VPP-ART: An Efficient Implementation of Fixed-Size-Candidate-Set Adaptive Random Testing Using Vantage Point Partitioning

Rubing Huang, *Senior Member, IEEE*, Chenhui Cui, Dave Towey, *Senior Member, IEEE*, Weifeng Sun, and Junlong Lian

***Abstract*—*Adaptive random testing* (ART) is an enhancement of *random testing* (RT), and aims to improve the RT failure-detection effectiveness by distributing test cases more evenly in the input domain. Many ART algorithms have been proposed, with *fixed-size-candidate-set* ART (FSCS-ART) being one of the most effective and popular. FSCS-ART ensures high failure-detection effectiveness by selecting as the next test case the candidate farthest from previously executed test cases. Although FSCS-ART has good failure-detection effectiveness, it also faces some challenges, including heavy computational overheads. In this article, we propose an enhanced version of FSCS-ART, *vantage point partitioning ART* (VPP-ART). VPP-ART addresses the FSCS-ART computational overhead problem using VPP, while maintaining the failure-detection effectiveness. VPP-ART partitions the input domain space using a *modified vantage point tree* (VP-tree) and finds the approximate nearest executed test cases of a candidate test case in the partitioned subdomains—thereby significantly reducing the time overheads compared with the searches required for FSCS-ART. To enable the FSCS-ART dynamic insertion process, we modify the traditional VP-tree to support dynamic data. The simulation results show that VPP-ART has a much lower time overhead compared to FSCS-ART, but also delivers similar (or better) failure-detection effectiveness, especially in the higher dimensional input domains. According to statistical analyses, VPP-ART can improve on the FSCS-ART failure-detection effectiveness by approximately 50–58%. VPP-ART also compares favorably with the *KD-tree-enhanced fixed-size-candidate-set ART* (KDFC-ART) algorithms (a series of enhanced ART algorithms based on the KD-tree). Our experiments also show that VPP-ART is more cost-effective than FSCS-ART and KDFC-ART.**

***Index Terms*—Software testing, adaptive random testing (ART), approximate nearest neighbor, vantage point partitioning (VPP), vantage point tree (VP-tree).**

Manuscript received 31 December 2021; revised 3 July 2022; accepted 21 October 2022. Date of publication 1 December 2022; date of current version 1 December 2023. This work was supported in part by the Science and Technology Development Fund of Macau, Macau SAR, under Grant 0046/2021/A and in part by the National Natural Science Foundation of China under Grant 61872167 and Grant 61502205. Associate Editor: F. Wotawa. *(Corresponding author: Rubing Huang.)*

Rubing Huang is with the School of Computer Science and Engineering, Macau University of Science and Technology, Taipa 999078, China (e-mail: rbhuang@must.edu.mo).

Chenhui Cui and Junlong Lian are with the School of Computer Science and Communication Engineering, Jiangsu University, Zhenjiang 212013, China (e-mail: 2211908012@stmail.ujs.edu.cn; 2211908018@stmail.ujs.edu.cn).

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

Weifeng Sun is with the School of Big Data and Software Engineering, Chongqing University, Chongqing 401331, China (e-mail: weifeng.sun@cqu.edu.cn).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TR.2022.3218602.

Digital Object Identifier 10.1109/TR.2022.3218602

I. Introduction

SOFTWARE testing is an important technique for evaluating and verifying the quality of the *system under test* (SUT), and is an important part of the software life cycle [1], [2]. Software testing involves executing the software, and aiming to find failures. It can be divided into the following four steps:

1) definition of test objectives;
2) generation of test cases;
3) execution of test cases;
4) examination and verification of test results.

Each test case is selected from the set of all possible inputs that constitute the *input domain*. When the output or behavior of the SUT during execution of the test case does not meet expectation (as determined by the *test oracle* [3], [4], [5]), then the test is considered to *fail*, otherwise, it *passes*.

Random testing (RT) [6] is a simple and efficient black-box testing method that generates test cases randomly within the input domain. RT has been used in a wide variety of environments and systems, including the following: in a stochastic scheduling algorithm for testing distributed systems [7]; for testing GCC, LLVM, and Intel C++ compilers [8]; and for GUI testing [9]. Research into RT enhancement has also been popular, with dynamic random testing (DRT) [10], [11], for example, improving the selection probability of subdomains with high failure-detection rates.

Because RT does not make use of any additional information beyond input parameter requirements, research is ongoing into how to improve its testing effectiveness. *Adaptive random testing* (ART) [12] is a family of RT-based testing techniques that aims to improve on RT testing effectiveness by more evenly spreading the test cases throughout the input domain. One of the first, and still most popular, ART implementations is *fixed-size-candidate-set* ART (FSCS-ART) [13]. Basically, for each next test case, FSCS-ART randomly generates k candidate test cases, calculates the distance between each candidate and each

previously executed test case (that did not reveal any failure), and selects the candidate farthest from them as the next test case to execute. Many previous studies have demonstrated the high effectiveness of FSCS-ART compared to RT [13], [14], [15], [16], [17], [18]. However, as reported by Wu et al. [19], although ART enhances RT, and is comparable to combinatorial testing in 96% of scenarios, it can be 3.5 times more computationally expensive than severely constrained combinatorial testing.

Although ART is very effective, many problems and challenges remain that need to be addressed [12]. One of these problems relates to the time required by FSCS-ART to select test cases, which can be much greater than the execution time: This is referred to as the *high computational overhead problem*. Many studies have investigated potential performance improvements for ART algorithms, including: a forgetting strategy [20] that reduces the number of distance calculations to previously executed tests; an approach, *distance-aware forgetting for fixed-size-candidate-set* ART (DF-FSCS-ART), that ignores executed test cases not in the line of *sight* of a given candidate [21]; implementations based on a K-dimensional tree (KD-tree) structure, KDFC-ART [22]; a *single-instruction-multiple-data* (SIMD) mechanism [23] to calculate all pairwise distances for a single distance calculation instruction; *adaptive random testing with divide-and-conquer* (ART-DC) [24], which uses a *divide-and-conquer* strategy to generate the test cases from the entire input domain; and *dynamic mirror ART* (DMART) [25], an enhancement of mirror ART (MART) [26] based on dynamic partitioning, that generates test cases using a specific ART algorithm in half of the subdomains, and then *mirrors* these test cases to the other half of the subdomains to generate the remaining test cases.

As stated, a significant problem faced by FSCS-ART is the heavy time overheads related to the large number of distance calculations required to find the nearest executed test cases for each candidate test case. Alleviating this problem will require a better way of identifying nearest executed test cases. In this article, we propose a new ART approach using *vantage point partitioning* (VPP-ART), to improve the efficiency of FSCS-ART. VPP-ART uses a modified vantage point tree (VP-tree) spatial partitioning structure to avoid redundant distance calculations, reducing the computational overheads of FSCS-ART. An original VP-tree [27], [28], [29] is a special kind of spatial partitioning tree that divides the input space into hyperspheres. Using a VP-tree, the space can be divided into inner and outer regions of the hypersphere, significantly reducing the number of computations when querying nearest neighbors (NNs) for a given query point. Therefore, VPP addresses the need for FSCS-ART time overheads reduction. To evaluate VPP-ART, we conducted a series of simulations and experiments on 22 subject programs, written in C++ and Java.

A standard VP-tree is only applicable to static data—the points must be known before the VP-tree is constructed. However, ART test case generation is a dynamic process: A newly generated test case tc depends on the information of previously executed test cases; if no failure is found by tc, then tc should be saved in the VP-tree. This process requires that the tree structure supports dynamic data, especially insert operations. Because the original VP-tree structure is constructed based on distances between the vantage point and other points, a worst-case scenario exists when a lower level node in the tree changes, causing upper level nodes to also (possibly) change, which may necessitate reconstruction of the entire tree. This problem can be addressed by revising the original VP-tree structure to support dynamic data.

The main contributions of this article are as follows:

1) We propose an improved VP-tree structure that can support dynamic insertion. This tree structure can identify an approximate NN that does not differ much from the exact NN, reducing the time cost of FSCS-ART. To the best of our knowledge, this is the first article to propose using VPP to address the ART time overheads problem.
2) We report on simulations and experiments investigating VPP-ART, from the perspectives of testing effectiveness and efficiency.
3) Compared with FSCS-ART, our approach significantly reduces computational overheads while delivering comparable, or better, failure-detection effectiveness. Compared with *KD-tree-enhanced fixed-size-candidate-set ART* (KDFC-ART) algorithms, our approach has similar or better performance, with reduced time costs in high dimensions.

The rest of this article is organized as follows. Section II introduces some background information about failure patterns, the original FSCS-ART method, and VPP. Section III presents a framework to enhance FSCS-ART, and introduces VPP-ART. Section IV describes the simulations and experiments, the results and analyses of which are presented in Section V. Section VI discusses the potential threats to the validity of our studies. Related work is discussed in Section VII, Finally, Section VIII concludes this article.

II. Background

In this section, we briefly present some background information about failure patterns and FSCS-ART. We also introduce some preliminary concepts about VPP.

A. Failure Regions

The inputs to a faulty program can be divided into two distinct types: *failure-causing inputs* (those inputs which, when executed, cause a test to fail); and *non-failure-causing inputs* (inputs that do not reveal a failure). The program's *failure region* consists of the set of all its failure-causing inputs. In software testing, knowledge of a failure region can be an extremely helpful guide for test case generation and selection. In general, two basic features are used to describe the failure region: the *failure pattern*, which is the distributions, shape, and locations of failure-causing inputs in the input domain; and the *failure rate*, denoted θ, which is the proportion of failure-causing inputs to all possible inputs in the entire input domain.

A number of studies [30], [31], [32], [33], [34] have reported that failure-causing inputs tend to cluster into contiguous regions. Chan et al. [35] classified failure patterns into three types: *block pattern*; *strip pattern*; and *point pattern*.

B. FSCS-ART

ART is a family of testing methods that improve over RT effectiveness by distributing the test cases more evenly throughout the input domain. One of the first, and still the most popular, ART implementations is FSCS-ART [13]. Many studies have shown FSCS-ART to be more effective than RT, in terms of failure-detection effectiveness [13], [14], [15], [16], test case distribution [17], and code coverage [18].

FSCS-ART [13] makes use of the concept of *distance* to evaluate similarities among test cases. It maintains two sets of test cases: the candidate set C and the executed set E. C stores k test cases that are randomly generated from the input domain, and E stores the test cases that have already been executed (but without causing any failure). Previous studies [13] have recommended a default value of 10 for k.

The FSCS-ART test case generation process can be described as follows. The first test case e_1 is randomly generated from the input domain, and executed. Assuming that e_1 does not reveal a failure, it is then stored in E. From now on, each time a new test case is needed, k test cases are randomly generated, and stored in the candidate set C. The *best* element from C is selected as the next test case to be executed—with FSCS-ART, best is defined as being farthest from the previously executed test cases (stored in E). As testing progresses (without failures being revealed), E grows larger. Formally, given a nonempty set of executed test cases E ($\{e_1, e_2, \ldots, e_{|E|}\}$), and a fixed number ($k$) of candidate test cases in C ($\{c_1, c_2, \ldots, c_k\}$), then the requirement for selecting the next (best) test case c_{best} is

$$\min_{\forall e_i \in E, c_j \in C} \text{dist}(c_j, e_i) \leq \min_{\forall e_i \in E} \text{dist}(c_{\text{best}}, e_i) \tag{1}$$

where $\text{dist}(x, y)$ is the distance between test cases x and y (typically the *Euclidean distance* for numeric input domains).

A challenge for the FSCS-ART algorithm is its *high computational overheads*: Each iteration of the FSCS-ART process requires distance calculations between each candidate test case in C and all the previously executed test cases. The time complexity of FSCS-ART is $O(kN^2)$, where k is the size of C and N is the number of previously executed test cases [12]. In order to detect a failure in a program, FSCS-ART could take an enormous amount of time to generate the required number of test cases. When the program's failure rate is very small, the FSCS-ART time cost will be very high. A key to reducing the computational overheads is to optimize the search for candidates' nearest executed test cases. Therefore, adoption of the highly efficient *NN Search* [36] strategies should enable a reduction in the search time.

C. VPP

Given a point set in a d-dimensional vector space, VPP [27] makes use of the relative distances between the points and a particular *vantage point* to enable a very efficient NN search.

VPP can be organized into a tree structure, a VP-tree [27], [28], [29]. The VP-tree can reduce unnecessary computations when solving NN search problems [36], and has been used in various contexts, including computational biology [37]; image processing [38]; databases [39], [40]; and computer vision [41].

