Postprint of article in *IEEE Access* **8**: 52475–52488 (2020)
Digital Object Identifier https://doi.org/10.1109/ACCESS.2020.2977777

Exploiting the Largest Available Zone: A Proactive Approach to Adaptive Random Testing by Exclusion

JINFU CHEN[1] (Member, IEEE), QIHAO BAO[1], T. H. Tse[2] (Senior Member, IEEE), TSONG YUEH CHEN[3] (Senior Member, IEEE), JIAXIANG XI[1], CHENGYING MAO[4] (Member, IEEE), MINJIE YU[1], AND RUBING HUANG[1] (Member, IEEE)

[1]School of Computer Science and Communication Engineering, Jiangsu University, Zhenjiang, 202000, China
[2]Department of Computer Science, The University of Hong Kong, Pokfulam, Hong Kong
[3]Department of Computer Science and Software Engineering, Swinburne University of Technology, Melbourne, VIC 3122, Australia
[4]School of Software and IoT Engineering, Jiangxi University of Finance and Economics, Nanchang, 330013, China

Corresponding author: Jinfu Chen (e-mail: jinfuchen@ujs.edu.cn).

This work was supported in part by the National Natural Science Foundation of China [grant numbers U1836116, 61762040, and 61872167] and the project of Jiangsu provincial Six Talent Peaks [grant number XYDXXJS-016].

ABSTRACT Adaptive random testing (ART) has been proposed to enhance the effectiveness of random testing (RT) through more even spreading of the test cases. In particular, restricted random testing (RRT) is an ART algorithm based on the intuition of skipping all the candidate test cases that are within the neighborhoods (or zones) of previously executed test cases. RRT has higher effectiveness than RT in terms of failure detection but incurs a higher time cost. In this paper, we aim to further reduce the time costs for RRT and improve the effectiveness for RT and ART methods. We propose a proactive technique known as "RRT by largest available zone" (RRT-LAZ). Like RRT, RRT-LAZ first defines an exclusion zone around every executed test case in order to determine the available zones. Unlike the original RRT, RRT-LAZ then compares all the available zones to proactively pick the largest one, from which the next test case is randomly generated. Both simulation analyses and empirical studies have been employed to investigate the efficiency and effectiveness of RRT-LAZ in relation to RT and related ART algorithms. The results show that RRT-LAZ has significantly lower time costs than RRT. Furthermore, RRT-LAZ is more effective than RT and related ART methods for block failure patterns in low-dimensional input spaces. In general, since RRT-LAZ employs a proactive technique instead of a passive one in generating next cases, it is much more cost-effective than RRT. RRT-LAZ is also more cost-effective than RT and other ART methods that we have studied.

INDEX TERMS Software testing, random testing, adaptive random testing, restricted random testing, exclusion zone, largest available zone

I. BACKGROUND

The advancement of information technology has triggered an increasing interest in the use of software applications for various activities such as online shopping, mobile apps, and train and taxi booking. Software has become an integral part of people's daily life in all aspects. By 2020, the market for software and software-based services is estimated to be nearly €290 billion for Europe alone [1]. While software brings convenience to us, it also causes many problems owing to the numerous faults that remain undetected. For instance, the inadequacy in software testing is estimated to cost the US 0.6 percent of its GDP [2], or $112 trillion based on the latest World Bank statistics [3]. The quality of software testing is, therefore, of vital importance.

Random testing (*RT*) is a fundamental software testing technique [4]. It randomly selects test cases from the entire input space and then executes them. Hence, the probability of selecting any location in the input space as the next test case is the same. It does not require any information other than the data types and ranges of the inputs. It is especially good at executing the software in unforeseen circumstances not obvious to human testers [5]. Furthermore, previous

experimental results show that RT is efficient in generating test cases [6].

Making use of the general property that failure-causing inputs tend to be clustered in contiguous regions, Chen et al. have proposed an improved method known as *adaptive random testing* (*ART*) [7]. It has subsequently been applied to different programming languages [8]–[10]. Compared with RT, ART requires fewer test cases to reveal program failures, and the test cases selected by ART are more evenly distributed throughout the input space [11]. The effectiveness of failure detection by ART has been validated to be better than that of RT [7]. Various ART algorithms have been developed to further improve the diversity of test cases based on different concepts.

ART by bisection (*ART-B*) [12], ART by random partitioning (*ART-RP*) [12], [13], and ART by two-point partitioning (*ART-TPP*) [14] are adaptive random testing algorithms based on the idea of partitions. ART-B divides the input space into equal-sized partitions. Test cases can only be selected from those partitions that do not contain previously executed test cases. When all the existing partitions contain previously executed test cases, further partitioning takes place. ART-RP partitions the input space at the location of every previously executed test case and selects the next test case from the largest region. ART-TPP also generates the new test case from the biggest region, but the partitioning does not occur at the locations of executed test cases. Instead, each partitioning location is determined from two points in the input space. The first is the location of a previously executed test case (or a random point if no test case has yet been executed). The second is selected according to farthest distance criterion. Then, the current region is partitioned at the midpoint of these two locations.

Restricted random testing (*RRT*) [15], [16] is an ART algorithm based on the idea of exclusion. It generates an exclusion zone in the input space around every previously executed test case. Then, it repeatedly generates random test cases until one of them is outside all the exclusion zones. This test case will be selected as the next one to be executed. Experimental results show that RRT performs better than RT and previous ART algorithms. On the other hand, it has to spend more time in test case selection by passively judging whether each candidate case is selectable. In addition, RRT is inappropriate in some situations, such as when the failure rate of the program is too small.

In view of the shortcomings of RRT, we propose a proactive algorithm known as *RRT by largest available zone* (*RRT-LAZ*), which reduces the effort in judging whether a candidate case is selectable. Experimental results show that RRT-LAZ uses significantly less time to select test cases when compared with RRT, but the effectiveness of failure detection is preserved.

The remaining parts of the paper are organized as follows: Section II lays the theoretical groundwork for the rest of the paper. Section III describes the original RRT algorithm and the RRT-LAZ algorithm. Section IV presents a set of experiments to compare RRT-LAZ with RT and other ART algorithms, followed by analyses of the results. Section V investigates the threats to validity of our study. Section VI presents selected related work. Finally, Section VII concludes the paper.

II. FAILURE PATTERNS

A program is said to fail if it produces an incorrect result. An input is said to be failure-causing if it leads to a failure of the program. Bishop [17] has observed that failure-causing inputs tend to be clustered in contiguous failure regions. Chan et al. [18] have further classified the patterns of failure-causing inputs into three categories, namely, point, strip, and block patterns. The main characteristic of the point pattern is that the failure-causing inputs are either stand-alone points or form many regions of very small sizes scattered over the input space. For the strip pattern, the failure-causing inputs form the shape of a narrow strip. A typical example is the domain error highlighted by White and Cohen [19], where "a specific input follows the wrong path." For the block pattern, the main characteristic is that the failure-causing inputs are concentrated in a more compact form than a strip pattern and in a small number of regions. Examples of the three patterns are shown in Figure 1 and the relevant code is shown in Table I. In the figure, the outer boundaries represent the borders of the input space and the filled regions denote failure-causing inputs.

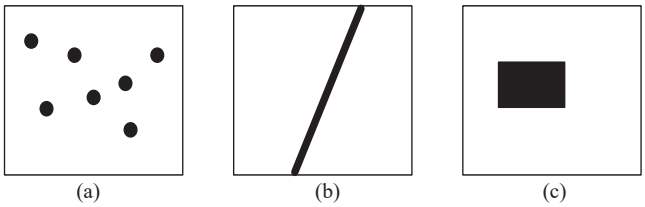


FIGURE 1. Three patterns of failure-causing inputs. (a) Point pattern. (b) Strip pattern. (c) Block pattern.

A straightforward improvement of RT would be to make use of the generic information on typical failure patterns. Intuitively, it would be more effective to select test cases far from non-failure-causing inputs because the failure patterns tend to be clustered. Adaptive random testing makes use of this additional information to enhance its failure-detection capability over random testing.

TABLE I. Sample faults causing the three patterns of failure-causing inputs.

Point Pattern	Strip Pattern	Block Pattern

if (x%4 == 0
&& y%6 == 0)
z = x/(2*y);
// It should be
// "z = x/2*y;"
else
z = x*y;
	if (3*x–y > 10)
	// It should be
	// "(3*x–y > 18)"
	z = x/2*y;
	else
	z = x*y;
		if (x > = 10
		&& x < = 11 && y > = 10
		&& y < = 12)
		z = x/(2*y);
		// It should be
		// "z = x/2*y;"
		else
		z = x*y;

III. IMPROVING THE PROACTIVENESS OF THE RRT ALGORITHM

A. THE ORIGINAL RRT ALGORITHM

The original RRT is an ART algorithm based on exclusion. The main idea of the RRT algorithm is as follows: First, when there are n (≥ 1) previously executed test cases, RRT defines an exclusion zone around each executed test case. The size of an exclusion zone is determined by the size of the input space s, the exclusion ratio R (which is set by human testers), and the number of previously executed test cases n. In a one-dimensional input space, the size of each exclusion zone is set to $R*s/n$ and the exclusion radius is $R*s/(2*n)$. RRT repeatedly generates candidate test cases randomly from the input space until a candidate test case falls outside of all the exclusion zones. Figure 2 shows a one-dimensional input space with values from 0 to 1. We set R as 0.75 and generate the first test case randomly. The second test case can only be selected from $(0, t_1 - 0.375*s/1)$ and $(t_1 + 0.375*s/1, 1)$. That is, if RRT generates a point not in $(0, t_1 - 0.375*s/1)$ or $(t_1 + 0.375*s/1, 1)$, it will repeatedly generate another point from the input space until it falls within $(0, t_1 - 0.375*s/1)$ or $(t_1 + 0.375*s/1, 1)$. Similarly, as shown in Figure 2 (b), the third test case can only be selected from $(0, t_2 - 0.375*s/2)$, $(t_2 + 0.375*s/2, t_1 - 0.375*s/2)$, or $(t_1 + 0.375*s/2, 1)$. Hence, RRT selects candidate test cases from the whole input domain and then discards the candidates that are inside the exclusion regions. This is a passive approach.

