An Adaptive Partition-Based Approach for Adaptive Random Testing on Real Programs

Yisheng Xia[a], Weifeng Sun[a], Meng Yan[a,*], Lei Xu[b] and Dan Yang[a]
[a]School of Big Data & Software Engineering, Chongqing University, Chongqing, China
[b]Qingdao Haier Smart Technology R & D Co., Ltd, Qingdao, China
Email: {yishengx, weifeng.sun, mengy, dyang}@cqu.edu.cn, xulei1@haier.com

***Abstract*—Adaptive random testing (ART) is a family of algorithms to enhance random testing (RT) by generating test cases extensively and evenly. For this purpose, many ART algorithms have been proposed, the most well-known and the first approach is the Fixed-Size-Candidate-Set ART (FSCS-ART). In recent years, researchers have also proposed many ART methods to continuously improve the performance of FSCS-ART, but the focus has been more on reducing the time overhead of FSCS-ART while retaining its failure detection effectiveness as much as possible due to the boundary effect. To alleviate the boundary effect and improve the effectiveness of FSCS-ART, this paper proposes an algorithm AP-FSCS-ART, an Adaptive Partition-based method on top of FSCS-ART. First, AP-FSCS-ART divides the entire input domain into external and internal sub-domains. Then, two different algorithms are adaptively applied to the two sub-domains to find the next test case from the randomly generated candidate test cases. During the selecting process, AP-FSCS-ART takes into account not only the most recently executed test case of a candidate test case but also its position relative to the input domain. Experiments using the 12 most common real programs and comparisons with other algorithms in this paper show that the AP-FSCS-ART algorithm has significantly better failure detection capability, with improvements from 8.8% to 11.4% compared to three state-of-the-art ART algorithms, including the FSCS-ART, FSCS-ctsr, and NNDC-ART.**

***Index Terms*—Adaptive random testing, Random testing, Software testing.**

I. Introduction

Software testing is a necessary process in the software lifecycle and one of the fundamental way to guarantee the quality of software products. Random testing (RT) is the common method of software testing and one of the baselines for the vast majority of software testing methods. The main process of RT is to randomly select a test case from the input domain, then execute it on the (software) system under test (SUT), and finally compare the expected result with the actual output result. The advantages of RT are its conceptual simplicity and ease of implementation, but it has attracted a great deal of controversy. Even Myers[14] comments on RT as the poorest test case design methodology, due to the fact that researchers found it redundant, ineffective and unable to use information from the SUT for targeted testing.

Adaptive Random Testing (ART)[9] is proposed to improve the effectiveness of RT by using more information from SUT. The main difference between ART and RT is that ART uses the distance (based on information from the SUT) to generate or select the next test case, while RT selects the next test case at random. Previous research[8][17] had found that inputs that cause errors usually tend to cluster together to form continuous failure regions, likewise inputs that do not cause errors. This seems to that the more evenly the test cases are spread throughout the input domain, the better its failure detection capability. Therefore, the goal of ART is to achieve a uniform distribution of test cases in the input domain if the previous inputs did not cause an error. There are many ART-based methods, but the first and well known implementation of ART is the Fixed-Size-Candidate-Set ART (FSCS-ART)[5]. First, it randomly selects a fixed number (denoted k) of candidate test cases from the input domain. Then, FSCS-ART selects an element from the candidates as the next test case, so that it is the furthest away from the previously executed test case. However, FSCS-ART has a potential boundary effect[7][10]. The boundary effect is the bias of the FSCS-ART algorithm to select test cases close to the boundary, which deviates from the initial purpose of even distribution.

As seen in recent years of ART-related papers[11][12], current research in this area has focused more on reducing ART computational overheads, but less on improving ART effectiveness. To improve the effectiveness of ART, we propose an adaptive partition-based ART method, namely AP-FSCS-ART. AP-FSCS-ART divides the entire input domain into two sub-domains (external and internal sub-domains) and generates separate test cases to alleviate the boundary effect of FSCS-ART. Taking this a step further, we designed two algorithms for different sub-domains, one selecting test cases in the internal sub-domain and the other towards selecting those in the external sub-domain. In addition, we have introduced additional information on the relative position of the test cases during the test case selection process.

