Hindawi
Mathematical Problems in Engineering
Volume 2019, Article ID 9381728, 15 pages
https://doi.org/10.1155/2019/9381728

Research Article

An Enhanced Adaptive Random Testing by Dividing Dimensions Independently

Zhibo Li, Qingbao Li, and Lei Yu

State Key Laboratory of Mathematical Engineering and Advanced Computing, Zhengzhou, China

Correspondence should be addressed to Zhibo Li; lizhibo1019@163.com

Received 11 April 2019; Revised 10 August 2019; Accepted 27 August 2019; Published 13 October 2019

Academic Editor: A. M. Bastos Pereira

Random testing (RT) is widely applied in the area of software testing due to its advantages such as simplicity, unbiasedness, and easy implementation. Adaptive random testing (ART) enhances RT. It improves the effectiveness of RT by distributing test cases as evenly as possible. Fixed Size Candidate Set (FSCS) is one of the most well-known ART algorithms. Its high failure-detection effectiveness only shows at low failure rates in low-dimensional spaces. In order to solve this problem, the boundary effect of the test case distribution is analyzed, and the FSCS algorithm of a limited candidate set (LCS-FSCS) is proposed. By utilizing the information gathered from success test cases (no failure-causing test inputs), a tabu generation domain of candidate test case is produced. This tabu generation domain is eliminated from the current candidate test case generation domain. Finally, the number of test cases at the boundary is reduced by constraining the candidate test case generation domain. The boundary effect is effectively relieved, and the distribution of test cases is more even. The results of the simulation experiment show that the failure-detection effectiveness of LCS-FSCS is significantly improved in high-dimensional spaces. Meanwhile, the failure-detection effectiveness is also improved for high failure rates and the gap of failure-detection effectiveness between different failure rates is narrowed. The results of an experiment conducted on some real-life programs show that LCS-FSCS is less effective than FSCS only when the failure distribution is concentrated on the boundary. In general, the effectiveness of LCS-FSCS is higher than that of FSCS.

1. Introduction

While software is increasing in scale and complexity, also the quality of software has attracted more and more attention. As an important task of software quality assurance, software testing is becoming increasingly important in software development [1].

The Software Under Test (SUT) usually has a large input domain space. Therefore, it is important to select test inputs that can effectively identify software failure as test cases. Test case generation technology, such as combinatorial testing [2], symbolic execution [3], random testing (RT) [4, 5], partition testing [6], test case generation technology based on finite state machine [7], or test case generation technology based on search [8], guides the generation of effective test cases. RT is a simple and easy-to-implement test method. It does not need complex software requirements or structural information of programs. It only requires selecting test cases randomly in the input domain. Since RT does not utilize any information of the SUT, it has the disadvantages of high redundancy, low coverage, and blindness in the test case generation. RT is even considered the worst testing method by Myers [9]. However, RT has the advantages of simplicity, easy implementation, low costs, unbiasedness, and fast execution. It is usually used in combination with other testing methods in software testing and in reliability evaluation field [10, 11]. At the same time, in theory all test cases that can be generated by any testing method can be generated by RT as well. Thus, RT has the potential to detect all failures [12].

Experimental studies [13] have found that failure-causing inputs tend to cluster in continuous areas. Based on this conclusion, Chen et al. [14] proposed adaptive random testing (ART). Compared with RT that does not use any

information to generate test cases randomly, ART achieves evenly distributed test cases by using the information of success test cases.

Experiments [14] show that in failure detection, ART performs better than RT, which means that the number of test cases that are needed to trigger the first failure is lower in ART. Various algorithms based on ART have been proposed, for example, distance ART algorithm (D-ART) [14], restricted ART algorithm (RRT) [15], partitioning adaptive random testing [16], and quasi-random testing [17].

FSCS [14] is one of the most well-known ART algorithms, but it does not perform well in high-dimensional spaces and at low failure rates [1, 12]. It is pointed out in the literature [18] that the best effectiveness is reached for 50% of RT. In order to improve the effectiveness of FSCS, the boundary effect of the test case distribution is analyzed, and a novel algorithm, the FSCS algorithm of a limited candidate set (LCS-FSCS), is proposed in this paper. LCS-FSCS effectively relieves this boundary effect and distributes test cases more evenly.

The rest of this paper is organized as follows: The distribution of test cases and the effectiveness of FSCS are analyzed in Section 2. Section 3 presents the LCS-FSCS approach. In Section 4, LCS-FSCS is compared with FSCS through simulation experiments. Settings and results of empirical studies are reported in Section 5. Threats to validity are discussed in Section 6. Finally, the conclusion and future work are presented in Section 7.

2. Analysis of FSCS

FSCS uses a distance-based selection criterion to evaluate a fixed set of randomly generated test case candidates. An initial test case is selected randomly. For each subsequent test case, this test case is selected from a candidate test case that has a maximum-minimum distance to any other existing test case. Let $\mathrm{TS} = \{T_1, T_2, \ldots, T_n\}$ be the executed set and $\mathrm{CS} = \{C_1, C_2, \ldots, C_k\}$ be the candidate set such that $\mathrm{TS} \cap \mathrm{CS} = \varnothing$. The best test case best $\in$ CS can be selected by the following formula, $\min_{i=1}^{n} \mathrm{dist}(\mathrm{best}, t_i) \geq \min_{i=1}^{n} \mathrm{dist}(c_j, t_i)$, where dist is defined as the Euclidean distance.

2.1. Analysis of Test Case Spatial Distribution of FSCS. For a 2-dimensional input domain, we assume that each dimension has a range of [1, 1000]. Suppose that FSCS generates 100 test cases continuously without failure and runs a total of 1000 times. Finally, the distribution of test cases is analyzed in each dimension, as shown in Figures 1 and 2.

The center of the input domain is uniformly distributed in each dimension. However, the number of test cases on the boundary is higher than the number of test cases within this center area. This is the so-called boundary effect; the FSCS is prone to generating more test cases on the boundary.

2.2. Analysis of Effectiveness of FSCS

2.2.1. Difference of Failure-Detection Effectiveness in Different Failure Rates. It is assumed that in the block failure pattern,

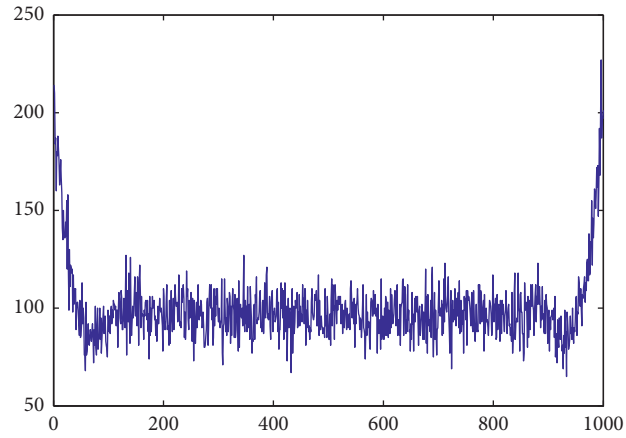

Figure 1: The distribution of test case in the X dimension.

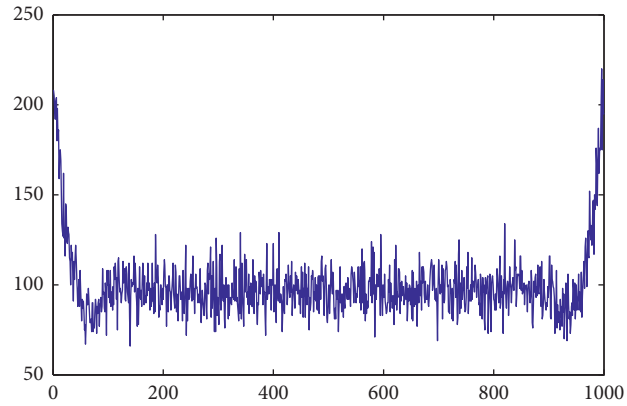

Figure 2: The distribution of test case in the Y dimension.

Table 1: The effectiveness of FSCS with different failure rates in 2D.

Failure rate	0.0005	0.001	0.005	0.01	0.05	0.1
F-ratio	0.6	0.61	0.63	0.65	0.69	0.73

the failure rates are 0.0005, 0.001, 0.005, 0.01, 0.05, and 0.1 respectively, in a 2-dimensional input domain.,running 2000 times for each failure rate; the average of F-count is calculated as the F-measure. The F-ratio is the ratio of F-FSCS to F-RT as shown in Table 1 (F-ratio = F-FSCS/F-RT).

From Table 1, it can be seen that the effectiveness of FSCS improves with a decrease in the failure rate. The reason for this is that for a larger failure rate, a smaller average number of test cases are needed to detect the first failure. According to the spatial distribution of FSCS test cases, the initial test cases generated by FSCS are easy to concentrate on the boundary [18]. As the number of test cases increases, the distribution of test cases becomes more even. For this reason, the advantage of FSCS is more obvious for lower failure rates.

2.2.2. Failure-Detection Effectiveness in Different Dimensions. We analyze the effectiveness of FSCS in a 2D-5D input domain under the assumption that the failure rate is 0.001 in the block failure pattern [19] and FSCS runs 2000 times.

Table 2: Effectiveness of FSCS in different dimensions for a failure rate of 0.001, in block pattern.

Dimension	2D	3D	4D	5D
F-ratio	0.61	0.707	0.777	0.958

As can be seen from Table 2, with an increase in the input space dimension, the failure-detection effectiveness of FSCS decreases rapidly. According to the analysis from the literature [18], the higher the dimension of the input domain is, the more likely the failure domain will concentrate on the middle of the input domain, whereas the test cases generated by FSCS prefer to focus on the boundary of the input domain; thus, the failure-detection effectiveness of FSCS is poor in a high-dimensional input domain.