Simulation and empirical studies have shown that RRT is more effective than RT in terms of failure detection capability [20]. However, in order to select the nth test case, RRT needs to generate, on average, $\log n$ candidates for each previously executed test case [21], and in order to determine whether the candidate test case is in the exclusion zone, RRT needs to measure the distance between the candidate test case and every executed test case. RRT requires $O(n \log n)$ time to generate the nth test case, and hence the time complexity of RRT is $O(n^2 \log n)$. As it can be seen, a disadvantage of RRT is that its time overhead is too large. An improved approach, known as RRT-LAZ, is proposed in the next section.

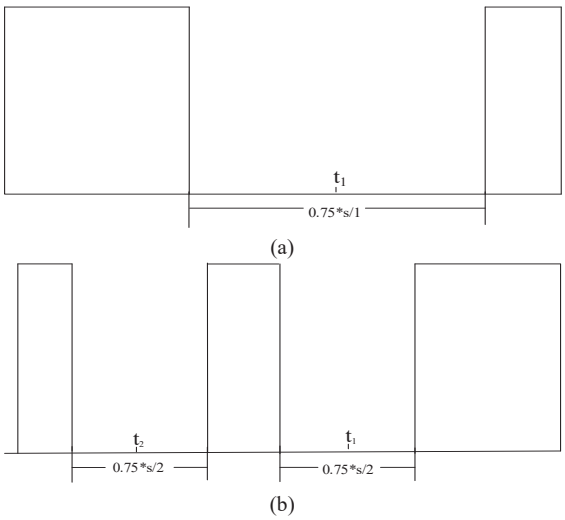


FIGURE 2. Example of applying RRT to a one-dimensional input space. (a) Exclusion zone for one test case. (b) Exclusion zones for two test cases

B. RRT-LAZ ALGORITHM FOR ONE-DIMENSIONAL INPUT SPACES

RRT-LAZ is an enhanced version of the RRT algorithm. The first test case t_1 is randomly selected from the input space in the same way as RRT. When there are n (≥ 1) previously executed test cases, RRT-LAZ similarly generates exclusion zones around each executed case. The size definition of exclusion zones is the same as that in RRT. Unlike RRT, however, RRT-LAZ maintains the exclusion zones in sequence of the Hilbert values of the test cases (which will be explained in Section III-C). After determining the exclusion zones and hence the *available zones* (that is, the regions outside the exclusion zones), RRT-LAZ picks the *largest available zone*, from which the next test case will be randomly generated. This method selects test case from the available regions and therefore no generated test cases will be discarded, greatly saving the time it takes to select the next test case. It represents a proactive approach that is different from the original passive approach.

Let us determine the time complexity of the RRT-LAZ algorithm. In each iteration, RRT-LAZ generates the nth test case from the largest available zone. To do so, it first computes all the available zones. It needs to traverse n–1 previously executed test cases to determine these zones, which takes $O(n–1)$ time. In other words, it needs $O(n–1)$ time to generate the nth test case. For the RRT-LAZ algorithm for one-dimensional input spaces, the time complexity is therefore $O(n^2)$. For multidimensional input spaces, we should also consider the time complexity of the Hilbert space-filling function (to be introduced in Section III-C). The latter time cost is only related to the dimension d and the order of the Hilbert curve m, both of which are constants. Hence, the time complexity of RRT-LAZ is also

Algorithm 1. RRT-LAZ for One-Dimensional Input Space

```
Inputs.  (1) Target exclusion ratio R
         (2) Boundaries min and max of input space
         (3) Predefined ceiling for number of test cases to be executed
1.  read inputs R, min, and max;
2.  set tcList = ⟨⟩;  /* tcList is the list of executed test cases */
3.  set azList = ⟨⟩;  /* azList is the list of available zones */
4.  set n = 0;  /* n is the number of executed test cases */
5.  set tc = GenRandomTC([min, max]);  /* Randomly select a test case from the set [min, max] */
6.  while not (Result(Execute(tc)) = fail or n = ceiling) do
        /* While the execution result of tc is not a failure and the number of executed test cases has not reached the predefined ceiling */
7.      set n = n + 1;
8.      insert tc into tcList, maintaining the list in ascending order through binary search;
9.      define exclusion zone for each test case in tcList;  /* Length of each exclusion zone = R*(max–min)/n */
10.     availableZone = FindAZ([min, tcList[1]]);  /* Find an available zone between min and the first test case. Return null if not found. */
11.     if (availableZone != null)  /* If found */
12.       then AddtoAZlist(availableZone);  /* then add it to azList */
13.     end if;
14.     if n > 1 then
15.       for i = 1 to n–1 do
16.         set thisTC = tcList[i];
17.         set nextTC = tcList[i+1];
18.         availableZone = FindAZ([thisTC, nextTC]);
19.         if (availableZone != null)
20.           then AddtoAZlist(availableZone);
21.         end if;
22.       end for;
23.     end if;
24.     set availableZone = null;
25.     availableZone = FindAZ([tcList[n], max]);
26.     if (availableZone != null)
27.       then AddtoAZlist(availableZone);
28.     end if;
29.     set LAZ = Largest(azList);  /* Find the largest available zone in azList */
30.     set tc = GenRandomTC(LAZ);  /* Randomly select a test case from LAZ */
31. end while;
32. return;
```

$O(n^2)$ for multidimensional input spaces, which is more favorable than RRT.

We will give a simple example to illustrate the RRT-LAZ process. Consider a one-dimensional input space with values from 0 to 1. We select the largest available zone as follows: Set the size of the largest available zone *maxlength* to an initial value of 0. The *lengthstart* and *lengthend* positions represent, respectively, the start and end points of the largest available zone. The size of an exclusion zone is equal to *R**s/*n*, where *R* is the exclusion ratio, *s* is the size of the input space, and *n* is the number of previously executed test cases. We set the initial size of the exclusion zone to 1. As in RRT, the exclusion radius for one-dimensional input spaces is equal to half the size of the exclusion zone. In RRT-LAZ, the first and the last executed test cases are processed separately. If the value of the first test case t_1 minus the exclusion radius is greater than zero, it means that there is an available zone between 0 and t_1 – radius. That is, *lengthstart* = 0, *lengthend* = t_1 – radius, and *maxlength* = *lengthend* – *lengthstart*. If the value of the last test case t_n plus the radius is less than 1, it indicates that t_n + radius is still in the input space. That is, *lengthstart* = t_n + radius, *lengthend* = 1, and *maxlength* = *lengthend* – *lengthstart*. When there is more than one test case, a vital condition in the algorithm must be satisfied. That is, the value of the current test case plus the radius cannot be greater than the next test case minus the radius; otherwise, there will be no available zone between the two test cases. The *maxlength* can be updated only if the above condition is met.

The implementation of the RRT-LAZ algorithm for one-dimensional input spaces is further illustrated in Figure 3. The actual RRT-LAZ process is shown in Algorithm 1. Consider again the input space [0, 1]. Following Line 5 of the algorithm, the first test case t_1 is randomly generated with a value of 0.6, as shown in Figure 3(a). Following Line 9 of the algorithm, taking R = 0.75, and based on the formula *R**s/*n*, the exclusion zone is computed to be [0.225, 0.975]. Following Lines 10 to 28 of the algorithm, the two available zones are [0, 0.225] and [0.975, 1]. The size of the available zone #1 is obviously larger than that of #2, and hence, following Line 29 of the algorithm, the next test case should be selected from the first available zone. This test case t_2 is randomly generated to be 0.2, as shown in Figure 3(b). It can be seen from the updated exclusion zones and available zones that the size of available zone #3 is the largest. As a result, the next test case is randomly generated in this zone.

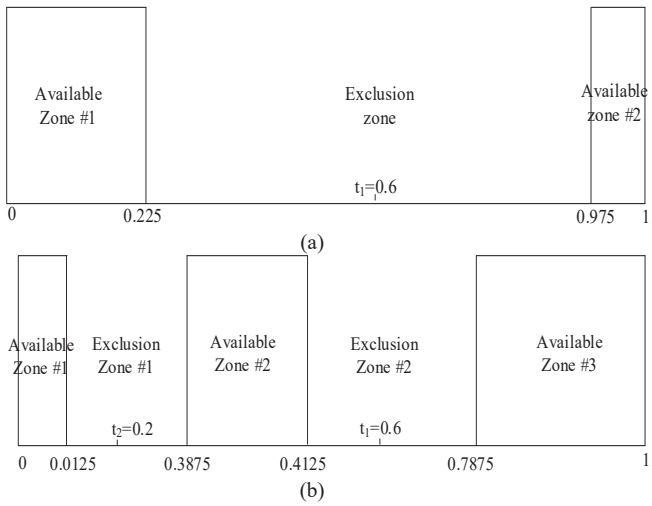


FIGURE 3. Example of applying RRT-LAZ to a one-dimensional input space. (a) Generation of test case t_1. (b) Generation of test case t_2.

