected). Consideration of the overheads involved with partitioning is therefore very important when developing partition-based ART strategies. It has also been shown that employing restriction in test input generation generally provides better failure detection effectiveness than the use of candidate selection [14].

This paper presents a new ART method that employs *Orthogonal Recursive Bisection* [24] as a partitioning strategy to significantly reduce overheads, and applies RRT [30] as an exclusion strategy in the test input generation to increase the failure detection effectiveness.

III. Method

The proposed ART-ORB method attempts to evenly spread test inputs throughout the input domain through a combination of partitioning and use of exclusion regions.

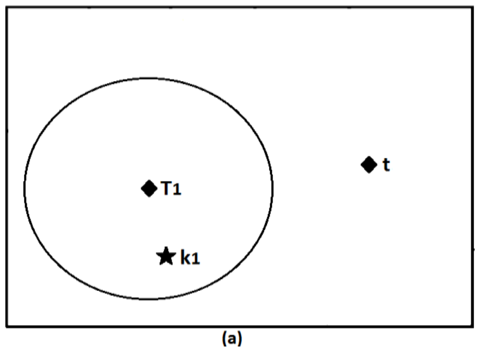
The first step is to randomly select a test input (T1) from the entire input domain and check whether or not it reveals a failure. If T1 does not reveal a failure, then, assuming a 2D input domain, a circular exclusion zone of radius r_1 is defined around T1 according to (1). (Higher dimensional input domains are dealt with similarly, but the exclusion zone is a corresponding hypersphere, and the radius is calculated accordingly.) ART-ORB then randomly generates the second test input, T2, from outside of this exclusion zone. T2 is executed to determine whether or not it reveals a failure: if it does not, then the entire input domain is partitioned into two sub-domains (regions), and the largest sub-domain is then identified. Using the area A_3 of this largest sub-domain, an exclusion zone of radius r_3 is created around the executed test input in the sub-domain, and the next test input (T3) is generated from outside this exclusion zone (but within the sub-domain). If T3 does not reveal a failure, then this sub-domain is also divided into two further sub-regions. ART-ORB continues by repeatedly selecting the next test from each successive largest region within the input domain and performing the exclusion and division operations until a generated test input reveals a failure.

When partitioning a region, ART-ORB uses the ORB strategy [24] with non-uniform partitioning (producing sub-regions of unequal sizes). ART-ORB uses the positions of the two test inputs in the region and the longest dimension of that region. The mid-point between the two test inputs is identified, and the region is split using a line perpendicular to the longest dimension through this point. This results in the two test inputs being separated, one at either side of the dividing line; one in each new sub-region. Because the dividing line's position is determined by the positions of the two test inputs, the resulting sub-regions are unlikely to have the same size. If $\boldsymbol{D}$ represents the input domain and $\boldsymbol{TS}$ represents a set of previously executed tests, then the process orthogonally divides the input domain into sub-domains $\{\boldsymbol{D}_1, \boldsymbol{D}_2, \ldots, \boldsymbol{D}_{s\}}$ such that $\bigcup_1^s \boldsymbol{D}_i = \boldsymbol{D}$ (where s denotes the number of sub-domains after each division). Because input domain division takes place only after a test has been executed, the number of sub-domains after each division process is equal to the total number of executed test cases ($s = |\boldsymbol{TS}|$).

The circular exclusion zone (in 2D) is chosen because previous research has shown this to provide the best RRT failure-finding performance [14], [34]. The size of the exclusion zone is partly determined by the target exclusion ratio (R) [31]. For example, in a 2D region, with a total area of 150, if R = 60%, ART-ORB will define an exclusion zone of area 150×0.6 = 90, centered on the single executed test in that region. The exclusion zone radius is calculated based on the dimensions of the input domain, using a formula for n dimensions:

$$Radius(r) = \sqrt[d]{\left(C_{d-2} \times \frac{d}{2}\right) \times \frac{AR}{N\pi^{\lceil d/2 \rceil}}} \qquad (2)$$

In (2), d is the dimension of the input domain, A is the area of the current region/partition, N is the number of previously executed tests, R is the target exclusion ratio and $[d/2]$ is the integer value of $d/2$. The result of $C_{d-2}\times d/2$ represents the formula coefficient for the dimension d, *and* C_{d-2} represents the formula coefficient for the d-2 dimension: For example, the radius formula for the 4D input domain [14] has a formula coefficient of 2 since the formula coefficient C_2 (that is C_{4-2}, where d=4) for 2D (as shown in (1)) is equal to 1. Chan *et al.* [14] provide a fuller description of various radius formulas, with formulas for n-dimensions also having been explained previously [35].



(a)

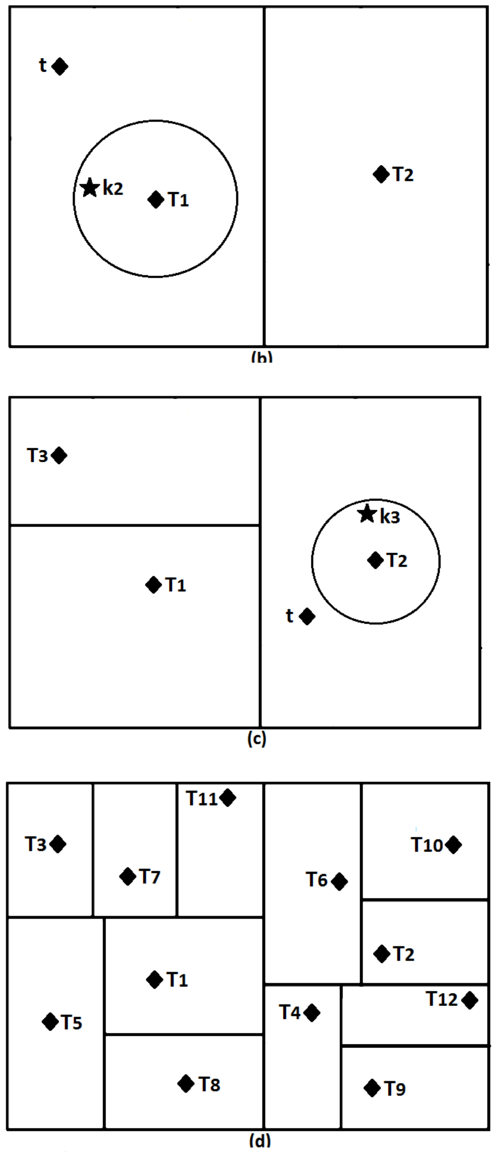


Fig. 3. Outline of ART by Orthogonal Recursive Bisection (ART-ORB). (**a**) Generation of the second test case. (**b**) Generation of the third test case. (**c**) Generation of the fourth test case. (**d**) Twelve generated test cases within the input domain.

A detailed example of the ART-ORB process is illustrated in Fig. 3. Here, ART-ORB randomly selects the first test input $\mathbf{T_1}$ from the input domain and defines an exclusion zone around it. Fig. 3(a) shows an exclusion zone defined around the first non-failure-revealing test case $\mathbf{T_1}$, and the next test (**t**) is randomly selected from outside this zone. When attempting to generate **t**, if a candidate is randomly selected from within the exclusion zone (such as $\mathbf{k_1}$, shown with a star symbol), it is discarded. If neither $\mathbf{T_1}$ nor $\mathbf{T_2}$ are failure-revealing, then the region is partitioned using a line perpendicular to the longest dimension through the mid-point of $\mathbf{T_1}$ and $\mathbf{T_2}$, as illustrated in Fig. 3(b). In the next steps, to generate the next test **t** (as 3rd and 4th tests), ART-ORB works within the largest region of the input domain (Figs. 3(b) and 3(c)), discarding test candidates $\mathbf{k_2}$ and $\mathbf{k_3}$, which were selected randomly but fell within the exclusion zones. Fig. 3(d) shows a possible distribution of test cases within the entire input domain after twelve test inputs have been selected.

A binary tree representation for the ART-ORB partitioning process shown in Fig. 3 is presented in Fig. 4. The nodes in the tree represent the regions within the input domain, and the percentage value in each node is the percentage of the total input domain area in that region at a particular stage of the partitioning process. The root of the binary tree **D** represents the entire input domain with a percentage area of 100%. The leaf nodes in Fig. 4 (highlighted with thick green circles) represent the current completely partitioned regions (corresponding to all regions in Fig. 3(d)). The sum of all current regions in the domain is equal to the complete input domain size, and thus the sum of the percentage area values in all the leaf nodes must be 100%. Because ART-ORB only allows one test per region, a test input generated from a particular region is assigned an identifier corresponding to that region: The test **T9** (Fig. 3(d)), for example, is in the current region **D9**.

At any stage of the partition process, the leaf node with the largest percentage size in the tree is partitioned, irrespective of its level in the tree. For example, the node in the tree that is divided after the root **D** has been partitioned is **D1**, because it has a larger proportion of the input domain (54.1%) than **D2** (45.9%). The numbers beneath parent nodes in Fig. 4 indicate the partitioning sequence. If the partitioning were to continue beyond the current twelve regions, the next node (region) to be partitioned would be **D5**, because its area is the largest. As this illustrates, it is possible for a node at a lower level in the tree to be partitioned before other nodes at higher levels—the nodes' levels do not influence the partition process.

Conventional partitioning-based testing strategies normally perform partitioning prior to the selection of any test cases. Although ART-ORB involves the notion of partitioning, it differs from conventional strategies in that the process is done progressively, and in real-time.

Because ART-ORB selects a new test input from the largest partition each time, and even within that specific partition, an exclusion region is defined around the previously executed test in it, ART-ORB can therefore be considered to use a "*double exclusion principle*" to generate the wide and even spread of test inputs.

A. *Algorithm Description*

We provide a formal description of the proposed ART-ORB method in Algorithm 1.

At each partitioning iteration, the largest subdomain is partitioned using a line perpendicular to the longest dimension through the mid-point of the two test inputs in the region. Each next test input is generated in the current region, outside of the exclusion zone around the previously executed input in the region [30], [14], [31]. The variable *curRegion* represents the current rectangular region, defined by its lower-left point and upper-right point coordinates.

The first test input is generated randomly from the entire input domain (line 4). In line 14, a test input (or point) in the current region is generated using RRT [30] with only one previously executed test input in the region. Any current region *curRegion* selected from *regionList* at any stage will contain one previously executed test input. The function findMaxRegion(*regionList*) (line 11) returns the index of the largest region in *regionList*. Because the exclusion zone size is

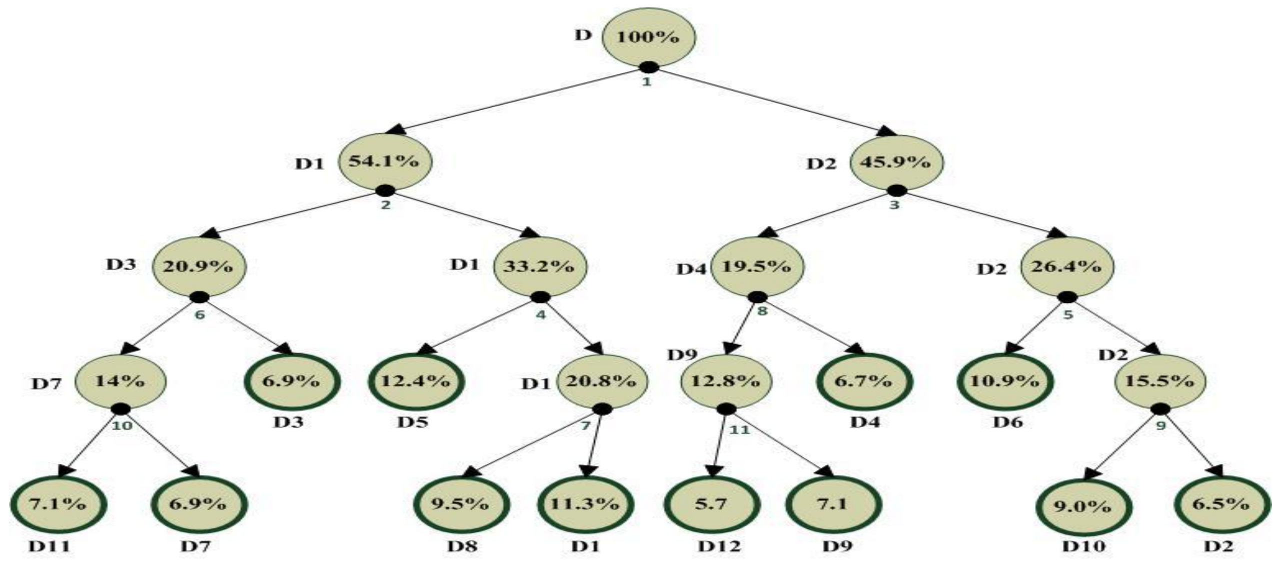


Fig. 4. Binary tree representation of the ART-ORB process.

proportional to the size of each region (refer to (2)), as the regions are recursively divided, the size of the exclusion region defined around the test input will also decrease.

Algorithm 1 *ART by Orthogonal Recursive Bisection (ART-ORB)*

```
Input: (1) D[]  // where D[] represents the input domain.
       (2) R  // exclusion ratio.
Output: TS ={T1; T2; _ _ _ ; Tn} // set of test cases

 1: Construct regionList = {}; // To store a list of regions or (sub-)domains.
 2: Construct TS = {}; //To store executed test cases.
 3: Set curRegion = D[]; // Assign the input domain to curRegion. curRegion
        represents the current region needed to be bisected recursively.
 4: tempT = generateRandPoint(curRegion); // Generate test case randomly
    from the entire input domain.
 5: TS = TS ∪ {tempT};
 6: if tempT finds failure then
 7:     break;
 8: end if
 9: Add curRegion to regionList;
10: while (stopping criteria not reached) do
11:     pIndex = findMaxRegion(regionList); //find the region with the largest
            size in regionList, and pIndex is the index of region for the next
            partition.
12:     curRegion = regionList.get(pIndex);
13:     T1= the existing test input in curRegion;
14:     T2=generateRandExPoint(curRegion., T1, R); // Generate a new test
          input by restricting region around T1 using exclusion ratio R within
          the current region.
15:     TS = TS ∪ {T2};
16:     if T2 finds failure then
17:         break;
18:     end if
19:     regionList.remove(pIndex); //remove the max-sized region from
          regionList
20:     Calculate the midpoint (median) of T1 and T2, divide curRegion
            orthogonally into two new sub-regions via this midpoint and using
            longest dimension of curRegion, and then add them into
            regionList;
21:     Locate T1 and T2 to their corresponding sub-regions;
22: end while
23: return TS ;
```

B. *Computational Efficiency*

The computational overhead of ART-ORB is analytically comparable to that of other ART methods. ART-ORB combines partitioning with an exclusion strategy. It does not employ any candidate selection, or require distance calculations to all previously executed test inputs in each selection process.

Assuming the size of the test case set is *N*. ART-ORB partitions a region of the input domain and applies an exclusion zone in the largest region around the executed test there. The time required for ART-ORB to identify the largest region (ie. *findMaxRegion()*) varies from 0 to *N*, therefore the complexity is *O(N/2)*. The original restriction algorithm selects the *N*th test input from the entire input domain with a complexity of *O(NlogN)*—each test input generation requires that the distances from each candidate test to all *N* previously executed tests be calculated. However, the ART-ORB algorithm only requires distance calculations associated with candidate tests from within the current region and the single executed test in that region; therefore using a constant time *k*. As a result, generating a new test input has a complexity of *O(k(logN))*. Hence, the worst-case time complexity of selecting *N* test cases using ART-ORB is *O(NlogN)*.

Unlike many other ART methods that use partitioning (e.g., ART-DC [27], ART-TPP [28], ART-RP [13], and ART-B [13]), ART-ORB is very efficient as it only reassigns the two executed test inputs in the divided sub-region after each partitioning.