3. Proposed Approach

3.1. Underlying Concept. With respect to the analysis results regarding the effectiveness and the spatial distribution of the test cases generated by FSCS, the FSCS of limited candidate set (LCS-FSCS) algorithm is proposed. By limiting the candidate test case generation domain, test cases are more evenly distributed and the boundary effect is eliminated.

To constrain the candidate test case generation domain D_c, first each dimension is divided into p equal subdomains. When the "best" test case tc_i is generated and it does not detect any failure, we transfer it to the execution test case set TS. The subdomains of each dimension of the tc_i are deleted from the candidate test case generation domain, so that candidate test cases are generated using the remaining $p-1$ subdomains. When the candidate test case generation domain D_c is empty, D_c is reinitialized with an input domain and divided into p subdomains. Then, the next test case is generated based on this procedure.

Assume that each dimension of an input domain is divided into five equal subdomains in 2D. That is, the input domain is divided into a 5×5 grid. The first test case tc_1 is randomly generated. Suppose that tc_1 does not detect any failure, it is therefore put into the executed test case set TS. (1) Using FSCS as shown in Figure 3, four candidate test cases (c_1, c_2, c_3, and c_4) are randomly generated in the input domain. On the basis of the Euclidean distance, the candidate test case c_3 with the max-min distance is selected as the next test case tc_2. The horizontal coordinate value of c_3 is very close to that of tc_1. This does not conform to the idea of uniform distribution from the perspective of the abscissa. (2) Using LCS-FSCS as shown in Figure 4, the success test case generated subdomain is removed from the candidate test case generation domain (that is, the shaded area in Figure 4 is excluded); as a next step, four candidate test cases (c_1', c_2', c_3', and c_4') are randomly generated in the remaining subdomains; Finally, the max-min distance between the test cases in TS and the candidate test cases is calculated, and the optimal candidate test case c_2' is selected as the next test case tc_2. Test case c_2' and test case tc_1 belong to different subdomains in each dimension. Diversity is a key characteristic of successful testing strategies [20].

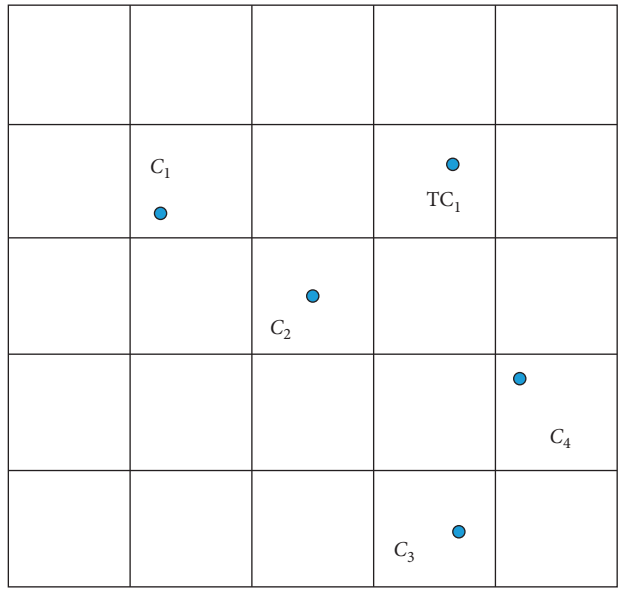

Figure 3: FSCS test case generation.

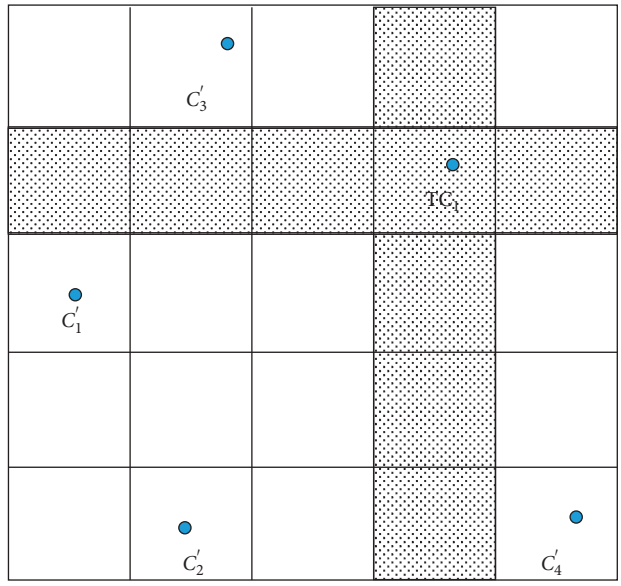

Figure 4: LCS-FSCS test case generation.

The proportional sampling strategy (PSS) [19] indicates that test cases should be randomly selected in proportion to the size of different partitions. The LCS-FSCS algorithm equally divides each dimension of the input domain. From the perspective of the independent dimension, this is consistent with PSS.

Definition 1 (equal subdomain in the independent dimension). For an n-dimensional input domain $D = \{D_1, D_2, \ldots, D_n\}$, if each dimension of a space is divided into p equal parts, then each part can be denoted as $D_i = \{d_{i1}, d_{i2}, \ldots, d_{ip}\}$, where $d_{ik}\,(1 \le k \le p)$ is the position identifier of partitioning in the $i_{\text{th}}\,(1 \le i \le n)$

Input:
(1) The size of candidate set (CS) is denoted as k $(k > 1)$
(2) The input domain D
(3) The dimension number (n) of input domain
(4) The partition number (p) of each dimension
Output:
The set of test cases TS
(1) Input parameters k, n, D, and p;
(2) Set TS = {}, D_c = {};
(3) Dividing each dimension into p equal parts $D_i = \{d_{i1}, d_{i2}, \ldots, d_{ip}\}$ //(Definition 1) Init D domain;
(4) Set $D_c = \{D_1, D_2, \ldots, D_n\}$; //init D_c
(5) while (termination condition is not satisfied) do
(6) tc = Call procedure GenTcByLCSFscs (D_c, TS); //to get a "best" test case tc from D_c domain;
(7) Add tc into TS;
(8) if tc is in the failure domain, then break;
(9) end while
(10) return TS;

Algorithm 1: LCS-FSCS algorithm.

dimension, with $1 \le k \le p$, $d_{ia} \cap d_{ib} = 0$, $a \ne b$, and $\cup_{j=1}^{p} d_{ij} = D_i$.

Definition 2 (location). In an n-dimensional input domain, assume a test case tc = $(x_1, x_2, \ldots, x_n)$, where x_i is the value in the i_{th} $(1 \le i \le n)$ dimension, $x_i \in d_{ik}$; then, the region of x_i in the i_{th} $(1 \le i \le n)$ dimension is denoted as $d_{ik} = \text{loc}(x_i)$, where $1 \le k \le p$.

Definition 3 (tabu subdomain). In an n-dimensional input domain, assume a test case tc = $(x_1, x_2, \ldots, x_n)$ that has not detected any failure; then, the generated tabu subdomain is denoted as $D_T(\text{tc}) = \{\text{loc}(x_1), \text{loc}(x_2), \ldots, \text{loc}(x_n)\} = \{d1_{1k}, d2_{2k}, \ldots, dn_{nk}\}$.

Definition 4 (candidate test case generation domain). In an n-dimensional input domain, assume a test case tc = $(x_1, x_2, \ldots, x_n)$ that has not detected any failure. The candidate test case generation domain is denoted as $D_c = D$, for $D_c = \varnothing$; or $D_c = D_c - D_T(\text{tc})$ for all other cases.

3.2. LCS-FSCS Algorithm. Based on the above definitions and our analysis, an independent dimensional division strategy for FSCS can be defined, which is referred to as LCS-FSCS algorithm. The LCS-FSCS algorithm is as follows (Algorithm 1).

At the initial stage (lines 1–4), each dimension of the input domain is divided into p equal parts, and then the ith dimension of the input domain is denoted as $D_i = \{d_{i1}, d_{i2}, \ldots, d_{ip}\}$. We initialize the candidate test case generation domain D_c with a divided dimension space. We generate a test case tc from the domain D_c (lines 5–9) and push tc into TS until the termination condition is satisfied (for example, a failure was detected). Line 6 calls the procedure GenTcByLCSFscs (D_c, TS) to select a "best" test case tc from the D_c domain using FSCS.

In the procedure GenTcByLCSFscs (D_c, TS), if there is no test case in TS (lines 1–3), the first test case is randomly selected from the input domain (equal to D_c domain). Otherwise (lines 4–9), first judge whether D_c is empty or not. We initialize D_c in case D_c is empty (lines 4–6). Subsequently, randomly generate k candidates $c_1, c_2, \ldots, c_k$ from the D_c domain and select an appropriate candidate as the next test case (lines 7–8). On lines 10–11, the tabu subdomain of tc is generated, denoted as $D_T(\text{tc})$. We then recalculate D_c by removing $D_T(\text{tc})$ domain (Definition 4). The last line returns the test case tc (Algorithm 2).

In the LCS-FSCS algorithm, the restricted candidate test case generation domain avoids to choose the next test case within the same subdomain of the previous generated test cases. Therefore, the distribution of test cases is more evenly distributed among independent dimensions.

4. Simulation Experiment

The effectiveness of LCS-FSCS is studied in experiments that focus on the following three problems:

RQ1: are the test cases of LCS-FSCS more evenly distributed than that of FSCS?

RQ2: what is the effect of the p value on the effectiveness of LCS-FSCS?

RQ3: does LCS-FSCS improve the effectiveness of FSCS in different dimensions?