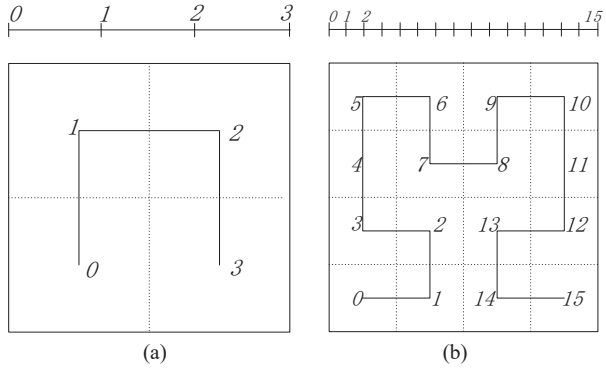


FIGURE 4. Examples of Hilbert space-filling curves. (a) First-order Hilbert curve. (b) Second-order Hilbert curve

C. RRT-LAZ ALGORITHM FOR MULTIDIMENSIONAL INPUT SPACES

In Sections III-A and III-B, we have introduced the RRT and RRT-LAZ algorithms for one-dimensional input spaces. This section presents the algorithm for multidimensional input spaces. We will use the Hilbert space-filling function to convert one-dimensional inputs into multidimensional test cases.

1) HILBERT SPACE-FILLING FUNCTION

The *Hilbert space-filling function* [22] can map bijectively the points on a one-dimensional line (known as the *Hilbert space-filling curve*) to the points in a multidimensional space. Figure 4 shows how the points in a one-dimensional space are mapped to a two-dimensional space. The square is first divided into four sub-squares. A *first-order curve* is constructed by connecting the centroids of the sub-squares by straight edges. The sequencing of the sub-squares is so arranged that that a common edge is always shared by a pair of sub-squares. The number at each centroid is known as the *Hilbert value*. Figure 4(b) shows the *second-order curve*. In this step, each of the four sub-squares is further divided into four new sub-squares. The first and last sub-squares need to be rotated to ensure that a common edge is always shared by a pair of sub-squares. There are $2^{1*2} = 4$ Hilbert values in the first-order curve for two-dimensional spaces, and $2^{2*2} = 16$ Hilbert values in the second-order curve for two-dimensional spaces.

Let r be the Hilbert value in an m-order space-filling curve to be converted into a d-dimensional test case. The value is represented by a binary number consisting of m groups with d bits in each group. The m groups are indexed by $i = 1, 2, ..., m$.

To initiate the dimensional conversion process, we obtain the binary representation of the value r that needs to be transformed. We fill in the most significant part of the integer portion of the binary number with zeros and the least significant part of the fractional portion with zeros, making the length of binary number equal to $d*m$.

Then, the conversion process undergoes a sequence of transformations as follows:

$$r \xrightarrow{\boldsymbol{\rho}} \rho_i \xrightarrow{\mathbf{J}} J_i \xrightarrow{\boldsymbol{\sigma}} \sigma_i \xrightarrow{\boldsymbol{\tau}} \tau_i \xrightarrow{\mathbf{q}} q_i \xrightarrow{\mathbf{t}} t_i \xrightarrow{\boldsymbol{\omega}} \omega_i \xrightarrow{\boldsymbol{\alpha}} \alpha_i$$

TABLE II. Transformation operators for dimensional conversion process in Hilbert space-filling function

Transformation Operator	Action
ρ	Split the binary value of r into m groups $\rho_1, \rho_2, ..., \rho_m$, each group ρ_i having d bits $\rho_i^1 \rho_i^2 ... \rho_i^d$.
J	Determine the *principal position* J_i = the last position in $\rho_i = \rho_i^1 \rho_i^2 ... \rho_i^d$ such that $\rho_i^j \neq \rho_i^d$. In case that all the bits of ρ_i are equal, the principal position is d.
σ	Determine σ^i such that $\sigma_1^i = \rho_1^i$, $\sigma_2^i = \rho_2^i \oplus \rho_1^i$, ..., $\sigma_d^i = \rho_d^i \oplus \rho_{d-1}^i$, where $\oplus$ is the "exclusive or" operator.
τ	Determine τ^i by first complementing σ^i in the dth position. If τ^i is of odd parity, complement the principal position.
q	Determine q_i as follows: For $i > 1$, shift σ^i right circular a number of positions equal to $(J_1 - 1) + (J_2 - 1) + ... + (J_i - 1)$.
t	Determine t_i as follows: For $i > 1$, shift τ^i right circular a number of positions equal to $(J_1 - 1) + (J_2 - 1) + ... + (J_i - 1)$.
ω	Set ω_1 = all zeros and $\omega_i = \omega_{i-1} \oplus t_{i-1}$.
α	Set $a_i = \omega_i \oplus q_i$.

1. We use operator **ρ** to split the binary representation of r into m groups $\rho_1, \rho_2, ..., \rho_m$.
2. We perform the following processes for $i = 1, 2, ..., m$:
   a) Use operator **J** to record the principal position of ρ_i.
   b) Use operator **σ** to transform p_i into σ_i.

Algorithm 2. RRT-LAZ for Multidimensional Input Space

```
Inputs. (1) Target exclusion ratio R
        (2) Dimension d
        (3) Boundaries of multidimensional input space
        (4) Predefined ceiling for number of test cases to be executed
1.  read inputs R, d, and boundaries of input space;
2.  set hvList = ⟨⟩;  /* hvList is the list of Hilbert values in one-to-one correspondence with the executed test cases */
3.  set azList = ⟨⟩;  /* azList is the list of available zones */
4.  set n = 0;  /* n is the number of executed test cases */
5.  Map the multidimensional input space to [0, 1] based on Hilbert space-filling curve;
6.  set hv = GenRandomHV([0, 1]);  /* Randomly select a Hilbert value hv from the set [0, 1] */
7.  set tc = HVtoTC(hv);  /* Map hv to multidimensional test case tc based on Hilbert space-filling curve */
8.  while not (Result(Execute(tc)) = fail or n = ceiling) do
      /* While the execution result of tc is not a failure and the number of executed test cases has not reached the predefined ceiling */
9.    set n = n + 1;
10.   insert hv into hvList, maintaining the list in ascending order through binary search;
11.   define exclusion zone for each Hilbert value in hvList;  /* Length of each exclusion zone = R/n */
12.   availableZone = FindAZ([0, hvList[1]]);  /* Find an available zone between 0 and the first Hilbert value. Return null if not found. */
13.   if (availableZone != null)  /* If found */
14.     then AddtoAZlist(availableZone);  /* then add it to azList */
15.   end if;
16.   if n > 1 then
17.     for i = 1 to n–1 do
18.       set thisHV = hvList[i];
19.       set nextHV = hvList[i+1];
20.       availableZone = FindAZ([thisHV, nextHV]);
21.       if (availableZone != null)
22.         then AddtoAZlist(availableZone);
23.       end if;
24.     end for;
25.   end if;
26.   set availableZone = null;
27.   availableZone = FindAZ([hvList[n], 1]);
28.   if (availableZone != null)
29.     then AddtoAZlist(availableZone);
30.   end if;
31.   set LAZ = Largest(azList);  /* Find the largest available zone in azList */
32.   set hv = GenRandomHV(LAZ);  /* Randomly select a Hilbert value from LAZ */
33.   set tc = HVtoTC(hv);
34. end while;
35. return
```

c) Use operator $\boldsymbol{\tau}$ to transform σ_i into τ_i.

d) Copy σ_1 to q_1. Use operator $\mathbf{q}$ to transform σ_i into q_i.

e) Copy τ_1 to t_1. Use operator $\mathbf{t}$ to transform τ_i into t_i.

f) Use operator $\boldsymbol{\omega}$ to transform t_i into ω_i.

g) Use operator $\boldsymbol{\alpha}$ to transform q_i into α_i.

3. The result is a sequence $\langle a_1, a_2, ..., a_m \rangle$.

The detailed actions of the transformation operators are shown in Table II.

For example, given $d = 3$ and $m = 4$, consider a one-dimensional point $r^1 = \langle 0.247314453125 \rangle_{10} = \langle 0.001111110101 \rangle_2$, where $\langle \cdot \rangle_{10}$ and $\langle \cdot \rangle_2$ denote the decimal and binary representations, respectively. We have $\rho_1 = \mathbf{001}, \rho_2 = \mathbf{111}, \rho_3 = \mathbf{110}$, and $\rho_4 = \mathbf{101}$. According to the transformation rule of the Hilbert curve, ρ_1, ρ_2, ρ_3, and ρ_4 are converted to two sequences $\langle j_1, j_2, j_3, j_4 \rangle = \langle 2, 3, 2, 1 \rangle$ and $\langle \sigma_1, \sigma_2, \sigma_3, \sigma_4 \rangle = \langle 001, 100, 101, 111 \rangle$. Applying the next operator $\boldsymbol{\tau}$, we obtain $\tau_1 = 000$, $\tau_2 = 101$, $\tau_3 = 110$, and $\tau_4 = 110$. Then, we use the operator $\mathbf{q}$ to transform σ_i into q_i to obtain $q_1 = 001$, $q_2 = 010$, $q_3 = 101$, and $q_4 = 111$. Similarly, we have $t_1 = 000$, $t_2 = 110$, $t_3 = 110$, and $t_4 = 011$. Applying the next operator $\boldsymbol{\omega}$, we have $\omega_1 = 000$, $\omega_2 = 000$, $\omega_3 = 110$, and $\omega_4 = 000$. Finally, applying the operator $\boldsymbol{\alpha}$, we obtain the sequence $\langle a_1, a_2, a_3, a_4 \rangle = \langle 001, 010, 011, 111 \rangle$. Therefore, the three-dimensional point $r^3 = \langle 0.0001, 0.0111, 0.1011 \rangle_2 = \langle 0.0625, 0.4375, 0.6875 \rangle_{10}$.