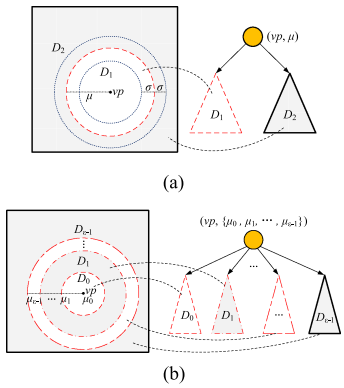


Fig. 1. VP-tree partitioning strategy in a 2-D input domain. (a) Binary VP-tree. (b) ε-ary VP-tree.

We have applied some modifications to the original VPP algorithm to enable its use with ART. Our research is, to the best of our knowledge, the first time VP-trees have been applied to test case generation in the field of ART.

1) VP-Tree Construction Process: To illustrate the VP-tree construction process, we use an example of binary (2-ary) partitioning. This can easily be generalized to ε-ary cases, where $\varepsilon > 2$ [42].

Generally speaking, a binary VP-tree is constructed by splitting a dataset into two subsets using a *distance partitioning criterion* and a *vantage point*. The vantage point is stored in the root node of the binary VP-tree, and the two subsets are organized into the left and right subtrees of the root node. Then, the subtrees are both processed recursively, constructing in each next level subtree according to newly selected vantage points. This continues until each node contains only one data point, and the construction of the tree is completed. Formally, given a set $\mathcal{D}$ of n data points, a point vp is randomly chosen as the vantage point. Next, the distances between vp and other points in $\mathcal{D}$ are calculated: $\mathcal{S} = \{\text{dist}(p, \text{vp}) | p \in \mathcal{D} - \{\text{vp}\}\}$. The entire dataset can be partitioned into two subsets using the median distance value μ in $\mathcal{S}$. As shown in Fig. 1(a), $\mathcal{D}_1$ refers to the points within a distance of μ from vp; and $\mathcal{D}_2$ refers to the points that are more than a distance of μ from vp.

The construction process for an ε-ary VP-tree when $\varepsilon > 2$ is similar to the binary tree case: For a given set of points ($\mathcal{D}$), a vantage point vp is again randomly chosen, and the distances between vp and all points in $\mathcal{D}$ are calculated and stored in ascending order. The differences compared with the binary case are as follows: 1) the dataset is not partitioned into only two subsets, but into ε approximately equal subsets; and 2) vp is stored in the first subset (in this article). As shown in Fig. 1(b), μ_i ($i = 1, 2, \ldots, \varepsilon - 1$) denotes the boundary distance values that split $\mathcal{D}_{i-1}$ and $\mathcal{D}_i$. Formally, for a sequence of distances, stored in ascending order, $\mathcal{S} = \{\text{dist}(a_j, \text{vp}) | a_j \in \mathcal{D}, j = 0, 1, \ldots, |\mathcal{D}| - 1\}$, where $|\mathcal{D}|$ is the number of elements in $\mathcal{D}$, the boundary distance values μ_i can be calculated as

$$\mu_i = \begin{cases} 0, & i = 0 \\ \dfrac{\mathcal{S}\left(i \times \left\lfloor |\mathcal{D}|/\varepsilon \right\rfloor - 1\right) + \mathcal{S}\left(i \times \left\lfloor |\mathcal{D}|/\varepsilon \right\rfloor\right)}{2}, & i \in [1, \varepsilon - 1] \end{cases} \tag{2}$$

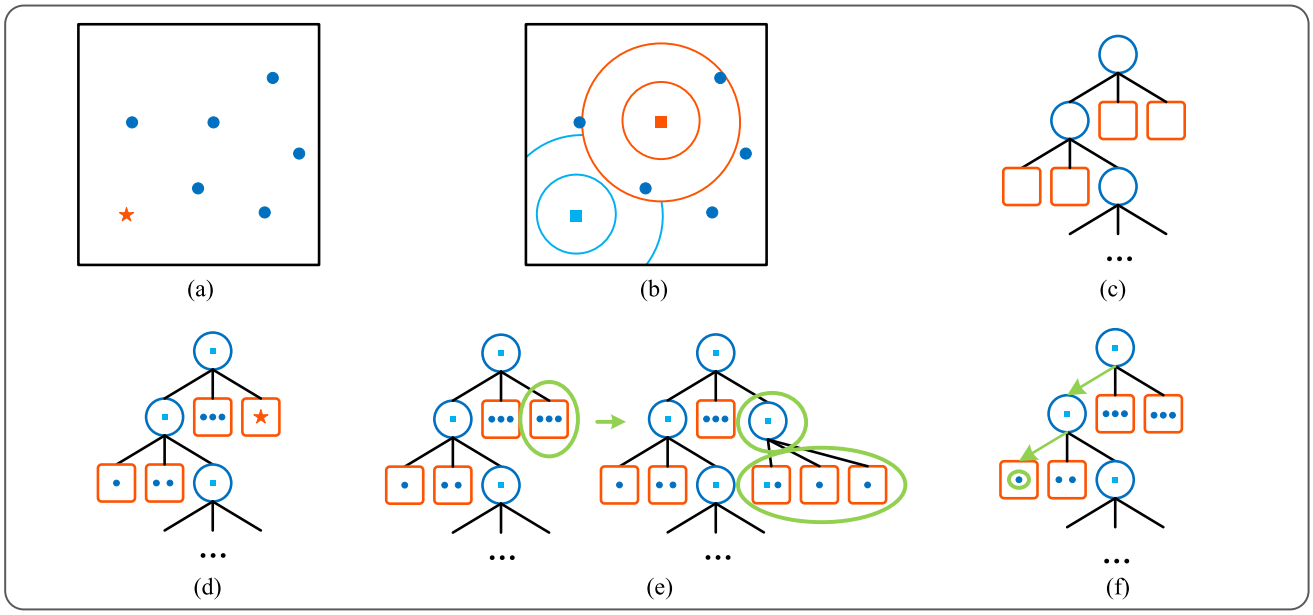


Fig. 2. VPP-ART framework, with six main stages. (a) Generate test cases in the input domain. (b) Partition the input domain using VPP. (c) Construct the modified VPP VP-tree structure. (d) Insert test cases into the modified VP-tree. (e) Promote a leaf node into a common node. (f) Search for the nearest neighbor test cases in the modified VP-tree.

where $\mathcal{S}(j)$ represents the jth ordered element in $\mathcal{S}$ ($j = 0, 1, \ldots, |\mathcal{D}| - 1$), and $\lfloor |\mathcal{D}|/\varepsilon \rfloor$ indicates the rounding down integer $|\mathcal{D}|/\varepsilon$. Therefore, for each point p in $\mathcal{D}_i$, the value of $\text{dist}(p, \text{vp})$ is between μ_i and μ_{i+1}.

2) NN Search in a VP-Tree: This section describes the algorithm for an NN search in a VP-tree, which involves identifying the nearest neighboring point of a query point q with the requirement that the maximum distance between q and the point be less than a specific threshold σ. This means that if the distance between q and its neighbor is greater than σ, then this cannot be the NN. If $\text{dist}(q, \text{vp})$ is the distance between the query point q and the vantage point vp, then the algorithm focuses on finding the approximate NN of q within the range $\text{dist}(q, \text{vp}) \pm \sigma$. With these requirements, the search for the NN of q in a binary VP-tree only needs to explore both $\mathcal{D}_1$ and $\mathcal{D}_2$ if q is in the range of $[\mu - \sigma, \mu + \sigma]$ (as shown in Fig. 1(a))—otherwise, only one subset needs to be searched, which effectively prunes one half of the input space. The approach is based on the principles of triangular inequality [29]. Formally, if $\text{dist}(q, \text{vp}) \leq \mu - \sigma$, for $p \in \mathcal{D}_2$, the distance between p and q is lower-bounded by σ [29]:

$$\begin{aligned} \text{dist}(p, q) &\geq \Big| \big|\text{dist}(p, \text{vp})\big| - \big|\text{dist}(q, \text{vp})\big| \Big| \\ &\geq \Big| \big|\text{dist}(p, \text{vp})\big| - (\mu - \sigma) \Big| \\ &> |\mu - \mu + \sigma| \\ &= \sigma \end{aligned} \tag{3}$$

therefore, the subset $\mathcal{D}_2$ can be ignored. Similarly, if $\text{dist}(q, \text{vp}) > \mu + \sigma$, for $p \in \mathcal{D}_1$, then $\mathcal{D}_1$ can be ignored. For the ε-ary cases, the system needs to explore $\mathcal{D}_i$ if

$$\mu_i - \sigma < d(q, \text{vp}) \leq \mu_{i+1} + \sigma \tag{4}$$

for $i = 0, 1, \ldots, \varepsilon - 2$ (the special case of $i = \varepsilon - 1$ will be discussed in Section III-B3): It is also based on triangular inequality, and can be derived in a similar way to the binary case.

III. VPP-ART: ART Based on VPP

In this section, we present our proposed ART approach, VPP-ART.

A. Framework

A main challenge for FSCS-ART lies in the high time cost in generating test cases. In this article, we combine VPP with the original FSCS-ART to improve the efficiency, mainly by organizing the set of executed test cases into a new storage structure. As noted, ART requires that the data structure used to store the executed test cases be able to support dynamic insertion.

Fig. 2 shows the framework, where small squares represent the vantage points, dots represent executed test cases, and the star represents a newly generated test case. The framework consists of six main stages:

1) generate test cases in the input domain;
2) partition the input domain using VPP;
3) construct the modified VP-tree structure;
4) insert test cases into the modified VP-tree;
5) promote a leaf node (terminal node) into a common (non-terminal) node;
6) search for the NN test cases in the modified VP-tree.

Stage 1: The test cases are generated in the same way as in FSCS-ART. The candidate test case set (C) contains k test cases randomly generated in the input domain. The executed test cases are stored in a modified VP-tree. Similar to FSCS-ART, the Euclidean distance is used to measure the similarity (distance) between test cases.

Stage 2: As the testing process proceeds, VPP is used to partition the input domain into different concentric hypersphere subdomains, bounded by different vantage points. Each subdomain contains far fewer test cases than the number of executed test cases in the entire input domain.

Stage 3: A modified VP-tree structure that supports dynamic data is used to store executed test cases, and to support the NN searches.

Stage 4: As VPP-ART proceeds, the candidate test cases (C) are generated randomly within the input domain, and the best candidate is identified and applied to the SUT. If an SUT failure is not revealed (and no other testing termination criteria are met), then the test case is added to the modified VP-tree. As the testing continues, the number of executed test cases in the VP-tree increases.

Stage 5: Test cases are only stored in leaf nodes of the modified VP-tree. During test case insertion, the number of test cases in a leaf node may reach the storage capacity, causing a promotion operation to be performed, which expands the storage capacity of the leaf node. Each round of test case insertion only needs to be executed, at most, once. After at most one promotion operation, a suitable leaf node (with spare capacity) will be identified and used to store the current test case.

Stage 6: The executed NN for each candidate test case is identified using the ε-ary VP-tree, with a series of query thresholds σ used to perform the searches. Starting from the root node of the modified VP-tree, the distances to vantage test cases are compared, layer by layer, until the leaf node containing the NN is identified.

B. Algorithm

The original VP-tree structure is constructed according to the distance criterion. The *top-down* partitioning strategy makes the management of VP-tree updates complicated—the partitioning of upper level nodes has an impact on the partitioning of lower level nodes [42]. In a worst-case scenario, reconstruction of the entire tree structure may be required. The VP-tree update operation remains a problem requiring further study. Fu et al. [42] proposed a dynamic VP-tree structure, but this strategy involves upward backtracking and node-splitting/merging, which may incur significant time overheads.

In this article, we 1) introduce a *modified VP-tree structure* in which to store the executed FSCS-ART test cases; and 2) propose an *insertion approach* for this modified structure. The inputs to the algorithm are the candidate set size k, the input domain dimension d, the leaf node capacity λ, and the partitioning parameter ε. Due to the page limit constraints, our detailed VPP-ART algorithms are provided online [43].

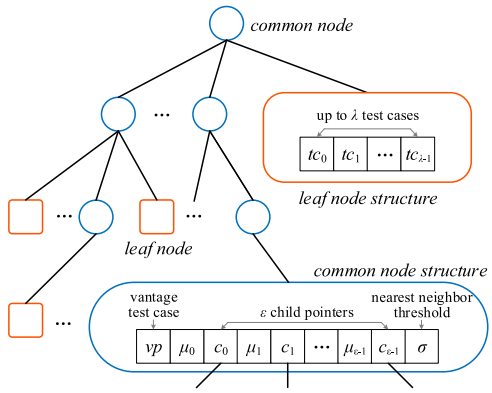


Fig. 3. Modified VPP-ART VP-tree structure.