IV. Empirical Studies and Analysis

A. *Setup of the Empirical Studies*

Because ART methods are enhancement to RT, our focus when evaluating the failure detection capability of ART-ORB is on its improvement over RT. ART-ORB presents a new partition-based ART strategy that aims to improve failure detection effectiveness and efficiency. Our empirical analysis had three phases.

Firstly, we performed simulations to evaluate ART-ORB's ability to evenly spread test cases throughout the input domain, or to analyze how close together the generated test cases are, compared to RT.

Secondly, we performed a series of simulations using different failure patterns and varying failure rates, to investigate the impact of different failure regions on the failure detection effectiveness of ART-ORB, again comparing with RT. Since all ART methods share the aim of improving on RT, we also determined how the performance of ART-ORB compares with some similar ART methods (ART-RP, ART-B, ART-DC, and ART-TPP). We performed a series of simulations in a 2D input space to ascertain: (1) the failure detection effectiveness performance of ART-ORB compared with RT and the other ART methods; and (2) the efficiency of the proposed method, compared with the other ART methods (the ART test input selection process typically incurs increased time costs).

Lastly, we performed experiments with 16 real, previously published, fault-seeded programs [12], [35], [28] to further validate the results obtained in the simulations. The programs were selected due to their varying dimensions and failure rates.

1) *Research Questions*

Our empirical study was guided by the following research questions:

RQ1: How evenly spread is the distribution of test cases generated by the ART-ORB method?

RQ2: Does ART-ORB perform better than RT for all failure patterns, in terms of the F-ratio?

RQ3: How does ART-ORB compare with other partition-based ART methods, in terms of the F-ratio, E-measure, Fm-time, and Execution time?

RQ4: What is the statistical significance of the ART-ORB performance compared to other ART methods, in terms of E-measure?

Although ART-ORB's use of the exclusion strategy with minimum distance computations significantly reduces the test generation costs, it may also potentially lead to a situation where several inputs are close to one another instead of being evenly distributed, due to *boundary effect* [36]. This could have a negative impact on the failure detection ability of the method. The first research question (RQ1) was designed to empirically evaluate the extent of this potential undesirable effect, if any. This is also vital for determining ART-ORB's ability to distribute test inputs, as it has been shown that more evenly distributed tests have higher failure detection [37]. The second research question (RQ2) was designed to establish the extent to which ART-ORB improves on ordinary RT, for different failure patterns. The third and fourth research questions (RQ3 and RQ4), were designed to compare ART-ORB to similar ART methods, to help identify situations in which ART-ORB should be applied instead of the other ART methods.

2) *Experimental Environment*

The environment used to conduct the simulations and experiments was the Windows 10 Professional (64 bits) OS, running on Intel Core i3 Duo processors, with a speed of 3.70 GHz each, and memory of 4 GB. We implemented all algorithms in Java and ran them on the Eclipse neon platform with JDK 1.7. We employed the Spyder utility within the Anaconda platform for generating the charts and used the R language platform for the statistical analysis.

3) *Test Distribution Metrics*

We adopted two diversity metrics [37] to measure how well-spread the distribution of test inputs generated by ART-ORB was. These two metrics, *Discrepancy* and *Dispersion*, are commonly used for measuring the equidistribution of sample points. Discrepancy indicates whether or not different regions inside the input domain D have similar densities of tests:

$$Discrepancy = \max_{i=1}^{m} \left| \frac{|T_i|}{|T|} - \frac{|D_i|}{|D|} \right| \tag{3}$$

where $D_1, D_2, D_3, \ldots, D_m$ are m randomly defined rectangular sub-domains of the input domain D, and $|D_i|$ is the size of D_i. T is the set of all selected test cases from D, and T_i is a subset of T from sub-domain D_i, such that $|T_i| = |T \cap D_i|$. The value of m cannot be too small; otherwise, a reliable approximation of the discrepancy may not be possible. Similarly, m cannot be too large either, because the computational overhead increases as the value of m increases. To balance the overheads and accuracy, we set m to be 1000, which is consistent with previous studies [37], [38].

Dispersion indicates whether or not there is a large empty region (containing no tests) in the input domain D, and is reflected by the maximum distance that any test input has from its nearest neighbor:

$$Dispersion = \max_{i=1}^{n} dist(t_i, \eta(t_i, T \setminus \{t_i\})) \tag{4}$$

where $dist(u,v)$ denotes the Euclidean distance between two points u and v, $\eta\ (u, N)$ refers to u's nearest neighbor in set N, and $T = \{t_1, t_2, \ldots, t_n\}$ is the set of all test cases.

Discrepancy and Dispersion have been used previously [37], [38], [39] to measure the test case distributions of various ART algorithms, and have provided evidence of the existence of a strong correlation between the even spread of test inputs and the failure detection effectiveness. Intuitively speaking, smaller Discrepancy and Dispersion values indicate more evenly distributed sets of generated test inputs.

4) *Failure Region Definition*

In order to simulate the testing of faulty programs in different situations, we defined a 2D input domain and created randomly located failure regions of the required patterns and sizes (based on the predefined failure rates). We applied the different testing methods in this input domain to generate test inputs.

The block pattern failure region was obtained by randomly defining a square region that provided the failure rate required within the input domain. The strip failure pattern was obtained by randomly choosing two points each on adjacent borders of the domain, and connecting them to form a strip representing the failure region. We then adjusted the width of the strip by varying moving the points to achieve the desired size of the failure region. The point failure pattern was created by randomly choosing 50 circular and non-overlapping regions from within the input domain. Suppose each of the 50 point failure regions is denoted as P_i, where $i = 1, 2, 3, \ldots, 50$, and

$|D|$ denotes the size of the input domain. We defined the size of each P_i as $|P_i| = (\rho_i / \sum_j^{50} \rho_j) \cdot \theta \cdot |D|$, where ρ_i is randomly chosen from [0, 1) based on a uniform distribution. We avoided points that were close to the corners of the input domain to prevent excessively wide strips. Similar to previous ART studies, we set the failure rates (θ) at 0.01, 0.005, 0.002 and 0.001, for each failure pattern.

When applying a testing method to generate test inputs in the simulations, if a generated input fell inside the failure region, then the testing method was considered to have detected a failure. The test input generation process was repeated until a failure was detected.

5) *Effectiveness and Efficiency Measure Criteria*

Chen and Yu [40] refer to elements of an input domain that do not produce correct outputs as failure-causing inputs. The failure rate (θ) is obtained by dividing the number of failure-causing inputs by the total number of inputs in the input domain.

The F-measure [18] is defined as the (expected) number of test cases executed before detecting the first failure. The failure detection effectiveness of a testing strategy can be reflected by the F-measure because lower F-measure means the testing strategy is more effective, as fewer test cases are needed to detect the first failure. In practice, a test process may be terminated whenever a failure is detected and resumed only after the detected fault is fixed. Hence, the F-measure is also realistic from a practical point of view. For an input domain with size $|D|$ and number of failure-causing inputs represented by m; the failure rate (θ) is calculated as $m/|D|$. The F-measure value for random test case selection (with replacement) is equal to $1/\theta$, or equivalently $|D|/m$. We also adopt the ART F-ratio (F_{ART}/F_{RT}), which is the ratio of ART's F-measure (F_{ART}) to RT's F-measure (F_{RT}), to compare ART's and RT's failure-finding performance. For example, if RT executes 100 test cases before detecting the first failure, and an ART method executes 20 before detecting a failure, then the ART method requires 20/100=0.2=20% of RT's test cases to detect the first failure. The F-ratio is computed as:

$$\%F - ratio = \frac{F_{ART}}{F_{RT}} \times 100 \tag{5}$$

Smaller F-ratio values for an ART method indicate better (faster) failure detection effectiveness.

We also used the E-measure (Em) to evaluate the failure detection effectiveness of our method. The E-measure is the (expected) total number of distinct failures detected by a specific number of generated test cases. A testing approach is considered more to be more effective in detecting failure if it has a lower F-measure, a lower F-ratio, and a higher E-measure.

To examine the significance of the performance differences between ART-ORB and other ART methods, we computed both *p-value* (probability value) and *effect size* (at the 5% significance level)[41] for the E-measure results. The p-value determines whether the difference between two ART methods is statistically significant. To measure the p-value, we used the unpaired two-tailed Mann-Whitney-Wilcoxon test [41]. A p-value less than 0.05 means that there is a significant difference between the two methods being compared. The effect size (ES) measure indicates the probability of one method being better than another. To measure the ES, we used the non-parametric Vargha and Delaney effect size measure [42]. An ES value for any two methods X and Y indicates the probability that X is better than Y. In this study, we used R language [43] to obtain the p-value and ES value for pairs of ART methods.

We employed two efficiency metrics to compare the time costs of ART-ORB with other ART methods. These metrics are: *Fm-time* (the time required to detect the first failure); and *Execution time* (the time required to execute a specific number of test cases). The efficiency of a testing approach is more intuitively reflected by these measures as a lower Fm-time indicates that less time is required to detect the first failure, and a lower execution time indicates that less time is required to execute a set of test cases.

6) *Experimental Parameters*

We conducted a comparative analysis of our ART algorithm against RT, ART-RP, ART-B, ART-DC, and ART-TPP.

ART-DC has two different implementations that achieve similar results: RRT-DC and FSCS-DC. In this study, we applied the RRT-DC version, and refer to it as ART-DC. We set the threshold (λ) of ART-DC to 50: Higher thresholds (such as λ =100) may provide better failure detection effectiveness, but also increase the computational overheads; lower threshold (such as λ =4 or λ =10), on the other hand, increase the chance of pure random generation of test cases, thus defeating the goal of even spreading [27]. In ART-TPP, as in previous work [28], we set the candidate set size, *k*, to 3.

In the simulations, as in previous studies [14], [44], we varied the target exclusion ratios for both ART-DC and ART-ORB between 1% and 150%, and used the best result each time. We extended the target exclusion ratio range up to 220% for the experiment with fault-seeded programs. The F-measure results in the simulations were averaged over 5000 runs, while the results from the experiments with real programs were averaged over 3000 runs—this was due to the significantly longer amount of time required for the real program execution and they are also consistent with previous studies [12], [14], [35]. For the E-measure, Fm-time, and Execution time results, we repeated both simulations and experiments 200 times. The choice of this repeated run was due to the time constraints in executing some of the subject programs. For example, the average time taken by ART-RP to execute each run of 4000 test cases for the *calGCD* program was as much as 19 hours. Additionally, we used as many as 16 subject programs with varying input dimensions in the experiments. Thus, the choice of 200 repetitions strikes a balance between generalization and statistical analysis [41].

B. *Simulations*

A major advantage of the use of simulations to evaluate a testing method is that they can provide a more complete picture of the performance under various scenarios. We conducted a series of simulations to address the research questions defined in Section IV-A-1. This section presents and discusses these simulations' results.

1) *Test Input Distribution of ART-ORB*

We conducted a series of simulations, in input domains of 1D to 4D, to investigate both the Discrepancy and Dispersion of ART-ORB compared to RT and other ART methods. To avoid bias, the exclusion ratio for both ART-ORB and ART-DC was set to 75%. In the simulations, we generated 100, 1000, and 10000 sets of test inputs for each testing method and input domain, and calculated the Discrepancy and Dispersion values using the formulas (3) and (4) from Section IV-A-1. Table II presents the simulation result, with the best values (the lowest values) highlighted.

It can be observed from Table II that the Discrepancy values of ART-ORB are lower than RT for all cases, and that they generally increase with the increasing dimensions of the input domain. Although the other ART methods generally have lower Discrepancies values than RT, they are not all lower in all cases. For example, the other ART methods had higher Discrepancy results than RT for 1000 tests in 4D input domains. ART-ORB has lower Discrepancy values than all other testing methods, both RT and ART, especially for the 1D and 2D input domains.

As expected, Table II also shows that ART-ORB performs better in terms of the Dispersion metric than RT, and again, increasing as the dimension increases. Compared with the other ART methods, ART-ORB usually has lower, or among the lowest, Dispersion values. The only time when RT had a better Dispersion result than ART-ORB was for 100 tests in the 3D input domain—in this case, the dispersions of the other ART methods were also higher than that of RT. Table II also shows that the Dispersion results for all testing methods increases as the dimension increases. When the dimension of input domain is low, ART methods evenly spread test cases and therefore have lower dispersion values. With the increase of dimension, the test cases they select show a certain degree of uneven distribution resulting in larger dispersion values. Hence, the reason for their higher dispersion as dimension increases. Surprisingly, this phenomenon is observed for RT. Our investigations show similar observations in other studies [37], [38].

TABLE II
DISCREPANCY AND DISPERSION RESULTS FOR EACH METHOD AND DIMENSION FOR DIFFERENT NUMBERS OF TEST INPUTS

Number of test inputs	Testing Strategy	Discrepancy				Dispersion			
		1D	2D	3D	4D	1D	2D	3D	4D
100	RT	0.064924	0.110079	0.080605	0.076196	0.024296	0.146948	**0.232739**	0.411993
	ART-RP	0.038009	0.036375	0.04085	0.049667	0.017792	0.12096	0.265197	0.382913
	ART-B	0.031748	0.078038	0.059664	**0.021465**	0.020372	**0.118384**	0.252942	0.353152
	ART-DC	0.034188	0.034128	**0.035375**	0.036382	0.015507	0.123422	0.300325	0.401826
	ART-TPP	0.030378	0.032806	0.041056	0.042993	0.018208	0.132635	0.293382	**0.339824**
	ART-ORB	**0.029091**	**0.023167**	0.041947	0.043958	**0.014654**	0.136443	0.268746	0.404833
1000	RT	0.021063	0.016977	0.015908	0.016411	0.00281	0.049948	0.142325	0.243003
	ART-RP	0.007591	0.007574	0.019485	0.018483	0.001947	**0.042786**	0.137002	0.229629
	ART-B	**0.00444**	0.01549	0.0216	0.021132	0.002464	0.045012	0.145932	0.237007
	ART-DC	0.019588	**0.004642**	0.017609	0.020325	0.002883	0.047424	0.151221	**0.221336**
	ART-TPP	0.009262	0.010459	**0.006511**	0.021547	0.002335	0.050908	0.139035	0.269923
	ART-ORB	0.008654	0.009616	0.006662	**0.012181**	**0.00148**	0.043875	**0.132397**	0.226125
10000	RT	0.002889	0.005012	0.005939	0.008203	0.000489	0.020114	0.102807	0.149492
	ART-RP	0.002812	0.003826	0.007169	0.005706	0.000195	0.016112	0.0702	0.143263
	ART-B	0.002	0.005408	0.004584	0.010245	0.000271	0.015831	0.07305	**0.133702**
	ART-DC	0.003189	0.00405	**0.002133**	**0.001015**	0.000314	0.017366	0.066335	0.135845
	ART-TPP	0.002094	0.004671	0.006738	0.013356	0.000286	0.017208	0.066119	0.141453
	ART-ORB	**0.000917**	**0.001114**	0.003977	0.006171	**0.000161**	**0.014128**	**0.058487**	0.135359

TABLE III
F-MEASURE AND F-RATIO COMPARISONS FOR ART BY ORTHOGONAL RECURSIVE BISECTION FOR DIFFERENT FAILURE PATTERNS

Failure Rate (θ)	Expected F-measure (F_{RT})	Block Pattern		Strip Pattern		Point Pattern	
		Mean F-measure of ART (F_{ART})	(F_{ART}/F_{RT}) (%)	Mean F-measure of ART (F_{ART})	(F_{ART}/F_{RT}) (%)	Mean F-measure of ART (F_{ART})	(F_{ART}/F_{RT}) (%)
0.01	100	69.5	**69.5%**	89.0	**89.0%**	95.6	**95.6%**
0.005	200	138.9	**69.5%**	182.2	**91.1%**	192.4	**96.2%**
0.002	500	351.1	**70.2%**	468.0	**93.6%**	481.9	**96.4%**
0.001	1000	701.5	**70.2%**	943.6	**94.4%**	965.3	**96.5%**

2) *Failure Detection Effectiveness*

We performed simulations using three failure pattern types (block, strip, and point), with different failure rates, averaging results over 5000 executions. These simulations were categorized into three main parts.