To evaluate the effectiveness of AP-FSCS-ART, we conducted experiments on 12 widely-used real programs, while we compared the proposed algorithm with FSCS-ART, FSCS-ctsr[11], and NNDC-ART[12] and performed statistical analysis of the experimental results. In summary, the main contributions of this paper are as follows.

1) We propose a novel adaptive partition-based ART method on top of FSCS-ART, namely AP-FSCS-ART. The key novelty is to adaptively utilize different test case selecting algorithms in different sub-domains. Such an adaptive mechanism can effectively alleviate the boundary effect of prior ART methods.

*Corresponding Author

2) We conduct experiments using the 12 most common real numerical programs for AP-FSCS-ART, FSCS-ART, FSCS-ctsr, and NNDC-ART, and perform statistical analyses of the results. Compared to the other algorithms, AP-FSCS-ART has better failure detection capability. In addition, we have made our code and related experimental results data publicly available.[1]

II. Approach

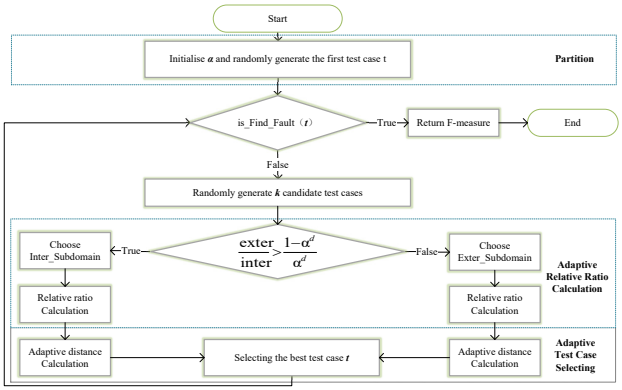

Fig. 1. AP-FSCS-ART framework.

A. Framework

Fig. 1 shows the framework of the AP-FSCS-ART. The first step is the initialization, α is the partition parameter (see section Partition for details). The first test case t is generated randomly from the input domain and executed in the SUT.

Next, if *t* finds a failure, F-measure is returned, otherwise continue to generate *k* (chen[9] et al. consider *k* = 10 to be optimal) random test cases from the input domain and add them all to candidate set (C). F-measure in Fig. 1 indicates the number of test cases that had been executed by the time the first failure was found.

Then we compare the magnitude of the two values $\frac{exter}{inter}$ and $\frac{1-\alpha^d}{\alpha^d}$ to determine which algorithm to calculate the adaptive relative ratio. In this component, we calculate the adaptive relative ratio to obtain information about the relative position of the test case points in the input domain.

Finally, we combine the nearest neighbor values[2] of the test cases with the adaptive relative ratio to calculate the adaptive distance and select the best test cases based on the maximum adaptive distance. The best test case is assigned to t and the above operation is repeated. The specific steps can be found in the pseudo-code for AP-FSCS-ART in Algorithm 1. The relevant terms mentioned above are described in detail in Sections Partition, Adaptive Relative Ratio Calculation, and Adaptive Test Case Selecting.

B. Partition

As already mentioned, FSCS-ART suffers from boundary effects, and we want to alleviate this problem by partitioning and using some adaptive components. In this article, AP-FSCS-ART divides the entire input domain into external and internal sub-domains with a fixed scale α. Given a d-dimensional input domain, and the α^d denotes the proportion of internal sub-domains in the overall input domain, and then 1 - α^d indicates the proportion of external sub-domains. Thus, $\frac{1-\alpha^d}{\alpha^d}$ represents the ratio of the size of the external sub-domain to the size of the internal sub-domain.