1) Modified VPP-ART VP-Tree Structure: Fig. 3 shows the modified VP-tree structure, where nodes are divided into *leaf nodes* and *common nodes*, denoted by squares and circles, respectively. *Leaf nodes* contain only test cases, the maximum number of which that can be stored in a single leaf node is denoted by λ. *Common nodes* contain no test cases, but do contain the vantage test case information, several boundary distance values with child pointers, and an NN threshold.

The *partitioning parameter* ε specifies the number of subsets per division. This determines the number of child pointers in each common node ($c_0, c_1, \ldots, c_{\varepsilon-1}$ in Fig. 3). The child pointers point to nodes in the next level of the tree. The same ε value is used for each partition, with this value being set by the tester.

Vantage test cases are at the core of the structure, and their choice, in each level of the VP-tree, plays an important role in the performance of the algorithm. An ideal vantage test case should have a uniform distribution of distances between it and other test cases. This minimizes the number of test cases in concentric regions, thereby reducing the probability that all subtrees must be explored. However, finding the ideal vantage test case for a given test case set can require very heavy computational costs. In practice, therefore, a test case is randomly selected from the λ test cases stored in a leaf node, yielding an approximate (instead of exact) vantage test case. This selection method has been shown to be effective, experimentally.

2) VPP-ART Test Case Insertion Algorithm: The VP-tree update operations, especially insertion, have a crucial role in the effectiveness of VPP-ART. In the following, we focus on the VPP-ART test case storage process, and propose *insertion* and node *promotion* strategies to make this process more dynamic.

Insertion: As testing progresses, it becomes necessary to insert newly generated test cases into the modified VP-tree. For an executed test case e, if the current node is a leaf node, then the leaf node is said to be the *quasi-belonging-node* for e (denoted e-QBN). If the number of test cases in e-QBN is less than the maximum storage capacity ($\alpha < \lambda$), then e is inserted into e-QBN, and e-QBN is said to be the *belonging-node* for e (denoted e-BN). If $\alpha = \lambda$, then it is necessary to promote (reconstruct) e-QBN to find e-BN. If the current node is a

common node, then dist(e, vp) is calculated (where vp is the vantage test case in this node), and compared with the boundary values μ_i to determine which child pointer should be followed to find e-BN.

Promotion: This step is only performed when $\alpha = \lambda$, which means that it is necessary to transform e-QBN from a leaf node into a common node. A test case t_i $(0 < i \leq \lambda)$ is randomly selected from the test cases stored in e-QBN as the vantage test case for this node; the remaining test cases and e are reorganized into child nodes (new leaf nodes) of this new common node; and e is assigned to its new e-BN. This promotion process is executed, at most, one time when inserting e: After (at most) one promotion process, VPP-ART will find an e-BN in which to store e.

3) Approximate NN Search in VPP-ART: The NN obtained by VPP-ART could be an *approximate NN*, because this search process is likely to find an inaccurate neighbor in the leaf node that does not contain an *exact NN*, which can be explained as follows. 1) Some specific subdomains need special consideration, and the entire search may be carried out in these subdomains, which perhaps do not contain the exact NNs. An example of this is the last subdomain ($\mathcal{D}_{\varepsilon-1}$) after each round of partitioning. 2) Because the boundary distance μ and the NN threshold σ of each node are calculated according to the fixed number of test cases, these two values of each node will be affected when a new point is inserted. To a certain extent, VPP-ART deliberately ignores the impact of the insertion process on these two values. Specifically, when a new point is inserted, VPP-ART ignores the change of the two values of the QBN's upper level nodes—it does not update their μ and σ values. This may cause the NN of the query point to no longer be in the original subdomain, which reduces the accuracy of the search algorithm.

Although VPP-ART adopts an approximate NN search, it still has advantages in some cases.

1) An exhaustive search is required to find an exact NN, which requires that the distances between the executed test cases and the candidate all be calculated. The closest executed test case will be identified as the NN. When the number of test cases is very large, or the dimension is relatively high, the search efficiency decreases sharply. However, VPP-ART can obtain a better efficiency by using an approximate search: VPP-ART can identify an acceptable NN using fewer distance calculations when searching for an approximate NN.
2) An approximate NN search process, by not being limited to identifying the exact NN, can improve the search efficiency at an acceptable cost in the accuracy. When searching some subdomains, the accuracy of some NNs may be lost, but VPP-ART may achieve similar, or even better, results than exact NN-based ART algorithms. The difference between approximate NN and exact NN can be very small, making it possible for VPP-ART to have comparable failure-detection effectiveness.
3) Using the property that test cases closer to the vantage point are more likely to be divided into the same subdomain, when the number of test cases increases, test cases with greater similarity will aggregate together. Using the VP-tree structure, VPP-ART, according to the distance relationship between candidate test cases and vantage points, searches the possible subdomains (VP-tree leaf nodes) of executed test cases with greater similarity to candidate test cases.

IV. Experimental Studies

This section introduces the design and settings of the simulations and experiments that we conducted to evaluate VPP-ART.

A. Research Questions

The proposed VPP-ART algorithm aims to reduce the time overheads of the original FSCS-ART algorithm; thus, measurement and examination of the test case generation time are necessary. VPP-ART is also expected to maintain the FSCS-ART failure-detection effectiveness, which requires evaluation and verification in various scenarios. The experimental studies were guided by the following research questions:

RQ1 *How well does VPP-ART perform, in terms of software failure-detection, compared with other ART algorithms?*

RQ2 *Compared with FSCS-ART and KDFC-ART, to what extent can VPP-ART reduce computational overheads?*

B. Variables and Evaluation Metrics

This section describes the independent and dependent variables in our research. The evaluation metrics used to examine the effectiveness and efficiency of the different ART algorithms are also introduced.

1) Independent Variable: The independent variable in the experimental study was the different ART algorithms used to generate test cases. VPP-ART, the new algorithm proposed in this article, is compared with FSCS-ART [13] and KDFC-ART [22].

VPP-ART is an enhanced version of FSCS-ART, and we want to know the effects of using VPP on FSCS-ART. VPP-ART is an ART algorithm based on a dynamic VP-tree structure. The KD-tree, as an efficient spatial indexing mechanism, is a similar tree structure often compared with the VP-tree. Mao et al. [22] introduced KD-trees into ART, and proposed three KDFC-ART algorithms: Naive-KDFC; SemiBal-KDFC; and LimBal-KDFC. Naive-KDFC and SemiBal-KDFC search for the exact NN of candidate test cases, and thus, their generated test cases are the same as those generated by FSCS-ART. LimBal-KDFC uses a limited backtracking method, identifying an *approximate* NN, similar to VPP-ART. The simulations and experiments sought to examine two things: 1) the impact of the difference between exact and approximate NN searching on ART effectiveness and efficiency; and 2) the differences in effectiveness and efficiency between the two approximate NN-search-based ART algorithms, VPP-ART and LimBal-KDFC. KDFC-ART, to some degree, can achieve the goal of balancing the effectiveness and efficiency of exact and approximate NNs. We also compare the impact of different spatial partition

methods on FSCS-ART, based on similar data structures—*tree* structures. The strategies adopted by some other algorithms (including DMART and ART-DC) are not highly correlated with, or comparable to, those employed by VPP-ART—and thus cannot easily be used as baselines for comparison. The differences between some existing ART algorithms and VPP are discussed in Section VII.

2) Dependent Variables: The dependent variables in our studies are the evaluation metrics, for both effectiveness and efficiency.

Effectiveness Metrics: The *F-measure* [14] is the expected number of test case executions before detecting the first failure in the SUT, and has been widely used in ART studies [12]. We therefore also used the *F-measure* in our study, with F_{RT} and F_{ART} denoting the *F-measure* when conducting RT and ART, respectively. Theoretically, F_{RT} is equals $1/\theta$ (where θ is the SUT failure rate). The *ART F-ratio* [14] denotes the ratio of F_{ART} to F_{RT}, showing the improvement of ART over RT: A lower *ART F-ratio* indicates better ART performance.

Efficiency Metrics: There are several measurements commonly taken when examining the efficiency of a testing methodology, including *generation time*; *execution time*; and *F-time* [12], [44]. The generation time refers to the cost of generating N test cases; the execution time refers to the time cost of executing the SUT with N test cases; and the *F-time* [44] is defined as the entire time cost for finding the first failure in the SUT. Generally speaking, the test case generation time has a great impact on the entire test cost. In the simulation studies, we recorded the average generation time to generate a certain number of test cases.

C. Experimental Environment

The simulations and experiments were conducted using a 16-GB RAM laptop PC with an i7 CPU, running at 2.20 GHz, running under the 64-bit Windows 10 operating system. All the algorithms under study were implemented in Java with JDK 1.8. The IDEs used were Eclipse (Version 4.15.0) and Microsoft Visual Studio 2019.

D. Data Collection and Statistical Analysis

The *F-measure* and *F-time* values were calculated by running each of the ART algorithms until a failure was detected. In the simulations, a failure was considered to be detected whenever a test case was generated from within a simulated failure region. In the experiments, the actual output was compared with the expected output (the test oracle [3], [4], [5]): A difference indicated a failure being detected. To minimize the error caused by randomness, and to provide confidence in the comparison, each experiment was run 3000 times, with the average being recorded.

When analyzing the experimental data, the *p-value* (probability value) and *effect size* for the different ART algorithms were calculated [45], [46], [47]. These can describe any significant differences or improvements between the two compared methods [48]. The unpaired two-tailed Mann–Whitney–Wilcoxon test [47] was used to verify whether or not there was a significant difference among the investigated ART algorithms. A *p-value* less than 0.05 indicates a significant difference between the two algorithms [49]. The *effect size* [47] shows the possibility that one method is better than the other: We used the nonparametric Vargha and Delaney effect size [50]. For two methods, *A* and *B*, an *effect size* between *A* and *B* of 0.50 means that *A* and *B* are equivalent; a value greater than 0.50 means that *A* is better than *B*; and a value less than 0.50 means that *B* is better than *A*.

E. Simulations and Experiments

The original FSCS-ART and KDFC-ART algorithms were compared with our proposed algorithm, VPP-ART, through a series of simulations and experiments [12]. The simulations involved simulated software faults, while the experiments used real-life subject programs altered through mutation analysis techniques [51].

1) Simulations Design: The simulations used a d-dimensional hypercube as the program input domain ($\mathcal{D}$). $\mathcal{D}$ was set as $\{(x_1, x_2, \ldots, x_d) | 0.0 \leq x_1, x_2, \ldots, x_d < 1.0\}$, with the dimension d set as 1, 2, 3, 4, 5, 8, and 10.

To address RQ1, the simulated failure regions were randomly placed in the input domain $\mathcal{D}$. Once a program has been written, the failure regions are fixed, but their locations are unknown to developers and testers (before testing). Failure regions have geometric shape (described by the *failure patterns*) and size (described by the *failure rate*), and distribution [12]. In the simulations, the failure rate and pattern were set in advance, allowing the failure regions to be located randomly in $\mathcal{D}$. As described in Section II-A, failure patterns have often been categorized into three main types: *strip*; *block*; and *point*. The simulations included all three failure patterns types. Block patterns used a randomly located, single-solid shape with equal lengths of side—a square in 2-dimensions, cube in 3-dimensions, etc. The strip patterns were each constructed using two points on adjacent boundaries that were connected with a width/volume to yield the desired size. According to the predetermined failure rate θ, the point patterns used 25 randomly located regions. The simulations used seven different θ settings: 0.01, 0.005, 0.002, 0.001, 0.0005, 0.0002, and 0.0001.

To address RQ2, the simulations recorded the test case generation times for both FSCS-ART and VPP-ART. A total of 20 000 test cases were generated by each algorithm in the d-dimensional input domain, and the generation times recorded at intervals of 500.

We conducted a series of simulations to investigate the best values for the two VPP-ART parameters ε and λ, which we determined as 3 and 10, respectively (based on the experimental results from Table I). Due to the page limit constraints, the detailed simulation results are presented online [43].