Firstly, we compared the F-measure performances for ART-ORB with the expected F-measure values for RT, for the three failure patterns. For each test run, we calculated the average F-measure value, and the F-ratio for each failure rate and failure pattern. Table III presents these results.

Table III shows that ART-ORB has a best improvement of 30.5% over RT for the block failure pattern, and 11% for the strip pattern. As expected, ART-ORB's spreading of test cases evenly over the input domain did not result in a significant improvement in failure detection for the point pattern, with the best improvement of only 4.4%.

Secondly, we compared the ART-ORB F-ratio performance to those of the other ART methods considered in this study, again using the block, strip, and point failure patterns. Table IV shows these results.

Table IV shows that, for non-point patterns, the ART failure detection performance generally increases as the failure rate increases. An increase in failure rate increases the probability that a failure-revealing test will be selected as the next input; therefore the increase in performances of the ART methods. As a result, the lowest F-ratio results were obtained for the highest failure rate (0.01) for most non-point failure patterns.

Table IV also shows that ART-ORB slightly outperforms the other ART methods for all failure rates, when the block failure pattern is used. The F-ratio results obtained for strip failure pattern also showed a slightly better performance for ART-ORB for half of the failure rates, and a comparable performance for the others (0.002 and 0.001). As expected, none of the ART methods showed much improvement over RT for the point failure pattern—some even had a worse performance. Although ART-ORB had a better performance than the other ART methods for point failure pattern, it also had a less significant improvement over RT, with the maximum improvement being 4.4%.

TABLE IV
F-RATIO RESULTS OF ART METHODS FOR THE BLOCK, STRIP, AND POINT FAILURE PATTERNS AVERAGED OVER 5000 TEST RUNS

Failure Rate (θ)	**(%) F_{ART}/F_{RT}**				
	ART-RP	**ART-B**	**ART-DC (λ=50)**	**ART-TPP**	**ART-ORB**
Block pattern					
0.01	76.0%	75.1%	78.7%	79.5%	**69.5%**
0.005	77.5%	73.8%	79.0%	77.5%	**69.5%**
0.002	80.9%	73.1%	79.9%	76.2%	**70.2%**
0.001	80.1%	73.8%	80.1%	76.6%	**70.2%**
Strip pattern					
0.01	92.0%	91.7%	90.1%	94.3%	**89.0%**
0.005	93.3%	94.6%	92.2%	95.0%	**91.1%**
0.002	95.0%	93.7%	**93.4%**	95.7%	93.6%
0.001	**94.2%**	95.5%	95.2%	96.6%	94.4%
Point pattern					
0.01	102.9%	98.8%	96.8%	100.8%	**95.6%**
0.005	103.9%	100.0%	98.3%	102.1%	**96.2%**
0.002	100.0%	100.6%	97.0%	98.6%	**96.4%**
0.001	100.3%	99.7%	97.8%	100.0%	**96.5%**

In the third set of simulations, although the proposed method showed relatively better failure finding effectiveness (F-measure) than the other previous ART methods, for almost all failure rates, we further investigated this effectiveness for a fixed number of test cases (E-measure).

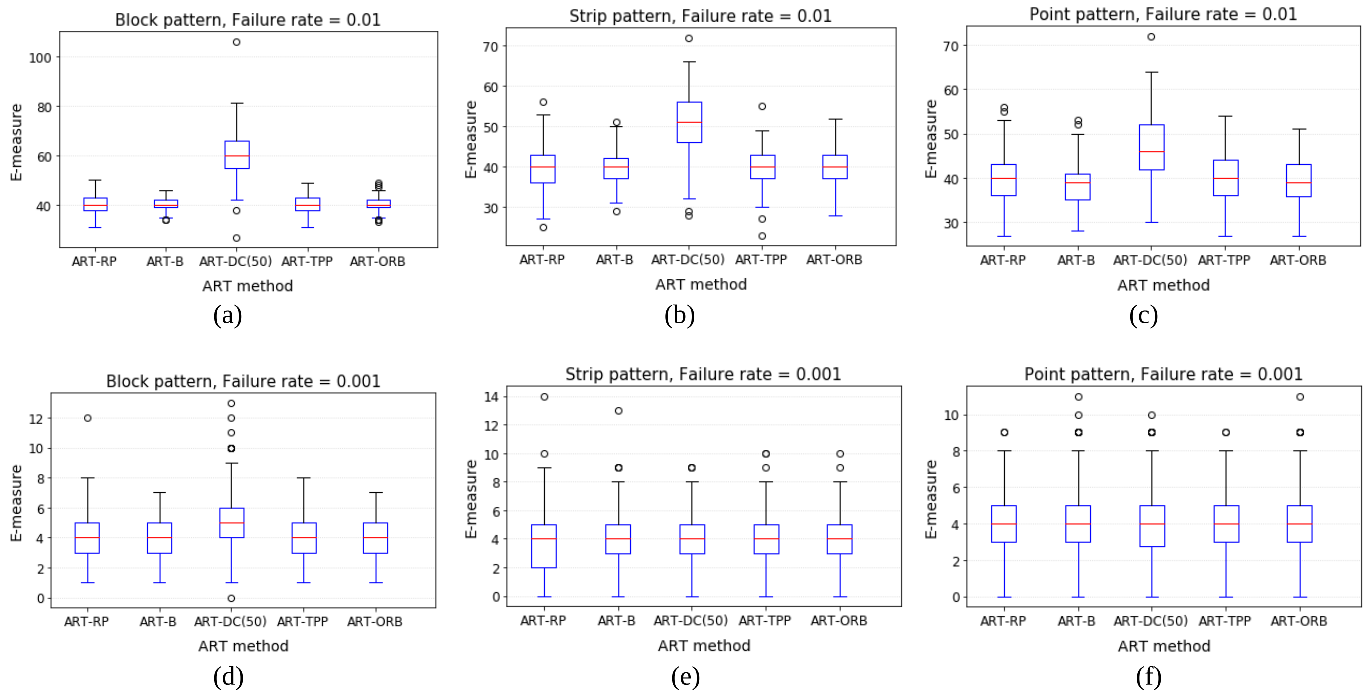


Fig. 5. E-measure comparisons of ART methods using the block, strip, and point failure patterns, for failure rates (θ) of 0.01 and 0.001.

We performed simulations for each failure pattern using failure rates of 0.01 and 0.001. For each E-measure simulation, we generated a fixed test set (n) of 4000 test cases and averaged the results over 200 runs. The results are presented in Fig. 5.

Fig. 5 indicates that the ART-ORB E-measure performance is comparable to that most of the other ART methods, for all failure patterns. An exception is ART-DC, which has better performance, especially for higher failure rates. This is due to the relatively large threshold (λ=50) considered in this study. Larger thresholds increase the probability of fault detection; however, larger thresholds come with higher computational costs.

3) Failure Detection and Computational Efficiency

The time required for a testing method to detect failure can be a good determinant of its efficiency. Generally, testing proceeds by repeatedly generating test cases until a failure is detected. Therefore, the time required to detect the first failure (Fm-time) may be a good indicator of a method's failure-detecting efficiency. To evaluate this for our proposed method, we compared ART-ORB with the other ART methods in terms of the Fm-time. Fig. 6 shows the Fm-time results for all failure patterns, using failure rates of 0.01 and 0.001, averaged over 200 results. As the results show, ART-ORB requires far less time to detect the first failure, outperforming the other methods for all failure patterns and failure rates, especially for the lower failure rates.

The failure-detection speed alone does not provide a complete representation of the efficiency of a testing method. The average time required to execute a fixed number of test cases (Execution time) is also indicative of the method's efficiency. We therefore further investigated ART-ORB by comparing its execution time with those of the other ART methods. For this, we generated 4000 test cases, repeating the simulation 200 times. Fig. 7 presents boxplots that summarize the execution time results for each ART method, for all failure patterns, using failure rates of 0.01, 0.005, 0.002, and 0.001.

For all failure patterns, ART-ORB provides the lowest (best) execution time. It can also be seen that ART-DC again has the worst execution time.

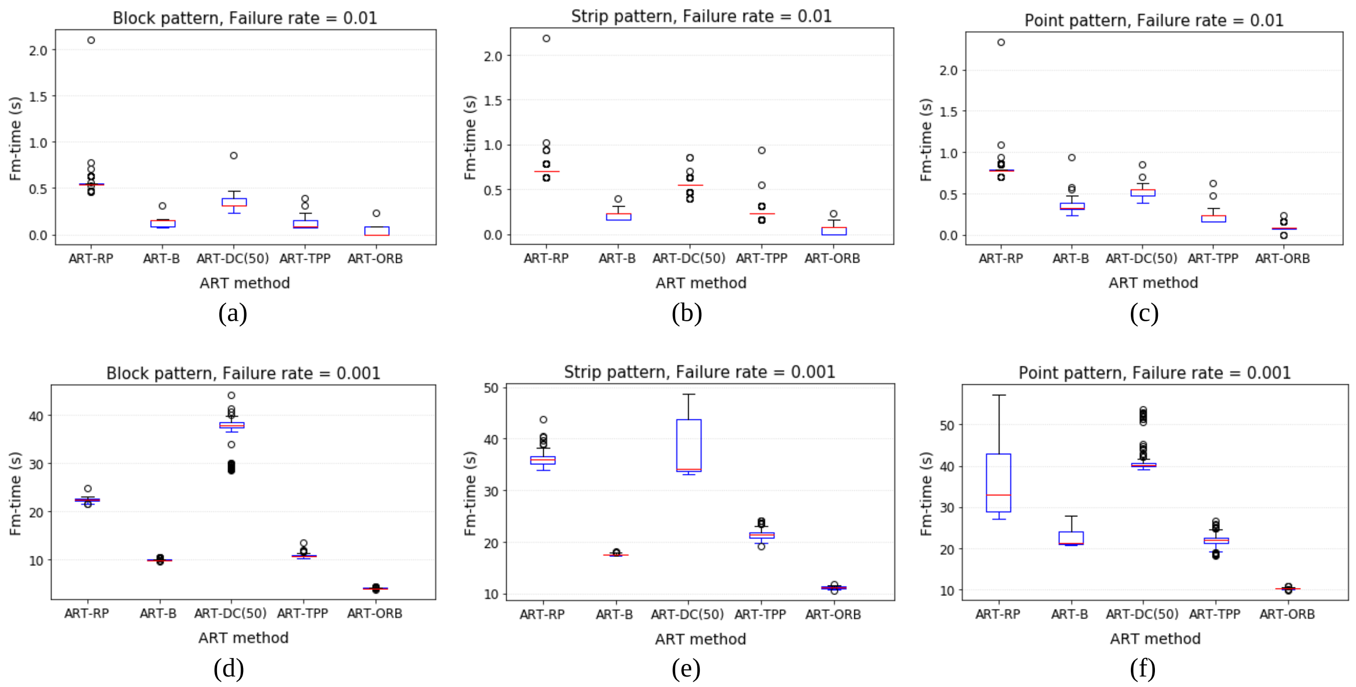


Fig. 6. Fm-time comparison of ART methods using the block, strip, and point failure patterns, for failure rates (θ) of 0.01 and 0.001.

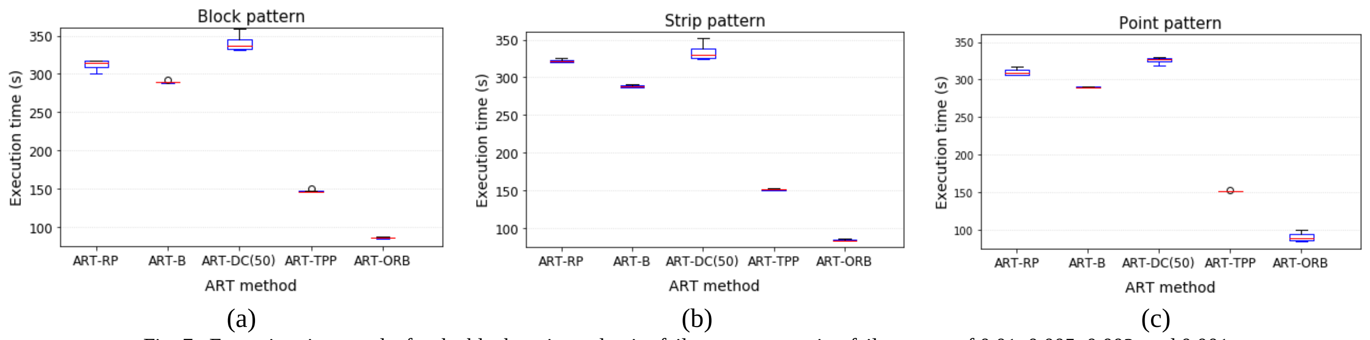


Fig. 7. Execution time results for the block, strip, and point failure patterns, using failure rates of 0.01, 0.005, 0.002, and 0.001.

C. *Experiments with Fault-seeded Programs*

The results from the simulations indicate that our proposed method outperforms RT and compares well with ART-RP, ART-B, ART-DC, and ART-TPP, in terms of failure detection effectiveness. The simulation results also show that the time taken by ART-ORB is less than the other ART methods. In order to further validate these results, we performed several similar experiments with a number of real-life benchmark programs.

The experiments involved 16 fault-seeded programs that were implemented in Java, with varying dimensions and input domains. Table V gives details about these programs, including their dimensions, inputs types, input domains, the number of each fault type used, and the failure rates.

The first 12 programs are published, fault-seeded programs [45], [46], that are commonly used in ART research [12], [14], [35]. They involve numerical calculations, and range in length from 30 to 200 lines of code. They have varying dimensions, and some have varying program input types. We converted these 12 published programs that were originally written in C and C++, to Java.

TABLE V
SUBJECT PROGRAMS ORDERED BY DIMENSION

Program Name	Dimension (d)	Input Type	Input domain		Types of Faults						Failure Rate
			From	To	SDL	RSR	AOR	CR	SVR	ROR	
airy	1	double	(−5000.0)	5000.0				1			0.000716
erfcc	1	double	(−30000.0)	30000.0			1	1	1	1	0.000574
probks	1	double	(−50000.0)	50000.0			1	1	1	1	0.000387
bessj0	1	double	(−300000.0)	300000.0			2	1	1	1	0.001373
tanh	1	double	(−500.0)	500.0			1	1	1	1	0.001817
bessj	2	int, double	(2.0,−1000.0)	(300.0, 15000.0)			2	1		1	0.001298
gammq	2	double, double	(0.0, 0.0)	(1700.0, 40.0)				1		3	0.000830
sncndn	2	double, double	(−5000.0, −5000.0)	(5000.0, 5000.0)				1	4		0.001623
golden	3	double, double, double	(−100.0, −100.0, −100.0)	(60.0, 60.0, 60.0)				1	1	3	0.000550
plgndr	3	int, int, double	(10.0, 0.0, 0.0)	(500.0, 11.0, 1.0)			1	2		2	0.000368
cel	4	double, double, double, double	(0.001, 0.001, 0.001, 0.001)	(1.0, 300.0, 10000.0, 1000.0)			1	1		1	0.000332
el2	4	double, double, double, double	(0.0, 0.0, 0.0, 0.0)	(250.0, 250.0, 250.0, 250.0)			1	3	2	3	0.000690
calDay	5	int, int, int, int, int	(1, 1, 1, 1, 1800)	(12, 31, 12, 31, 2200)	1						0.000632
triangle	6	int, int, int, int, int, int	(-25, -25, -25, -25, -25, -25)	(25, 25, 25, 25, 25, 25)				1			0.000713
line	8	int, int, int, int, int, int, int, int	(-10, -10, -10, -10, -10, -10, -10, -10)	(10, 10, 10, 10, 10, 10, 10, 10)						1	0.000303
calGCD	10	int, int, int, int, int, int, int, int, int, int	(1, 1, 1, 1, 1, 1, 1, 1, 1, 1)	(1000, 1000, 1000, 1000, 1000, 1000, 1000, 1000, 1000, 1000)			1				0.000984

The remaining four subject programs all have higher dimensionality, were downloaded from other sources [47], [48], and were implemented in Java. All four have integer-only input parameters (input domains). The *calDay* and *line* programs were obtained from Ferrer *et al.* [47]. The *calDay* test program computes the days of the week, while the *line* program checks if two rectangles overlap. The *triangle* program is a classification program (for acute-, obtuse-, and right-angled triangles) whose implementation is based on a programming exercise [48]. The *calGCD* program computes the greatest common divisor of ten integer values.