Algorithm 1: AP-FSCS-ART

```
   Input: input domain(𝔻)
   Output: count
1  Selected set(E), Candidate set(C) ← {};
2  count, exter, inter ← 1;
3  α ← 0.7;
4  Randomly generate the first test case t from 𝔻;
5  while ! is_Find_Fault( t ) do
6      Insert t into E as an executed test case;
7      Randomly generate k (k = 10) test cases from 𝔻
         and assign them to C;
8      Calculate the dimensionality of t and assign it to d;
9      adp_dist_(C,E) ← 0;
10     if exter/inter > (1−α^d)/α^d then
11         inter++;
12         for c_i ∈ C do
13             Calculating the rel_ratio_ci;
14             Calculating the mindist_(c_i,E) value by
                 Euclidean Distance;
15             Calculating the adp_dist_(c_i,E);
16             if adp_dist_(c_i,E) > adp_dist_(C,E) then
17                 adp_dist_(C,E) ← adp_dist_(c_i,E);
18                 t ← c_i;
19     else
20         exter++;
21         for c_i ∈ C do
22             Calculating the rel_ratio_ci;
23             Calculating the mindist_(c_i,E) value by
                 Manhattan Distance;
24             Calculating the adp_dist_(c_i,E);
25             if adp_dist_(c_i,E) > adp_dist_(C,E) then
26                 adp_dist_(C,E) ← adp_dist_(c_i,E);
27                 t ← c_i;
28     count++;
29     C ← {};
30 return count;
```

C. Adaptive Relative Ratio Calculation

In this component, we mainly calculate the adaptive relative ratio (*rel_ratio*) of test case points. The *rel_ratio* represents the relative position of the test case point from the boundary of the input domain, i.e. when a candidate test case point c_i is d-dimensional, in each dimension c_i has a coordinate value c_{i_j} (j ∈ d). Similarly, the input domain is also d-dimensional, and each of its dimensions is bounded by the maximum $c_{i_{j-max}}$ and minimum $c_{i_{j-min}}$ values. Each dimension has an adaptive relative ratio ($rel\_ratio_{c_{i_j}}$), which is shown in equation 2. The minimum of the $rel\_ratio_{c_{i_j}}$ of all dimensions is the *rel_ratio* of c_i, followed by $rel\_ratio_{c_i}$, which is shown in equation 1. Theoretically, the closer the test case is to the boundary of the input domain, the smaller the *rel_ratio* (in the range of [0, 1]).

$$rel\_ratio_{c_i} \leq \min_{j=1}^{|c_i|}\left(rel\_ratio_{c_{i_j}}\right), \quad (c_i \in C) \tag{1}$$

[1]The source code of the AP-FSCS-ART and the relevant experimental data are available at: https://github.com/No-oneS/AP-FSCS-ART

[2]as shown in Algorithm 1, lines 14 and 23, and see [9] for details.

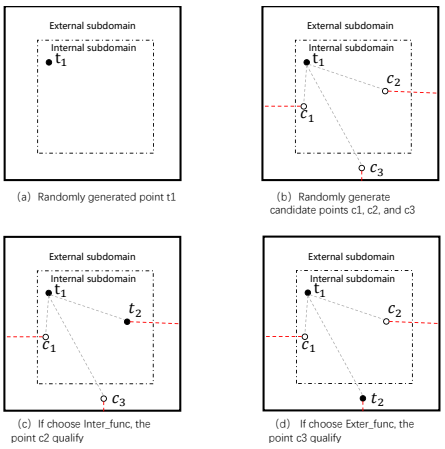

Fig. 2. Example of *Exter_func* and *Inter_func*.

$$rel\_ratio_{c_{i_j}} = 2 * \frac{\min\left(\left|c_{i_j} - c_{i_{j-min}}\right|, \left|c_{i_{j-max}} - c_{i_j}\right|\right)}{\left|c_{i_{j-max}} - c_{i_{j-min}}\right|} \tag{2}$$

D. Adaptive Test Case Selecting

In this component, we will use two algorithms, based on different ideas combining nearest neighbor and adaptive relative ratio, which are used to select the best target candidate test cases.

The *exter* and *inter* are used to count the number of calls to the two algorithms (*Exter_func* and *Inter_func*). *Inter_func* (as shown in Algorithm 1, lines 11-18) is more biased towards selecting test cases close to the center of the input domain, while *Exter_func* (as shown in Algorithm 1, lines 20-27) is biased towards the boundary of the input domain (as shown in Fig. 2). We want to constrain the number of calls to the two algorithms by the inequality ($\frac{exter}{inter} > \frac{1-\alpha^d}{\alpha^d}$) to achieve an even distribution of test cases. If the result of the inequality is true, we will choose the *Inter_func*, otherwise, we will choose the *Exter_func*.