4.1. Experimental Setup and Measures. LCS-FSCS is compared with RT and FSCS to analyze its failure-detection effectiveness. The relevant parameter settings for these experimental studies are discussed in this section. The parameters involved in the simulation experiments are mainly dimension, failure rate, failure pattern, test method, and number of experiments.

(1) Dimension: the dimension of the input domain indicates the number of parameters of SUT. In the

```
(1) If TS = ∅, then
(2)     tc = Random (D_c); //randomly select a test case tc from the D_c domain as the first TC;
(3) else
(4)     If D_c = ∅, then
(5)         Init D_c domain; //D_c = {D_1, D_2, ..., D_n};
(6)     end if
(7)     Randomly generate k candidates c_1, c_2, ..., c_k from the D_c domain; //CS = {c_1, c_2, ..., c_k}
(8)     select the best one as the next test case tc; //Max-Min Euclidean Distance (FSCS)
(9) end if
(10) generated tabu subdomain of tc, noted as D_T(tc); (Definition 3)
(11) recalculate D_c domain by (formula 2);
(12) return tc
```

Algorithm 2: Procedure GenTcByLCSFscs (D_c, TS).

simulation experiment, the 2D-5D input domain is used as the representative for analysis and comparison. Moreover, it is assumed that each dimension is an equidistant continuous space. The coordinate range of each dimension is [1, 1000]. This means the 2D input domain spans a square, the 3D input domain spans a cube, and so on.

(2) Failure rate: the failure rate is obtained as the ratio of failure-causing input domains to all input domains. The failure rate is represented by $\theta \in [0, 1]$. It is assumed that the failure pattern is a block failure pattern [19], and the failure-causing input field is an equidistant continuous space. The location of the block failure domain appears randomly within the input domain. The failure rate range is $\theta \in [0.001, 0.5]$ in this experiment.

(3) Test case generation algorithm: the following four test case generation algorithms are compared:

   (a) RT: test cases were generated randomly with replacement as a benchmark test.

   (b) ART with random partitioning (RP) [16]: ART with random partitioning generates a test case randomly, divides the subdomain according to its coordinates, generates the next test case randomly from the largest subdomain, and divides the subdomain according to its coordinates. Repeat this procedure until the first failure is found. This is a classic partition-based ART algorithm.

   (c) FSCS: FSCS is the prototype of the improved algorithm. The parameter K represents the size of the candidate set. It is shown [14] that for numerical programs, the failure-detection effectiveness of this algorithm does not improve significantly for K larger than 10. Therefore, K is set to 10 in the experiment.

   (d) LCS-FSCS: a new enhanced FSCS strategy is proposed in this paper. The number of divisions p in each dimension is (1, 5, 10, 25, 50, 100, 200, 500, and 1000). When p is 1, LCS-FSCS and FSCS are equivalent.

(4) Number of experiments: the number of experiments is set to 2000. These repetitions of the experiments are needed to effectively avoid the influence of randomness on the experimental results.

(5) Metric of failure-detection effectiveness: the F-measure is used as a measure of effectiveness within this paper. The F-measure is defined as the average number of test cases needed to detect the first failure. The F-count is defined as the number of test cases needed to detect the first failure for each run, $F_{\text{measure}} = \overline{F_{\text{count}}}$. The smaller the value of the F-measure is, the stronger the effectiveness of the algorithm is.

In order to further evaluate the effectiveness improvement of the ART algorithm in comparison with RT, the ratio of the F-measure of ART to the F-measure of RT (theoretical value is $1/\theta$) is denoted as F-ratio and is used to measure the improvement in comparison with RT. If the F-ratio is less than 1, the ART algorithm outperforms RT. This means ART requires fewer test cases to detect the first failure.

According to the configuration of the experimental parameters, the simulation process is as follows: the input domain is generated and the failure domain is calculated according to the failure rate. In each experiment, the failure domain is randomly generated in the input domain. The test case is generated using different test methods. When the test case falls into the failure domain, that is, a failure is detected, the number of test cases executed is captured as the F-count. When the number of experiments reaches 2000, the average of F-count is recorded as F-measure.

4.2. Analysis of the Distribution of Test Cases. Because LCS-FSCS evenly divides the independent dimensions and constrains the test case generation domains, in theory the spatial distribution of test cases is more even. Simulation experiments are conducted to investigate whether the test cases generated by LCS-FSCS are indeed more evenly distributed than the test cases generated using FSCS.

Answer RQ1: distribution of test cases generated by LCS-FSCS.

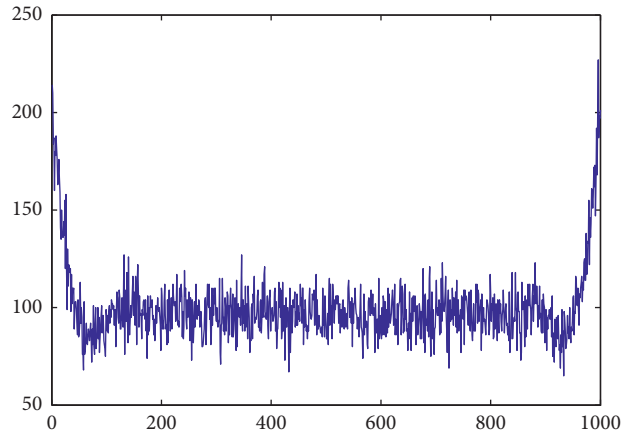

Figure 5: Test case distribution of FSCS.

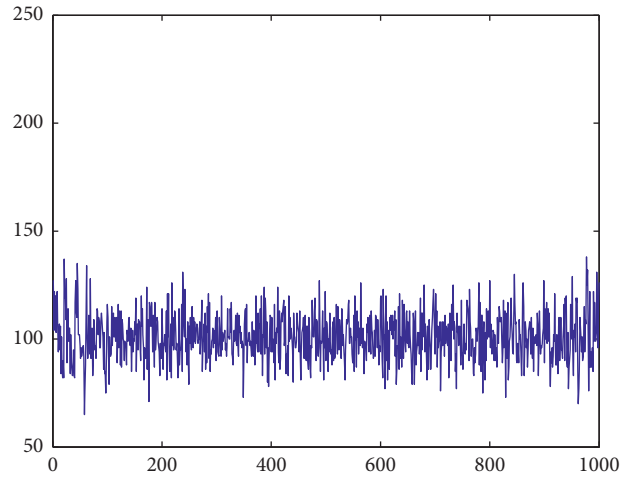

Figure 6: Test case distribution of LCS-FSCS.

In the 2D input domain, the range of each dimension is (1, 1000). 100 test cases are generated using LCS-FSCS and FSCS with no failure domain, and a total of 1000 runs are performed. Because the distribution of test cases on each dimension is similar, the distribution of test cases is compared as shown in Figures 5 and 6.

The distribution of test cases generated by FSCS is shown in Figure 5. There are a large number of test cases near the boundary of the input domain, which demonstrates the so-called boundary effect. Figure 6 shows the distribution of test cases generated by LCS-FSCS. From the perspective of the independent dimensions, the distribution of the test case generated by LCS-FSCS is more even than that generated by FSCS.

4.3. Analysis on Failure-Detection Effectiveness. The experimental results in Section 4.2 show that LCS-FSCS distributes test cases more evenly. However, does this affect the failure-detection effectiveness? To answer this question, we research the failure-detection effectiveness of LCS-FSCS at different failure rates in RQ2. Furthermore, we aim to determine the effectiveness of LCS-FSCS in a multidimensional input domain in RQ3.

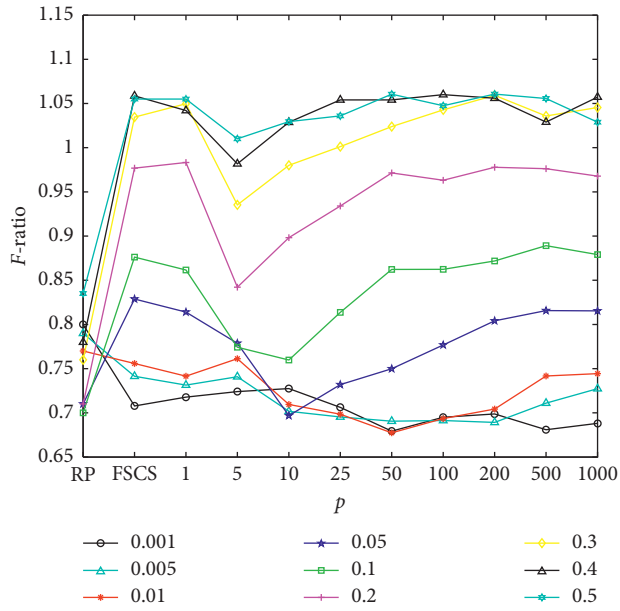

Figure 7: *F*-ratio comparison of different ARP algorithms in 3D.

4.3.1. Failure-Detection Effectiveness at Different Failure Rates. Experiments are conducted to analyze the effect of the p value on the improvement of the failure-detection effectiveness within the same dimensional input domain at different failure rates.

Answer RQ2: the effect of different p values on the failure-detection effectiveness of LCS-FSCS.

We assume the following failure rates: 0.5, 0.4, 0.3, 0.2, 0.1, 0.05, 0.01, 0.005, and 0.001, respectively, whereas the abscissa is P in LCS-FSCS with values of 1, 5, 10, 25, 50, 100, 200, 500, and 1000, respectively, in the 3D input domain. The Y-axis is the F-ratio as shown in Figure 7. The F-ratios of RP and FSCS serve as a benchmark comparison value for the improvement of the effectiveness of LCS-FSCS.