Previous ART researchers [23] set the order of the Hilbert curve to 32. By further experimental analysis, we found that 32 was not large enough to ensure the precision of the mapping. If the order of the Hilbert curve was set to 64, the precision of the mapping was better than that of 32. Hence, in our experiment, the order of the Hilbert curve was set to 64 based on the cost-effective analysis.

2) MULTIDIMENSIONAL RRT-LAZ ALGORITHM

We will apply the Hilbert space-filling function to support test case generation for programs with multidimensional input spaces. First, we generate one-dimensional Hilbert values similar to the RRT-LAZ algorithm for one-dimensional input spaces in Section III-B. Then, we convert

these values into multidimensional test cases using the Hilbert spacing-filling function. The specific RRT-LAZ algorithm for multidimensional input spaces is shown in Algorithm 2.

IV. EXPERIMENTAL EVALUATIONS

In this section, we report the comparisons of RRT-LAZ with related algorithms by applying them to simulated and real-life programs. Since RRT-LAZ involves two concepts, namely, exclusion and partitioning, we only compare it with RT, RRT (which is based on exclusion), as well as ART-B, ART-RP, and ART-TPP (which are based on partitioning). In our experiments, the exclusion ratio R was set according to the original RRT paper [15]. All the experiments were conducted on a PC with Intel Core i3-380M 2.53GHz dual core processor and 6GB RAM running the Windows 7 operating system. We made sure that the performance of the PC was steady in all the conducted experiments.

A. SIMULATION STUDY

We apply two metrics to compare the proposed RRT-LAZ algorithm with RT and popular ART techniques. First, we use the *test case generation time*, which does not include the time for test case execution as this cannot be simulated without bias. Second, we use the *F-measure* [7], which refers to the expected number of executed test cases to reveal the first failure.

1) TEST CASE GENERATION TIME

The main purpose of this part of the study is to observe and analyze the time costs of the algorithms. We do not require the ART algorithms to test programs, but simply use them to produce test cases. The test case generation time is the average of the sums of times obtained from 2000 independent tests for input spaces of all dimensions.

According to Figures 5 and 6, which illustrate the time costs for input spaces of one and two dimensions, RRT-LAZ spends less time to generate test cases than RRT and ART-RP. From Figures 7 and 8, which show the time costs for input spaces of three and four dimensions, it can further be observed that RRT-LAZ spends less time to generate test cases than RRT, ART-RP, and ART-TPP.

As explained before, a significant disadvantage of the RRT algorithm is that it spends excessive time in test case generation. RRT-LAZ clearly improves on the time cost of its predecessor. Furthermore, RRT-LAZ uses less time than ART-TPP as the dimension increases, because our method does not need to handle d-dimensional points or d-dimensional zones. On the other hand, RRT-LAZ uses more time than RT and ART-B.

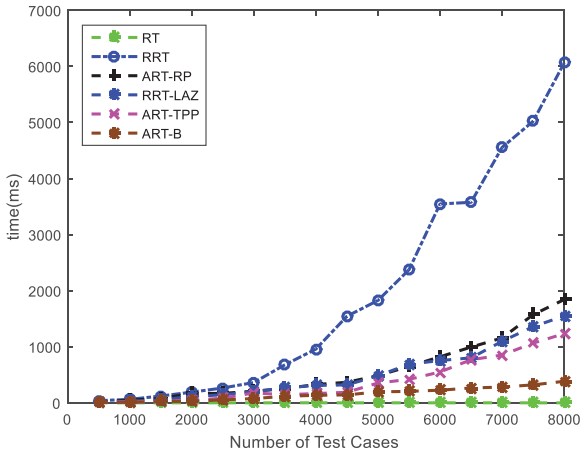


FIGURE 5. Time costs of test cases in 1-dimensional input spaces.

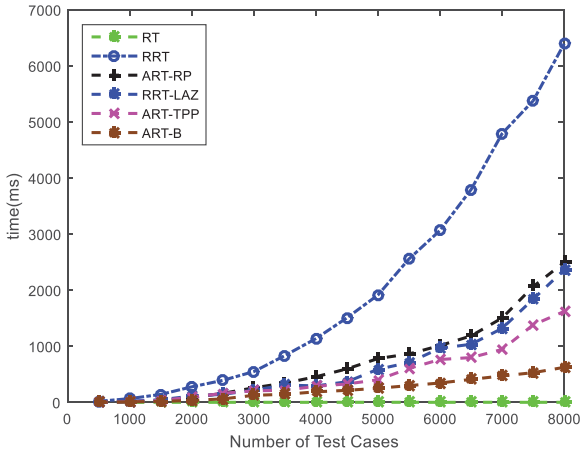


FIGURE 6. Time costs of test cases in 2-dimensional input spaces.

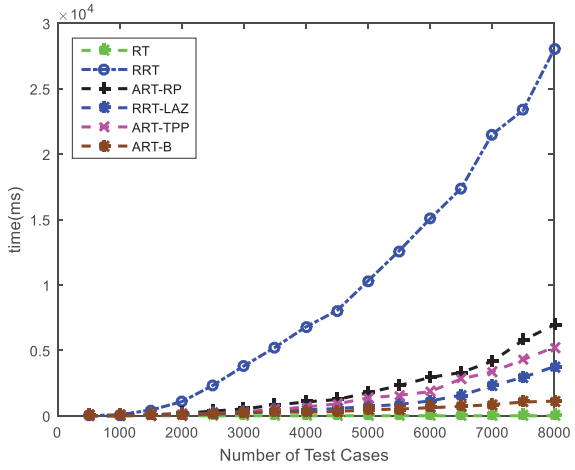


FIGURE 7. Time costs of test cases in 3-dimensional input spaces.

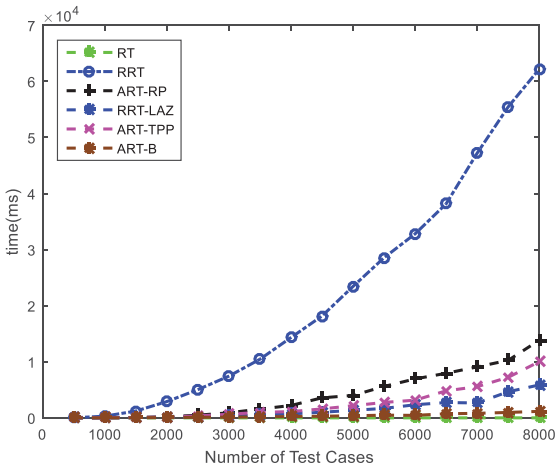


FIGURE 8. Time costs of test cases in 4-dimensional input spaces.

2) EFFECTIVENESS OF TEST CASE GENERATION

We recall that the F-measure evaluates the expected number of test cases to expose the first failure. Thus, a lower F-measure indicates that an algorithm is more effective. We conducted a simulation study of the F-measure with respect to varying failure rates θ. A failure rate denotes the fraction of failure-causing inputs over the entire input space. A lower failure rate means that it will be more difficult to reveal failures.

We used failure rates of 0.0050, 0.0020, 0.0010, and 0.0015. For each failure rate, the seed of random numbers was set between 0 and 10000, and the interval between two seeds was 5. Each experiment was repeated 2000 times for each failure rate, and the average was taken as the overall result. We conducted the experiments in 1-, 2-, 3-, 4-, and 10-dimensional input spaces. As said before, there are three main patterns of failure-causing inputs. All the three patterns were tested in the experiments. The results of the F-measures for testing the simulated programs with the RRT-LAZ and other ART algorithms are shown in Tables III to VII.

It can be observed from the first four of these tables that RRT-LAZ is better than other methods for block failure patterns in low-dimensional input spaces. As shown in Tables III to VII, when the dimension increases, the effectiveness of ART algorithms decreases. RRT-LAZ is no longer the best algorithm for block failure patterns in 10-dimensional input spaces. Nevertheless, the F-measure of RRT-LAZ is comparable to the best one. For strip patterns, Tables III to VII show that RRT-LAZ is better than other methods in some cases but worse in other cases. Considering both the failure-detection effectiveness and overhead, we can conclude that RRT-LAZ is more cost-effective than other approaches. Since ART was not intended for point failure patterns, it is not expected to show much improvement over RT.

Figures 9 to 12 show the experimental results of various algorithms in box plots, categorized by different failure rates and dimensions. We have run the experiments 100 times to obtain 100 F-measures for each algorithm before drawing the box plots. The dispersions of the F-measures can be observed from these figures.

It can be seen from Figures 9 to 12 that the ranges of the boxes for RRT-LAZ are smaller than those for other ART algorithms. This means that the dispersions of RRT-LAZ are smaller than those of other algorithms.