All 16 programs had faults seeded in using different types of common mutant operations [49], including: statement deletion (SDL), return statement replacement (RSR), arithmetic operator replacement (AOR), constant replacement (CR), scalar variable replacement (SVR), and relational operator replacement (ROR). The failure rates of the subject programs range approximately from 0.0003 to 0.002. The failure rate for the *calGCD* program was not documented in the literature; therefore we estimated it by performing random testing with a large number of test inputs (1,000,000,000).

TABLE VI
FAILURE PATTERNS OF THE 16 SUBJECT PROGRAMS

Subject Program	**Failure Pattern Characteristics**
airy	A block in the center of the input domain
erfcc	A block in the center of the input domain
probks	A block in the center of the input domain
bessj0	A block in the center of the input domain
tanh	A block in the center of the input domain
bessj	Strip
gammq	A long narrow strip
sncndn	Points scattered over the entire input domain
golden	Points scattered around a very large hyperplane
plgndr	Strips near the edge of the input domain
cel	One failure region (strip) along the entire edge of the input domain
el2	Strips near the edges of the input domain
calDay	A combination of strips and points, but with the strips only in some dimensions.
triangle	Points scattered over the entire input domain
line	Point-like patterns scattered over different regions of the input domain
calGCD	Points scattered over the entire input domain

Previous researches [50], [51], and analysis reported the failure pattern types of the first 12 subject program. The failure patterns of the remaining four programs were not available in the literature. We therefore identified their failure patterns by executing RT and recording a large numbers of generated test inputs that cause failure in the programs. Plotting all dimension may produce an output that is complicated and may be impossible to properly analyze the failure patterns, hence, we chose to plot their values in 3D and at different angles (see figures in Appendix). Although their plots were in parts (three dimensions at a time), the results provided a good estimate of their failure patterns. The characteristics of the failure patterns in the 16 programs are described in the Table VI.

In the experiments, all subject programs were tested by each testing method, with failures being recorded as detected when the output for the faulty program (the mutant) and the original (correct) version were different.

1) *Failure Detection Effectiveness*

Similar to the simulations, we performed experiments to determine the improvement of ART-ORB's failure detection ability over RT's. We applied both methods to test each subject program, recording their average F-measure from 3000 repetitions, and calculating the F-ratio values. These results are shown in Table VII. The table also includes the exclusion ratios *R* used by ART-ORB which provided the best results for each subject program.

The results in Table VII clearly show that ART-ORB has better failure detection effectiveness than RT for all subject programs. For *airy*, *erfcc*, *probks*, *bessj0*, and *tanh*, ART-ORB strongly outperformed RT, with improvements ranging between 42% and 45%. For the *bessj*, *gammq*, *plgndr*, *el2*, and calDay, ART-ORB had between 7.8% and 38.2% improvement over RT. There was only small improvement over RT for *sncndn*, *golden*, *cel*, *triangle*, *line*, and *calGCD* (between 1.2% and 5.3%). We observed that ART-ORB generally provided better results for high exclusion ratios.

TABLE VII
F-MEASURE AND F-RATIO RESULTS FOR ART-ORB AND RT FOR EACH SUBJECT PROGRAM, AND BEST R FOR ART-ORB

Subject Program	**Mean F-measure of RT (F_{RT})**	**Mean F-measure of ART-ORB (F_{ART})**	**F-ratio of ART-ORB (FART/FRT) (%)**	**Best *R* for ART-ORB (%)**
airy	1504.4	827.6	**55.0%**	50%
erfcc	1905.7	1073.1	**56.3%**	80%
probks	2567.5	1490.0	**58.0%**	70%
bessj0	781.3	445.5	**57.0%**	60%
tanh	572.9	322.4	**56.3%**	80%
bessj	755.5	467.1	**61.8%**	150%
gammq	1210.8	1082.4	**89.4%**	110%
sncndn	624.3	609.9	**97.7%**	120%
golden	1834.8	1761.3	**96.0%**	100%
plgndr	2767.7	1910.0	**69.0%**	140%
cel	3123.2	3086.7	**98.8%**	140%
el2	1438.7	1101.7	**76.6%**	220%
calDay	86.3	79.6	**92.2%**	140%
triangle	1418.3	1368.0	**96.5%**	220%
line	3252.8	3081.8	**94.7%**	170%
calGCD	1047.3	991.4	**94.7%**	190%

To further validate the simulation results obtained from the comparison of ART-ORB with other ART methods, we also applied the other ART methods to the subject programs, calculating their F-measure and F-ratio values. Fig. 8 presents the F-ratio comparison of ART-ORB to the other ART methods for each subject program.
The results in Fig. 8 show that ART-ORB consistently uses amongst the fewest test cases to detect the first failure: It generally outperforms the other ART methods for 10 of the 16 fault-seeded programs, and has comparable performance for the other six. Although ART-ORB showed less significant improvement over RT for *sncndn*, *golden*, *cel*, *triangle*, *line*, and *calGCD*, its performances was similar to the other ART methods in the study.

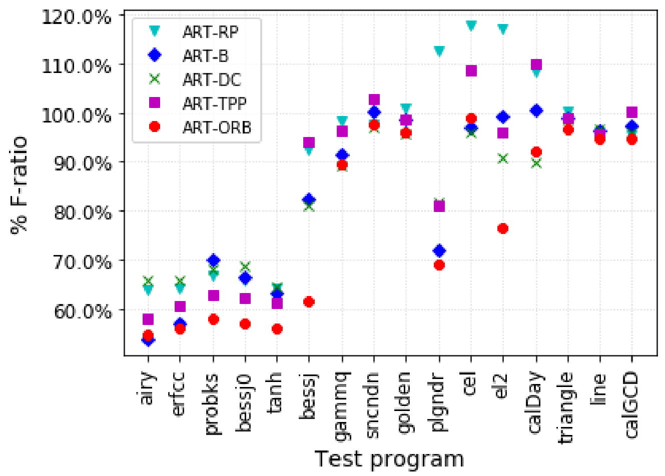
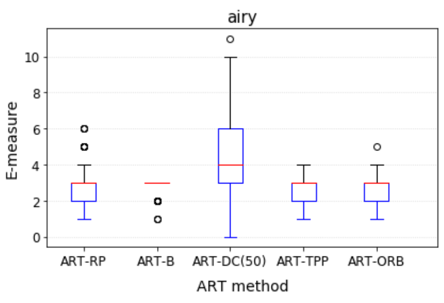


Fig. 8. F-ratio comparison for the ART methods.

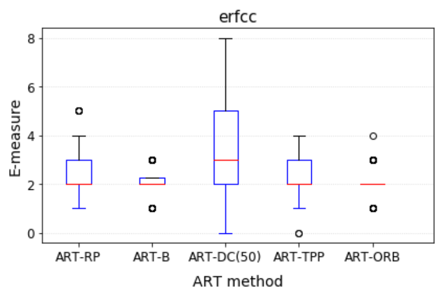
We also compared ART-ORB with the other ART methods in terms of the E-measure, using a test set of 4000, and averaging the result over 200 repetitions. Fig. 9 shows these results.

The E-measure results in Fig. 9 show that ART-ORB has comparable performance to the other ART methods. However, the ART-ORB performance varies in magnitude for each subject program. The results also show that ART-ORB E-measure performance improves slightly as higher dimensional programs are used. ART-DC has better E-measure performance for all the 1D programs, but its performance becomes comparable to the other ART methods in the higher dimensional programs.

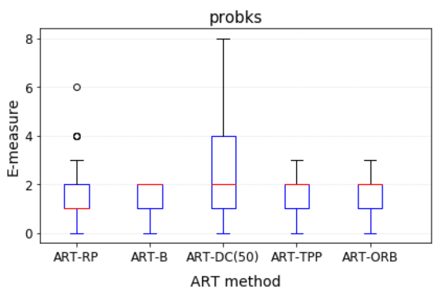
In order to further analyze the significance of the differences in E-measure, we calculated the p-value and effect size (ES) [41] for pair-wise comparisons between the individual ART methods. We generated 800 E-measure results for each subject program using test sets of size 1000, 2000, 3000, and 4000. The results are presented in Table VIII, where the column labeled "Preferred" identifies which ART method of the pair, based on ES, has better performance.
The results for most subject programs show that there is no significant E-measure difference between ART-ORB and ART-RP, ART-B, or ART-TPP (the p-value is greater than 0.05). However, there is a significant E-measure difference between ART-ORB and ART-DC for most programs (the p-value is less than 0.05). The E-measure ES values indicate



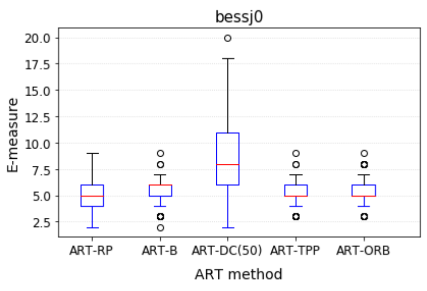
(a)



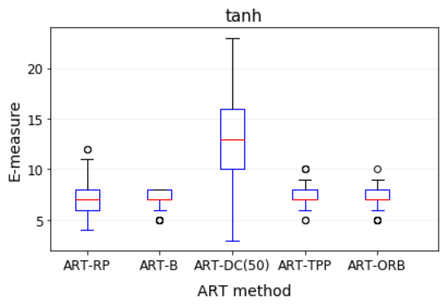
(b)



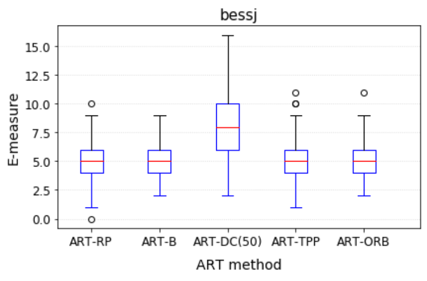
(c)



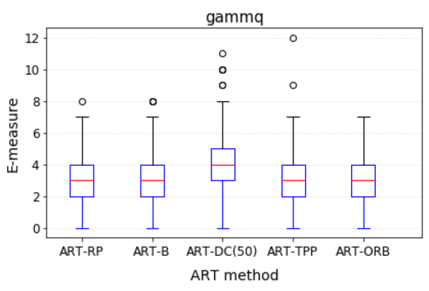
(d)



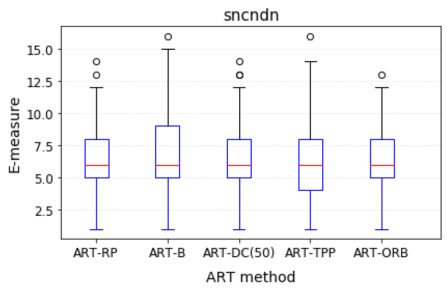
(e)



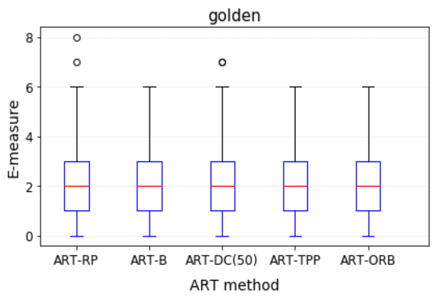
(f)



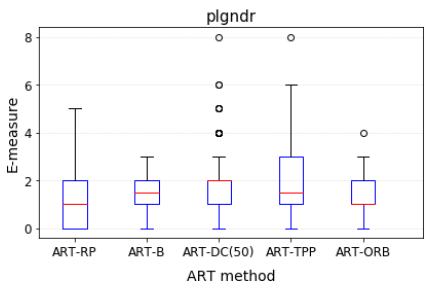
(g)



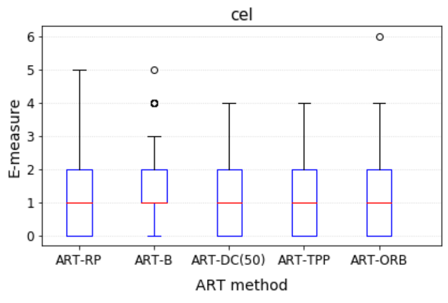
(h)



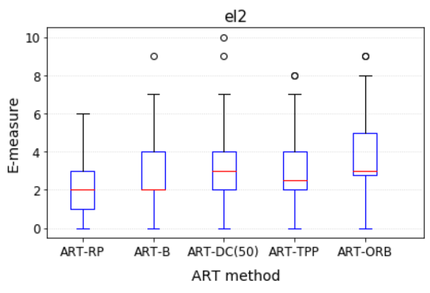
(i)



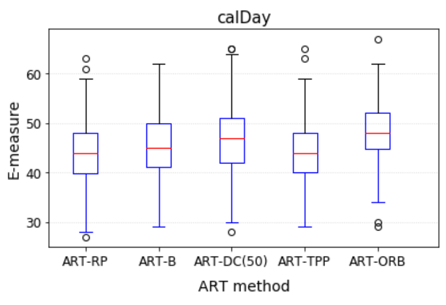
(j)



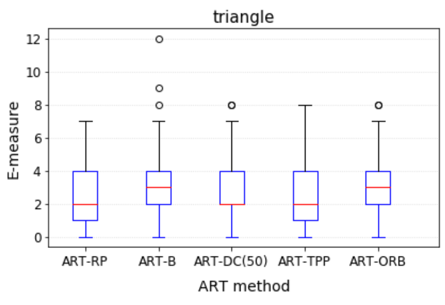
(k)



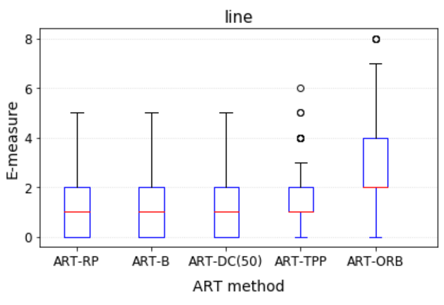
(l)



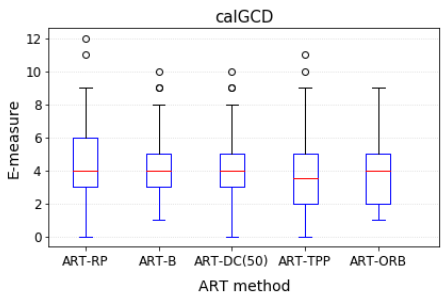
(m)



(n)



(o)



(p)

Fig. 9. E-measure comparison of ART methods for each subject program using a set of 4000 tests.

that ART-ORB is generally better than ART-RP in terms of probability of detecting more failures (the ES values for most programs is greater than 0.5, with few programs having ES values close to 0.5). The comparison of ES values between ART-ORB and either ART-B or ART-TPP shows similar failure detection probabilities (there is a similar number of ES values above and below 0.5 for all programs). ART-DC has a higher probability of detecting failures than ART-ORB (the ES values are below 0.5 for most programs). However, for higher dimensional programs (*d*≥4), the failure detection performance of ART-DC becomes worse than ART-ORB, in most cases. The table also shows that ART-ORB is preferred to the other methods in high dimensional situations (it has better ES performance for almost all the programs with higher dimensional input space).