As can be seen in Figure 7, the failure-detection efficiency of FSCS and LCS-FSCS varies greatly depending on the failure rate, while that of the RP algorithm varies little with changes in the failure rate within the 3D input domain.

With increasing failure rates, the failure-detection efficiency of the RP algorithm provides an advantage. For a failure rate of 0.5, the F-ratio of RP is less than 0.85, while the F-ratio of FSCS is larger than 1, indicating that the failure-detection effectiveness of FSCS is inferior to that of RT. For different values of p in LCS-FSCS, the F-ratio of LCS-FSCS fluctuates. The minimum F-ratio of LCS-FSCS is slightly better than that of FSCS. The failure-detection efficiency of RP is poor when the failure rate is low. For example, for a failure rate of 0.005, the F-ratio of RP is close to 0.8, which indicates that the efficiency of failure detection is worse than those of FSCS and LCS-FSCS.

For a failure rate of 0.1 and p value of 5, 10, or 25, the F-ratio of LCS-FSCS is significantly lower than that of FSCS (the lowest F-ratio is reached for $p = 10$). This shows that the LCS-FSCS significantly improves the failure-detection effectiveness of FSCS. With the increase of the p value, the F-ratio of LCS-FSCS gradually resembles that of FSCS. As the

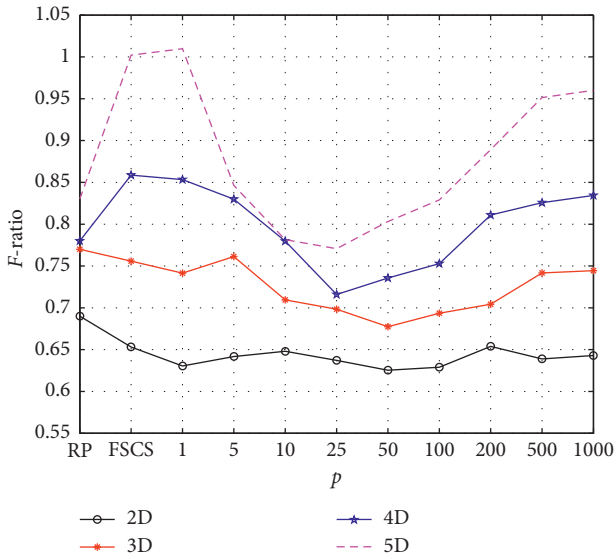

Figure 8: F-ratio comparison in 2-5D with $\theta = 0.01$.

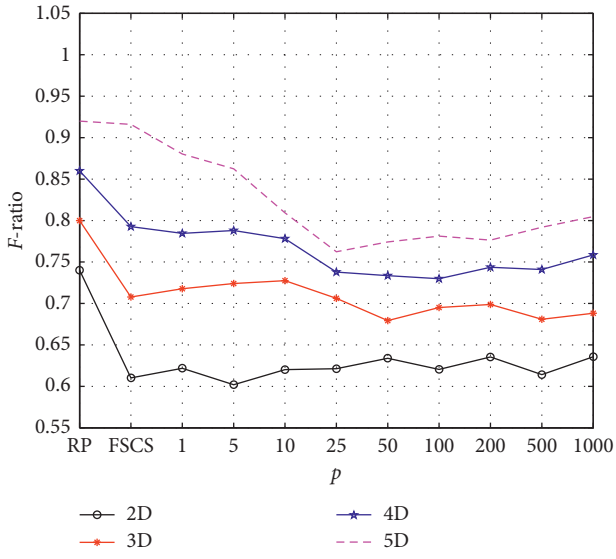

Figure 9: F-ratio comparison in 2-5D with $\theta = 0.001$.

failure rate continues to decrease, the gap between the F-ratio of LCS-FSCS and the F-ratio of FSCS shrinks. For a failure rate of 0.001, the minimum F-ratio of LCS-FSCS improves slightly to resemble that of FSCS.

According to the experimental results, the p value at the minimum F-ratio of LCS-FSCS is related to the failure rate. There are p values that minimize the F-ratio of LCS-FSCS for different failure rates. This proves that LCS-FSCS improves the effectiveness of FSCS. However, if the failure rate is too large or too small, the influence of different p values on the F-ratio of LCS-FSCS is reduced in the 3D input domain.

4.3.2. Failure-Detection Effectiveness in Different Dimensions. We further study the impact of LCS-FSCS on the effectiveness of the failure detection in the multidimensional input domain.

Answer RQ3: failure-detection effectiveness of LCS-FSCS in the high-dimensional space.

We assume that the failure rate is $\theta = \{0.001, 0.01\}$ in the 2D-5D input domain. Figures 8 and 9 show the comparison of the F-ratios. The abscissa is the p value in LCS-FSCS. p is set to 1, 5, 10, 25, 50, 100, 200, 500, and 1000, respectively. The F-ratio of RP and FSCS serves as a benchmark comparison value for the improvement regarding the effectiveness of LCS-FSCS.

It can be seen in Figures 8 and 9 that for a failure rate of 0.01 and an increase in the dimension, the failure-detection effectiveness of the RP algorithm is better than that of the FSCS algorithm, but not as good as that of the optimal LSC-FSCS algorithm. For a failure rate of 0.001, the effectiveness of RP is worse than that of FSCS and LCS-FSCS.

Comparing Figures 8 and 9, the improvement of LCS-FSCS is more significant for low failure rates. In different dimensional spaces, the effect of the test case distribution on the effectiveness of FSCS increases with the increase in the dimensions. This means the boundary effect of FSCS has a greater influence on the effectiveness in these cases. Because the side length of the failure domain increases with the increase of the dimension at the same failure rate, the failure domain is more likely to concentrate on the center of the input domain [12, 21]. This results in a rapid rise of the F-ratio with the increase of the dimension. LCS-FSCS effectively reduces the boundary effect and improves the effectiveness of FSCS in the high-dimensional space.

In general, with an increase of the dimension, the effectiveness of LCS-FSCS gradually improves (Figures 8 and 9). With an increase of the failure rate, the optimal p value becomes smaller (Figure 7). If the failure rate is low, a large p value is required to optimize LCS-FSCS. It is worth noting that for a low failure rate, LCS-FSCS slightly improves the failure-detection effectiveness.

5. Empirical Study

Although the simulations reported in the previous section can provide a comprehensive overview of LCS-FSCS's effectiveness under various conditions (different failure rates θ, dimensions n, etc.), it is still necessary to conduct an empirical study to investigate its failure-detection effectiveness for real-life programs.

5.1. Research Questions. We conduct an empirical study to answer the following research questions:

RQ1: how effective is LCS-FSCS at revealing failures in the real-life program?

RQ2: under what circumstances, is LCS-FSCS less effective than FSCS?

5.2. Object Program. There are three real-life programs in this experiment: the Trityp program and the TCAS program derived from the Software Artifact Infrastructure Repository (SIR) [22] at http://sir.unl.edu and the Integer program is the java.lang.Integer class in JDK.

Trityp is an implementation of the classic triangle classification program. The program has three input integers and determines whether they represent a triangle, and if so, the specific type of triangle. Faults are introduced into the object program based on the mutation analysis technique. A java mutation tool, mujava [23], is used to generate various mutants, each of which is related to a single fault injected into an object program. The total number of mutants generated for the Trityp program is 462. After removing 72 equivalent mutants (those mutants are equivalent to SUT), 390 mutants remain. Suppose that in the integer input domain space of [1, 100], the failure rate of each mutant is calculated by traversing each value of the input domain space. The 165 mutants of the failure rate were selected in [0.001, 0.01].

TCAS is one of the classic "Siemens" programs. TCAS is an aircraft collision avoidance system with 12 input parameters. The range of values is [0, 1000]. Mutants come from SIR. Real faults are introduced in these mutants. The failure rate of those mutants is between [0.00001, 0.04].

The Integer program has two input parameters, and the range of values is [1, 1000]. The Integer program also generates mutants using mujava. A total of 160 mutants are generated, of which 21 equivalent mutants and 16 mutants with a failure rate greater than 0.95 are removed. The failure rate of the remaining 123 mutants is between [0.0009, 0.004].

Details of the three real-life programs are shown in Table 3.

5.2.1. Independent Variables. The independent variables are the test case generation strategy and the implementation of LCS-FSCS. RT and FSCS are selected as baseline techniques for the comparison. RT is a natural baseline and LCS-FSCS is an enhancement to FSCS. Therefore, assessing whether LCS-FSCS is more effective than FSCS is important. In general, an automated oracle is assumed when RT is applied. In our experiments, the size of the candidate set is 10 for FSCS and LCS-FSCS. By results of the simulation experiments, we draw a conclusion that the optimal p value of LCS-FSCS increases with the decrease of the failure rate. Therefore, according to the range of failure rate, p is set to 50 in Trityp and Integer, while p is set to 100 in TCAS.

5.2.2. Dependent Variables. In this experiment, F_{RT} is recorded as the F-measure of RT, F_{FSCS} represents the F-measure of FSCS, and $F_{LCS-FSCS}$ is the F-measure of LCS-FSCS. The F-ratio is usually used as a measure of failure-detection effectiveness. Let $F_{FSCS/RT} = F_{FSCS}/F_{RT}$ and $F_{LCS-FSCS/RT} = F_{LCS-FSCS}/F_{RT}$ indicate the improvement of the effectiveness of FSCS in comparison with RT and the improvement of the effectiveness of LCS-FSCS in comparison with RT, respectively. $F_{FSCS/RT} < 1$ indicates that FSCS is more effective than RT. At the same time, $F_{LCS-FSCS/FSCS} = F_{LCS-FSCS}/F_{FSCS}$ is defined to compare the effectiveness of LCS-FSCS with FSCS. When $F_{LCS-FSCS/FSCS} < 1$, it shows that the effectiveness of LCS-FSCS is higher than that of FSCS.