TABLE III. F-measures for 1-dimensional input spaces

	Block Pattern				Strip Pattern				Point Pattern			
failure rate	**0.0050**	**0.0020**	**0.0010**	**0.0015**	**0.0050**	**0.0020**	**0.0010**	**0.0015**	**0.0050**	**0.0020**	**0.0010**	**0.0015**
RT	200.0	500.0	1000.0	666.7	200.0	500.0	1000.0	666.7	200.0	500.0	1000.0	666.7
RRT	117.2	297.0	603.0	404.0	117.2	297.0	603.0	404.0	192.4	487.0	998.0	652.0
ART-B	125.6	308.5	628.0	411.3	125.6	308.5	628.0	411.3	199.4	497.0	1019.0	665.3
ART-RP	135.6	339.0	668.0	440.7	135.6	339.0	668.0	440.7	**188.6**	**471.5**	988.0	649.3
ART-TPP	130.8	319.5	638.0	420.7	130.8	319.5	638.0	420.7	193.8	482.0	972.0	640.0
RRT-LAZ	**107.2**	**270.5**	**552.0**	**359.3**	**107.2**	**270.5**	**552.0**	**359.3**	200.4	495.5	980.0	660.7

TABLE IV. F-measures for 2-dimensional input spaces

	Block Pattern				Strip Pattern				Point Pattern			
failure rate	**0.0050**	**0.0020**	**0.0010**	**0.0015**	**0.0050**	**0.0020**	**0.0010**	**0.0015**	**0.0050**	**0.0020**	**0.0010**	**0.0015**
RT	200.0	500.0	1000.0	666.7	200.0	500.0	1000.0	666.7	200.0	500.0	1000.0	666.7
RRT	125.2	306.0	593.0	404.7	186.2	**467.0**	**928.0**	**613.3**	201.6	508.5	1015.0	691.3
ART-B	144.2	364.5	715.0	488.7	184.4	477.0	986.0	644.7	197.4	490.0	978.0	662.7
ART-RP	151.4	387.0	786.0	522.7	182.6	475.5	961.0	644.0	197.0	498.5	989.0	664.7
ART-TPP	161.0	384.5	784.0	523.3	**182.0**	478.0	947.0	622.7	198.4	**486.0**	**968.0**	**646.0**
RRT-LAZ	**124.0**	**305.5**	**591.0**	**401.3**	188.8	473.5	942.0	634.0	**190.6**	496.0	1005.0	667.3

TABLE V. F-measures for 3-dimensional input spaces

	Block Pattern				Strip Pattern				Point Pattern			
failure rate	**0.0050**	**0.0020**	**0.0010**	**0.0015**	**0.0050**	**0.0020**	**0.0010**	**0.0015**	**0.0050**	**0.0020**	**0.0010**	**0.0015**
RT	200.0	500.0	1000.0	666.7	200.0	500.0	1000.0	666.7	200.0	500.0	1000.0	666.7
RRT	160.0	373.5	741.0	502.0	188.2	468.0	955.0	**639.3**	207.4	518.0	1009.0	686.0
ART-B	165.0	394.5	824.0	532.0	193.6	489.5	990.0	676.0	201.4	**491.5**	995.0	658.7
ART-RP	169.0	436.0	875.0	577.3	198.8	492.5	982.0	674.7	198.4	492.5	1026.0	**652.7**
ART-TPP	171.8	434.0	859.0	582.7	189.6	494.0	955.0	656.7	**195.2**	505.0	1015.0	686.0
RRT-LAZ	**157.8**	**360.5**	**736.0**	**496.0**	**186.6**	**464.5**	**953.0**	644.0	201.2	496.5	**993.0**	667.3

TABLE VI. F-measures for 4-dimensional input spaces

	Block Pattern				Strip Pattern				Point Pattern			
failure rate	**0.0050**	**0.0020**	**0.0010**	**0.0015**	**0.0050**	**0.0020**	**0.0010**	**0.0015**	**0.0050**	**0.0020**	**0.0010**	**0.0015**
RT	200.0	500.0	1000.0	666.7	200.0	500.0	1000.0	666.7	200.0	**500.0**	1000.0	666.7
RRT	186.2	453.5	868.0	591.3	182.8	460.0	918.0	610.0	198.0	502.0	1053.0	690.0
ART-B	180.0	444.0	883.0	580.7	188.4	464.0	**868.0**	577.3	201.8	516.0	1060.0	**687.3**
ART-RP	164.6	435.0	903.0	597.3	179.2	477.5	916.0	620.0	199.6	519.5	**1045.0**	689.3
ART-TPP	197.8	506.5	977.0	650.0	192.0	463.5	911.0	614.7	**197.4**	502.0	1052.0	694.0
RRT-LAZ	**162.2**	**419.0**	**831.0**	**556.0**	**169.0**	**427.5**	889.0	**572.0**	197.8	507.0	1055.0	690.0

TABLE VII. F-measures for 10-dimensional input spaces

	Block Pattern				Strip Pattern				Point Pattern			
failure rate	**0.0050**	**0.0020**	**0.0010**	**0.0015**	**0.0050**	**0.0020**	**0.0010**	**0.0015**	**0.0050**	**0.0020**	**0.0010**	**0.0015**
RT	200.00	500.00	1000.00	666.67	**200.00**	500.00	1000.00	666.67	**200.00**	500.00	1000.00	666.67
RRT	275.34	597.35	1405.60	905.13	290.14	672.25	1281.30	872.07	208.02	473.45	918.50	675.80
ART-B	194.30	486.65	975.20	649.27	217.12	506.30	**952.40**	651.40	226.00	509.60	993.00	734.47
ART-RP	**192.06**	**429.45**	**826.40**	**551.00**	219.60	**449.10**	971.30	660.67	234.48	533.95	1039.60	746.80
ART-TPP	261.84	555.45	1292.10	845.80	284.94	589.00	1257.70	843.47	211.54	485.65	919.30	**647.53**
RRT-LAZ	195.18	476.40	954.20	658.80	214.56	484.45	938.80	**644.20**	205.00	**466.00**	**878.90**	664.00

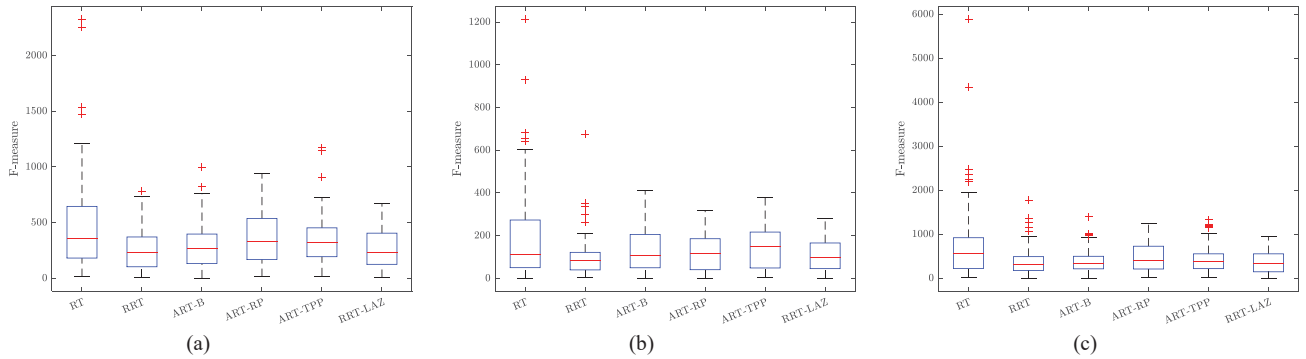


FIGURE 9. Box plots of F-measures of RT and ART methods for 1-dimensional input spaces. (a) Failure rate θ = 0.0020. (b) Failure rate θ = 0.0050. (c) Failure rate θ = 0.0015

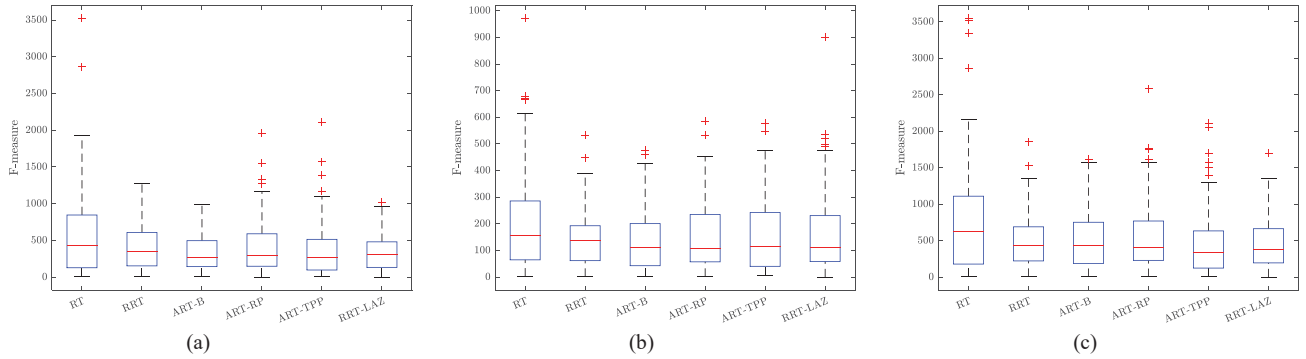


FIGURE 10. Box plots of F-measures of RT and ART methods for 2-dimensional input spaces. (a) Failure rate θ = 0.0020. (b) Failure rate θ = 0.0050. (c) Failure rate θ = 0.0015