2) Experimental Design: Although simulations can simulate the performance of the algorithms in different scenarios, the failure types may not be representative of real-life situations. For example, in reality, the failure types can be categorized into

TABLE I
ART F-RATIO RESULTS OF VPP-ART WITH DIFFERENT $\langle \varepsilon, \lambda \rangle$ PARAMETER PAIR VALUES

Parameter ε	Dimension (d)	Parameter λ				
		$\lambda = 10$	$\lambda = 20$	$\lambda = 30$	$\lambda = 40$	$\lambda = 50$
$\varepsilon = 2$	$d = 1$	0.5706	0.5523	0.5675	0.5631	0.5702
	$d = 2$	0.6790	0.6752	0.6645	0.6641	0.6440
	$d = 3$	0.8056	0.7809	0.7964	0.7957	0.7970
	$d = 4$	0.9611	0.9243	0.9506	0.9280	0.9462
	$d = 5$	1.0824	1.0984	1.0978	1.1260	1.1069
	$d = 8$	1.7050	1.7668	1.8200	1.7711	1.8289
	$d = 10$	2.4504	2.6366	2.6355	2.7046	2.7719
$\varepsilon = 3$	$d = 1$	0.5555	0.5686	0.5557	0.5599	0.5582
	$d = 2$	0.6510	0.6447	0.6377	0.6459	0.6523
	$d = 3$	0.7553	0.8083	0.7753	0.8063	0.7940
	$d = 4$	0.9390	0.9510	0.9385	0.9355	0.9421
	$d = 5$	1.0605	1.0709	1.1085	1.0997	1.1053
	$d = 8$	1.6631	1.7354	1.8558	1.8833	1.8493
	$d = 10$	2.2916	2.4765	2.6265	2.7906	2.7080
$\varepsilon = 4$	$d = 1$	0.5577	0.5621	0.5529	0.5567	0.5588
	$d = 2$	0.6570	0.6590	0.6534	0.6340	0.6529
	$d = 3$	0.7617	0.7571	0.7659	0.7900	0.7754
	$d = 4$	0.9144	0.9272	0.9418	0.9427	0.9393
	$d = 5$	1.0896	1.0734	1.1486	1.0964	1.1133
	$d = 8$	1.6900	1.8339	1.8048	1.8578	1.9717
	$d = 10$	2.3144	2.6492	2.8174	2.8207	2.9358

regular and *irregular* types [52], [53]). In addition to the simulations, we conducted experiments using 22 real-life programs with faults seeded in using mutation operators [51]. Table II presents the detailed information of these programs. A total of 12 of the programs come from *Numerical Recipes* [54] and ACM's *Collected Algorithms* [55], and have been widely studied in ART research [13], [16], [25], [56]. Programs `calDay`, `complex`, and `line` are from Ferrer et al. [57]. The `pntLinePos`, `pntTrianglePos` and `twoLinesPos` programs describe the positional relationships between a point and a line, a point and a triangle, and between two lines, respectively [58]. The `triangle` program classifies a triangle into one of three types (acute, right, and obtuse) [58]. The `nearestDistance` program uses five points to realize the nearest point pair function. The `calGCD` program calculates the greatest common divisor of ten integers, and `select` returns the ith largest element from an unordered array [59].

All subject programs were implemented in Java or C++, and had previously been used in the KDFC-ART experiments [22]. The following six mutation operators were used to generate mutants of the original subject programs [51]:

1) arithmetic operator replacement (AOR);
2) relational operator replacement (ROR);
3) scalar variable replacement (SVR);
4) constant replacement (CR);
5) statement deletion (SDL);
6) return statement replacement (RSR).

Five algorithms were applied in the experiments: original FSCS-ART; three KDFC-ART algorithms; and our proposed VPP-ART.

V. EXPERIMENTAL RESULTS

This section presents the results from the simulations and experiments. The differences in effectiveness and efficiency among FSCS-ART, KDFC-ART, and VPP-ART are discussed, and answers are provided to the two research questions from Section IV-A. In the tables in this section, a *blue bold* number indicates the minimum value of *ART F-ratio*, *F-measure*, or *F-time* across the several ART algorithms.

Due to the page limit constraints, this section only provides the mean ART F-ratio, F-measure, and F-time values. The detailed statistical comparisons (including for *p-value* and *effect size*) are available online [43].

A. Comparisons of Failure-Detection Effectiveness

This section reports on the effectiveness comparisons between VPP-ART and FSCS-ART, and between VPP-ART and the three kinds of KDFC-ART. The results and main findings address RQ1, as follows.

1) Answer to RQ1—Part 1: Results of Simulations: Tables III–V present the simulation *ART F-ratio* value comparisons among FSCS-ART, KDFC-ART, and VPP-ART, for *block*, *strip*, and *point patterns*, respectively.

In general, for all five ART algorithms, the *ART F-ratio* values decrease as θ decreases, for both block and point patterns. This indicates that ART has better failure-detection effectiveness when the failure rate is small. As the dimension d increases, the *ART F-ratio* values generally increase, showing that d has an important negative impact on the effectiveness of ART algorithms.

a) VPP-ART versus FSCS-ART:

1) *Block-pattern simulation findings:* When considering a fixed failure rate θ, the VPP-ART *ART F-ratio* values increase with increases in the dimension d: The higher the dimension is, the worse the failure-detection effectiveness of VPP-ART is. When $1 \leq d \leq 4$, the failure-detection effectiveness of VPP-ART is significantly better than RT, in most cases. Except in some cases ($d = 4, \theta = 0.01, 0.005$), the *ART F-ratio* values are less than 1.0, but when d increases, then the VPP-ART failure-detection effectiveness becomes weaker than RT. When $1 \leq d \leq 5$, the failure-detection effectiveness of VPP-ART and FSCS-ART are similar. When $d = 8, 10$, the failure-detection effectiveness of VPP-ART is obviously better than that of FSCS-ART.
2) *Strip-pattern simulation findings:* When $d = 1$, both VPP-ART and FSCS-ART have better failure-detection effectiveness than RT, with *ART F-ratio* values of about 0.55. In other dimensions, regardless of failure rate θ, the *ART F-ratio* values of VPP-ART are similar to those of FSCS-ART, indicating that the failure-detection performance of VPP-ART is not significantly different from FSCS-ART.
3) *Point-pattern simulation findings:* Similar to the block pattern case, for a given failure rate θ, the failure-detection effectiveness of both VPP-ART and FSCS-ART decrease as the dimension d increases—as seen from the increasing *ART F-ratio* values for increasing dimension. When $d = 1, 2$, both VPP-ART and FSCS-ART outperform RT, but when $d \geq 3$, VPP-ART has performance similar to, or better than, FSCS-ART—reflected in VPP-ART having lower *ART F-ratio* values than FSCS-ART, in most cases. In summary, when $1 \leq d \leq 4$, there is no significant difference in the failure-detection effectiveness of VPP-ART

TABLE II
22 SUBJECT PROGRAMS USED IN EXPERIMENTS

No.	Program	Dimension (d)	Input domain		Size (LOC)	Mutant operators	Total faults
			From	To			
1	airy	1	−5000	5000	43	CR	1
2	bessj0	1	−300000	300000	28	AOR, ROR, SVR, CR	5
3	erfcc	1	−30000	30000	14	AOR, ROR, SVR, CR	4
4	probks	1	−50000	50000	22	AOR, ROR, SVR, CR	4
5	tanh	1	−500	500	18	AOR, ROR, SVR, CR	4
6	bessj	2	(2, −1000)	(300, 15000)	99	AOR, ROR, CR	4
7	gammq	2	(0, 0)	(1700, 40)	106	ROR, CR	4
8	sncndn	2	(−5000, −5000)	(5000, 5000)	64	ROR, CR	5
9	golden	3	(−100, −100, −100)	(60, 60, 60)	80	ROR, SVR, CR	5
10	plgndr	3	(10, 0, 0)	(500, 11, 1)	36	AOR, ROR, CR	5
11	cel	4	(0.001, 0.001, 0.001, 0.001)	(1, 300, 10000, 1000)	49	AOR, ROR, CR	3
12	el2	4	(0, 0, 0, 0)	(250, 250, 250, 250)	78	AOR, ROR, SVR, CR	9
13	calDay	5	(1, 1, 1, 1, 1800)	(12, 31, 12, 31, 2200)	37	SDL	1
14	complex	6	(−20, −20, −20, −20, −20, −20)	(20, 20, 20, 20, 20, 20)	68	SVR	1
15	pntLinePos	6	(−25, −25, −25, −25, −25, −25)	(25, 25, 25, 25, 25, 25)	23	CR	1
16	triangle	6	(−25, −25, −25, −25, −25, −25)	(25, 25, 25, 25, 25, 25)	21	CR	1
17	line	8	(−10, −10, −10, −10, −10, −10, −10, −10)	(10, 10, 10, 10, 10, 10, 10, 10)	86	ROR	1
18	pntTrianglePos	8	(−10, −10, −10, −10, −10, −10, −10, −10)	(10, 10, 10, 10, 10, 10, 10, 10)	68	CR	1
19	twoLinesPos	8	(−15, −15, −15, −15, −15, −15, −15, −15)	(15, 15, 15, 15, 15, 15, 15, 15)	28	CR	1
20	nearestDistance	10	(1, 1, 1, 1, 1, 1, 1, 1, 1, 1)	(15, 15, 15, 15, 15, 15, 15, 15, 15, 15)	26	CR	1
21	calGCD	10	(1, 1, 1, 1, 1, 1, 1, 1, 1, 1)	(1000, 1000, 1000, 1000, 1000, 1000, 1000, 1000, 1000, 1000)	24	AOR	1
22	select	11	(1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1)	(10, 100, 100, 100, 100, 100, 100, 100, 100, 100, 100)	117	RSR, CR	2

TABLE III
ART F-RATIO COMPARISON RESULTS FOR *BLOCK PATTERNS*

Dimension (d)	Failure rate (θ)	*VPP-ART*	*FSCS-ART*	*Naive-KDFC*	*SemiBal-KDFC*	*LimBal-KDFC*
$d=1$	0.01	**0.5634**	0.5729	0.5664	0.5714	0.5658
	0.005	0.5666	0.5633	0.5670	0.5696	**0.5619**
	0.002	0.5639	0.5683	0.5665	0.5723	**0.5605**
	0.001	0.5634	0.5720	**0.5549**	0.5690	0.5617
	0.0005	**0.5555**	0.5564	0.5629	0.5556	0.5619
	0.0002	**0.5520**	0.5700	0.5658	0.5662	0.5614
	0.0001	0.5766	0.5765	**0.5527**	0.5545	0.5569
$d=2$	0.01	**0.6820**	0.6911	0.6822	0.6904	0.6953
	0.005	0.6750	0.6613	**0.6561**	0.6635	0.6671
	0.002	0.6712	**0.6536**	0.6574	0.6633	0.6561
	0.001	0.6742	0.6573	**0.6449**	0.6557	0.6595
	0.0005	0.6510	**0.6391**	0.6525	0.6484	0.6492
	0.0002	0.6489	**0.6268**	0.6409	0.6414	0.6388
	0.0001	**0.6244**	0.6248	0.6531	0.6389	0.6313
$d=3$	0.01	0.8840	0.8641	0.8431	0.8504	**0.8391**
	0.005	0.8391	0.8314	**0.8176**	0.8195	0.8177
	0.002	0.8214	0.7847	**0.7778**	0.7948	0.8052
	0.001	0.8189	0.7735	**0.7720**	0.7735	0.7772
	0.0005	0.7553	0.7549	**0.7504**	0.7618	0.7615
	0.0002	0.7881	0.7499	0.7603	**0.7441**	0.7464
	0.0001	0.7688	0.7358	**0.7252**	0.7518	0.7387
$d=4$	0.01	1.0886	1.0786	1.0739	1.0711	**1.0666**
	0.005	1.0523	1.0272	1.0350	**1.0200**	1.0202
	0.002	0.9948	0.9606	**0.9497**	0.9711	0.9754
	0.001	0.9398	0.9155	**0.9122**	0.9190	0.9366
	0.0005	0.9390	0.9033	0.8908	**0.8904**	0.9067
	0.0002	0.8965	0.8522	**0.8494**	0.8651	0.8708
	0.0001	0.8857	0.8357	**0.8234**	0.8687	0.8491
$d=5$	0.01	1.3417	1.3346	1.3416	1.3268	**1.3209**
	0.005	1.2809	1.2694	1.2638	1.2632	**1.2550**
	0.002	1.1671	1.1661	1.1932	1.1685	**1.1550**
	0.001	1.1130	1.1097	1.1185	1.1317	**1.0850**
	0.0005	1.0605	1.0462	1.0498	1.0584	**1.0217**
	0.0002	1.0404	1.0156	**0.9930**	1.0215	1.0054
	0.0001	1.0223	0.9935	0.9833	**0.9810**	0.9867
$d=8$	0.01	**2.4832**	2.6802	2.6413	2.6390	2.5701
	0.005	**2.2558**	2.4032	2.4134	2.3672	2.2685
	0.002	**1.9843**	2.1176	2.1177	2.0986	2.0333
	0.001	**1.7889**	1.9526	1.9312	1.9525	1.8306
	0.0005	**1.6631**	1.8607	1.8474	1.8163	1.7219
	0.0002	**1.5212**	1.7096	1.7099	1.6956	1.5956
	0.0001	**1.3741**	1.6325	1.5772	1.6110	1.5027
$d=10$	0.01	**3.3475**	3.9718	3.9114	3.9454	3.8735
	0.005	**3.1357**	3.5995	3.4993	3.5591	3.4597
	0.002	**2.8812**	3.1565	3.1775	3.1184	2.9093
	0.001	**2.6297**	2.8741	2.8712	2.9142	2.6868
	0.0005	**2.2916**	2.7049	2.7083	2.7002	2.4310
	0.0002	**1.9882**	2.4118	2.3827	2.4727	2.1770
	0.0001	**1.8902**	2.2568	2.2235	2.3156	2.0289