Table 3: Details regarding the three real-life programs.

Program	Number of input parameters	Range of parameters	Number of mutants	Range of failure rate
Trityp	3	[1, 100]	165	[0.001, 0.01]
Integer	2	[1, 1000]	123	[0.0009, 0.004]
TCAS	12	[0, 1000]	20	[0.00001, 0.04]

5.3. Generation of Test Cases. The experimental process is as follows: test cases are generated using RT, FSCS, and LCS-FSCS. As a next step, the source program and the mutants are executed. The source program execution result is used as a test oracle. If the mutant result is different from the source program result for the same test case, the mutant is killed. For each effective mutant, the number of test cases required to kill the mutant is recorded as F-count, and the average F-count over 2000 experiments is recorded as the F-measure. Let F_{RT}, F_{FSCS}, and $F_{LCS-FSCS}$ be the F-measure of RT, FSCS, and LCS-FSCS, respectively. The $F_{FSCS/RT}$, $F_{LCS-FSCS/RT}$, and $F_{LCS-FSCS/FSCS}$ for each mutant are calculated separately.

5.4. Data and Analysis

5.4.1. Failure-Detection Effectiveness. For 165 mutations of the Trityp program, the statistics of each mutation operator are shown in Table 4.

As shown in Table 4, FSCS and LCS-FSCS are compared with RT, respectively. For FSCS, there are 54 mutants with $F_{FSCS/RT} \geq 1$ and the remaining 111 mutants with $F_{FSCS/RT} < 1$. The failure-detection effectiveness of FSCS with 67.3% mutants is higher than that of RT. At the same time, for LCS-FSCS, there are 16 mutants with $F_{LCS-FSCS/RT} \geq 1$, whereas $F_{LCS-FSCS/RT} < 1$ for the remaining 149 mutants. This shows that the failure-detection effectiveness of LCS-FSCS with 90.3% mutants is better than that of RT. Overall, the advantage of LCS-FSCS over RT is much higher than that of the FSCS over RT.

FSCS is compared to LCS-FSCS. There are 74 mutants with $F_{LCS-FSCS/FSCS} \geq 1$ and the remaining 91 mutants with $F_{LCS-FSCS/FSCS} < 1$. This shows that the failure-detection effectiveness of LCS-FSCS with 55.15% of mutants is better than that of FSCS. Overall, the LCS-FSCS algorithm is superior to the FSCS algorithm.

To further analyze the difference in the failure-detection effectiveness between LCS-FSCS and FSCS when Ratio > 1 (where Ratio is $F_{FSCS/RT}$, $F_{LCS-FSCS/RT}$, and $F_{LCS-FSCS/FSCS}$, respectively), a Conditional Value-at-risk (CVaR) is introduced [24]. CVaR is a risk measurement method that measures the average loss when the loss exceeds VaR. The formula is as follows: $\mathrm{CVaR}(\Pr(r < \mathrm{VaR})) = E(r \mid r \geq \mathrm{VaR})$.

Given the risk threshold VaR, the smaller the CVaR value, the smaller the average loss and the overall risk. In this experiment, VaR is related to Ratio. When Ratio > 1, it is considered that loss occurs, so VaR = 1. When Ratio is $F_{LCS-FSCS}$ as shown in Figure 10, there are 55.15% of mutants with $F_{LCS-FSCS/FSCS} < 1$, so $\mathrm{CVaR}(\Pr(F_{LCS-FSCS/FSCS} < 1)) = \mathrm{CVaR}(0.5515) = E(F_{LCS-FSCS/FSCS} \mid F_{LCS-FSCS/FSCS} \geq 1) = 1.056$. This means there is the probability that

Table 4: Ratio results of experiments with the Trityp program.

Mutation operators	Number of mutants	$F_{FSCS/RT}$		$F_{LCS-FSCS/RT}$		$F_{LCS-FSCS/FSCS}$	
		<1	≥1	<1	≥1	<1	≥1
AOIS	32	21	11	25	7	18	14
AOIU	6	3	3	6	0	5	1
AORB	24	14	10	23	1	19	5
CDL	3	1	2	3	0	2	1
COI	3	3	0	3	0	2	1
COR	3	3	0	3	0	0	3
LOI	21	11	10	16	5	13	8
ODL	15	8	7	14	1	8	7
ROR	40	35	5	38	2	14	26
SDL	12	12	0	12	0	4	8
VDL	6	0	6	6	0	6	0
Total	165	111	54	149	16	91	74
Percentage	100%	67.27%	32.73%	90.30%	9.70%	55.15%	44.85%

$1 - \Pr(F_{LCS-FSCS/FSCS} < 1) = 0.4485$ makes $F_{LCS-FSCS/FSCS} \geq 1$. The average of $F_{LCS-FSCS/FSCS}$ with all $F_{LCS-FSCS/FSCS} \geq 1$ is 1.056. As shown in Figure 10, among the 74 mutants with $F_{LCS-FSCS/FSCS} \geq 1$, there are 46 mutants corresponding to $F_{LCS-FSCS/FSCS} \in [1, 1.05]$. The effectiveness of LCS-FSCS for most of the 74 mutants is not significantly inferior to that of FSCS. However, there are some mutants with $F_{LCS-FSCS/FSCS} > 1.1$. The following section specifically analyzes the failure domain distribution of these mutants. When Ratio is $F_{FSCS/RT}$ or $F_{LCS-FSCS/RT}$, CVaR is shown in Figures 11 and 12.

For 123 mutations of the Integer program, the statistics of each mutation operator mutations are shown in Table 5.

As shown in Table 5, the $F_{FSCS/RT}$ and $F_{LCS-FSCS/RT}$ of all 123 mutants are less than 1, which indicates that both FSCS and LCS-FSCS are more effective than RT in the failure detection. The reason can be seen in Table 3: all mutants of the Integer program range within [0.0009, 0.004]. However, the FSCS algorithm is more effective in the failure detection for low failure rates in the two-dimensional input domain. Compared with the LCS-FSCS algorithm, the failure-detection effectiveness of the LCS-FSCS algorithm is inferior to that of the FSCS algorithm for 11 mutants only. In general, LCS-FSCS performs better than FSCS for 123 mutants of the Integer program.

For 20 mutations of the TCAS program, the statistics of mutations are shown in Table 6.

Table 6 shows that the failure-detection effectiveness of the FSCS algorithm is inferior to that of RT in 40% of the mutants of the 12-dimensional input fields of the TCAS program. The LCS-FSCS algorithm is superior to RT with 80% failure-detection effectiveness. At the same time, among 85% of the mutants, LCS-FSCS algorithm is more effective than the FSCS algorithm in the failure detection.

5.4.2. Analysis of Mutation with Low Failure-Detection Effectiveness. This section researches mutants of LCS-FSCS that are ineffective in the Trityp program.

To study the mutants of $F_{LCS-FSCS/FSCS} > 1.1$ in Figure 10 (that is, mutants whose LCS-FSCS is less effective than FSCS), the distribution of their failure domains is analyzed as shown in Figures 13–17. A 3D view of each of mutant and a projection diagram in each dimension are given, respectively.

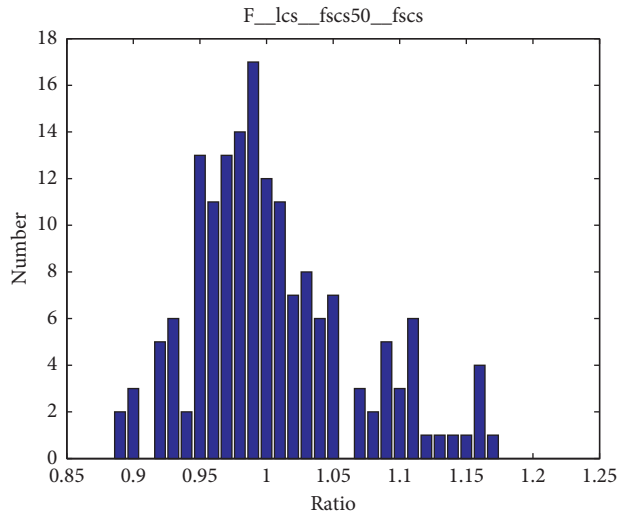

Figure 10: CVaR of $F_{LCS-FSCS/FSCS}$.

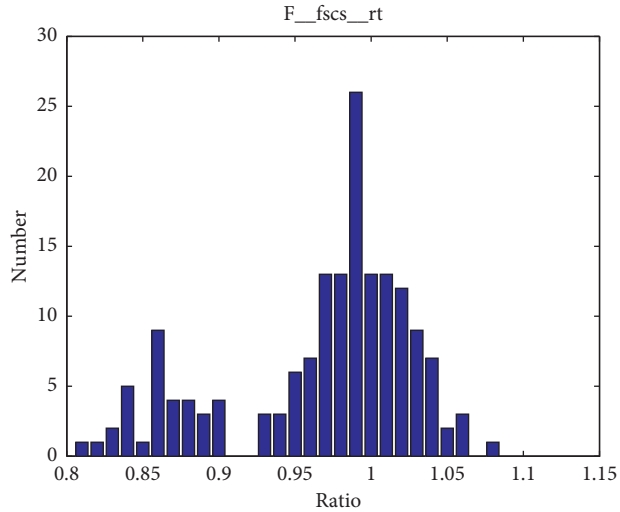

Figure 11: CVaR of $F_{FSCS/RT}$.