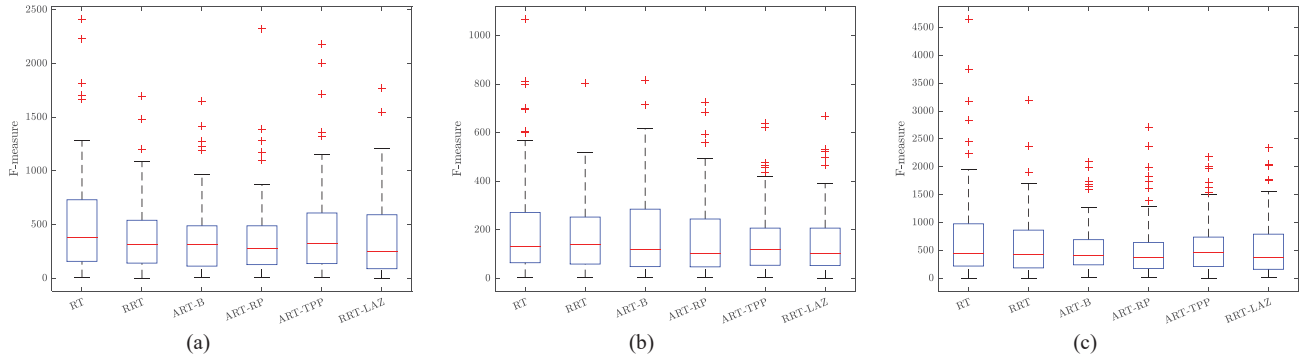


FIGURE 11. Box plots of F-measures of RT and ART methods for 3-dimensional input spaces. (a) Failure rate θ = 0.0020. (b) Failure rate θ = 0.0050. (c) Failure rate θ = 0.0015

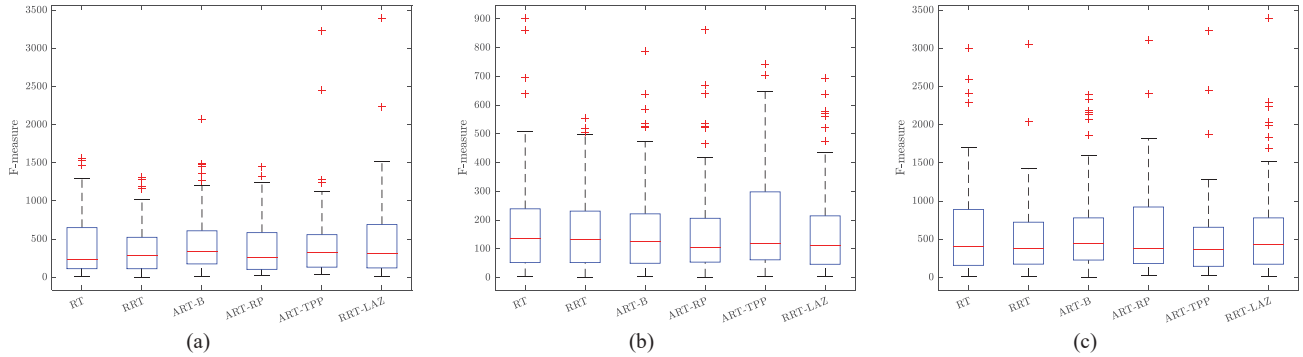


FIGURE 12. Box plots of F-measures of RT and ART methods for 4-dimensional input spaces. (a) Failure rate θ = 0.0020. (b) Failure rate θ = 0.0050. (c) Failure rate θ = 0.0015

TABLE VIII. Input spaces and fault information of subject programs.

Program	Input Space		Seeded Faults				Failure Rate
	From	To	AOR	ROR	SVR	CR	
airy	(−5000.0)	(5000.0)				1	0.000716
bessj	(2.0, −1000.0)	(300.0, 15000.0)	2	1		1	0.001298
bessj0	(−300000.0)	(300000.0)	2	1	1	1	0.001373
cel	(0.001, 0.001, 0.001, 0.001)	(1.0, 300.0, 10 000.0, 1000.0)	1	1		1	0.000332
el2	(0.0, 0.0, 0.0, 0.0)	(250.0, 250.0, 250.0, 250.0)	1	3	2	3	0.000690
erfcc	(−30000.0)	(30000.0)	1	1	1	1	0.000574
gammq	(0.0, 0.0)	(1700.0, 40.0)		3		1	0.000830
golden	(−100.0, −100.0, −100.0)	(60.0, 60.0, 60.0)		3	1	1	0.000550
plgndr	(10.0, 0.0, 0.0, −50000.0)	(500.0, 11.0, 1.0, 5000.0)	1	2		2	0.000368
probks	(−50000.0)	(50000.0)	1	1	1	1	0.000387
sncndn	(−5000.0, −5000.0)	(5000.0, 5000.0)			4	1	0.001623
tanh	(−500.0)	(500.0)	1	1	1	1	0.001817

TABLE IX. F-measures for subject programs.

	airy	bessj	bessj0	cel	el2	erfcc	gammq	golden	plgndr	probks	sncndn	tanh
RT	1398.648	767.416	738.332	3022.048	1437.275	1741.160	1204.819	1818.182	2717.391	2583.979	616.143	550.358
RRT	806.006	633.205	454.406	2936.747	825.362	1024.739	1094.337	1739.818	1689.402	1580.878	645.287	327.188
ART-B	763.687	**620.108**	495.339	2832.831	1435.072	1021.951	1137.710	1779.272	1904.619	1837.726	619.532	356.742
ART-RP	898.464	726.888	509.031	3491.867	1736.086	1161.150	1210.361	1782.000	3055.434	1790.697	626.186	365.768
ART-TPP	865.782	702.311	481.937	3129.518	1074.202	1096.690	1136.144	1780.727	1716.847	1693.023	619.039	364.942
RRT-LAZ	**703.352**	643.374	**408.594**	**2825.301**	**808.841**	**946.168**	**1091.687**	**1735.818**	**1643.478**	**1454.521**	**618.115**	**299.835**

B. EMPIRICAL STUDY

Previous ART studies used 12 real-life programs to evaluate the effectiveness of ART algorithms in practical situations. These programs were extracted from *Numerical Recipes* [24] and ACM's *Collected Algorithms* [25]. Some faults were seeded into the subject programs using the following 4 mutation operators [26].

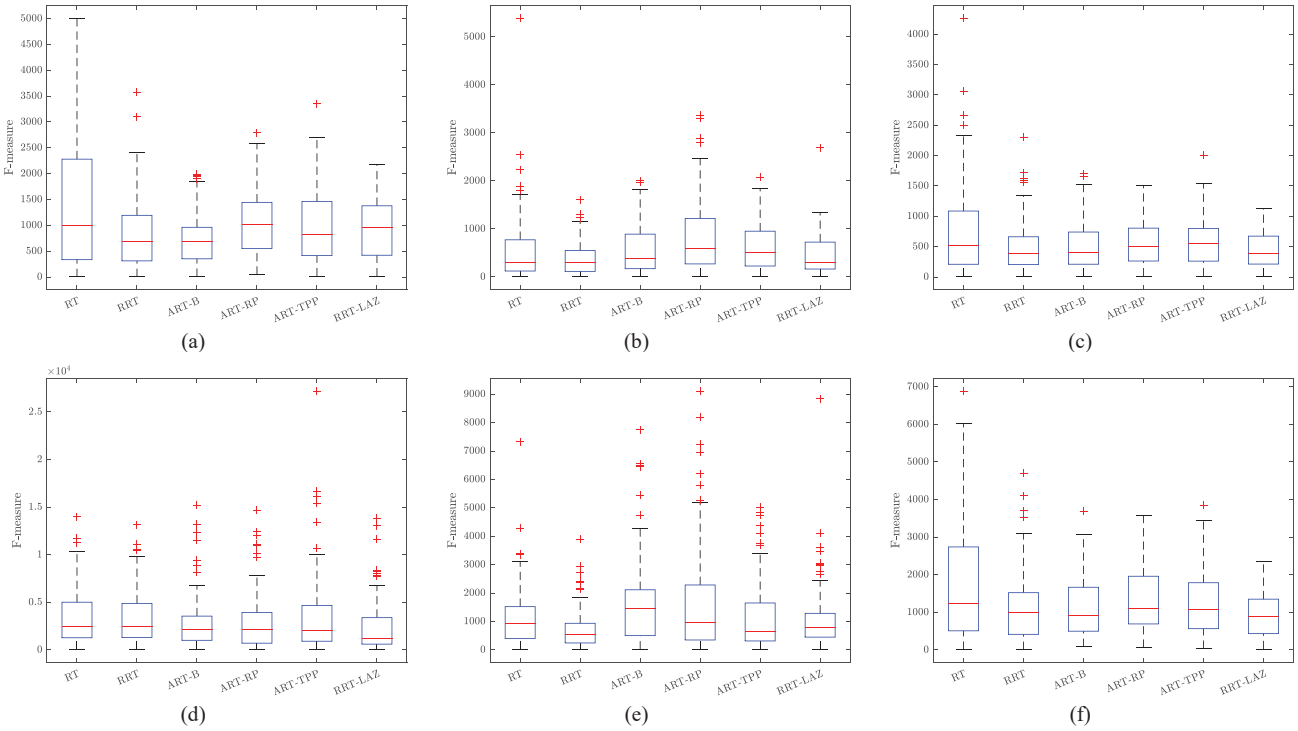
1) arithmetic operator replacement (AOR);
2) relational operator replacement (ROR);
3) scalar variable for scalar variable replacement (SVR);
4) constant replacement (CR).

The input spaces and fault information of these programs are shown in Table VIII. Note that a hyperrectangular input space was defined for each subject program.

We used related ART algorithms to test the subject programs in the empirical study. Table IX shows the F-measures of RT and related ART algorithms. All the results in this table are averaged over 2000 test runs for each subject program using different seeds. For 11 of the 12 real-life subject programs (except for Program **bessj**), we find that RRT-LAZ has a better F-measure than RT and related ART techniques. As for Program **bessj**, since there are two seeded faults via the AOR operator, and since its input space is two-dimensional, RRT-LAZ does not exhibit the best F-measure.