TABLE VIII

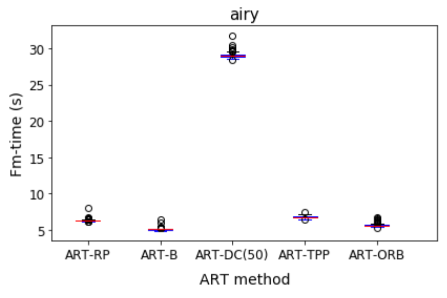
COMPARISON OF PAIRS OF ART METHODS FOR EACH SUBJECT PROGRAM USING P-VALUE AND EFFECT SIZE FOR 800 E-MEASURE VALUES

Subject Programs	**ART-ORB and ART-RP**			**ART-ORB and ART-B**			**ART-ORB and ART-DC**			**ART-ORB and ART-TPP**		
	P-value	**ES**	**Preferred**	**P-value**	**ES**	**Preferred**	**P-value**	**ES**	**Preferred**	**P-value**	**ES**	**Preferred**
airy	0.971619	0.500495	**ART-ORB**	0.531013	0.491352	**ART-B**	3.22E-20 *	0.369814	**ART-DC**	0.785059	0.496231	**ART-TPP**
erfcc	0.642482	0.506331	**ART-ORB**	0.416547	0.51098	**ART-ORB**	2.88E-09 *	0.417089	**ART-DC**	0.676117	0.505682	**ART-ORB**
probks	0.068810	0.524184	**ART-ORB**	0.488026	0.509203	**ART-ORB**	2.37E-07 *	0.429026	**ART-DC**	0.945897	0.499102	**ART-TPP**
bessj0	0.125268	0.521848	**ART-ORB**	0.933170	0.501195	**ART-ORB**	1.12E-24 *	0.353025	**ART-DC**	0.699894	0.505490	**ART-ORB**
tanh	0.644019	0.506617	**ART-ORB**	0.480267	0.489913	**ART-B**	6.71E-52 *	0.282084	**ART-DC**	0.637029	0.493245	**ART-TPP**
bessj	0.001900 *	0.544320	**ART-ORB**	0.986666	0.499761	**ART-B**	2.42E-17 *	0.378625	**ART-DC**	0.072443	0.525616	**ART-ORB**
gammq	0.626420	0.493118	**ART-RP**	0.184588	0.518748	**ART-ORB**	0.001376 *	0.454592	**ART-DC**	0.084432	0.524374	**ART-ORB**
sncndn	0.452202	0.489228	**ART-RP**	0.068236	0.473864	**ART-B**	0.292013	0.484895	**ART-DC**	0.494232	0.490203	**ART-TPP**
golden	0.353127	0.487076	**ART-RP**	0.083454	0.475838	**ART-B**	0.000974 *	0.453942	**ART-DC**	0.870945	0.497738	**ART-TPP**
plgndr	4.10E-09 *	0.579108	**ART-ORB**	0.931361	0.501166	**ART-ORB**	0.075820	0.475825	**ART-DC**	0.056164	0.473977	**ART-TPP**
cel	0.061232	0.524688	**ART-ORB**	0.27752	0.485499	**ART-B**	0.373224	0.511796	**ART-ORB**	0.463055	0.509727	**ART-ORB**
el2	1.56E-29 *	0.659459	**ART-ORB**	2.48E-21 *	0.634256	**ART-ORB**	1.65E-10 *	0.590543	**ART-ORB**	4.33E-18 *	0.622742	**ART-ORB**
calDay	8.91 E-05*	0.556563	**ART-ORB**	0.031725 *	0.531005	**ART-ORB**	0.209173	0.518129	**ART-ORB**	0.003829 *	0.541745	**ART-ORB**
triangle	0.157980	0.519853	**ART-ORB**	0.258723	0.515906	**ART-ORB**	0.178200	0.518957	**ART-ORB**	0.168949	0.519377	**ART-ORB**
line	8.93 E-40*	0.682969	**ART-ORB**	3.61 E-42*	0.688481	**ART-ORB**	7.43 E-43*	0.689995	**ART-ORB**	2.13 E-30*	0.659302	**ART-ORB**
calGCD	0.022253 *	0.467473	**ART-RP**	0.060388	0.473306	**ART-B**	0.255103	0.483816	**ART-DC**	0.418101	0.511512	**ART-ORB**

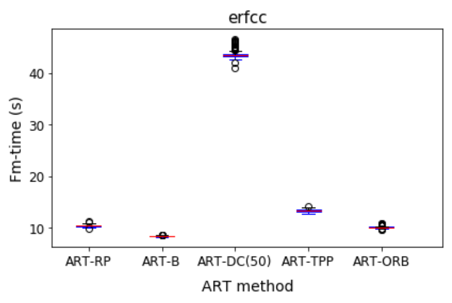
* *denotes statistically significant difference (p<0.05)*

2) *Failure Detection and Computational Efficiency*

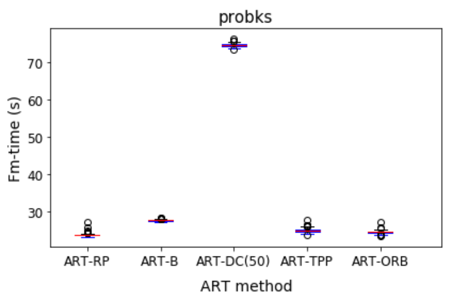
We evaluated the failure detection efficiency of ART-ORB by comparing the time taken detect the first failure (Fm-time) to those of other ART methods (as done in the simulation process). We tested each subject program and recorded the Fm-time averaged over 200 iterations. We performed the same experiment for all the ART methods and compared their Fm-time results: Fig. 10 presents the boxplot representations of the Fm-time results for each subject program.



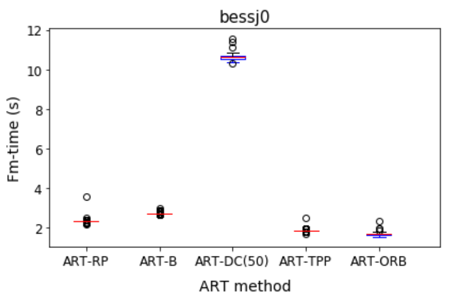
(a)



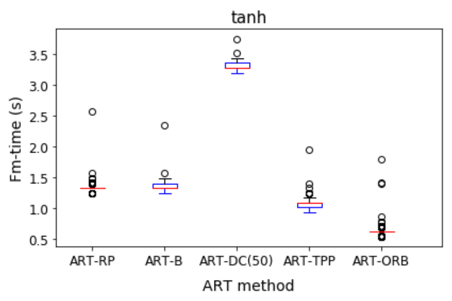
(b)



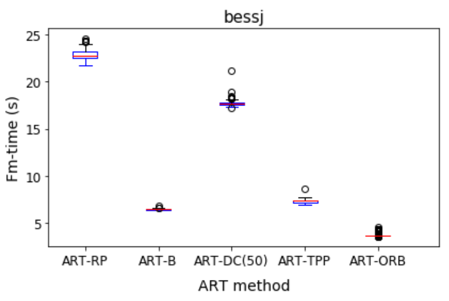
(c)



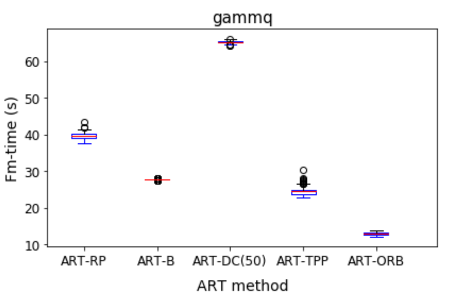
(d)



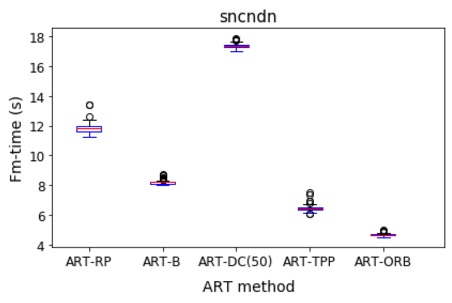
(e)



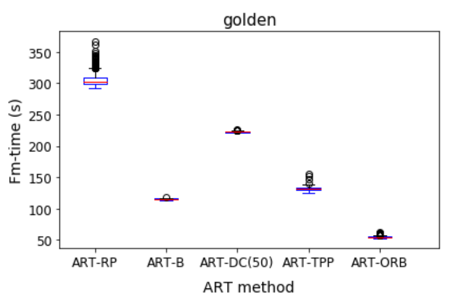
(f)



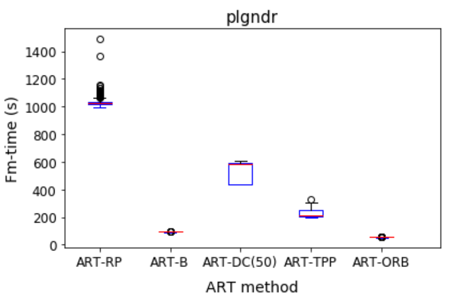
(g)



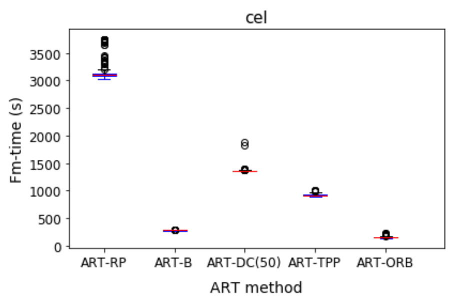
(h)



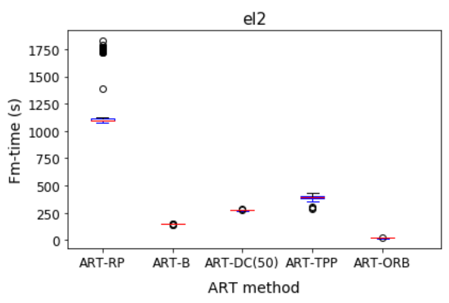
(i)



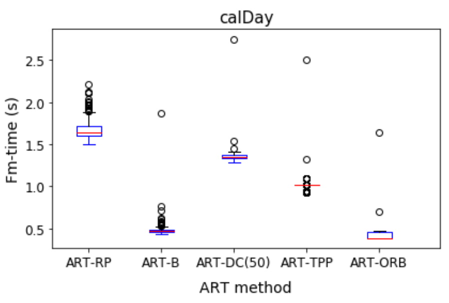
(j)



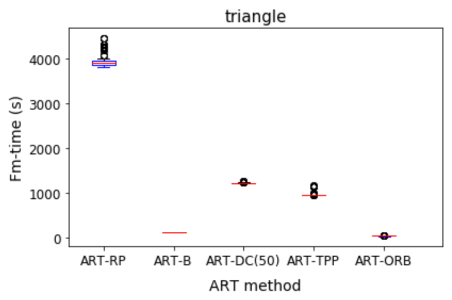
(k)



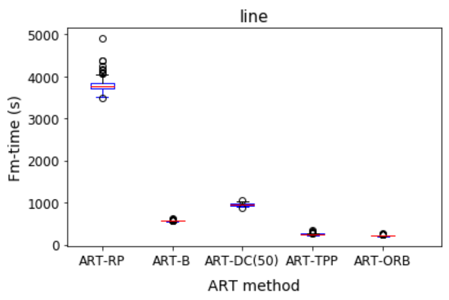
(l)



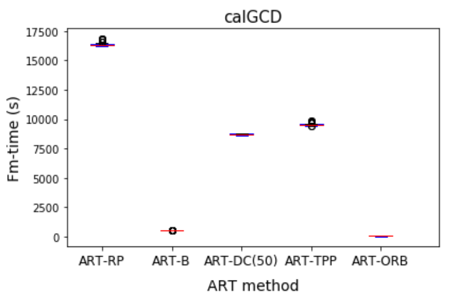
(m)



(n)



(o)



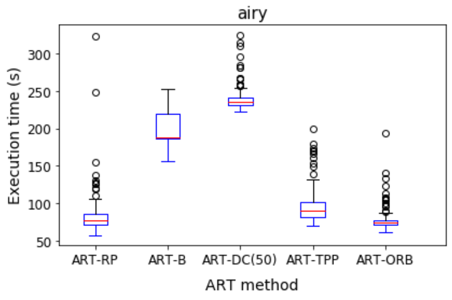
(p)

Fig. 10. Fm-time comparison of ART methods for each subject program.

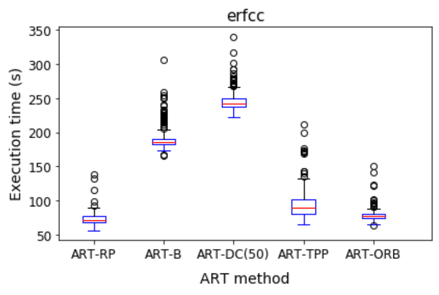
It can be seen from Fig. 10 that ART-ORB uses far less time to detect the first failure for almost all subject programs: ART-ORB provides better failure detection efficiency for 12 of the 16 real-life programs, and comparable time to other methods for four of the five 1D programs. The results also show that the Fm-time of ART-RP increases as higher dimensional programs are used. Conversely, the Fm-time results of ART-DC decreases with increasing program dimension. These results are in broad agreement with those obtained in the simulations (Fig. 6).

The time required to execute a number of test cases (Execution time) was also compared across the ART methods. We tested each subject program using each of the ART methods using a set of 4000 tests, repeating 200 times and averaging results. Fig. 11 presents this execution time comparison.

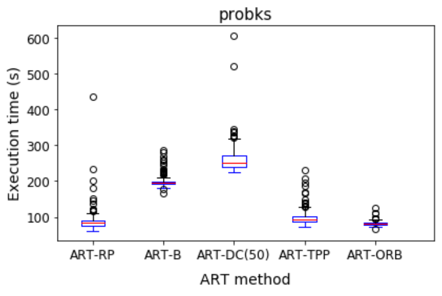
Fig. 11 shows that, generally, ART-ORB has better execution time than the other ART methods for almost all subject programs. Similar to the failure detection efficiency, ART-ORB's execution time is comparable to other ART methods for 1D subject programs, but is better in all other dimensions. A difference in efficiency can also be observed for varying dimensions of subject programs: When these subject programs are used, both ART-RP and ART-DC have similar characteristics as in the Fm-time experiment—that is, ART-DC has the worst execution time for lower dimensional programs while ART–RP has the worst for higher dimensional programs.