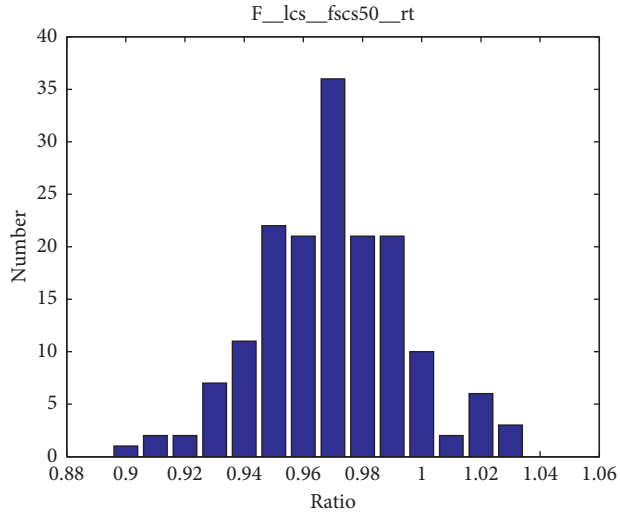

Figure 12: CVaR of $F_{\text{LCS–FSCS/RT}}$.

Table 5: Ratio results of experiments with the Integer program.

Mutation operators	Number of mutants	$F_{\text{FSCS/RT}}$		$F_{\text{LCS–FSCS/RT}}$		$F_{\text{LCS–FSCS/FSCS}}$	
		<1	≥1	<1	≥1	<1	≥1
AODS	1	1	0	1	0	1	0
AODU	3	3	0	3	0	3	0
AOIS	28	28	0	28	0	26	2
AOIU	4	4	0	4	0	4	0
AORB	11	11	0	11	0	11	0
CDL	1	1	0	1	0	1	0
COD	1	1	0	1	0	1	0
COI	5	5	0	5	0	5	0
LOI	16	16	0	16	0	15	1
ODL	13	13	0	13	0	12	1
ROR	25	25	0	25	0	18	7
SDL	10	10	0	10	0	10	0
VDL	5	5	0	5	0	5	0
Total	123	123	0	123	0	112	11
Percentage	100%	100%	0%	100%	0%	91.06%	8.94%

Table 6: Ratio results of experiments with the TCAS program.

Program	Number of mutants	$F_{\text{FSCS/RT}}$		$F_{\text{LCS–FSCS/RT}}$		$F_{\text{LCS–FSCS/FSCS}}$	
		<1	≥1	<1	≥1	<1	≥1
TCAS	20	12	8	16	4	17	3
Percentage	100%	60%	40%	80%	20%	85%	15%

As shown in Figure 13, the failure domain distribution of mutant COR_13 with a failure rate of 0.0075 corresponds to other COR mutants in Table 4. LCS-FSCS of these mutants has lower failure-detection effectiveness than FSCS. The failure regions of these mutants are the same; they consist of three failure domains and three 2-dimensional projection maps. It can be seen in Figures 13–17 that the failure domains of these mutants are concentrated on the boundary. These characteristics meet FSCS's feature to focus on the boundary. Therefore, compared to LCS-FSCS, FSCS with its boundary effect is of advantage when the failure region is concentrated on the boundary. FSCS can use fewer test cases to find the first failure.

Table 7 summarizes the distribution of the five failure domain types in the Trityp program in Figures 13–17. The type of failure domain shape is defined as the five cases in which the LCS-FSCS algorithm is inferior to the FSCS algorithm. The failure rate is corresponding to each type of failure domain. The total number of mutants of the five types is 32. All mutants with $F_{\text{LCS–FSCS/FSCS}} > 1.1$ are covered. In general, FSCS is superior to LCS-FSCS mainly in the failure domain type with obvious boundary effect

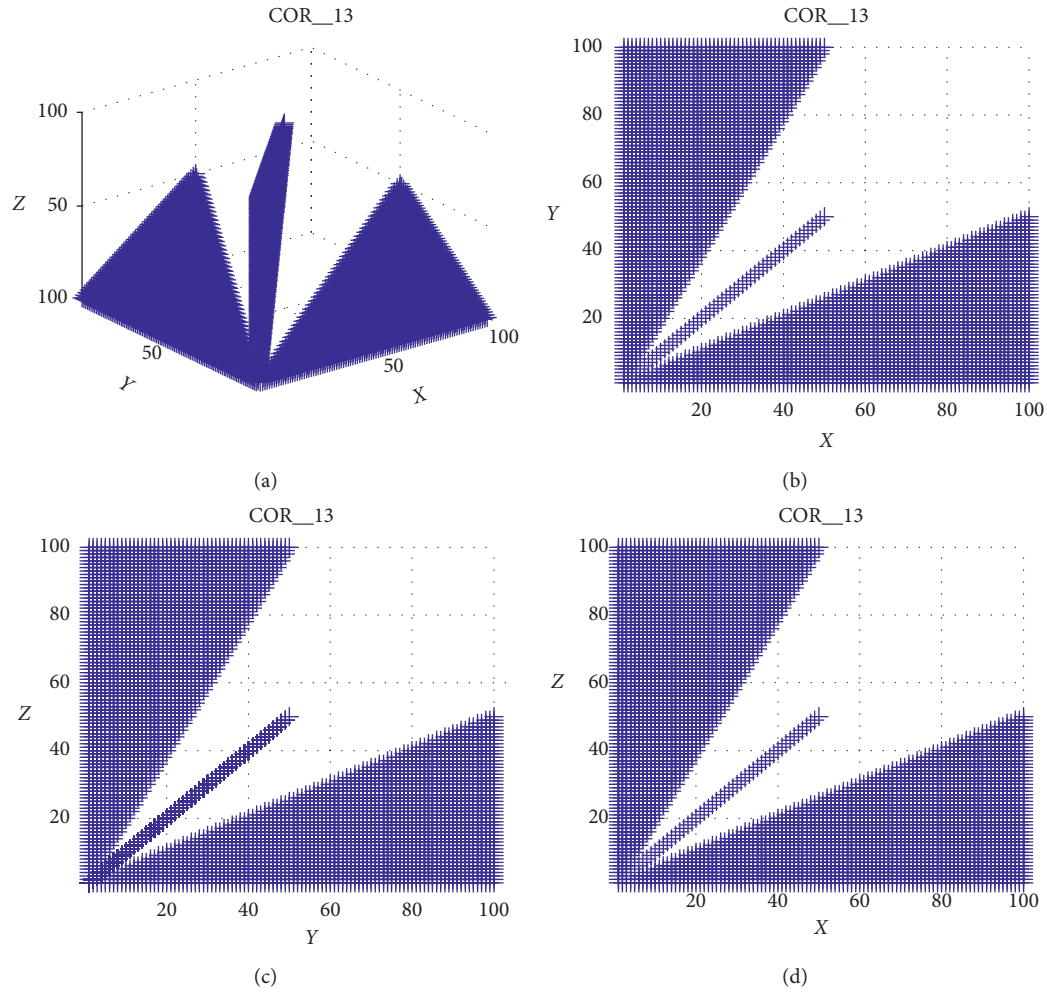

Figure 13: Failure domain distribution of mutant COR_13.

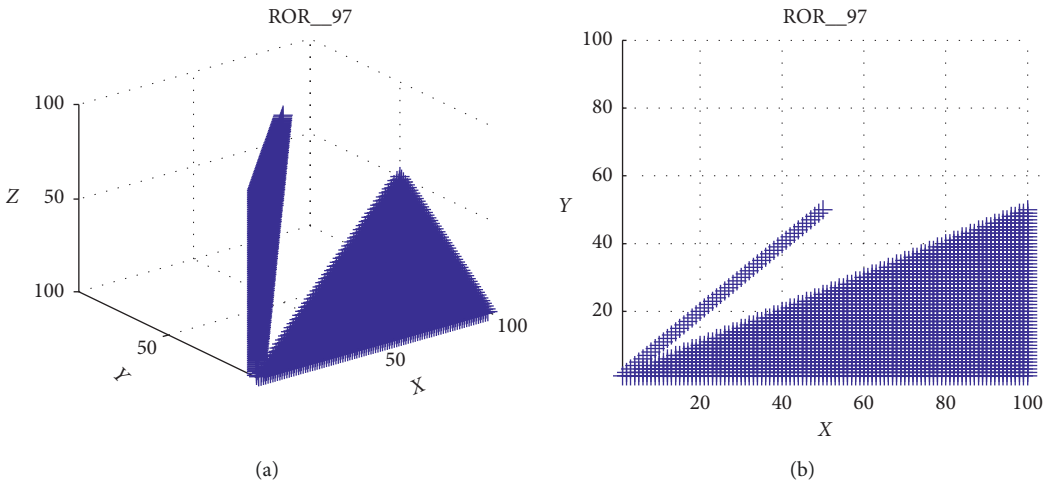

Figure 14: Continued.

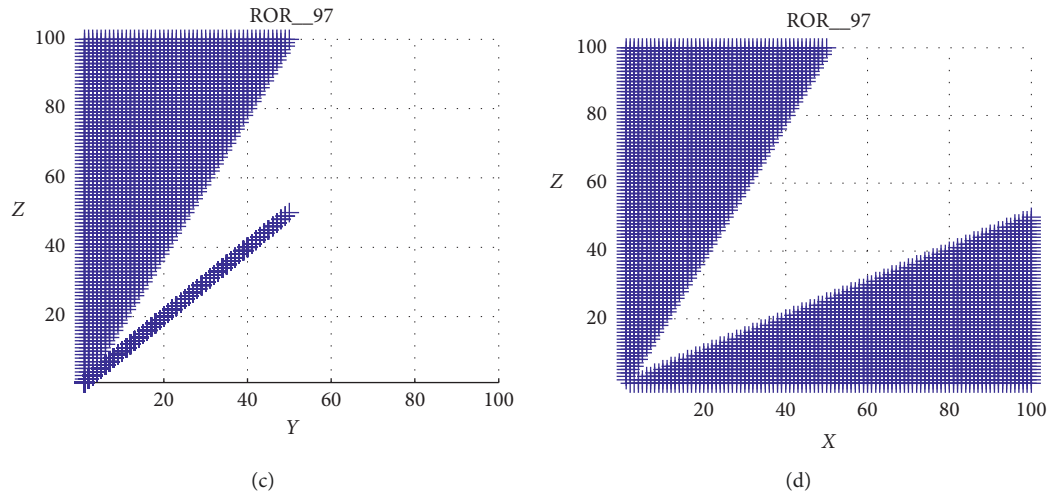

Figure 14: Failure domain distribution of mutant ROR_97.