For *Inter_func*, the nearest neighbor value ($mindist_{(c_i,E)}$) of the candidate test case c_i to E is calculated using the Euclidean distance. Next, the *adaptive distance* (*adp_dist*) is calculated by combining the adaptive relative ratios with the following formula:

$$adp\_dist_{c_i} = mindist_{(c_i,E)} * (1 + rel\_ratio_{ci}) \tag{3}$$

However, for algorithm *Exter_func*, the nearest neighbor value is calculated using the Manhattan distance. And its adaptive distance is calculated as follows:

$$adp\_dist_{c_i} = mindist_{(c_i,E)} * (1 - rel\_ratio_{ci}) \tag{4}$$

Similar to FSCS-ART, which uses the maximum nearest neighbor value to select the best test case, AP-FSCS-ART is conditional on the maximum *adaptive distance*.

E. Complexity Analysis

As shown in Algorithm 1, the space complexity of storing the set of executed test cases is O(N), and the space complexity of storing the set of *k* candidate test cases is O(K).In addition, we need O(1) space complexity to store multiple parameters, so the final space complexity of the algorithm AP-FSCS-ART is O(N).

Compared to FSCS-ART ($O(dKN^2)$ see [11]), the time complexity of AP-FSCS-ART additionally increases the computation of the relative ratio. The time complexity of calculating the relative ratio $_{c_i}$ is related to $|C|$, $|E|$, and $|c_i|$, so for the candidate sets C of k (= $|C|$) and dimension d (=$|c_i|$), and the executed test case sets E of N (=$|E|$), the time complexity of AP-FSCS-ART is $O(dKN^2 + dKN)$. The time complexity of AP-FSCS-ART is similar to that of FSCS-ART .

III. EXPERIMENTAL SETUP

In this section, we introduce the research questions, datasets and evaluation metrics, and the statistical analysis.

A. Research Questions

We used 12 real programs to verify the validity of AP-FSCS-ART, which have been widely adopted in research work related to ART[3][4][6][8][9]. We used three state-of-the-art algorithms as our baselines, including FSCS-ART, FSCS-ctsr, and FSCS-NNDC-ART. FSCS-ART is the most classical ART algorithm and the proposed algorithm is based on it, therefore FSCS-ART should be used as our comparison algorithm. FSCS-ctsr and FSCS-NNDC-ART have been shown to alleviate the boundary effects of FSCS-ART to varying degrees. We have chosen these two algorithms as our comparison algorithms. All experiments followed the setup of previous studies except for the parameters α. The range of the size of the α, was set to vary from 0.1 to 1 with an increment of 0.1. We observed that the size of the value of α affects the number of calls to the two algorithms and thus the distribution of test cases. After extensive experimental proof, we found that α = 0.7 works best. Therefore, in this thesis, all experiments using α have a value of 0.7. The empirical study can help answer the following research questions:

1) How does AP-FSCS-ART perform on 12 real programs compared to other ART algorithms?
2) How do the different components affect the effectiveness of the AP-FSCS-ART?

B. Datasets and Evaluation Metrics

We conducted an empirical study using 12 real C++ programs from published ACM programs[1] and Numerical Recipes[15], which are the most widely used in ART-related research, the details of which are given in Table I. These real programs are generated primarily from six mutant seeds, including arithmetic operator replacement (AOR), constant replacement (CR), relational operator replacement (ROR), return statement replacement (RSR), statement deletion (SD), and scalar variable replacement (SVR). We run the test cases in the original program and in the variant program and collect the results. By comparing the results of their executions, we can determine whether the generated test cases detect failures.

Following previous research, we use F_{RT} to refer to the F-measure of RT and F_{ART} to refer to the F-measure of ART, and then we use F-ratio as a validity criterion, which refers the

TABLE I
12 REAL PROGRAMS.