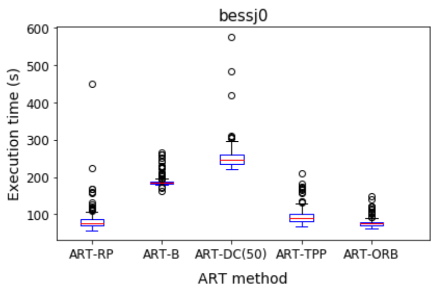
(a)



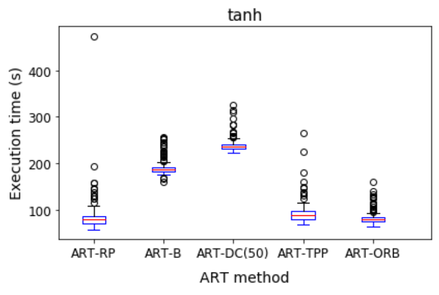
(b)



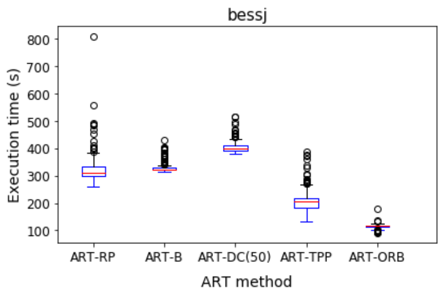
(c)



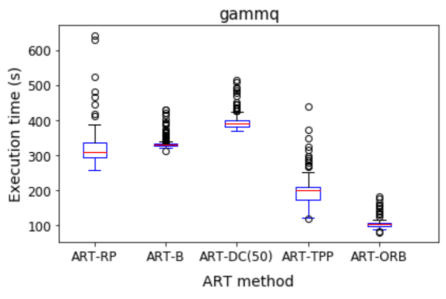
(d)



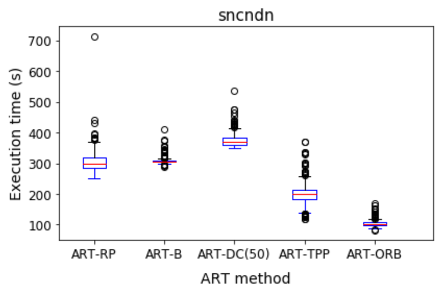
(e)



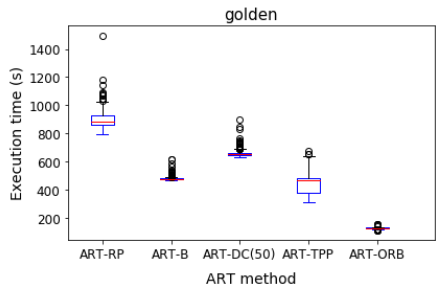
(f)



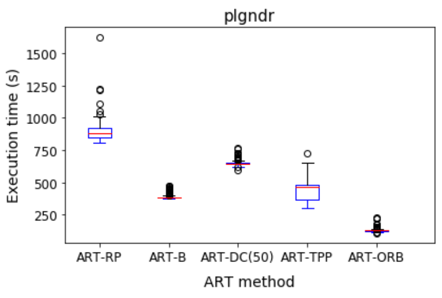
(g)



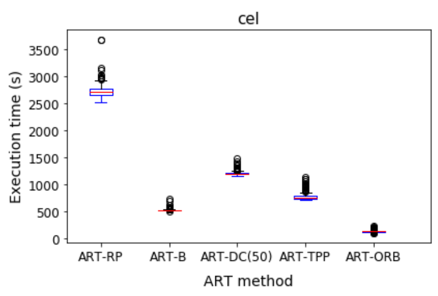
(h)



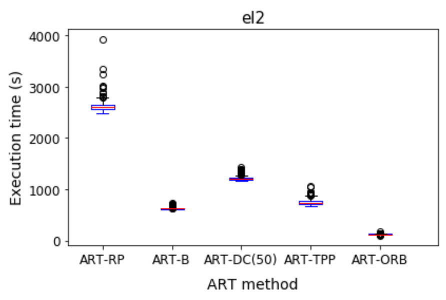
(i)



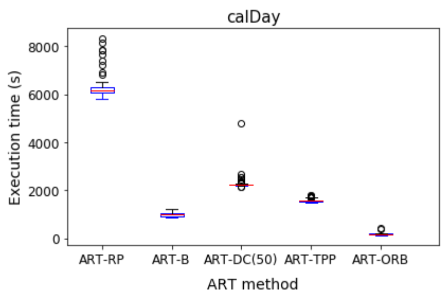
(j)



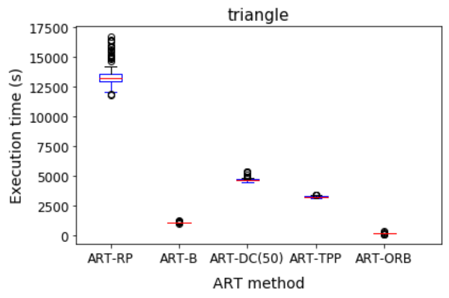
(k)



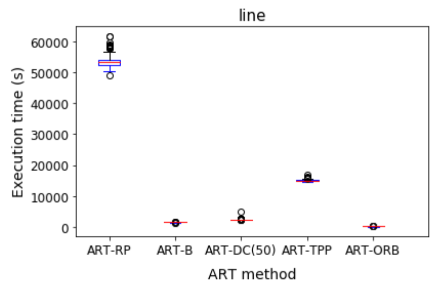
(l)



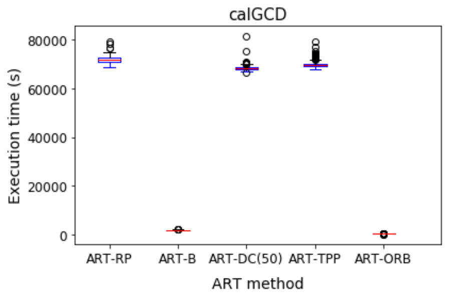
(m)



(n)



(o)



(p)

Fig. 11. Execution time comparison for the ART methods for each subject program using a set of 4000 tests.

V. Discussion

In this paper, we have presented a new ART method (ART-ORB) that dynamically spreads test cases by incrementally and orthogonally bisecting regions of the input domain. The method is based on the intuition that combining a partitioning strategy (with acceptable overheads) with an exclusion strategy should enable a more even spread of test inputs over the input domain. This is partly motivated through the review of ART methods that employ partitioning, as described in Section II. We have empirically evaluated the performance of the proposed ART method based on four research questions posed in this paper. This section now provides a discussion of the results obtained in response to the research questions.

RQ1: In our initial investigation, we used both Discrepancy and Dispersion metrics to evaluate the potential of ART-ORB to distribute test inputs evenly over the input domain.

The Discrepancy results show that ART-ORB can distribute test more evenly than RT, for all input domain dimensions. The Dispersion results also indicate that, compared with RT, smaller empty regions exist among tests generated by ART-ORB, for almost all input domain dimensions. This reflects a better even spreading of tests. For both metric, the ART-ORB results are comparable to other ART methods, with generally very small values in all cases. Thus, we can conclude that ART-ORB does provide an effective strategy for evenly distributing test inputs.

The more evenly spread distribution of test is achieved through ART-ORB's use of a restriction zones around a previous test within a region to enhance the diversity [52] of selected tests. Furthermore, the partitioning of a region through the midpoint of two generated tests makes it less likely for tests to be very close to the border of each region. This also limits how close test inputs within different regions of the input domain can be to each other.

RQ2: Previous studies [12], [53] have shown that for non-point failure patterns, evenly spread test cases have a higher probability of detecting the first failure faster. ART, which is based on this observation, therefore requires fewer test executions than RT to detect the first (for non-point failure patterns). This is consistent with our empirical results (Table III). ART-ORB was also shown to execute more tests to find the first failure in the case of point failure patterns than for non-point patterns. These results provide strong evidence that ART-ORB outperforms RT for all failure patterns and failure rates, most significantly outperforming when the failure pattern is of block type.

The results of the experiments with real programs (Table VII) are consistent with the simulation results (Table III). It has been suggested that ART-ORB can obtain better failure detection effectiveness for *airy*, *erfcc*, *probks*, *bessj0*, and *tanh* partly because their failure patterns are of block type. Similarly, since *bessj*, *plgndr*, and *el2* have been identified to have strip type failure patterns [18], this may explain why the improved performances of ART-ORB over RT are less pronounced compared to programs with block patterns. Both *gammq* and *calDay* show similar improved performances, as they have been identified to have strip failure patterns. The marginal improvement of ART-ORB over RT for *sncndn* and *golden*, is also arguably related to these programs having point failure patterns [19], [30]. The experiments, has shown that the failure patterns for *triangle*, *line*, and *calGCD* (see figures in Appendix) are similar to the point patterns discussed in Section II.A.2, and therefore, their failure detection efficiency were not expected to improve [20]. The *cel* program also shows some small improvement although its failure pattern is non-point [18]. Previous study has attributed this performance to the shape and nature of its input domain [30]—that is, there are significant variations in the dimensional magnitudes of its input domain. However, its performance may improve for extremely high target exclusion (*R*), as seen in some other studies [14]. For all the subject programs used in the experiment, ART-ORB outperformed RT in terms of the F-measure.

RQ3: Comparing the failure detection improvements of ART-ORB to those of other ART methods, ART-ORB showed more improvement in most failure patterns than ART-RP, ART-B, ART-DC, and ART-TPP. Similarly, as shown in Fig. 8, ART-ORB outperformed the other ART methods for most of the fault-seeded programs. These improvements may be attributed to ART-ORB's ability to more evenly spread tests over the input domain.

Regarding ART-ORB's potential for finding multiple failures (E-measure), the empirical results indicate that it is comparable to the other ART methods: For all failure rates and failure patterns used in the simulation, ART-ORB performed similarly to the other ART methods, except ART-DC. Comparable E-measure performances were also observed in the experiments with fault-seeded programs. Although the ART-DC E-measure performance of was better in the simulations (especially for higher failure rates), the experimental results also indicate that ART-DC only has better performance for input domains with lower dimensions.

The Fm-time results obtained in both the simulations and experiments provide show that ART-ORB outperforms the other ART methods in terms of failure detection efficiency. ART-ORB's lower time cost for detecting the first failure is attributed to its much reduced computational overheads: it does not have candidate selections; distance calculations for all but one executed test are not necessary; repartitioning is only one dimension of one subregion at a time; and reassignment/reclassification of previous tests to new regions only involves two tests for each new partition. ART-ORB has reduced distance computations compared to ART-TPP and ART-DC; and reduced test reassignments compared to ART-RP, ART-B, and ART-DC. In addition, the ART-ORB partitioning process is done progressively, and in real-time, and hence the time cost of detecting the first failure is proportional to the number of previously executed tests. This is a major advantage in practice, since the testing process may often terminate one a failure has been detected.

ART-ORB also has lower execution time than the other ART methods for all failure rates and patterns. These observations were consistent across both simulations and experiments. The lower execution time indicates that ART-ORB is more computationally efficient, taking less time to execute tests. ART-ORB's efficiency is attributed to its minimized computational overheads.

RQ4: The findings from the statistical evaluation of the E-measure results generally indicate that ART-ORB has similar failure detection ability to the other ART methods, except ART-DC, which had better failure detection probability, possibly due to the relatively large threshold (λ=50) used in our study. Larger thresholds increase the probability of fault detection, but also incur higher computational costs—ART-DC has higher execution time than ART-ORB in terms of both Fm-time and Execution time. On the other hand, some interesting observations were made when higher dimensional subject programs (*d*≥5) were used. As seen from the statistical analysis of the E-measure results in Table VIII, the performance of ART-DC tends to decrease for higher dimensional programs. ART-ORB, in contrast, was observed to outperform all the other ART methods for almost all higher dimensional programs. This observation was also evident in the comparison of the E-measure results shown in Fig. 9. This provides evidence that ART-ORB is also more effective for testing higher dimensional programs.

VI. Threats to Validity

We have proposed and analyzed a new partition-based ART method, ART-ORB, that aims to improve on RT's failure detection effectiveness and efficiency. Evaluation of threats to the validity of a study is very important, and this section discusses the potential threats to the validity of this study.

The ART-ORB algorithm ensures an even spread of tests within the input domain through the combination of bisection-partitioning and an exclusion zone implementation. A potential threat to the validity of our study is that it may be possible for two test inputs from different regions of the input domain to fall very close to each other. Our investigation into the impact of this possible threat (RQ1) involved evaluating the distribution of tests generated by ART-ORB, and showed that the effect of this potential threat on the method's performance is not significant: The possibility of it happening is very small.

A common threat to the validity of any empirical study relates to the generalization of the results obtained to other situations. This study has shown that ART-ORB is applicable to both lower and higher dimensional programs with varying input domain sizes. The evaluations in this study employed a set of 16 real-life programs with input domains of up to 10 dimensions (higher dimensional programs require very high system configurations to run due to their time and space complexity). In addition, some subject programs with the same dimensionality were identified to have similar failure pattern types—the 1D programs' patterns, for example, were all characterized as block type. Although the choice of subject programs for this study may not fully represent a generalization of the proposed method's applicability, it does serve the purpose of introducing and evaluating ART-ORB. Furthermore, the simplicity of the approach (especially the incremental partitioning of the input space and selection of tests from within the regions of the input domain) make it applicable to other types of programs, including very large ones. In our future work, we will apply ART-ORB to programs with input domains beyond those used in this study, including of higher dimensions, larger input domains, and involving other programming constructs. Such application will, we anticipate, further validate the approach, and support more generalization of the conclusions.

The choice of other ART algorithms used in the comparisons may also represent a threat to the validity of this study. In both the simulations and experiments, we compared ART-ORB with four partition-based ART algorithms: ART-RP, ART-B, ART-DC, and ART-TPP. Since the four compared ART implementations are common variations of the basic ART algorithms [13], [27], [28], they are suitable candidates for the comparisons. Nonetheless, our future work will include evaluation against other ART methods, based on notions other than partitioning, which we anticipate will further improve the method's validity.

VII. Conclusion

In this paper, we have introduced an innovative, new test case generation method (ART-ORB) aimed at reducing the number of test cases executed before detecting the first failure (compared to pure RT), and providing selection overheads that are comparable to previous ART methods. The method integrates the concepts of both partition testing and exclusion.

The method involves sequentially bisecting the largest region within the entire input domain into further sub-regions. This partitioning process is activated whenever two previously executed tests are found within a single region. The process splits the region with a divisor orthogonal to the largest dimension of the region, through the midpoint of the previously executed tests (in that region). The method repeatedly divides the regions orthogonally and selects new inputs randomly from outside of a restricted zone in the region. This process has low overheads related to reassigning (two) tests in the region, and avoids computing distances between all previously executed tests. This results in an enhanced failure detection capability and reduced test input generation overheads.

We performed a series of simulations to examine the method's test case distribution, compared to RT and other ART methods (ART-RP, ART-B, ART-DC, and ART-TPP). We also performed simulations using the different categories of failure pattern, and experimented with real-life, fault-seeded subject programs, again comparing the method with both RT and the other ART methods.

The evaluation and empirical results indicate that the proposed approach is simple to implement, provides acceptable complexity, and a better even spread of test inputs because of its *one-domain-one-input* approach. Our evaluations have demonstrated that ART-ORB can distribute tests more evenly over the input domain than RT, and in a distribution comparable to other ART methods.

ART-ORB performs better than RT in terms of ability to detect failure, using significantly fewer tests than RT. It also compares well with other ART methods in terms of failure detection effectiveness.

In terms of efficiency of detecting failure, ART-ORB has demonstrated minimized execution overheads, and outperforms all the comparison ART methods in terms of both the time required to find a first failure, and the execution time. The significance of the overhead reductions was particularly evident in high dimensional input domains.

In conclusion, we recommend that ART-ORB should be considered whenever RT may be used, especially in situations where test input execution is expensive. ART-ORB should also be preferred among other ART methods in situations where cost-efficient test input selection is required**,** especially in higher dimensional input space**.** Due to the simplicity of this approach, it will be of great interest to combine it with other ART strategies in the future, to further improve on its failure-detection effectiveness and efficiency. Our future work will include enhancing the failure detection effectiveness of the method by extending the exclusion to neighboring regions of a candidate test case or by applying probability test profiles to the algorithm.

Acknowledgment

We thank Prof. T. Y. Chen for the invaluable suggestions and guidance related to the study of adaptive random testing.

References

[1] D. Hamlet and R. Taylor, "Partition testing does not inspire confidence (program testing)," *IEEE Transactions on Software Engineering,* vol. 16, no. 12, pp. 1402-1411, Dec. 1990. doi:10.1109/32.62448.

[2] T. Y. Chen, F.-C. Kuo, R. G. Merkel, and T. H. Tse, "Adaptive random testing: The art of test case diversity," *Journal of Systems and Software,* vol. 83, no. 1, pp. 60-66, Jan. 2010. doi:10.1016/j.jss.2009.02.022.