To show the distribution of F-measures for each algorithm in these programs, we also draw box plots of the respective F-measures as shown in Figure 13. For each F-measure, we ran the experiment 100 times and obtained the average value. The dispersions of the F-measures can also be observed in the figure. From these statistical data, we have the following observations.

1) For the majority of the subject programs (7 out of 12, except for Programs **airy**, **el2**, **gammq**, **plgndr**, and **probks**), the median of RRT-LAZ is smaller than those of related approaches.
2) For the majority of the subject programs (except for **airy**, **bessj**, **el2**, **golden**, and **sncndn**), the interquartile range (the box length) and the maximum (the top end of the whiskers) of RRT-LAZ are smaller than those of related approaches, which shows that RRT-LAZ has a more stable data distribution than the others.



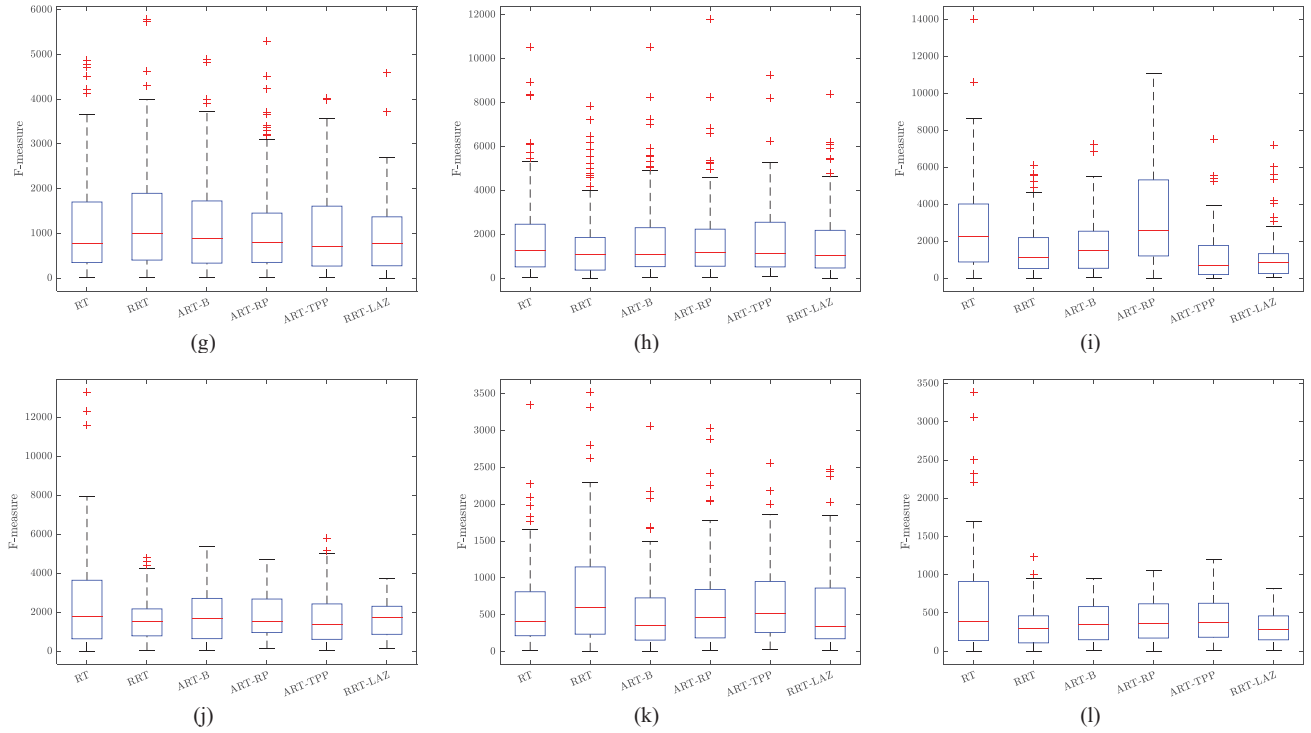


FIGURE 13. **Box plots of F-measures for 12 subject programs. (a) airy. (b) bessj. (c) bessj0. (d) cel. (e) el2. (f) erfcc. (g) gammq. (h) golden. (i) plgndr. (j) probks. (k) sncndn. (l) tanh**

V. THREATS TO VALIDITY

Construct validity refers to the extent that our experimental studies evaluate the actual comparisons of RT and ART techniques. We have adopted the F-measure and the test case generation time as the effectiveness and efficiency metrics. The use of other metrics may give different results.

Internal validity refers to how well our experimental studies are conducted, especially with respect to the dependency of the variables. We have used both simulated and real-life programs in our studies. We have carefully implemented and tested our evaluation tools to assure correctness. Also, the effectiveness of the RRT and RRT-LAZ algorithms depend on the exclusion rate R. In our studies, we have set the value of R to be the same for both algorithms. Setting distinct values of R may produce different results.

External validity refers to the degree that our experimental studies can be generalized. We have used 12 subject programs from *Numerical Recipes* and ACM's *Collected Algorithms*. While they are commonly adopted in other ART studies, the use of other programs may produce different results. Having more subject programs with larger sizes can strengthen the generalization.

VI. FURTHER DISCUSSIONS ON RELATED WORK

Numerous adaptive random testing algorithms have been developed by various authors. It is practically impossible to compare a new ART approach with all existing algorithms experimentally.

We have compared RRT-LAZ with RT because the latter is the original technique that adaptive random testing targeted to improve.

RRT-LAZ involves two concepts, namely, exclusion and partitioning. It is, therefore, appropriate to compare it with RRT (which is based on exclusion) as well as ART-B, ART-RP, and ART-TPP (which are based on partitioning).

The fixed-size-candidate-set (FSCS) algorithm using the max-min candidate selection criterion [7] is based on the idea of distances. It first generates a fixed number of candidate test cases and calculates the minimum distance between each candidate and all the previously executed test cases. Then, the candidate with the maximum distance among all the computed minimum distances is selected as the next test case. As we have explained, it is unrealistic to compare a new ART approach with all existing algorithms. Since FSCS is not related to the three basic concepts in RRT-LAZ, we have not included it in our experimental studies.

Chen et al. [27] proposed a mirror adaptive random testing (MART) approach. First, it partitions the given input space into m equivalent disjoint subspaces. It picks one of the

subspaces and applies an ART algorithm (say, RRT) to select a test case. It then uses the mirroring function to generate a test case for each of the remaining m–1 subspaces without the need to apply RRT again. MART exhibits the same F-measure as the RRT algorithm but needs only $1/m^2$ of the time for distance computation using RRT.

Chan et al. [28] recommended the concept of forgetting for enhancing adaptive random testing. It restricts the execution of an ART algorithm (say, RRT) to a limited number of previously executed test cases. In this way, the time complexity can be improved.

ART with dynamic non-uniform candidate distribution (ART-DNC) [29] is another attempt to improve adaptive random testing by the use of failure-driven test profiles. It reduces the time cost by tuning the process according to the test case distribution in each dimension.

The enhancement of RRT by mirroring, forgetting, and dynamic non-uniform candidate distribution are orthogonal to the enhancement of RRT through the largest available zone. In other words, RRT-LAZ can also be further improved by mirroring, forgetting, or DNC.

Shahbazi and others [30], [31] presented another alternative approach to random testing. They use centroidal Voronoi tessellations to maximize the test case coverage of the input space and presented an RBCVT-Fast algorithm with a time complexity of $O(n)$. However, testers need to specify the target number of test cases (n) at the beginning. If testers overestimate n, they will waste a lot of time in test case generation. On the other hand, if testers underestimate n, after executing the first batch of targets, another set of n evenly spread test cases will need to be generated without taking the first set into consideration. Our algorithm does not need to assume the target number of test cases prior to testing. There is no problem related to overestimation or underestimation.

Recently, ART algorithms have also been proposed for non-numerical inputs. For instance, a linear-time ARTsum algorithm was recommended by Barus et al. [32] using the concepts of categories and choices in category-partition testing. Ciupa et al. [8] and Chen et al. [9] introduced the ARTOO and the OMISS metrics for modeling the distances between objects and between method innovation sequences. We have not compared RRT-LAZ with these algorithms experimentally because they cater for non-numerical inputs.

VII. CONCLUSION AND FUTURE WORK

Adaptive random testing enhances the effectiveness of random testing by making use of the failure distribution properties of most programs under test. Restricted random testing makes use of exclusion zones around previously executed test cases and selects the next test case outside these zones. A major shortcoming of RRT lies in the time overhead due to the generation of candidate cases, which may or may not be outside the exclusion zones. To alleviate the problem, we propose in this paper an RRT-LAZ approach that proactively determines the largest available zone from which the next test case is randomly selected. Compared with RRT, the RRT-LAZ algorithm significantly reduces the time cost while preserving the effectiveness of failure detection. Further experimental results show that, for block failure patterns in low-dimensional input spaces, RRT-LAZ is more effective than RT and the related ART techniques that we have studied. By considering the time cost of test cases, RRT-LAZ is more cost-effective than the other approaches for both block and stripe failure patterns in the input spaces for all dimensions

As future work, we will extend our empirical study to higher-dimensional input spaces. We will test larger-scale programs with RRT-LAZ. We will also customize our RRT-LAZ tool to support the testing of OO software.

REFERENCES

[1] "The Economic and Social Impact of Software & Services on Competitiveness and Innovation (SMART 2015/0015): Final Study Report," European Commission, 2015.

[2] "The Economic Impacts of Inadequate Infrastructure for Software Testing: Final Report," National Institute of Standards and Technology, Gaithersburg, MD, 2002.