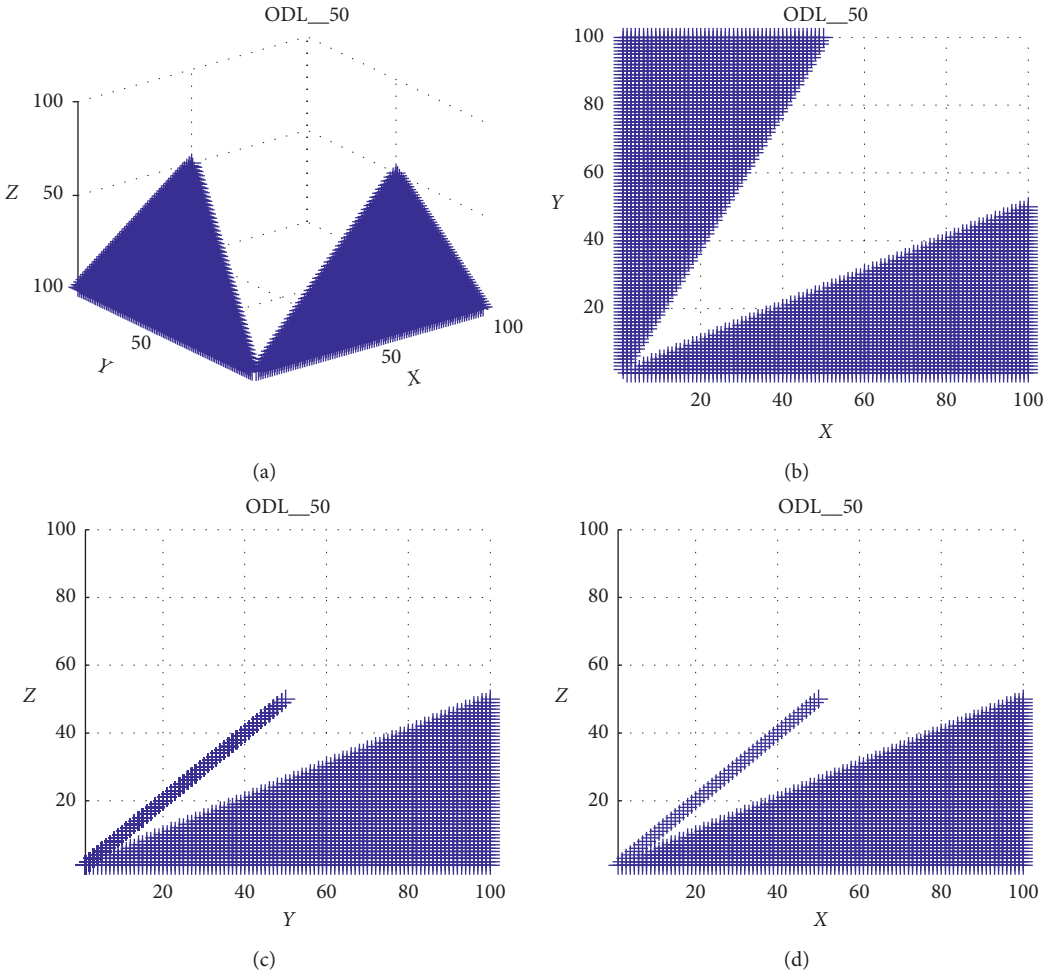

Figure 15: Failure domain distribution of mutant ODL_50.

($F_{\text{LCS–FSCS/FSCS}} > 1.1$). In other cases, LCS-FSCS either has higher failure-detection effectiveness than FSCS ($F_{\text{LCS–FSCS/FSCS}} < 1$), or its failure-detection effectiveness is not much worse than that of FSCS ($1 \le F_{\text{LCS–FSCS/FSCS}} \le 1.1$). Considering all cases where LCS-FSCS is inferior to FSCS, the effectiveness gap between LCS-FSCS and FSCS is small ($\text{CVaR}(\Pr(F_{\text{LCS–FSCS/FSCS}} < 1)) = E(F_{\text{LCS–FSCS/FSCS}} \mid F_{\text{LCS–FSCS/FSCS}} \ge 1) = 1.056$).

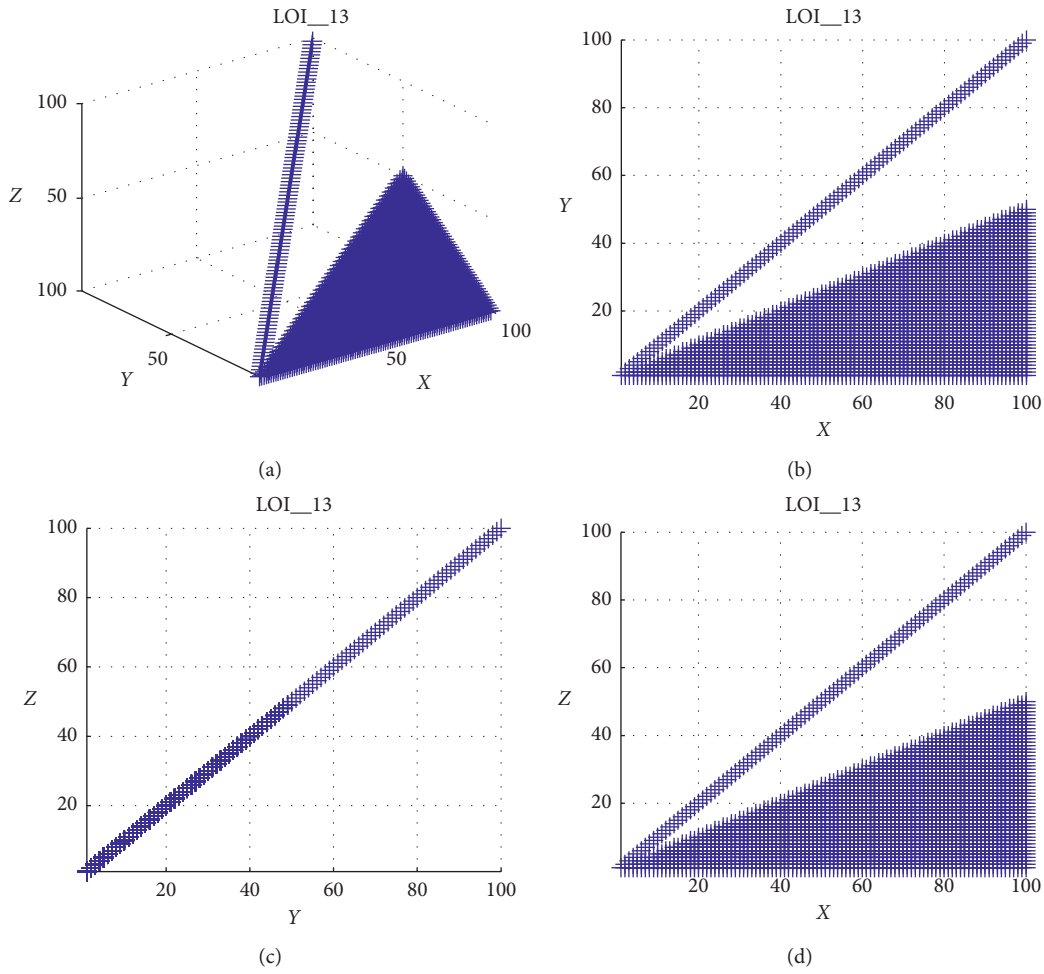

Figure 16: Failure domain distribution of mutant LOI_13.

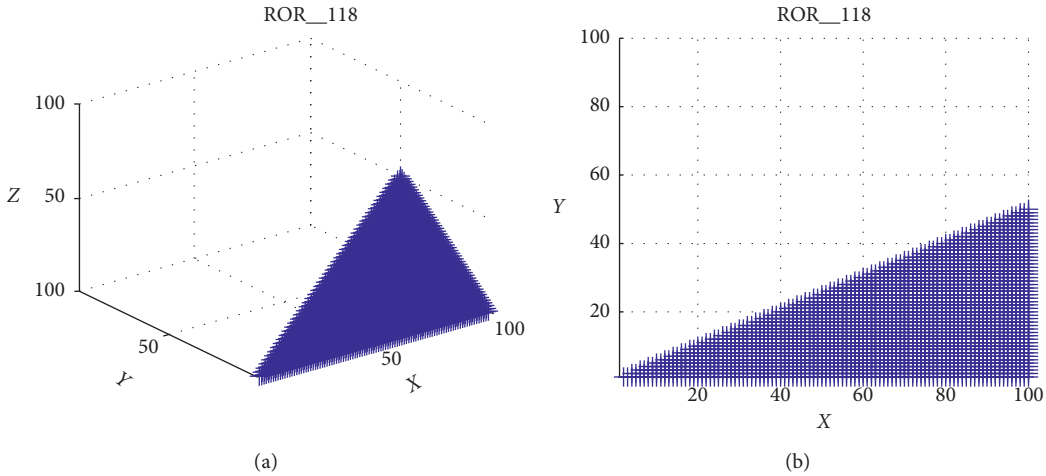

Figure 17: Continued.