[3] T. Y. Chen, F.-C. Kuo, H. Liu, P.-L. Poon, D. Towey, T. Tse, and Z. Q. Zhou, "Metamorphic testing: A review of challenges and opportunities," *ACM Computing Surveys (CSUR),* vol. 51, no. 1, pp. 4:1–4:27, 2018. doi:10.1145/3143561.

[4] R. Hamlet, *Random Testing. Encyclopedia of Software Engineering. Edited by J. Marciniak*: Wiley, 1994.

[5] T. Yoshikawa, K. Shimura, and T. Ozawa, "Random program generator for Java JIT compiler test system," in *Proceedings of the 3rd International Conference on Quality Software (QSIC 2003)*, Dallas, TX, USA, Nov, 2003, pp. 20-23. doi:10.1109/QSIC.2003.1319081.

[6] B. P. Miller, D. Koski, C. P. Lee, V. Maganty, R. Murthy, A. Natarajan, and J. Steidl, *Fuzz revisited: A re-examination of the reliability of UNIX utilities and services*, Technical report, University of Wisconsin, 1995.

[7] 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-13)*, Lisbon, Portugal, Sept, 2005, pp. 263-272. doi:10.1145/1081706.1081750.

[8] G. J. Myers, *The Art of Software Testing. Revised and updated by T. Badgett, T.M. Thomas and C. Sandler, Hoboken*: NJ: John Wiley and Sons, 2004.

[9] L. J. White and E. I. Cohen, "A domain strategy for computer program testing," *IEEE Transactions on Software Engineering,* vol. SE-6, no. 3, pp. 247-257, May 1980. doi:10.1109/TSE.1980.234486.

[10] P. G. Bishop, "The variation of software survival time for different operational input profiles (or why you can wait a long time for a big bug to fail)," in *Proceedings of the 23rd International Symposium on Fault-Tolerant Computing (FTCS-23 Digest of Papers, 1993).* Jun. 22-24, 1993, pp. 98-107. doi:10.1109/FTCS.1993.627312.

[11] P. E. Ammann and J. C. Knight, "Data diversity: An approach to software fault tolerance," *IEEE Transactions on Computers,* vol. 37, no. 4, pp. 418-425, Apr. 1988. doi:10.1109/12.2185.

[12] T. Y. Chen, H. Leung, and I. K. Mak, "Adaptive random testing," in *Proceedings of the 9th Asian Computing Science Conference (ASIAN'04), Lecture Notes in Computer Science*, Chiang Mai, Thailand, Dec., 2004, pp. 320-329. doi:10.1007/978-3-540-30502-6_23.

[13] T. Y. Chen, R. G. Merkel, P.-K. Wong, and G. Eddy, "Adaptive random testing through dynamic partitioning," in *Proceedings of the 4th International Conference on Quality Software, 2004 (QSIC 2004)* Braunschweig, Germany, Sept., 2004, pp. 79-86. doi:10.1109/QSIC.2004.1357947.

[14] K. P. Chan, T. Y. Chen, and D. Towey, "Restricted random testing: Adaptive random testing by exclusion," *International Journal of Software Engineering and Knowledge Engineering,* vol. 16, no. 04, pp. 553-584, Oct. 2006. doi:10.1142/S0218194006002926.

[15] R. Huang, H. Liu, X. Xie, and J. Chen, "Enhancing mirror adaptive random testing through dynamic partitioning," *Information and Software Technology,* vol. 67, pp. 13-29, Nov. 2015. doi:10.1016/j.infsof.2015.06.003.

[16] J. Chen, F.-C. Kuo, T. Y. Chen, D. Towey, C. Su, and R. Huang, "A Similarity Metric for the Inputs of OO Programs and Its Application in

Adaptive Random Testing," *IEEE Transactions on Reliability,* vol. 66, no. 2, pp. 373-402, 2017. doi:10.1109/TR.2016.2628759.

[17] J. Chen, L. Zhu, T. Y. Chen, D. Towey, F.-C. Kuo, R. Huang, and Y. Guo, "Test case prioritization for object-oriented software: An adaptive random sequence approach based on clustering," *Journal of Systems and Software,* vol. 135, no. C, pp. 107-125, 2018. doi:10.1016/j.jss.2017.09.031.

[18] T. Y. Chen, F.-C. Kuo, and Z. Q. Zhou, "On favourable conditions for adaptive random testing," *International Journal of Software Engineering and Knowledge Engineering,* vol. 17, no. 06, pp. 805-825, Dec. 2007. doi:10.1142/S0218194007003501.

[19] T. Y. Chen, T. H. Tse, and Y.-T. Yu, "Proportional sampling strategy: A compendium and some insights," *Journal of Systems and Software,* vol. 58, no. 1, pp. 65-81, Aug. 2001. doi:10.1016/S0164-1212(01)00028-0.

[20] F. T. Chan, T. Y. Chen, I. K. Mak, and Y.-T. Yu, "Proportional sampling strategy: Guidelines for software testing practitioners," *Information and Software Technology,* vol. 38, no. 12, pp. 775-782, Feb. 1996. doi:10.1016/0950-5849(96)01103-2.

[21] C. Mao, T. Y. Chen, and F.-C. Kuo, "Out of sight, out of mind: a distance-aware forgetting strategy for adaptive random testing," *Science China Information Sciences,* vol. 60, no. 9, Apr. 2017, Art. no. 092106. doi:10.1007/s11432-016-0087-0.

[22] K. K. Sabor and S. Thiel, "Adaptive random testing by static partitioning," in *Proceedings of the 10th International Workshop on Automation of Software Test*, Florence, Italy, May 16-24, 2015, pp. 28-32. doi:10.1109/AST.2015.13.

[23] I. K. Mak, "On the effectiveness of random testing," M.S. thesis, Faculty of Science, University of Melbourne, 1998.

[24] J. K. Salmon, "Parallel hierarchical N-body methods," Ph.D Dissertation California Institute of Technology, 1991.

[25] M. Reumann, B. G. Fitch, A. Rayshubskiy, D. U. J. Keller, G. Seemann, O. Dossel, M. C. Pitman, and J. J. Rice, "Orthogonal recursive bisection data decomposition for high performance computing in cardiac model simulations: Dependence on anatomical geometry," in *Proceedings of the Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC 2009)*, Minneapolis, MN, USA, Sept. 3-6, 2009, pp. 2799-2802. doi:10.1109/IEMBS.2009.5333803.

[26] J. Makino, "A fast parallel treecode with grape," *Publications of the Astronomical Society of Japan,* vol. 56, no. 3, pp. 521-531, Jun. 2004. doi:10.1093/pasj/56.3.521.

[27] C. Chow, T. Y. Chen, and T. H. 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, 2013)*, Nanjing, China, Jul. 29-30, 2013, pp. 268-275. doi:10.1109/QSIC.2013.19.

[28] C. Mao, "Adaptive random testing based on two-point partitioning," *Informatica,* vol. 36, no. 3, Sept. 2012.

[29] T. Y. Chen, D. H. Huang, and F.-C. Kuo, "Adaptive random testing by balancing," in *Proceedings of the 2nd International Workshop on Random Testing: co-located with the 22nd IEEE/ACM International Conference on Automated Software Engineering (ASE 2007)*, Atlanta, Georgia, Nov. 06, 2007, pp. 2-9. doi:10.1145/1292414.1292418.

[30] K. P. Chan, T. Y. Chen, and D. Towey, "Restricted random testing," in *Proceedings of the 7th European Conference on Software Quality—ECSQ 2002,* Helsinki, Finland, Jun, 2002, pp. 321-330. doi:10.1007/3-540-47984-8_35.

[31] K. P. Chan, T. Y. Chen, and D. Towey, "Controlling restricted random testing: An Examination of the exclusion ratio parameter," in *Proceedings of the 19th International Conference on Software Engineering and Knowledge Engineering (SEKE 2007)*, Boston, MA, Jul. 9-11, 2007, pp. 163-166

[32] E. J. Weyuker and B. Jeng, "Analyzing partition testing strategies," *IEEE Transactions on Software Engineering,* vol. 17, no. 7, pp. 703-711, Jul. 1991. doi:10.1109/32.83906.

[33] T. Y. Chen, D. H. Huang, and Z. Q. Zhou, "Adaptive random testing through iterative partitioning," in *Proceedings of the International Conference on Reliable Software Technologies*, Porto, Portugal, Jun. 5-9, 2006, pp. 155-166. doi:10.1007/11767077_1.

[34] K. P. Chan, T. Y. Chen, and D. Towey, "Normalized restricted random testing," in *Proceedings of the 8th Ada-Europe International Conference on Reliable Software Technologies*, Toulouse, France, Jun. 16-20, 2003, pp. 368-381.

[35] H. Ackah-Arthur, J. Chen, J. Xi, M. Omari, H. Song, and R. Huang, "A cost-effective adaptive random testing approach by dynamic restriction," *IET Software,* vol. 12, no. 6, pp. 489 – 497, Dec. 2018. doi:10.1049/iet-sen.2017.0208.

[36] J. Geng and J. Zhang, "A new method to solve the "boundary effect" of adaptive random testing," in *Proceedings of the 2010 International Conference on Educational and Information Technology (ICEIT)*, Chongqing, China, Sept., 2010, pp. V1-298-V1-302. doi:10.1109/ICEIT.2010.5607704.

[37] 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*, Boston, United States July 9-11, 2007, pp. 141-144.

[38] H. Liu, X. Xie, J. Yang, Y. Lu, and T. Y. Chen, "Adaptive random testing through test profiles," *Software: Practice and Experience,* vol. 41, no. 10, pp. 1131-1154, Apr. 2011. doi:10.1002/spe.1067.

[39] H. Liu, X. Xie, J. Yang, Y. Lu, and T. Y. Chen, "Adaptive random testing by exclusion through test profile," in *Proceedings of the 10th International Conference on Quality Software (QSIC 2010)* Zhangjiajie, China, 2010, pp. 92-101. doi:10.1109/QSIC.2010.61.

[40] T. Y. Chen and Y.-T. Yu, "On the relationship between partition and random testing," *IEEE Transactions on Software Engineering,* vol. 20, no. 12, pp. 977-980, Dec. 1994. doi:10.1109/32.368132.

[41] A. Arcuri and L. Briand, "A Hitchhiker's guide to statistical tests for assessing randomized algorithms in software engineering," *Software Testing, Verification and Reliability,* vol. 24, no. 3, pp. 219-250, 2014.

[42] A. Vargha and H. D. Delaney, "A critique and improvement of the CL common language effect size statistics of McGraw and Wong," *Journal of Educational and Behavioral Statistics,* vol. 25, no. 2, pp. 101-132, Jun. 2000. doi:10.3102/10769986025002101.

[43] R. C. Team, "R: A language and environment for statistical computing," *R Foundation for Statistical Computing*, Vienna, Austria, 2018. Available: http://www.R-project.org/.

[44] K. P. Chan, T. Y. Chen, and D. Towey, "Forgetting test cases," in *Proceedings of the 30th Annual International Computer Software and Applications Conference (COMPSAC'06)*, Chicago, IL, USA, Sept. 17-21, 2006. doi:10.1109/COMPSAC.2006.43.

[45] ACM, *Collected Algorithms from ACM*, Volume 1, 2, 3, (1980) ed., New York, USA: Association for Computer Machinery, 1980.

[46] B. P. Flannery, W. H. Press, S. A. Teukolsky, and W. T. Vetterling, *Numerical Recipes 3rd Edition: The Art of Scientific Computing*, 3rd ed.: Cambridge university press, 2007.

[47] J. Ferrer, F. Chicano, and E. Alba, "Evolutionary algorithms for the multi‐objective test data generation problem," *Software: Practice and Experience*, vol. 42, no. 11, pp. 1331-1362, Nov. 2, 2012. doi:https://doi.org/10.1002/spe.1135.

[48] Y. D. Liang, *Introduction to Java Programming and Data Structures*, Eleventh ed.: Pearson Education, 2017.

[49] Y. Jia and M. Harman, "An analysis and survey of the development of mutation testing," *IEEE Transactions on Software Engineering,* vol. 37, no. 5, pp. 649-678, Sept.-Oct. 2011. doi:10.1109/TSE.2010.62.

[50] F.-C. Kuo, "On adaptive random testing," PhD Dissertation, Faculty of Information & Communication Technologies, Swinburne University of Technology, 2006.

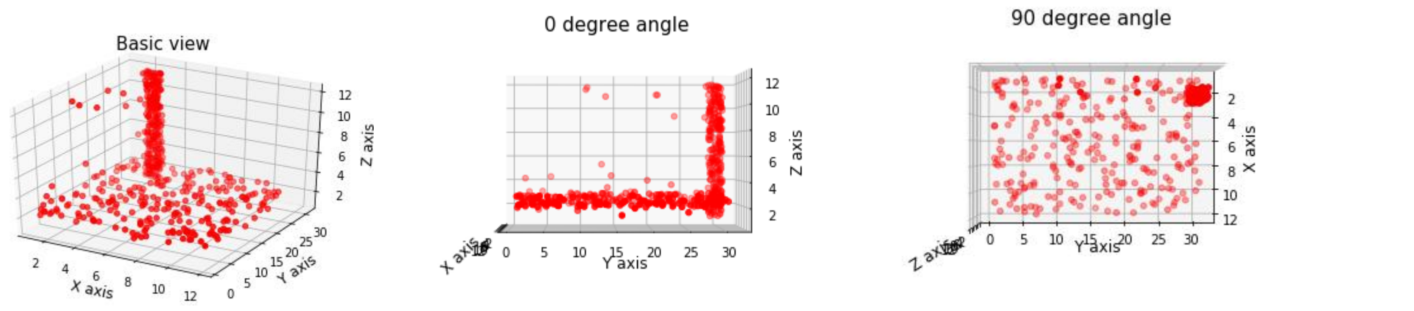
[51] D. Towey, "Studies of different variations of adaptive random testing," PhD Dissertation, University of Hong Kong, 2006.

[52] T. Y. Chen, F.-C. Kuo, D. Towey, and Z. Q. Zhou, "A revisit of three studies related to random testing," *Science China Information Sciences,* vol. 58, no. 5, pp. 052104:1--052104:9, Apr. 2015. doi:10.1007/s11432-015-5314-x.

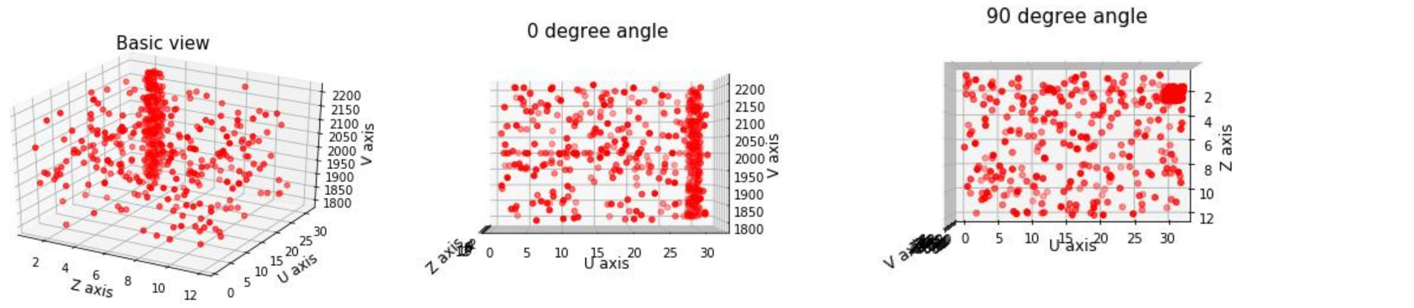
[53] T. Y. Chen, F.-C. Kuo, and Z. Zhou, "On the relationships between the distribution of failure-causing inputs and effectiveness of adaptive random testing," in *Proceedings of the 17th International Conference on Software Engineering and Knowledge Engineering (SEKE 2005)*, Taipei, Taiwan, Jun 14-16, 2005, pp. 306-311.