[3] *GDP*, The World Bank. [Online]. Available: https://data.worldbank.org/indicator/NY.GDP.MKTP.CD/.

[4] T. Y. Chen and R. G. Merkel, "An upper bound on software testing effectiveness," *ACM Trans. Softw. Eng. Methodol.*, vol. 17, no. 3, pp. 16:1–16:27, 2008.

[5] T. Y. Chen, F.-C. Kuo, R. G. Merkel, and T. H. Tse, "Adaptive random testing: The ART of test case diversity," *J. Syst. Softw.*, vol. 83, no. 1, pp. 60–66, 2010.

[6] W. J. Gutjahr, "Partition testing vs. random testing: The influence of uncertainty," *IEEE Trans. Softw. Eng.*, vol. 25, no. 5, pp. 661–674, 1999.

[7] T. Y. Chen, H. Leung, and I. K. Mak, "Adaptive random testing," *in Advances in Computer Science: Proc. 9th Asian Comput. Sci. Conf. (ASIAN) (Lecture Notes in Computer Science 3321)*, 2004, pp. 320–329.

[8] I. Ciupa, A. Leitner, M. Oriol, and B. Meyer, "ARTOO: Adaptive random testing for object-oriented software," in *Proc. 30th Int. Conf. Softw. Eng. (ICSE)*, 2008, pp. 71–80.

[9] J. Chen, F.-C. Kuo, T. Y. Chen, D. Towey, C. Su, and R. Huang, "A similarity metric for the inputs of OO programs and its application in adaptive random testing," *IEEE Trans. Rel.*, vol. 66, no. 2, pp. 373–402, 2017.

[10] Y. Lin, X. Tang, Y. Chen, and J. Zhao, "A divergence-oriented approach to adaptive random testing of Java programs," in *Proc. 24th IEEE/ACM Int. Conf. Autom. Softw. Eng. (ASE)*, 2009, pp. 221–232.

[11] 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. (SEKE)*, 2007, pp. 141–144.

[12] T. Y. Chen, G. Eddy, R. G. Merkel, and P. K. Wong, "Adaptive random testing through dynamic partitioning," in *Proc. 4th Int. Conf. Qual. Softw. (QSIC)*, 2004, pp. 79–86.

[13] H. Ackah-Arthur, J. Chen, J. Xi, M. Omari, H. Song, and R Huang, "A cost-effective adaptive random testing approach by dynamic restriction," *IET Softw.*, vol.12, no. 6, pp. 489–497,2018.

[14] C. Mao, "Adaptive random testing based on two-point partitioning," *Informatica*, vol. 36, no. 3, pp. 297–303, 2012.

[15] K. P. Chan, T. Y. Chen, and D. Towey, "Restricted random testing," in *Proc. 7th European Conf. Softw. Qual. (ECSQ)* (*Lecture Notes in Computer Science 2349*), 2002, pp. 321–330.

[16] 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.
[17] P. G. Bishop, "The variation of software survival time for different operational input profiles (or why you can wait a long time for a big bug to fail)," in *Dig. Papers 23rd Int. Symp. Fault-Tolerant Comput. (FTCS)*, 1993, pp. 98–107.
[18] 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.
[19] L. J. White and E. I. Cohen, "A domain strategy for computer program testing," *IEEE Trans. Softw. Eng.*, vol. 6, no. 3, pp. 247–257, 1980.
[20] K. P. Chan, T. Y. Chen, and D. Towey, "Normalized restricted random testing," in *Proc. 8th Int. Conf. Rel. Softw. Technol. (Ada-Europe) (Lecture Notes in Computer Science 2655)*, 2003, pp. 368–381.
[21] J. Mayer and C. Schneckenburger, "An empirical analysis and comparison of random testing techniques," in *Proc. 2006 ACM/IEEE Int. Symp. Empirical Softw. Eng.* (ISESE), 2006, pp. 105–114.
[22] A. R. Butz, "Alternative algorithm for Hilbert's space-filling curve," *IEEE Trans. Comput.*, vol. 20, no. 4, pp. 424–426, 1971.
[23] H. Liu, X. Xie, J. Yang, Y. Lu, and T. Y. Chen, "Adaptive random testing by exclusion through test profile," in *Proc. 10th Int. Conf. Qual. Softw. (QSIC)*, 2010, pp. 92–101.
[24] W. H. Press, *Numerical Recipes 3rd Edition: The Art of Scientific Computing*, Cambridge, UK: Cambridge University Press, 2007.
[25] *Collected Algorithms*, ACM. [Online]. Available: http://calgo.acm.org/.
[26] R. A. DeMillo, R. J. Lipton, and F. G. Sayward, "Hints on test data selection: Help for the practicing programmer," *IEEE Comput.*, vol. 11, no. 4, pp. 34–41, 1978.
[27] T. Y. Chen, F.-C. Kuo, R. G. Merkel, and S. P. H. Ng, "Mirror adaptive random testing," *Inf. Softw. Technol.*, vol. 46, no. 15, pp. 1001–1010, 2004.
[28] K. P. Chan, T. Y. Chen, and D. Towey, "Forgetting test cases," in *Proc. 30th Annu. Int. Comput. Softw. Appl. Conf.* (COMPSAC), 2006, pp. 485–494.
[29] T. Y. Chen, F.-C. Kuo, and H. Liu, "Application of a failure driven test profile in random testing," *IEEE Trans. Rel.*, vol. 58, no. 1, pp. 179–192, 2009.
[30] A. Shahbazi, A. F. Tappenden, and J. Miller, "Centroidal Voro-noi tessellations: A new approach to random testing," *IEEE Trans. Softw. Eng.*, vol. 39, no. 2, pp. 163–183, 2013.
[31] A. F. Tappenden and J. Miller, "A novel evolutionary approach for adaptive random testing," *IEEE Trans. Rel.*, vol. 58, no. 4, pp. 619–633, 2009.
[32] A. C. Barus, T. Y. Chen, F.-C. Kuo, H. Liu, R. G. Merkel, and G. Rothermel, "A cost-effective random testing method for programs with non-numeric inputs," *IEEE Trans. Comput.*, vol. 65, no. 12, pp. 3509–3523, 2016.

JINFU CHEN received the BE degree in 2004 from Nanchang Hangkong University, Nanchang, China and the PhD degree in 2009 from Huazhong University of Science and Technology, Wuhan, China, both in computer science. He is currently a full professor in the School of Computer Science and Communication Engineering, Jiangsu University, Zhenjiang, China. His major research interests include software testing, software analysis, and trusted software.

QIHAO BAO received the BE degree in software engineering in 2017 from Jiangsu University, Zhenjiang, China, where he is currently working toward an MS degree in the School of Computer Science and Communication Engineering. His research interests include software testing and software analysis.

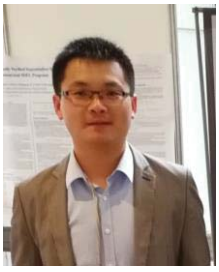
T.H. TSE received the PhD degree from the London School of Economics and was a visiting fellow at the University of Oxford. He is an honorary professor in computer science at The University of Hong Kong after retiring from the full professorship in 2014. His research interest is in program testing and debugging. He is the steering committee chair of the IEEE International Conference on Software Quality, Reliability, and Security, an associate editor of *IEEE Transactions on Reliability*, and an editorial board member of *Software Testing, Verification and Reliability* and *Software: Practice and Experience*. He also served on the search committee for the editor-in-chief of *IEEE Transactions on Software Engineering*. He is a fellow of the British Computer Society. He was awarded an MBE by The Queen of the United Kingdom.

TSONG YUEH CHEN received the BSc and MPhil degrees from The University of Hong Kong, the MSc degree and DIC from the Imperial College of Science and Technology, University of London, UK, and the PhD degree from the University of Melbourne, Australia. He is currently a professor of software engineering in the Department of Computer Science and Software Engineering, Swinburne University of Technology, Australia. He is the inventor of metamorphic testing and adaptive random testing. His current research interests include software testing, debugging, and program repair.

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

CHENGYING MAO received the BE degree in Computer Science and Technology in 2001 from Central South University, Changsha, China and the PhD degree in Computer Software and Theory in 2006 from Huazhong University of Science and Technology, Wuhan, China. He worked as a post-doctoral researcher in the College of Management at Huazhong University of Science and Technology from 2006 to 2008. He is now a full professor in the School of Software and IoT Engineering at Jiangxi University of Finance and Economics, Nanchang, China. His current research interests include software engineering and service computing. He is a member of the ACM, IEEE, IEEE Computer Society, and CCF.

MINJIE YU received the MS degree in software engineering in 2017 from Jiangsu University, Zhenjiang, China. Her research interests include software testing and software analysis.



RUBING HUANG received the Ph.D. degree in computer science and technology from the Huazhong University of Science and Technology, Wuhan, China, in 2013. From 2016 to 2018, he was a visiting scholar at Swinburne University of Technology and at Monash University, Australia. He is an associate professor in the Department of Software Engineering, School of Computer Science and Communication Engineering, Jiangsu University, Zhenjiang, China. His current research interests include software testing (including adaptive random testing, random testing, combinatorial testing, and regression testing), debugging, and maintenance. He has more than 50 publications in journals and proceedings, including in IEEE Transactions on Software Engineering, IEEE Transactions on Reliability, Journal of Systems and Software, Information and Software Technology, IET Software, The Computer Journal, International Journal of Software Engineering and Knowledge Engineering, ICSE, ICST, COMPSAC, QRS, SEKE, and SAC. He is a senior member of the China Computer Federation, and a member of the IEEE and the ACM. More about him and his work is available online at https://huangrubing.github.io/.