Program	Input Domain			Fault Types	Failure Rate
	Dimension	Input Types	(From, To)		
Airy	1D	Double	(-5000,5000)	CR	0.000716
Bessj0	1D	Double	(-300000,300000)	AOR, ROR, SVR, CR	0.001373
Erfcc	1D	Double	(-30000,30000)	AOR, ROR, SVR, CR	0.000574
Probks	1D	Double	(-50000,50000)	AOR, ROR, SVR, CR	0.000387
Tanh	1D	Double	(-500,500)	AOR, ROR, SVR, CR	0.001817
Bessj	2D	Double	(2,300),(-1000,15000)	AOR, ROR, CR	0.001298
Gammq	2D	Double	(0,1700),(0,40)	ROR, CR	0.000830
Sncndn	2D	Double	(-5000,5000),(-5000,5000)	SVR, CR	0.001623
Golden	3D	Double	(-100,60),(-100,60),(-100,60)	ROR, SVR, CR	0.000550
Plgndr	3D	Double	(10,500),(0,11),(0,1)	AOR, ROR, CR	0.000368
Cel	4D	Double	(0.001,1),(0.001,300), (0.001,10000),(0.001,1000)	AOR, ROR, CR	0.000332
El2	4D	Double	(0, 250), (0, 250), (0, 250), (0, 250)	AOR, ROR, SVR, CR	0.000690

ratio of F_{ART} to F_{RT}. F-ratio is used to measure the F-measure improvement of ART compared to RT, and the smaller the value of F-ratio, the greater the improvement in F-measure.

C. Statistical Analysis

We used the P-values[2] and the Effect Sizes[16] for the statistical analysis. We wanted the P-value from the Wilcoxon rank sum test to determine whether the two algorithms are significantly different; when P-value is less than 0.05, it means that the AP-FSCS-ART has significantly different from the target algorithm. We used non-parametric Vargha and Delaney Effect Size to determine the likelihood that the AP-FSCS-ART has an advantage in failure detection capability. The Effect Size is greater than 0.5, implying that AP-FSCS-ART is likely to be better than the baseline method.

We determine the likelihood that the algorithm AP-FSCS-ART is superior by Effect Size, and we use the non-parametric Vargha and Delaney Effect Size without considering preconditions such as the presence of a normal distribution in the experimental results data. If the Effect size is greater than 0.5, it means that AP-FSCS-ART is very likely to be better than the target algorithm.

To ensure the reliability of the experimental results, we ran each experiment 5000 times and took the average value as the final result.

IV. RESULTS

A. Answer to RQ1

From Table II we can obtain the following observations:

1) From the F-ratio results: AP-FSCS-ART outperforms both FSCS-ART and FSCS-ctsr, regardless of real programs. Additionally, AP-FSCS-ART outperforms NNDC-ART in most cases, except the Golden and Plgndr programs. We have accumulated the F-ratio values of the comparison methods for all 12 real programs and the results show that the AP-FSCS-ART algorithm can deliver 8.8% - 11.4% performance improvement.
2) From the P-values results: Compared to FSCS-ART, AP-FSCS-ART had P-values less than 0.05 in the majority of cases except for the Bessj and Golden programs. As compared to FSCS-ctsr, AP-FSCS-ART has P-values less than 0.05 in almost all cases except for the Golden program. And Compared to NNDC-ART, AP-FSCS-ART had P-values less than 0.05 in most cases except the Sncndn and Golden programs. From these P-values results, it can be seen that in most cases, the proposed algorithm is significantly different from baselines; but the proposed algorithm does not seem to be very different from these comparison algorithms in the Golden program.
3) From the Effect Size results: Compared to FSCS-ART and FSCS-ctsr, AP-FSCS-ART had effect size values greater than 0.5 for all real programs, indicating that AP-FSCS-ART failure detection is better than the FSCS-ART and FSCS-ctsr. However, compared to NNDC-ART, AP-FSCS-ART had effect size values greater than 0.5 in most cases and even effect sizes up to 0.7096 in the Cel program, except for the Golden and Plgndr programs. It can therefore be concluded that AP-FSCS-ART is better than the comparison algorithm in almost all cases, except possibly worse than NNDC-ART on Golden and Plgndr programs.