APPENDIX

The figures below are illustrates the distributions of failure causing inputs that are generated from the domains of the calDay, triangle, line and calGCD programs respectively. For each program, 500 failure causing inputs are generated and plotted in 3D with three views angles.

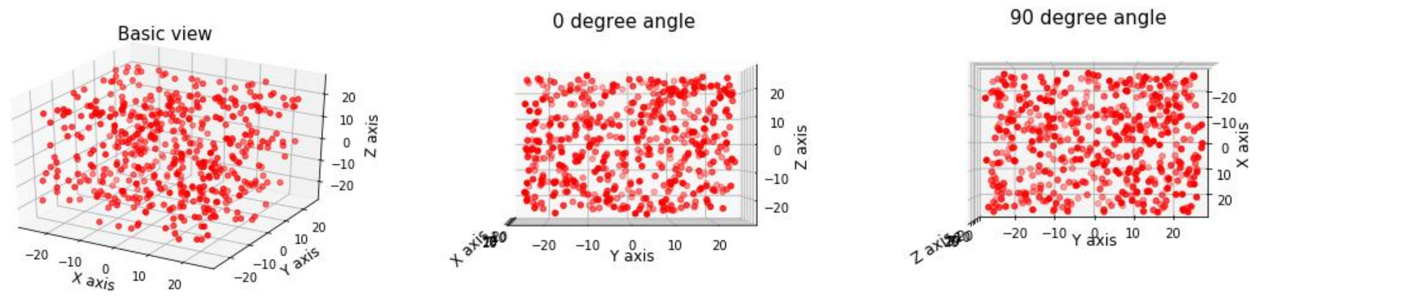


(a)

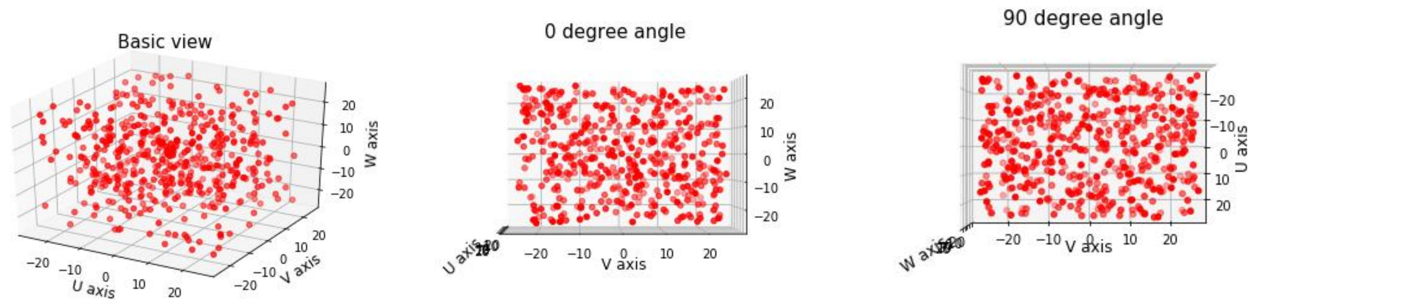


(b)

Fig. A1. The distribution of failure-causing test inputs for the *calDay* program represented in 3D. **a** shows the distribution in the first three axes (X, Y, Z). **b** shows the distribution in the last three axes (Z, U, V).

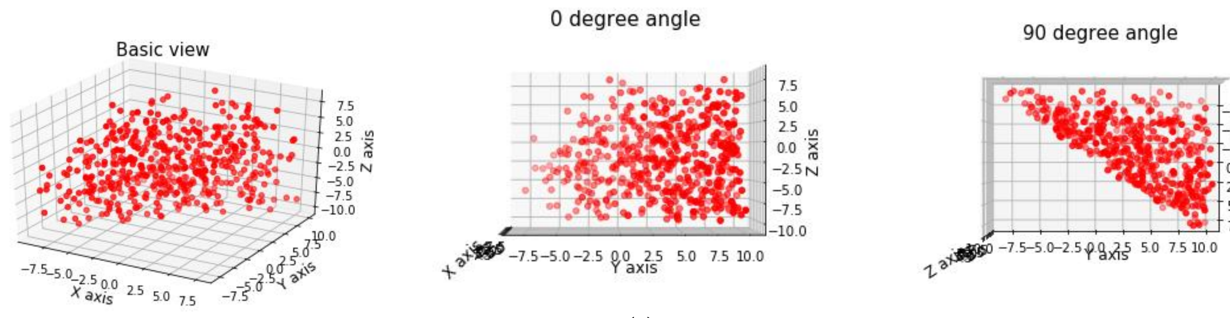


(a)

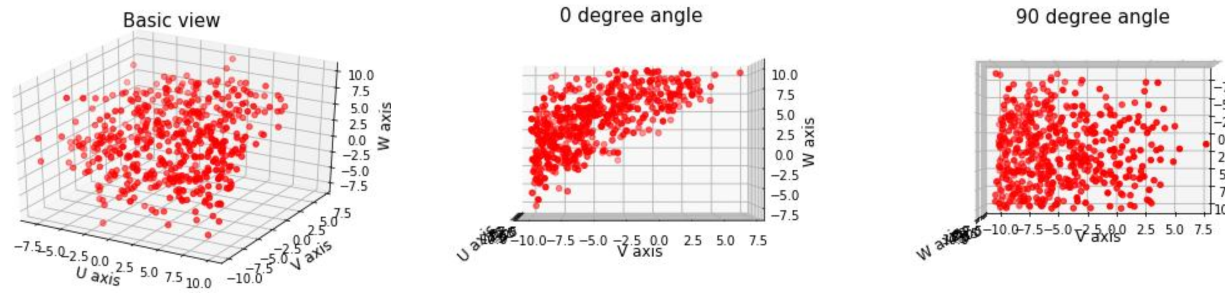


(b)

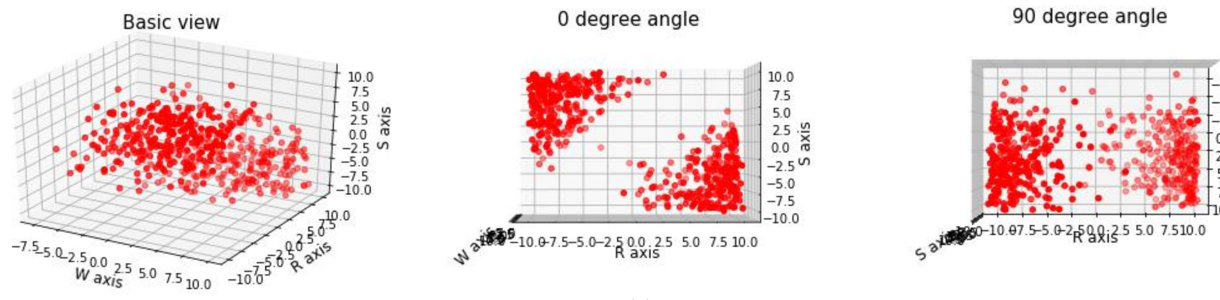
Fig. A2. The distribution of failure-causing test inputs for the *triangle* program represented in 3D. **a** shows the distribution in the first three axes (X, Y, Z) . **b** shows the distribution in the last three axes (U, V, W).



(a)

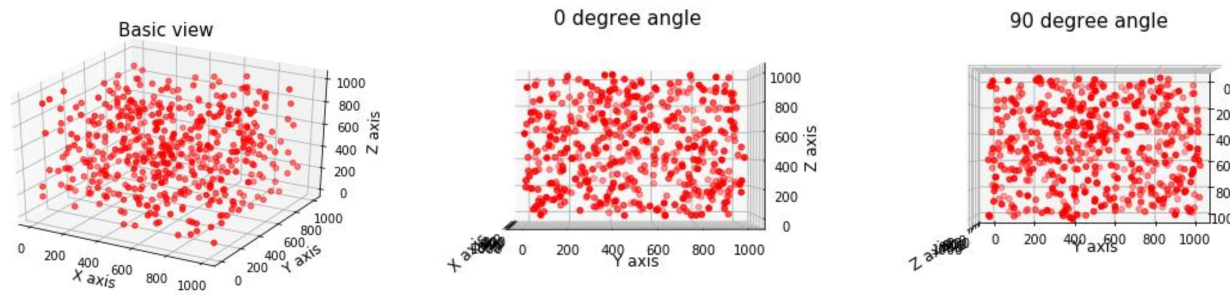


(b)

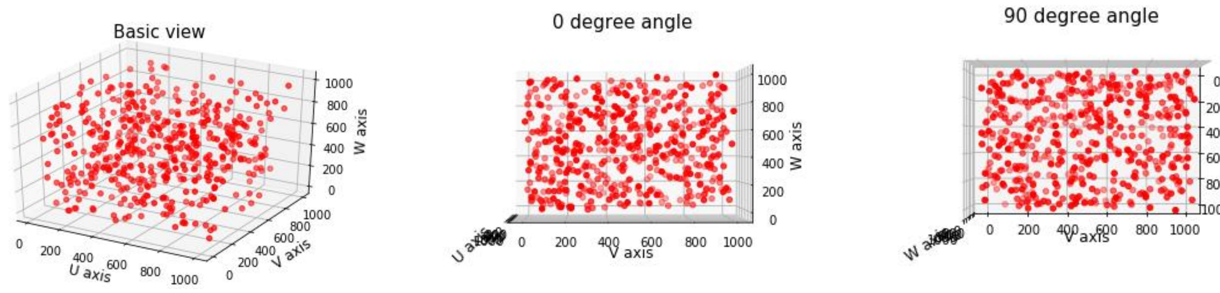


(c)

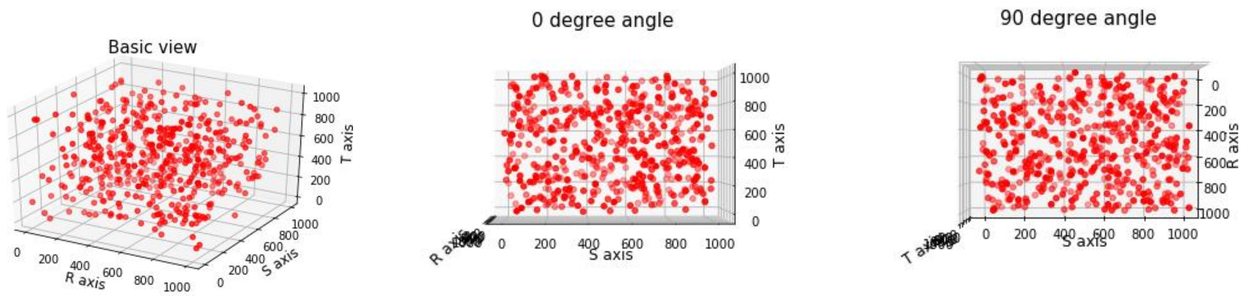
Fig. A3. The distribution of failure-causing test inputs for the *line* program represented in 3D. **a** shows the distribution in the first three axes (X, Y, Z) . **b** shows the distribution in the fourth to sixth axes (U, V, W). **c** shows the distribution in the last three axes (W, R, S).



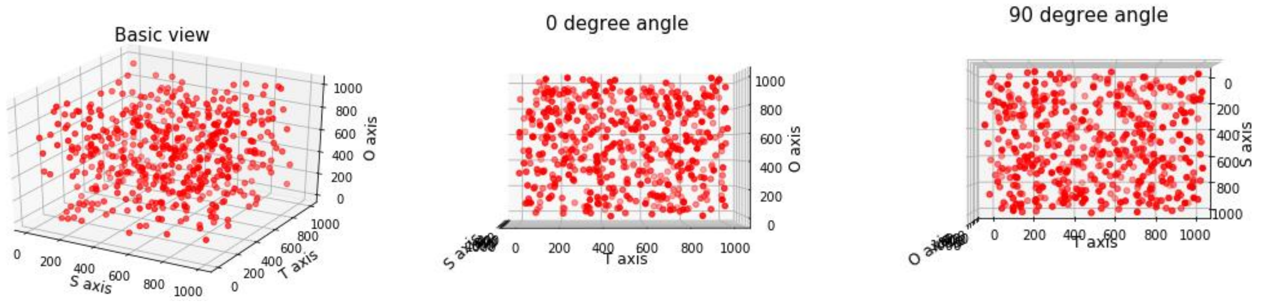
(a)



(b)



(c)



(d)

Fig. A4. The distribution of failure-causing test inputs for the *calGCD* program represented in 3D. **a** shows the distribution in the first three axes (X, Y, Z) . **b** shows the distribution in the fourth to sixth axes (U, V, W). **c** shows the distribution in the seventh to ninth axes (R,S,T). d shows the distribution in the last three axes (S, T, O).

Hilary Ackah-Arthur received the B.Sc. degree in computer science from University of Cape Coast, Ghana, in 2007; the M.S. degree in telecommunications management from HAN University of Applied Sciences, The Netherlands, in 2011; and the Postgraduate diploma in wireless and mobile computing from the Centre for Development of Advanced Computing (CDAC), India, in 2013. Since 2011, he has been a lecturer with the Computer Science Department, Takoradi Technical University, Ghana. He is currently pursuing the doctorate degree in computer applied technology at the School of Computer Science and Telecommunication Engineering, Jiangsu University, China. His research interests include software analysis and testing, service computing, information systems and security. Mr. Ackah-Arthur is a member of the IEEE Computer Society.

Jinfu Chen (M'13) received the B.E. degree from Nanchang Hangkong University, Nanchang, China, in 2004, and the Ph.D. from Huazhong University of Science and Technology, Wuhan, China, in 2009, both in computer science. He is a professor in the School of Computer Science and Communication Engineering, Jiangsu University, Zhenjiang, China. His major research interests include software testing, software analysis, and trusted software. Prof. Chen is a Member of the ACM and the China Computer Federation.

Dave Towey is an associate professor at University of Nottingham Ningbo China (UNNC), in Zhejiang, China, where he serves as the director of teaching and learning, and deputy head of school, for the School of Computer Science. He is also the deputy director of the International Doctoral Innovation Centre at UNNC. He is a member of the UNNC Artificial Intelligence and Optimization research group. His current research interests include software testing (especially adaptive random testing, for which he was amongst the earliest researchers who established the field, and metamorphic testing), computer security, and technology-enhanced education. He received the B.A. and M.A. degrees in computer science, linguistics, and languages from the University of Dublin, Trinity College, Ireland; the M.Ed. degree in education leadership from the University of Bristol, U.K.; and the Ph.D. degree in computer science from The University of Hong Kong, China. He co-founded the ICSE International Workshop on Metamorphic Testing in 2016. Towey is a member of both the IEEE and the ACM.

Michael Omari received the B.Sc. in computer science from University of Ghana, Legon in 2007. He obtained the masters degree from Coventry University, UK in 2014 in information technology. He worked in Takoradi Technical University as a research assistant from 2008 to 2013. Since then he has been working as an instructor in assembly language programming and software engineering. He began his doctoral studies in 2016 at Jiangsu University, China. His research interests include software testing and embedded systems.

Jiaxiang Xi received the B.E. degree in software engineering in 2015 from Jiangsu University, Zhenjiang, China, where he is currently working toward the master’s degree in the School of Computer Science and Communication Engineering. His research interests include software testing and service computing.

Rubing Huang is an associate professor in the Department of Software Engineering, School of Computer Science and Communication Engineering, Jiangsu University, China. He received the Ph.D. degree from Huazhong University of Science and Technology, China, in computer science and technology. His current research interests include software testing and software maintenance, especially adaptive random testing, random testing, combinatorial testing, and regression testing. He has more than 40 publications in journals and proceedings, including in ICSE, IEEE-TR, JSS, IST, IET Software, IJSEKE, SCN, COMPSAC, SEKE, and SAC. He has served as the program committee member of SEKE14-19, SAC17-19, and CTA17-19. He is a senior member of the China Computer Federation, and a member of the IEEE and the ACM.