TABLE IV
ART F-RATIO COMPARISON RESULTS FOR *STRIP PATTERNS*

Dimension (d)	Failure rate (θ)	*VPP-ART*	*FSCS-ART*	*Naive-KDFC*	*SemiBal-KDFC*	*LimBal-KDFC*
$d=1$	0.01	**0.5634**	0.5729	0.5664	0.5714	0.5658
	0.005	0.5666	0.5633	0.5670	0.5696	**0.5619**
	0.002	0.5639	0.5683	0.5665	0.5723	**0.5605**
	0.001	0.5634	0.5720	**0.5549**	0.5690	0.5617
	0.0005	**0.5555**	0.5564	0.5629	0.5556	0.5619
	0.0002	**0.5520**	0.5700	0.5658	0.5662	0.5614
	0.0001	0.5766	0.5765	0.5527	0.5545	0.5569
$d=2$	0.01	0.9302	0.9816	0.9365	0.9490	**0.9276**
	0.005	0.9434	0.9716	0.9521	**0.9279**	0.9456
	0.002	**0.9457**	0.9961	0.9749	0.9611	0.9859
	0.001	0.9852	0.9561	0.9783	0.9775	**0.9547**
	0.0005	0.9978	0.9784	0.9769	**0.9641**	0.9808
	0.0002	0.9871	0.9827	**0.9574**	0.9915	0.9811
	0.0001	0.9678	1.0130	0.9726	**0.9534**	0.9760
$d=3$	0.01	0.9515	0.9639	0.9606	0.9850	**0.9491**
	0.005	0.9844	**0.9404**	0.9817	0.9803	0.9809
	0.002	0.9946	0.9853	**0.9569**	0.9918	0.9653
	0.001	0.9861	**0.9514**	0.9852	0.9757	1.0010
	0.0005	1.0068	0.9978	0.9859	**0.9510**	0.9832
	0.0002	0.9862	0.9734	0.9834	**0.9730**	0.9974
	0.0001	0.9996	**0.9945**	1.0162	1.0572	1.0066
$d=4$	0.01	0.9984	0.9733	1.0022	0.9895	**0.9723**
	0.005	0.9913	0.9830	0.9971	0.9604	**0.9602**
	0.002	0.9949	1.0274	1.0084	**0.9749**	0.9919
	0.001	0.9839	0.9982	0.9874	**0.9767**	0.9807
	0.0005	0.9997	1.0038	1.0264	0.9968	**0.9792**
	0.0002	**0.9832**	1.0081	1.0013	1.0117	1.0206
	0.0001	**0.9693**	1.0268	0.9943	1.0038	0.9911
$d=5$	0.01	**0.9714**	1.0162	0.9736	0.9806	1.0228
	0.005	0.9787	1.0210	1.0097	1.0002	**0.9613**
	0.002	1.0196	1.0108	0.9807	**0.9871**	1.0363
	0.001	1.0095	**0.9791**	1.0039	1.0275	1.0236
	0.0005	0.9939	1.0236	1.0298	**0.9708**	1.0223
	0.0002	0.9987	**0.9751**	0.9881	1.0208	0.9881
	0.0001	0.9954	1.0039	**0.9648**	0.9953	0.9832
$d=8$	0.01	**0.9446**	0.9907	0.9836	0.9847	1.0045
	0.005	1.0329	1.0145	**0.9723**	1.0094	0.9781
	0.002	**0.9842**	0.9905	1.0159	1.0024	1.0316
	0.001	1.0189	1.0107	**0.9999**	1.0069	1.0411
	0.0005	1.0247	1.0123	1.0064	**0.9830**	0.9866
	0.0002	0.9792	1.0166	0.9967	**0.9596**	0.9942
	0.0001	**0.9932**	1.0207	1.0074	0.9979	0.9935
$d=10$	0.01	**0.9760**	1.0068	1.0196	0.9771	0.9967
	0.005	1.0035	1.0265	**0.9950**	1.0067	1.0089
	0.002	0.9856	0.9933	**0.9827**	0.9882	0.9974
	0.001	1.0031	1.0083	1.0074	0.9946	**0.9834**
	0.0005	1.0161	1.0054	**1.0052**	1.0388	1.0265
	0.0002	1.0008	1.0073	1.0096	**0.9823**	1.0246
	0.0001	**0.9743**	0.9945	1.0097	0.9832	1.0149

TABLE V
ART F-RATIO COMPARISON RESULTS FOR *POINT PATTERNS*

Dimension (d)	Failure rate (θ)	*VPP-ART*	*FSCS-ART*	*Naive-KDFC*	*SemiBal-KDFC*	*LimBal-KDFC*
$d=1$	0.01	**0.9592**	0.9607	0.9755	0.9621	0.9763
	0.005	**0.9262**	0.9543	0.9563	0.9576	0.9320
	0.002	**0.9355**	0.9568	0.9627	0.9825	0.9788
	0.001	1.0026	**0.9346**	0.9771	0.9623	0.9651
	0.0005	0.9779	**0.9380**	0.9815	0.9446	0.9422
	0.0002	0.9708	0.9693	0.9798	**0.9282**	0.9655
	0.0001	0.9750	0.9721	0.9807	0.9610	**0.9530**
$d=2$	0.01	0.9979	0.9988	0.9918	**0.9894**	1.0207
	0.005	**0.9662**	0.9762	1.0042	0.9825	0.9917
	0.002	0.9918	0.9675	0.9718	**0.9557**	0.9877
	0.001	0.9688	0.9995	**0.9550**	0.9672	0.9817
	0.0005	**0.9427**	0.9663	0.9650	0.9806	0.9777
	0.0002	0.9681	1.0034	0.9522	**0.9392**	0.9428
	0.0001	0.9758	0.9792	**0.9511**	0.9673	0.9556
$d=3$	0.01	**1.0376**	1.1231	1.0930	1.1084	1.0795
	0.005	**1.0609**	1.0744	1.0973	1.0665	1.1051
	0.002	1.0269	**1.0235**	1.0297	1.0746	1.0499
	0.001	1.0221	1.0343	**1.0151**	1.0548	1.0551
	0.0005	**0.9988**	1.0017	1.0121	1.0113	1.0077
	0.0002	1.0183	1.0093	**1.0036**	1.0122	1.0074
	0.0001	0.9807	1.0023	**0.9795**	0.9905	0.9824
$d=4$	0.01	**1.2336**	1.3211	1.2789	1.3035	1.3037
	0.005	**1.2100**	1.2614	1.2517	1.2633	1.2192
	0.002	1.1283	1.1809	1.1735	1.1524	**1.1212**
	0.001	**1.0915**	1.1137	1.1287	1.1401	1.1370
	0.0005	**1.0788**	1.1117	1.1062	1.0980	1.1065
	0.0002	**1.0444**	1.0521	1.1007	1.0487	1.0510
	0.0001	1.0506	1.0837	**1.0500**	1.0589	1.0509
$d=5$	0.01	**1.4384**	1.5695	1.5413	1.5603	1.5243
	0.005	**1.3364**	1.4785	1.4519	1.4385	1.4456
	0.002	**1.2637**	1.3549	1.3642	1.3691	1.3510
	0.001	**1.1862**	1.2964	1.2948	1.3005	1.2110
	0.0005	**1.1718**	1.2559	1.2236	1.2361	1.1938
	0.0002	1.1638	1.1746	1.1636	**1.1562**	1.1708
	0.0001	1.1276	1.1257	1.1474	1.1553	**1.1074**
$d=8$	0.01	**2.0945**	2.4049	2.3487	2.4215	2.3374
	0.005	**2.0220**	2.3711	2.3543	2.3313	2.2386
	0.002	**1.8539**	2.1827	2.2076	2.1710	2.0722
	0.001	**1.7151**	2.0761	2.1198	2.0804	1.9098
	0.0005	**1.6117**	1.9976	1.9948	1.9881	1.8038
	0.0002	**1.4995**	1.7979	1.7829	1.8695	1.6424
	0.0001	**1.4719**	1.7575	1.7223	1.7567	1.5938
$d=10$	0.01	**2.3399**	2.5080	2.5382	2.5738	2.5994
	0.005	**2.4368**	2.8216	2.6878	2.7286	2.7150
	0.002	**2.3325**	2.8479	3.0133	2.8899	2.6569
	0.001	**2.2444**	2.8835	2.8983	2.8322	2.6249
	0.0005	**2.0856**	2.7033	2.7247	2.6135	2.4393
	0.0002	**1.8928**	2.4606	2.4871	2.5065	2.2301
	0.0001	**1.7778**	2.0912	2.3241	2.3366	2.1341

and FSCS-ART; but when $d \geq 5$, VPP-ART has a much better performance.

b) VPP-ART versus KDFC-ART:

1) *Block-pattern simulation findings:* For a given failure rate θ, the failure-detection effectiveness of FSCS-ART, VPP-ART, and all three KDFC-ART versions appear similar, as the dimension d increases. When $1 \leq d \leq 5$, the three KDFC-ART versions have lower *ART F-ratio* values than VPP-ART, indicating that KDFC-ART has failure-detection similar, or better, in these dimensions. However, when $d = 8, 10$, VPP-ART has the lower *ART F-ratio* values, showing a much better failure-detection effectiveness. Overall, VPP-ART has similar failure-detection effectiveness to KDFC-ART in lower dimensional input domains, and better performance than Naive-KDFC and SemiBal-KDFC in higher dimensions. VPP-ART also performs similar to or better than LimBal-KDFC in some high-dimensional cases.

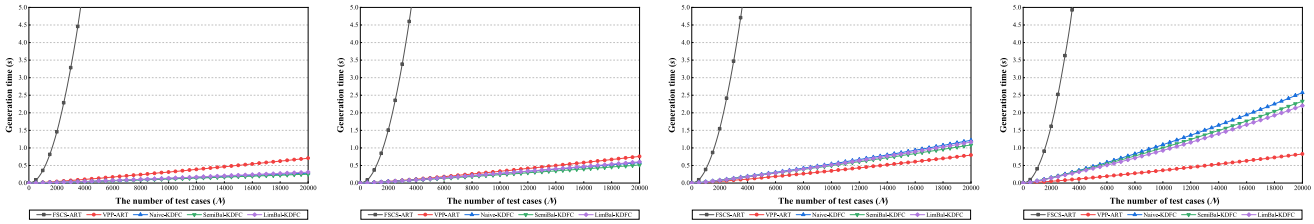
TABLE VI
F-MEASURE COMPARISON RESULTS FOR THE 22 SUBJECT PROGRAMS

Program	d	*VPP-ART*	*RT*	*FSCS-ART*	*Naive-KDFC*	*SemiBal-KDFC*	*LimBal-KDFC*
airy	1	**797.28**	1448.70	816.03	806.91	803.29	809.33
bessj0	1	450.05	758.91	448.20	443.44	**440.31**	449.13
erfcc	1	**1019.32**	1832.56	1054.65	1040.58	1045.86	1033.00
probks	1	1452.91	2683.85	1469.21	**1450.82**	1452.57	1475.86
tanh	1	312.42	566.12	319.82	**306.91**	309.85	309.36
bessj	2	462.96	784.93	**452.49**	457.52	457.60	461.69
gammq	2	**1063.88**	1208.68	1087.52	1066.34	1100.74	1172.50
sncndn	2	640.22	**629.03**	643.40	629.63	649.75	655.74
golden	3	1824.80	1881.15	1831.29	**1806.09**	1816.82	1902.52
plgndr	3	1733.83	2636.11	**1572.82**	1648.26	1618.40	1665.89
cel	4	1628.50	3049.35	**1572.56**	1577.88	1593.71	1586.13
el2	4	796.70	1381.35	714.58	**710.58**	714.01	749.98
calDay	5	**1101.30**	1394.67	1312.37	1295.35	1262.40	1226.75
complex	6	1134.81	**1120.51**	1283.25	1155.82	1150.68	1142.01
pntLinePos	6	1397.57	**1318.12**	1589.92	1444.04	1490.97	1458.47
triangle	6	1411.86	1430.20	1396.55	1415.63	1389.04	**1324.95**
line	8	3322.27	**3178.10**	3435.07	3343.48	3269.93	3370.50
pntTrianglePos	8	**4584.95**	4708.16	5067.49	5046.12	4955.54	4659.86
TwoLinesPos	8	7415.10	**6226.52**	9814.67	8297.09	8430.90	8909.71
nearestDistance	10	**2145.31**	3964.67	2259.57	2277.88	2161.20	2188.53
calGCD	10	**1004.15**	1054.66	1016.98	1016.14	1017.74	1056.02
select	11	5490.70	**4759.86**	5599.86	5907.50	5634.03	5808.85

2) *Strip-pattern simulation findings:* Similar to other ART algorithms, there is no significant difference in *ART F-ratio* values among VPP-ART and KDFC-ART. When $d = 1$, VPP-ART and KDFC-ART have lower *F-measures* than RT. In other dimensions, the *ART F-ratio* values are about 1.0, indicating similar failure-detection effectiveness to RT.
3) *Point-pattern simulation findings:* For a given failure rate θ, as the dimension d increases, the failure-detection effectiveness of both VPP-ART and KDFC-ART declines. When $d = 1, 2$, the effectiveness of VPP-ART and KDFC-ART is better than that of RT, with their *ART F-ratio* values being less than 1.0; when $d \geq 3$, the VPP-ART *ART F-ratio* values are similar to or better than those of KDFC-ART. In summary, when the dimension is low, VPP-ART and KDFC-ART have comparable failure-detection effectiveness; and in high dimensions, VPP-ART performs better.