In summary, from the experimental results of these 12 real programs, AP-FSCS-ART performs significantly better than FSCS-ART, FSCS-ctsr, and NNDC-ART.

B. Answer to RQ2

We conducted ablation experiments to explore the contribution of each component of the AP-FSCS-ART. As the AP-FSCS-ART contains two adaptive components, adaptive relative ratio calculation, and adaptive test case selecting. We sequentially removed the formation variants and obtained the corresponding F-ratio results. Table III provides the results of the ablation experiments, from which we can make the following observations:

1) For the adaptive relative ratio: Compared to AP-FSCS-ART-a, AP-FSCS-ART had lower F-ratio values on 11 real programs, except for the Bessj program. However, for the program Bessj, the difference between the two algorithms is small, suggesting that the adaptive relative ratio has a significant improvement for AP-FSCS-ART.
2) For the two different algorithms (adaptive test case selecting): Compared to AP-FSCS-ART-b, AP-FSCS-ART achieved significantly worse than AP-FSCS-ART-b for the programs Airy, Bessj0, Erfcc, Probks, and Tanh. However, significantly better F-ratio results were obtained on the other seven programs. Interestingly, all five programs are 1D and the failure pattern (the shape of failure-causing inputs) is the block (from the work of Kuo et al.[13] and Huang et al.[12]), making it clear that AP-FSCS-ART-b easier to select test cases in the center of the input domain. Then compared to AP-FSCS-ART-c, AP-FSCS-ART is significantly worse on the five real programs Bessj, Gammq, Plgndr, Cel, and El2, but the other seven are significantly better. And for the 5 real programs Airy, Bessj0, Erfcc, Probks, and Tanh, it is difficult to find failures, whose F-ratio is denoted by

TABLE II
F-RATIO OF AP-FSCS-ART AND OTHER ALGORITHMS ON 12 REAL PROGRAMS.

Program	AP-FSCS-ART	FSCS-ART			FSCS-ctsr			NNDC-ART		
	F-ratio	F-ratio	P-value	Effect Size	F-ratio	P-value	Effect Size	F-ratio	P-value	Effect Size
Airy	**0.5267**	0.5785	0.0000	0.5367	0.5779	0.0000	0.5302	0.5531	0.0019	0.5179
Bessj0	**0.5603**	0.6137	0.0000	0.5380	0.6158	0.0000	0.5364	0.6057	0.0031	0.5170
Erfcc	**0.5401**	0.5926	0.0000	0.5381	0.6071	0.0000	0.5441	0.5509	0.0268	0.5128
Probks	**0.5212**	0.5626	0.0000	0.5316	0.5735	0.0000	0.5345	0.5681	0.0025	0.5174
Tanh	**0.5134**	0.5619	0.0000	0.5365	0.5673	0.0000	0.5346	0.5511	0.0271	0.5128
Bessj	**0.5765**	0.5851	0.7903	0.5015	0.6117	0.0042	0.5165	0.7779	0.0000	0.5658
Gammq	**0.7748**	0.8765	0.0000	0.5288	0.9083	0.0000	0.5361	0.9113	0.0000	0.5290
Sncndn	**0.9957**	1.0405	0.0121	0.5145	1.0270	0.0203	0.5134	1.0074	0.3903	0.5050
Golden	1.0073	1.0184	0.6208	0.5029	1.0297	0.2613	0.5065	**0.9897**	0.3648	0.4948
Plgndr	0.5202	0.5818	0.0000	0.5277	0.6011	0.0000	0.5364	**0.3601**	0.0000	0.3889
Cel	**0.3981**	0.5290	0.0000	0.5777	0.5670	0.0000	0.5858	0.9453	0.0000	0.7096
El2	**0.3962**	0.4974	0.0000	0.5549	0.5598	0.0000	0.5856	0.4526	0.0000	0.5283
SUM	**7.3310(x)**	8.0383 (y)	**Improvement: 8.8%**		8.2469 (y)	**Improvement: 11.1%**		8.2736 (y)	**Improvement: 11.4%**	

(**Bolded red** indicates the best result for that row, and the improvement as **1 - x/y**)

TABLE III
F-RATIO OF AP-FSCS-ART ON 12 REAL PROGRAMS FOR RQ2.