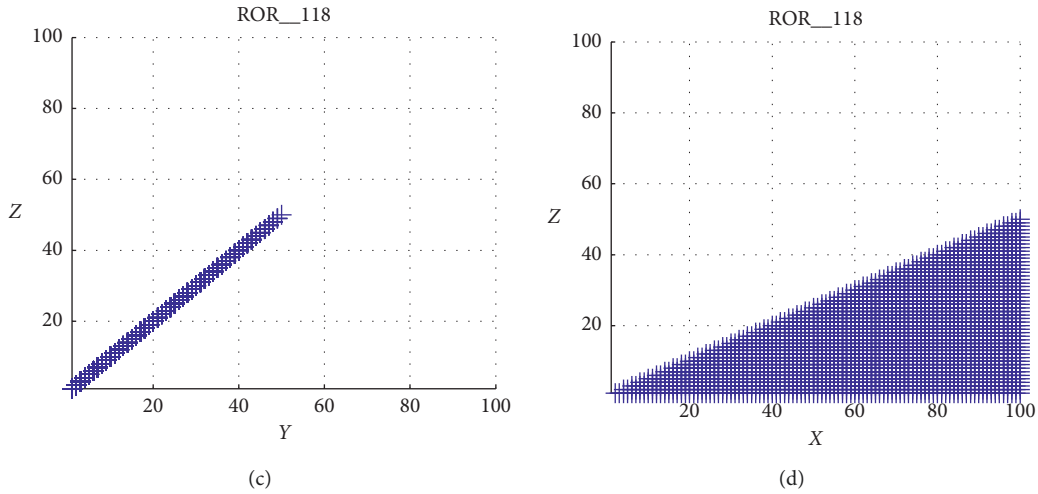

Figure 17: Failure domain distribution of mutant ROR_118.

Table 7: Distribution statistics of five shape failure domains in the Trityp program.

Type of failure domain shape	Failure rate	Number of mutants	$F_{\text{LCS–FSCS/FSCS}}$		
			Avg.	Min.	Max.
A	0.0075	7	1.12	1.07	1.17
B	0.005	5	1.12	1.07	1.16
C	0.005	3	1.1	1.07	1.14
D	0.0026	2	1.05	1.04	1.05
E	0.0025	15	1.09	0.99	1.17

6. Threats to Validity

Some of the potential threats to the effectiveness of this experimental research are as follows.

The threat to the internal effectiveness lies in the conduct of an unbiased experimental design. The experiments include simulations and real-life programs. However, this does not represent the various possible types of faulty program in real life. Further study will mitigate the threat to internal effectiveness.

The threat to construct effectiveness is primarily a measure of the effectiveness of the testing strategy. There are many metrics regarding the failure-detection effectiveness. No single metric can paint a complete picture of the effectiveness of a test technique. The F-measure is commonly used to evaluate the effectiveness of ART testing algorithms. It represents the expectation of the number of test cases needed to detect the first failure. The F-measure is commonly used to compare with other algorithms.

7. Conclusion and Future Work

This paper proposes a novel algorithm for the enhancement of FSCS. By constraining the candidate test case generation domain, the number of test cases on the boundary is reduced, and the boundary effect is effectively alleviated. More importantly, LCS-FSCS reduces the sensitivity of FSCS regarding the dimension and the failure rate.

Future work mainly includes the following: (1) The improvement of the LCS-FSCS algorithm: the effectiveness of the algorithm is related to the p value (the number of dimensions divided), and the p value is related to the failure rate. How to adaptively adjust the p value to achieve higher failure-detection effectiveness is part of our future work and requires further improvements to the algorithm. (2) Extension of failure modes: further analysis of the effect of the algorithmic complexity on more complex failure domains is needed. (3) The application of real-life complex programs: in this paper, we have verified the effectiveness of the LCS-FSCS algorithm in a variety of simulation environments and in three real-life programs. In future work, the scale and number of test sets need to be expanded to further verify the failure-detection effectiveness of the algorithm.

Data Availability

The data used to support the findings of this study are available from the corresponding author upon request.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This work was supported by the National Social Science Foundation of China (15AJG012) and National Science and Technology Major Project of China (2013JH00103).

References

[1] A. Bertolino, "Software testing research: achievements, challenges, dreams," in *Proceedings of the IEEE 2007 Future of Software Engineering*, pp. 85–103, Washington, DC, USA, June 2007.

[2] R. Huang, X. Xie, T. Y. Chen, and Y. Lu, "Adaptive random test case generation for combinatorial testing," in *Proceedings of the 2012 IEEE 36th Annual Computer Software and Applications Conference*, pp. 52–61, Izmir, Turkey, July 2012.

[3] A. Griesmayer, B. Aichernig, and E. B. Johnsen, "Dynamic symbolic execution for testing distributed objects," in *IEEE 2009 Tests and Proofs, Third International Conference; 2–3 July 2009*, pp. 105–120, Springer, Berlin, Germany, 2009.

[4] W. Huayao, C. Nie, P. Justyna, J. Yue, and H. Mark, "An empirical comparison of combinatorial testing, random testing and adaptive random testing," *IEEE Transactions on Software Engineering*, p. 1, 2018.

[5] E. Selay, Z. Q. Zhou, T. Y. Chen, and F.-C. Kuo, "Adaptive random testing in detecting layout faults of web applications," *International Journal of Software Engineering and Knowledge Engineering*, vol. 28, no. 10, pp. 1399–1428, 2018.

[6] X. F. Zhang, Z. Z. Zhang, X. Y. Xie, and Y. C. Zhou, "An approach of iterative partition testing based on priority sampling," *Chinese Journal of Computers*, vol. 39, no. 11, pp. 2307–2323, 2016.

[7] B. Zhou, H. Okamura, and T. Dohi, "Enhancing performance of random testing through Markov chain Monte Carlo methods," *IEEE Transactions on Computers*, vol. 62, no. 1, pp. 186–192, 2013.

[8] M. R. Girgis, "Automatic test data generation for data flow testing using a genetic algorithm," *Journal of Universal Computer Science*, vol. 11, no. 6, pp. 898–915, 2005.

[9] G. J. Myers, *The Art of Software Testing*, Wiley, Hoboken, NJ, USA, 2nd edition, 2004.

[10] H. Xiao, W. Li, T. Zhang, Z. Ma, W. Shao, and C. Hu, "Randomized approach to software model generation," *Chinese Journal of Software*, vol. 28, no. 4, pp. 907–924, 2017.

[11] A. Orso and G. Rothermel, "Software testing: a research travelogue (2000–2014)," in *Proceedings of the ACM 2014 on Future of Software Engineering; ACM*, pp. 117–132, Hyderabad, India, May 2014.

[12] T. Y. Chen, F.-C. Kuo, and Z. Q. Zhou, "On favourable conditions for adaptive random testing," *International Journal of Software Engineering and Knowledge Engineering*, vol. 17, no. 6, pp. 805–825, 2007.

[13] P. E. Ammann and J. C. Knight, "Data diversity: an approach to software fault tolerance," *IEEE Transactions on Computers*, vol. 37, no. 4, pp. 418–425, 1988.

[14] T. Y. Chen, H. Leung, and I. K. Mak, "Adaptive random testing," in *Advances in Computer Science-ASIAN 2004. Higher-Level Decision Making*, pp. 57–76, Springer, Berlin, Germany, 2004.

[15] K. P. Chan, T. Y. Chen, and D. Towey, "Restricted random testing," in *Springer 2002 PInternational Conference on Software Quality*, pp. 321–330, Springer, Berlin, Germany, 2002.

[16] T. Y. Chen, R. Merkel, G. Eddy, and P. K. Wong, "Adaptive random testing through dynamic partitioning," in *Proceedings of the IEEE 2004 International Conference on Quality Software*, pp. 79–86, Braunschweig, Germany, September 2004.

[17] H. Liu and T. Y. Chen, "Randomized quasi-random testing," *IEEE Transactions on Computers*, vol. 65, no. 6, pp. 1896–1909, 2016.

[18] R. Merkel, *Analysis and enhancements of adaptive random testing*, Ph.D. thesis, Swinburne University of Technology, Boroondara, Australia, 2005.

[19] F. T. Chan, T. Y. Chen, I. K. Mak, and Y. T. Yu, "Proportional sampling strategy: guidelines for software testing practitioners," *Information and Software Technology*, vol. 38, no. 12, pp. 775–782, 1996.

[20] T. Y. Chen, F.-C. Kuo, D. Towey, and Z. Q. Zhou, "A revisit of three studies related to random testing," *Science China Information Sciences*, vol. 58, no. 5, pp. 1–9, 2015.

[21] F. C. Kuo, *On adaptive random testing*, Ph.D. thesis, Swinburne University of Technology, Boroondara, Australia, 2006.

[22] H. Do, S. Elbaum, and G. Rothermel, "Supporting controlled experimentation with testing techniques: an infrastructure and its potential impact," *Empirical Software Engineering*, vol. 10, no. 4, pp. 405–435, 2005.

[23] Y. S. Ma, J. Offutt, and Y. R. Kwon, "MuJava: a mutation system for java," in *Proceedings of the ACM 2006 International Conference on Software Engineering; ACM*, pp. 827–830, Shanghai, China, May 2006.

[24] R. T. Rockafellar and S. Uryasev, "Conditional value-at-risk for general loss distributions," *Journal of Banking & Finance*, vol. 26, no. 7, pp. 1443–1471, 2002.