Discussion of effectiveness simulation results: We examined the failure-detection effectiveness of VPP-ART, FSCS-ART, and three versions of KDFC-ART through simulations. Naive-KDFC and SemiBal-KDFC search for the exact NN of a candidate test case (the same as FSCS-ART); but VPP-ART and LimBal-KDFC use approximate NN searches. Through the simulations, it was found that VPP-ART has a similar failure-detection effectiveness to FSCS-ART and other ART algorithms that use the exact NN search approach, when the input domain dimension is low, and has better effectiveness in high dimensions. There was no significant difference between VPP-ART and LimBal-KDFC, in low dimensions, in terms of their failure-detection effectiveness. However, in high dimensions, VPP-ART can perform comparably or better than LimBal-KDFC.

2) Answer to RQ1–Part 2: Results of Experiments: Table VI presents the *ART F-ratio* data for VPP-ART, FSCS-ART, and KDFC-ART with the 22 subject programs.



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

Fig. 4. Generation times for various test suite sizes in (a) $d = 2$. (b) $d = 3$. (c) $d = 4$. (d) $d = 5$.

a) VPP-ART versus RT: In 16 of the 22 programs, VPP-ART has a lower *F-measure* than RT, indicating that VPP-ART can generate more effective test cases than RT, which is consistent with the simulation results.

b) VPP-ART versus FSCS-ART: VPP-ART has lower *F-measures* than FSCS-ART in 16 of the 22 real-life programs (73%), indicating that VPP-ART can effectively ensure or improve on the failure-detection effectiveness of FSCS-ART in real-life programs. Generally speaking, VPP-ART performs similarly to or slightly worse than FSCS-ART in low-dimensional programs, but has comparable or better failure-detection effectiveness in high-dimensional programs—from the perspective of the *F-measure*, VPP-ART may require fewer test cases to find the first program failure in high-dimensional programs.

c) VPP-ART versus KDFC-ART: In low-dimensional input domain programs, KDFC-ART can usually achieve lower *F-measure* values, but this changes for high dimensions, when VPP-ART performs best.

Discussion of effectiveness experiments using subject programs: In the experiments using real-life programs, the VPP-ART failure-detection effectiveness was found to be comparable to that of other ART algorithms. VPP-ART had comparable failure-detection effectiveness in low-dimensional programs, and often had better performance than other ART algorithms with the high-dimensional programs—VPP-ART required fewer test cases to detect the first failure in the high-dimensional programs.

Summary and Discussion for Answer to RQ1: Both the simulations and experimental data suggest that VPP-ART can achieve comparable, or better, failure-detection effectiveness, especially in higher dimensions.

B. Comparisons of Efficiency

This section reports on the efficiency comparisons between VPP-ART and the other ART algorithms. The results and main findings address RQ2, as follows.

1) Answer to RQ2—Part 1: Results of Simulations: Fig. 4 shows the test case generation times of VPP-ART, FSCS-ART, and the three KDFC-ART algorithms, for various test suite sizes, in different dimensional input domains. In the figures, the x-axis shows the size of the test suite (N), and the y-axis shows the time taken to generate the N test cases. Due to the page limit constraints, only the results for $d = 2$, 3, 4, and 5 are provided here. The results for $d = 1$, $d = 2$, and $d = 3$ are similar; as are those for $d \geq 4$. More details can be found online [43].

a) VPP-ART versus FSCS-ART: VPP-ART has a significant advantage over FSCS-ART in terms of test case generation time, across different input domain dimensions. Compared with FSCS-ART, VPP-ART uses VPP and approximate NN strategies to increase the efficiency. The simulation results indicate that the VPP-ART test case generation time is not obviously impacted by increases in dimension: VPP-ART maintains a strong ability to reduce the time overheads in high dimensions.

b) VPP-ART versus KDFC-ART: Fig. 4 shows that when $d = 3$, there is little difference in test case generation time for the three KDFC-ART versions, all of which have better performance than VPP-ART. However, when the input domain dimension increases to $d > 3$, the KDFC-ART performance becomes worse than VPP-ART (as shown in [43]).

As the dimension increases ($d \geq 5$), the performance differences among the three KDFC-ART versions gradually increase, with LimBal-KDFC emerging as the best. However, the performance of all three KDFC-ART versions is worse than that of VPP-ART, with the difference increasing as the dimension increases (as shown in [43]).

Discussion of efficiency simulation results: The simulation results show that VPP-ART requires much less time than FSCS-ART to generate an equal number of test cases. VPP-ART performs slightly worse than KDFC-ART when the dimension is low, but increasingly becomes the better performer as the dimensions increase. The two ART algorithms based on approximate NN (LimBal-KDFC and VPP-ART) have better test case generation time. As the number of test cases increases, the time cost of the approximate NN search is lower than that of the exact NN search.

2) Answer to RQ2—Part 2: Results of Experiments: This section reports on the investigation into the efficiency of the five ART algorithms (VPP-ART, FSCS-ART, Naive-KDFC, SemiBal-KDFC, and LimBal-KDFC) on the 22 subject programs. Table VII reports the average time taken to detect the first failure (*F-time*) in each program, by each algorithm. In the table, the values in bold blue typeface represent the minimum *F-time* values in the comparison among VPP-ART and the other ART algorithms (RT is not compared here). Based on the data, the main findings are as follows.

a) VPP-ART versus RT: For all 22 programs, the time cost of RT was significantly lower than that of VPP-ART. That is,

TABLE VII
F-TIME COMPARISON RESULTS FOR THE 22 SUBJECT PROGRAMS

Program	d	*VPP-ART*	*RT*	*FSCS-ART*	*Naive-KDFC*	*SemiBal-KDFC*	*LimBal-KDFC*
airy	1	12.74	0.34	329.08	**3.31**	3.33	3.71
bessj0	1	6.38	0.17	98.95	**1.66**	1.66	1.76
erfcc	1	17.33	0.49	529.69	**4.46**	4.60	4.80
probks	1	37.18	20.37	1081.17	18.11	**17.95**	18.57
tanh	1	3.98	0.10	50.72	**1.08**	1.17	1.21
bessj	2	7.55	0.70	132.59	8.43	**3.08**	3.97
gammq	2	20.60	0.45	775.74	16.10	**13.08**	14.28
sncndn	2	11.18	0.23	287.05	**5.53**	9.03	9.06
golden	3	45.73	3.60	2417.30	**45.16**	72.10	66.76
plgndr	3	42.52	7.79	1714.59	94.44	**19.43**	21.65
cel	4	36.14	0.82	1858.20	155.37	**23.46**	28.03
el2	4	**15.42**	0.46	379.12	25.32	29.34	28.23
calDay	5	**27.57**	2.19	968.30	169.88	36.91	36.18
complex	6	**27.62**	0.69	1317.92	136.16	136.54	113.95
pntLinePos	6	**34.46**	0.52	1895.50	182.67	193.30	155.95
triangle	6	**35.23**	0.61	1562.98	177.93	180.97	142.86
line	8	**102.64**	1.57	10274.08	1542.17	1659.12	936.30
pntTrianglePos	8	**153.93**	2.86	21846.93	2864.84	3077.41	1450.36
TwoLinesPos	8	**272.39**	3.27	70499.32	6015.84	6919.79	3177.67
nearestDistance	10	**69.81**	2.76	3493.54	3113.37	3467.71	1067.29
calGCD	10	**27.96**	2.48	916.10	483.69	554.96	364.48
select	11	**216.93**	5.93	26894.43	24325.93	23561.25	4338.21

because there is no need to calculate distances among test cases, RT requires less time to find the first failure.

b) VPP-ART versus FSCS-ART: The experimental data show the VPP-ART *F-time* results to be much lower than those of FSCS-ART, which shows a significant difference between VPP-ART and FSCS-ART. In summary, VPP-ART greatly improves on the testing efficiency of FSCS-ART for the real-life subject programs.

c) VPP-ART versus KDFC-ART: The results show that when the input domain dimension is low, all three KDFC-ART versions require less time than VPP-ART to find the first failure. When $d = 1, 2$, Naive-KDFC has better performance than VPP-ART, except for the `bessj` program. For higher dimensions, according to the *effect size* scores, VPP-ART outperforms Naive-KDFC. When $1 \leq d \leq 4$, except for the `golden` and `el2` programs, SemiBal-KDFC and LimBal-KDFC outperform VPP-ART. For the other programs/dimensions, VPP-ART outperforms SemiBal-KDFC and LimBal-KDFC, which shows that the differences in VPP-ART and KDFC-ART performances are significant.

Discussion of efficiency experiments using subject programs: In general, for all subject programs, VPP-ART outperforms FSCS-ART, in terms of the time taken to find the first failure. In low dimensions, KDFC-ART shows better efficiency than VPP-ART, but as the dimension increases, VPP-ART outperforms KDFC-ART. Due to the distance calculations, all ART algorithms take longer than RT.

Summary and Discussions for Answer to RQ2: VPP-ART is more efficient than FSCS-ART, according to both the simulation and experimental results. KDFC-ART is more efficient in low-dimensional input domains, but VPP-ART outperforms KDFC-ART when the dimension is higher.

Conclusion: Compared with FSCS-ART, the proposed VPP-ART approach not only shows comparable or better failure-detection effectiveness in most cases, but also significantly improves on the FSCS-ART efficiency in most cases. VPP-ART has similar failure-detection effectiveness to KDFC-ART, outperforming KDFC-ART in higher dimensions. VPP-ART, due to the use of VPP and the approximate NN search strategy, has the following advantages.

1) VPP-ART can generate an equivalent number of test cases to FSCS-ART in very short time.
2) VPP-ART can find software failures more efficiently than FSCS-ART.
3) VPP-ART can relieve the influence of dimension on the efficiency of KDFC-ART, with VPP-ART not showing performance degradation as the input domain dimension increases.

In conclusion, VPP-ART is more cost-effective than FSCS-ART, and in high dimensions, VPP-ART is more cost-effective than KDFC-ART.

VI. THREATS TO VALIDITY

A. Construct Validity

Construct validity refers to how well a study examines and measures its assertions. In this article, we used the *F-measure* (and related *ART F-ratio*) [14] to measure the failure-detection effectiveness of RT and different ART algorithms. In addition to the *F-measure*, two other failure-detection effectiveness metrics that are commonly used are the *E-measure* and the *P-measure* [15]. The *E-measure* refers to the expected number of failures to be identified by a set of test cases, and the *P-measure* is the probability of a test set identifying at least one program failure. The *E-measure* and *P-measure* are most suitable for the evaluation of automated testing systems when the number of test cases is fixed [60]. The *F-measure*, in contrast, is more suitable when testing continues until a failure is detected [12]. Previous studies have shown that ART outperforms RT in terms of the *P-measure* [61]. A criticism of the *E-measure* is that multiple failures may be associated with a single fault [60]. In this article, we examined FSCS-ART and the enhanced RT algorithms VPP-ART and KDFC-ART in simulations and experiments, with the algorithm stopping whenever a (first) program failure was detected. Accordingly, the *F-measure* was the metric used to evaluate the failure-finding effectiveness of the three algorithms.

B. External Validity

External validity refers to the extent to which our experiments are generally valid, and to what degree can the results and findings be generalized.

The performance of the algorithm is mainly affected by the input domain dimension, and the failure pattern and rate. The subject programs selected in this study have been widely used in ART research. The programs have multiple input domain dimensions, and can thus be used to effectively reflect the performance of the algorithm for different dimensions. The programs also have common failure patterns and failure rate ranges, and can thus also be used to effectively evaluate the algorithm for real failure patterns and rates. Although the actual program size was not considered to be a very influential factor for this study, our

future work will involve other programs with a wider variety of scale and size.

Because VPP-ART is an application in Euclidean space, it currently only supports numeric programs. The VPP strategy proposed in this article uses input domain partitioning and an approximate NN search based on the distances among test cases. Distance calculations between two points in Euclidean space are relatively easy to calculate and compare. However, for other types of programs (such as object-oriented programs and deep learning programs), digitization and vectorization of the relevant inputs may be required before the VPP strategy could be applied. The impact of this preprocessing is unknown, as is the applicability of Euclidean distance measurements in this context. This exploration of the application of VPP to non-Euclidean space will form part of our future research.

C. Internal Validity

Internal validity refers to the accuracy and completeness of the experiments. Each of our experiments was repeated 3000 times, allowing confidence in the calculated average data. Different parameter settings for VPP-ART will lead to different outcomes, which is also a potential threat. For example, we recommended setting the parameter pair values as $\langle \varepsilon, \lambda \rangle = \langle 3, 10 \rangle$: VPP-ART may produce different behavior and results when different parameter values are chosen. However, the exhaustive enumeration and examination of every possible parameter value combination is clearly not possible. Nevertheless, although we are confident of our experimentally informed parameter-value choices, the further exploration of these parameter-pair values will be part of our future work.

VII. Related Work

Many techniques have been proposed to address the ART computational overhead problem, including *forgetting* [20], ART-DC [24], DMART [25], DF-FSCS-ART [21], KDFC-ART [22], and *FSCS-ART using SIMD instructions* (FSCS-SIMD) [23].

A. Forgetting

With ART, generation of the next test case depends on the $|E|$ test cases that were already executed, without revealing any failures. As the number of test cases in E grows, the computational overhead of generating test cases becomes greater. Chan et al. [20] proposed a *forgetting* strategy that only examines up to a constant M number of executed test cases when generating new test cases, typically with M being $\ll |E|$. The forgetting strategy thus makes generation of the next test case independent of the size of E. VPP-ART uses the concept of partitioning, with all $|E|$ test cases, but only λ test cases are considered in the search process. Both forgetting and VPP-ART can reduce the searching scale of E, but there is a difference between the algorithms: The forgetting strategy ignores some executed test cases without considering the relationship between them; with VPP-ART, however, the attribute (distance) information of each test case is indexed by vantage points, thereby implicitly including the information when reducing the search scale.

B. DF-FSCS-ART

DF-FSCS-ART [21] uses information about the distribution of executed test cases, ignoring those not in the *sight* of a given candidate test case. The input domain is first divided into $p \times p$ subdomains (where p is a predefined parameter). During test case generation, FSCS-ART is applied to the entire input domain, but for each candidate test case, only those executed test cases lying in adjacent subdomains are included in the distance calculation process. When DF-FSCS-ART is dividing the input domain, as long as the division conditions are satisfied, a new round of division will be performed. To avoid too many test cases within the subdomains, DF-FSCS-ART randomly filters them to keep the number below a threshold value τ.

DF-FSCS-ART faces some challenges as follows. 1) Two parameters (p and τ) need to be set before testing, with different values of these two parameters causing the performance of DF-FSCS-ART to vary widely. This is also a threat to our proposed VPP-ART, which requires that the parameter pair $\langle \varepsilon, \lambda \rangle$ be set before testing. 2) As the number of executed test cases increases, more distance calculations need to be performed, increasing the time overheads. Although VPP-ART and DF-FSCS-ART share the same goal of reducing the FSCS-ART computational overheads, the dimension and number of executed test cases are significantly stronger limitations for DF-FSCS-ART: As seen in this article, the VPP-ART time cost changes very little in response to increases in the executed test set size and input domain dimension.

C. ART-DC

ART-DC [24] uses standard ART procedures to generate test cases in the entire input domain. When the number of test cases generated reaches the preset threshold δ, each dimension will be divided equally, resulting in the entire input domain being divided into small, equal-sized subdomains. Finally, the same ART processes are used independently in each subdomain to generate test cases. ART-DC has been reported to effectively reduce the time overheads, while ensuring the effectiveness of ART [24].

ART-DC and VPP-ART both have a similar drawback: Certain parameters need to be set and adjusted to ensure the effectiveness of ART. A higher threshold parameter δ for ART-DC can deliver better effectiveness. Similarly, a smaller parameter pair $\langle \varepsilon, \lambda \rangle$ for VPP-ART will have better test effectiveness. Nevertheless, the core concepts used by ART-DC and VPP-ART are quite different: 1) ART-DC uses the idea of divide-and-conquer, while VPP-ART uses the distance relationship to realize the tree spatial index structure (VP-tree). ART-DC divides the input domain equally, with the size of each subdomain being the same, but VPP-ART uses concentric hyper-spheres, with no concept of equal partitioning. 2) After the input domain is divided, ART-DC generates test cases in each subdomain, while VPP-ART continues to generate test cases in the entire input domain. Generally speaking, on the one hand, both ART-DC

and VPP-ART are affected by the selection of parameter values. On the other hand, VPP-ART conducts ART in the entire input domain, but accelerates the (approximate) NN search process, while ART-DC applies ART in each sub-domain.

In theory, ART-DC addresses, to some extent, the boundary-area test case generation bias by ensuring that test cases are generated in each subdomain in the entire input domain. However, VPP-ART may still have a bias toward generating test cases near the input domain boundary—test cases are not generated independently in each subdomain, but throughout the entire input domain (with a preference for the boundary area). Therefore, ART-DC may have better diversity in the test case distribution than VPP-ART.

D. DMART

DMART [25] was proposed to overcome some shortcomings of Mirror ART (MART) [26]. DMART selects half of the subdomains to generate test cases using a specific ART algorithm, and then mirrors these test cases to the other half of the subdomains to generate the remaining test cases. On the one hand, DMART uses a mechanism similar to ART-DC to divide the input domain, which is obviously different from VPP-ART. On the other hand, similar to ART-DC and VPP-ART, DMART also requires parameter setting, i.e., the maximum number of test cases required to be generated in each subdomain, δ.

E. KDFC-ART

To reduce the high computational overheads of FSCS-ART, Mao et al. [22] proposed three KDFC-ART algorithms, based on the KD-tree [62], [63], [64], [65] structure: *Naive-KDFC*, *SemiBal-KDFC*, and *LimBal-KDFC*. *Naive-KDFC* sequentially divides the input space in each dimension to construct the KD-tree. To improve the balance of the KD-tree, *SemiBal-KDFC* prioritizes the partitioning using a splitting strategy [22]. Because the KD-tree structure has a drawback that, in the worst case (especially when the dimension increases), all nodes may need to be traversed, the third algorithm, *LimBal-KDFC*, was proposed: *LimBal-KDFC* uses an upper limit to control the number of traversed nodes in the backtracking. *Naive-KDFC* and *SemiBal-KDFC* are based on an exact NN search, while *LimBal-KDFC* uses an approximate NN search—the same as VPP-ART. As shown in this article, both VPP-ART and KDFC-ART focus on reducing the high computational overheads of ART, and each has its own advantages and disadvantages. VPP-ART shows the improvement in high-dimensional efficiency due to its use of VPP for distances among test cases: VPP-ART test case generation time changes little as the dimension increases.

F. FSCS-SIMD

Ashfaq et al. [23] proposed an efficient FSCS-ART implementation, FSCS-SIMD, which uses the SIMD instruction architecture to calculate the distances among multiple test cases simultaneously, in a many-to-many manner. FSCS-SIMD loads a batch of multiple test cases from the candidate and executed test case sets at once. FSCS-SIMD accelerates the distance computation by increasing the CPU execution utilization, but is dependent on the available hardware resources [23]. VPP-ART, in contrast, improves the efficiency at the level of the algorithm and data structures. In poorer hardware environments (such as with a low number of CPU cores), FSCS-SIMD has greater limitations than VPP-ART.

VIII. Conclusion

ART [13] improves on the failure-detection effectiveness of RT [6], by making the test cases more evenly distributed in the input domain. The FSCS-ART [13] was among the first ART implementations, and has remained among the most popular [12]: FSCS-ART is widely used in practice, and has been shown to have better failure-detection effectiveness [13], [14], [15], [16], test-case distribution [17], and code coverage [18] than RT. However, FSCS-ART still has a significant computational overhead problem [12]: The time required to generate/select FSCS-ART test cases is much greater than their execution time. In this article, we have proposed the VPP-ART algorithm, which makes use of VPP and an approximate NN search strategy. VPP-ART achieves a significant reduction in the time overheads for FSCS-ART while maintaining its effectiveness.

We conducted a series of simulations and experimental studies to validate the VPP-ART approach, using FSCS-ART and three KDFC-ART algorithms for comparison. The simulations showed that VPP-ART not only requires less time to detect software failures compared with FSCS-ART, but also retains comparable failure-detection effectiveness. For selected values of ε and λ, VPP-ART and KDFC-ART are, to some extent, complementary in failure-detection effectiveness: VPP-ART performs better in high dimensions, and KDFC-ART performs better in low dimensions (although their performance in low dimensions is similar). In terms of efficiency, KDFC-ART has lower overheads in low dimensions, but VPP-ART performs better in high dimensions. Furthermore, the VPP-ART time overheads only increase slightly for increases in the input-domain dimension. VPP-ART effectively reduces the computational overhead problem of FSCS-ART, and is cost-effective.

The ART algorithms based on exact NN searches have better failure-detection effectiveness in lower dimensional input domains, while those based on approximate NN are better in higher dimensions. This is because of the clustering of similar test cases, which is more pronounced as the dimension and number of test cases increases, resulting in more similar test cases clustering in the same subdomain. Moreover, when there is little difference between the approximate and the exact NNs, the approximate NN can replace the exact NN, improving the search efficiency of test case generation at the cost of a tolerable loss in accuracy.

In this article, we introduced a modified VP-tree to support the dynamic nature of FSCS-ART. Because many other ART approaches and implementations also suffer from the same drawbacks and overheads as FSCS-ART, our future work will include exploration of application of the insertable VP-tree to these other ART algorithms. Furthermore, as noted in Section VI, further exploration of the VPP-ART parameter settings for λ and ε is

also one of our future research directions. The influence of the parameter pairs (λ and ε) on the effectiveness of the experiments has been discussed, with the experimental parameter pair values determined from the simulations. However, because there has not yet been much research into these two parameters, exploring the functional or dependency relationship between them, so that our proposed insertable VP-tree structure can be better applied, will also form part of our future work. VPP-ART is an ART algorithm based on an approximate NN search (like LimBal-KDFC): Our experimental analysis has shown that VPP-ART and LimBal-KDFC can effectively improve on the high-dimensional failure-detection effectiveness of the original ART algorithm, and greatly reduce the test case generation time. This gives us some inspiration to identify and create other algorithms or data structures that may be enhanced through an approximate, rather than exact, NN search.

ACKNOWLEDGMENT

The authors would like to thank the anonymous reviewers for their many constructive comments.

REFERENCES

[1] S. Anand et al., "An orchestrated survey of methodologies for automated software test case generation," *J. Syst. Softw.*, vol. 86, no. 8, pp. 1978–2001, 2013.
[2] A. Orso and G. Rothermel, "Software testing: A research travelogue (2000-2014)," in *Proc. Future Softw. Eng.*, 2014, pp. 117–132.
[3] E. J. Weyuker, "On testing non-testable programs," *Comput. J.*, vol. 25, no. 4, pp. 465–470, 1982.
[4] E. T. Barr, M. Harman, P. McMinn, M. Shahbaz, and S. Yoo, "The Oracle problem in software testing: A survey," *IEEE Trans. Softw. Eng.*, vol. 41, no. 5, pp. 507–525, 2015.
[5] G. Jahangirova, "Oracle problem in software testing," in *Proc. 26th ACM SIGSOFT Int. Symp. Softw. Testing Anal.*, 2017, pp. 444–447.
[6] A. Arcuri, M. Iqbal, and L. Briand, "Random testing: Theoretical results and practical implications," *IEEE Trans. Softw. Eng.*, vol. 38, no. 2, pp. 258–277, 2012.
[7] B. K. Ozkan, R. Majumdar, F. Niksic, M. T. Befrouei, and G. Weissenbacher, "Randomized testing of distributed systems with probabilistic guarantees," *Proc. ACM Program. Lang.*, vol. 2, no. OOPSLA, pp. 160:1–160:28, 2018.
[8] V. Livinskii, D. Babokin, and J. Regehr, "Random testing for C and C compilers with YARPGen," *Proc. ACM Program. Lang.*, vol. 4, no. OOPSLA, pp. 196:1–196:25, 2020.
[9] T. D. White, G. Fraser, and G. J. Brown, "Improving random GUI testing with image-based widget detection," in *Proc. 28th ACM SIGSOFT Int. Symp. Softw. Testing Anal.*, 2019, pp. 307–317.
[10] H. Pei, K. Cai, B. Yin, A. P. Mathur, and M. Xie, "Dynamic random testing: Technique and experimental evaluation," *IEEE Trans. Rel.*, vol. 68, no. 3, pp. 872–892, 2019.
[11] H. Pei, B. Yin, M. Xie, and K. Cai, "Dynamic random testing with test case clustering and distance-based parameter adjustment," *Inf. Softw. Technol.*, vol. 131, 2021, Art. no. 106470.
[12] R. Huang, W. Sun, Y. Xu, H. Chen, D. Towey, and X. Xia, "A survey on adaptive random testing," *IEEE Trans. Softw. Eng.*, vol. 47, no. 10, pp. 2052–2083, 2021.
[13] T. Y. Chen, H. Leung, and I. K. Mak, "Adaptive random testing," in *Proc. 9th Asian Comput. Sci. Conf.*, 2004, pp. 320–329.
[14] T. Y. Chen, F.-C. Kuo, and R. G. Merkel, "On the statistical properties of the F-measure," in *Proc. 4th Int. Conf. Qual. Softw.*, 2004, pp. 146–153.
[15] T. Y. Chen, F. Kuo, and R. G. Merkel, "On the statistical properties of testing effectiveness measures," *J. Syst. Softw.*, vol. 79, no. 5, pp. 591–601, 2006.
[16] T. Y. Chen, F.-C. Kuo, and Z. Q. Zhou, "On favourable conditions for adaptive random testing," *Int. J. Softw. Eng. Knowl. Eng.*, vol. 17, no. 6, pp. 805–825, 2007.
[17] T. Y. Chen, F.-C. Kuo, and H. Liu, "On test case distributions of adaptive random testing," in *Proc. 19th Int. Conf. Softw. Eng. Knowl. Eng.*, 2007, pp. 141–144.
[18] T. Y. Chen, F.-C. Kuo, H. Liu, and W. E. Wong, "Code coverage of adaptive random testing," *IEEE Trans. Rel.*, vol. 62, no. 1, pp. 226–237, 2013.
[19] H. Wu, C. Nie, J. Petke, Y. Jia, and M. Harman, "An empirical comparison of combinatorial testing, random testing and adaptive random testing," *IEEE Trans. Softw. Eng.*, vol. 46, no. 3, pp. 302–320, 2020.
[20] K.-P. Chan, T. Y. Chen, and D. Towey, "Forgetting test cases," in *Proc. 30th Annu. Int. Comput. Softw. Appl. Conf.*, 2006, pp. 485–494.
[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," *Sci. China Inf. Sci.*, vol. 60, no. 9, pp. 092106:1–092106:21, 2017.
[22] C. Mao, X. Zhan, T. H. Tse, and T. Y. Chen, "KDFC-ART: A KD-tree approach to enhancing fixed-size-candidate-set adaptive random testing," *IEEE Trans. Rel.*, vol. 68, no. 4, pp. 1444–1469, 2019.
[23] M. Ashfaq, R. Huang, and M. Omari, "FSCS-SIMD: An efficient implementation of fixed-size-candidate-set adaptive random testing using SIMD instructions," in *Proc. 31st IEEE Int. Symp. Softw. Rel. Eng.*, 2020, pp. 277–288.
[24] 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 *Proc. 13th Int. Conf. Qual. Softw.*, 2013, pp. 268–275.
[25] R. Huang, H. Liu, X. Xie, and J. Chen, "Enhancing mirror adaptive random testing through dynamic partitioning," *Inf. Softw. Technol.*, vol. 67, pp. 13–29, 2015.
[26] T. Y. Chen, F.-C. Kuo, R. Merkel, and S. P. Ng, "Mirror adaptive random testing," *Inf. Softw. Technol.*, vol. 46, no. 15, pp. 1001–1010, 2004.
[27] P. N. Yianilos, "Data structures and algorithms for nearest neighbor search in general metric spaces," in *Proc. 4th Annu. ACM/SIGACT-SIAM Symp. Discrete Algorithms*, 1993, pp. 311–321.
[28] J. K. Uhlmann, "Satisfying general proximity/similarity queries with metric trees," *Inf. Process. Lett.*, vol. 40, no. 4, pp. 175–179, 1991.
[29] T. Chiueh, "Content-based image indexing," in *Proc. 20th Int. Conf. Very Large Data Bases*, 1994, pp. 582–593.
[30] L. J. White and E. I. Cohen, "A domain strategy for computer program testing," *IEEE Trans. Softw. Eng.*, vol. SE-6, no. 3, pp. 247–257, 1980.
[31] P. E. Ammann and J. C. Knight, "Data diversity: An approach to software fault tolerance," *IEEE Trans. Comput.*, vol. 37, no. 4, pp. 418–425, 1988.
[32] G. B. Finelli, "NASA software failure characterization experiments," *Rel. Eng. Syst. Saf.*, vol. 32, no. 1/2, pp. 155–169, 1991.
[33] P. G. Bishop, "The variation of software survival times for different operational input profiles," in *Proc. 23rd Int. Symp. Fault-Tolerant Comput.*, 1993, pp. 98–107.
[34] C. Schneckenburger and J. Mayer, "Towards the determination of typical failure patterns," in *Proc. 4th Int. Workshop Softw. Qual. Assurance: Conjunction 6th Joint Meeting Eur. Softw. Eng. Conf. ACM SIGSOFT Int. Symp. Found. Softw. Eng.*, 2007, pp. 90–93.
[35] F. T. Chan, T. Y. Chen, I. K. Mak, and Y. T. Yu, "Proportional sampling strategy: Guidelines for software testing practitioners," *Inf. Softw. Technol.*, vol. 38, no. 12, pp. 775–782, 1996.
[36] J. S. Sánchez, J. M. Sotoca, and F. Pla, "Efficient nearest neighbor classification with data reduction and fast search algorithms," in *Proc. IEEE Int. Conf. Syst., Man, Cybern.*, 2004, pp. 4757–4762.
[37] T. DeFreitas, H. Saddiki, and P. Flaherty, "GEMINI: A computationally-efficient search engine for large gene expression datasets," *BMC Bioinf.*, vol. 17, 2016, Art. no. 102.
[38] I. Markov, "VP-tree: Content-based image indexing," in *Proc. 4th Spring Young Researcher's Colloq. Database Inf. Syst.*, vol. 256, 2007.
[39] M. Shishibori, S. S. Lee, and K. Kita, "A method to improve metric index VP-tree for multimedia databases," in *Proc. 17th Korea- Jpn. Joint Workshop Front. Comput. Vis.*, 2011, pp. 1–6.
[40] M. Skalak, J. Han, and B. Pardo, "Speeding melody search with vantage point trees," in *Proc. 9th Int. Conf. Music Inf. Retrieval*, 2008, pp. 95–100.
[41] N. Kumar, L. Zhang, and S. K. Nayar, "What is a good nearest neighbors algorithm for finding similar patches in images?," in *Proc. 10th Eur. Conf. Comput. Vis.*, 2008, pp. 364–378.
[42] A. W. Fu, P. M. Chan, Y. L. Cheung, and Y. S. Moon, "Dynamic VP-tree indexing for n-nearest neighbor search given pair-wise distances," *VLDB J.*, vol. 9, no. 2, pp. 154–173, 2000.
[43] R. Huang, C. Cui, D. Towey, W. Sun, and J. Lian, "More details for VPP-ART," 2022. [Online]. Available: https://github.com/huangrubing/VPP-ART

[44] T. Y. Chen, D. H. Huang, and Z. Q. Zhou, "Adaptive random testing through iterative partitioning," in *Proc. 11th Ada-Europe Int. Conf. Reliable Softw. Technol.*, 2006, pp. 155–166.
[45] N. L. Leech and A. J. Onwuegbuzie, "A call for greater use of nonparametric statistics," in *Proc. Annu. Meeting MidSouth Educ. Res. Assoc.*, 2002, pp. 1–24.
[46] R. J. Grissom and J. J. Kim, *Effect Sizes for Research: A Broad Practical Approach*, Hillsdale, NJ, USA: Lawrence Erlbaum Associates Publishers, 2005.
[47] A. Arcuri and L. Briand, "A hitchhiker's guide to statistical tests for assessing randomized algorithms in software engineering," *Softw. Testing, Verification Rel.*, vol. 24, no. 3, pp. 219–250, 2014.
[48] T. Menzies, S. Williams, B. W. Boehm, and J. Hihn, "How to avoid drastic software process change (using stochastic stability)," in *Proc. 31st Int. Conf. Softw. Eng.*, 2009, pp. 540–550.
[49] M. Cowles and C. Davis, "On the origins of the. 05 level of statistical significance," *Amer. Psychol.*, vol. 37, no. 5, pp. 553–558, 1982.
[50] A. Vargha and H. D. Delaney, "A critique and improvement of the CL common language effect size statistics of McGraw and Wong," *J. Educ. Behav. Statist.*, vol. 25, no. 2, pp. 101–132, 2000.
[51] Y. Jia and M. Harman, "An analysis and survey of the development of mutation testing," *IEEE Trans. Softw. Eng.*, vol. 37, no. 5, pp. 649–678, 2010.
[52] T. Y. Chen and F.-C. Kuo, "Is adaptive random testing really better than random testing," in *Proc. 1st Int. Workshop Random Testing: Conjunction 15th ACM SIGSOFT Int. Symp. Softw. Testing Anal.*, 2006, pp. 64–69.
[53] T.-Y. Chen, F.-C. Kuo, and C.-A. Sun, "Impact of the compactness of failure regions on the performance of adaptive random testing," *J. Softw.*, vol. 17, no. 12, pp. 2438–2449, 2006.
[54] W. Press, B. P. Flannery, S. A. Teulolsky, and W. T. Vetterling, *Numerical Recipes*. Cambridge, U.K.: Cambridge Univ. Press, 1986.
[55] *ACM, Collected Algorithms from ACM*, New York, NY, USA: ACM, 1980.
[56] K. P. Chan, T. Y. Chen, and D. Towey, "Restricted random testing: Adaptive random testing by exclusion," *Int. J. Softw. Eng. Knowl. Eng.*, vol. 16, no. 4, pp. 553–584, 2006.
[57] J. Ferrer, F. Chicano, and E. Alba, "Evolutionary algorithms for the multi-objective test data generation problem," *Softw.: Pract. Experience*, vol. 42, no. 11, pp. 1331–1362, 2012.
[58] Y. D. Liang, *Introduction to Java Programming and Data Structures. Comprehensive Version*, 11th ed.London, U.K.: Pearson Education, 2017.
[59] P. S. May, "Test data generation: Two evolutionary approaches to mutation testing," Ph.D. dissertation, Sch. Comput., Univ. Kent, Canterbury, U.K., 2007.
[60] A. Shahbazi, A. F. Tappenden, and J. Miller, "Centroidal Voronoi Tessellations—A new approach to random testing," *IEEE Trans. Softw. Eng.*, vol. 39, no. 2, pp. 163–183, 2013.
[61] T. Y. Chen, D. H. Huang, T. H. Tse, and Z. Yang, "An innovative approach to tackling the boundary effect in adaptive random testing," in *Proc. 40th Annu. Hawaii Int. Conf. Syst. Sci.*, 2007, pp. 1–10.
[62] J. L. Bentley, "Multidimensional binary search trees used for associative searching," *Commun. ACM*, vol. 18, no. 9, pp. 509–517, 1975.
[63] J. Kubica, J. Masiero, A. W. Moore, R. Jedicke, and A. J. Connolly, "Variable KD-tree algorithms for spatial pattern search and discovery," in *Proc. 18th Int. Conf. Neural Inf. Process. Syst.*, 2005, pp. 691–698.
[64] A. Adams, N. Gelfand, J. Dolson, and M. Levoy, "Gaussian KD-trees for fast high-dimensional filtering," *ACM Trans. Graph.*, vol. 28, no. 3, pp. 21:1–21:12, 2009.
[65] T. Foley and J. Sugerman, "KD-tree acceleration structures for a GPU raytracer," in *Proc. 5th ACM SIGGRAPH/EUROGRAPHICS Symp. Graph. Hardware*, 2005, pp. 15–22.