Program	AP-FSCS-ART F-ratio	AP-FSCS-ART-a F-ratio	AP-FSCS-ART-b F-ratio	AP-FSCS-ART-c F-ratio
Airy	0.5267	0.5662	0.4597	——
Bessj0	0.5603	0.6215	0.4889	——
Erfcc	0.5401	0.6047	0.4725	——
Probks	0.5212	0.5610	0.4474	——
Tanh	0.5134	0.5631	0.4523	——
Bessj	0.5765	0.5706	0.7686	0.4142
Gammq	0.7748	0.9007	0.5662	0.5519
Sncndn	0.9957	1.0455	1.0038	1.0196
Golden	1.0073	1.0082	0.8862	1.2290
Plgndr	0.5202	0.6212	0.8469	0.4554
Cel	0.3981	0.5242	0.5670	0.3366
El2	0.3962	0.4986	0.5598	0.3474
SUM	**7.3310** (x)	8.0861 (y) **Improvement: 9.3%**	8.9902 (y) **Improvement: 18.4%**	—— (y) **Improvement: —— %**

AP-FSCS-ART-a: without adaptive relative ratio; **AP-FSCS-ART-b**: just use Inter_func; **AP-FSCS-ART-c**: just use Exter_func

'—'. This is because AP-FSCS-ART-c always selects test cases close to the boundary of the input domain, and the failures of these programs are located in the center of the input domain. After the final accumulation of the F-ratio, the AP-FSCS-ART improved by 18.4% compared to the AP-FSCS-ART-b; the degree of improvement could not be derived because the fact that the AP-FSCS-ART-c was difficult to find the failure.

In summary, both adaptive relative-ratio calculation and adaptive test case selecting components are helpful for the effectiveness of our AP-FSCS-ART.

V. CONCLUSIONS AND FUTURE WORK

We propose AP-FSCS-ART, an adaptive partition-based ART method on top of FSCS-ART. First, it divides the entire input domain into external and internal sub-domains. Then, two different algorithms are adaptively used to select the randomly generated set of candidate test cases in these two sub-domains to find the best one. To evaluate the effectiveness of our method, we perform experiments in the classical 12 real programs and compare them with the most classical method FSCS-ART and two recently proposed ART methods FSCS-ctsr and NNDC-ART. And the experimental results show that AP-FSCS-ART showed a significant improvement in failure detection capability, with improvements from 8.8% to 11.4% compared to the comparison algorithms. In the future, we will conduct experiments using more real programs, especially in high dimensions; and validate them using some SUTs.

ACKNOWLEDGMENT

This work was supported in part by the National Key Research and Development Project (No. 2021YFB1714200), the National Natural Science Foundation of China (No. 62002034), the Fundamental Research Funds for the Central Universities (No. 2022CDJDX-005), and the Starry Night Science Fund of Zhejiang University Shanghai Institute for Advanced Study (No. SN-ZJU-SIAS-001).

REFERENCES

[1] ACM. Collected algorithms from acm. Association for Computer Machinery, New York, 1980.

[2] Andrea Arcuri and Lionel Briand. A hitchhiker's guide to statistical tests for assessing randomized algorithms in software engineering. *Software Testing, Verification and Reliability*, 24(3):219–250, 2014.

[3] Muhammad Ashfaq, Rubing Huang, and Michael Omari. Fscs-simd: An efficient implementation of fixed-size-candidate-set adaptive random testing using simd instructions. In *2020 IEEE 31st International Symposium on Software Reliability Engineering (ISSRE)*, pages 277–288. IEEE, 2020.

[4] Kwok Ping Chan, Tsong Yueh Chen, and Dave Towey. Restricted random testing. In *European Conference on Software Quality*, pages 321–330. Springer, 2002.

[5] T. Y. Chen, H. Leung, and I. K. Mak. Adaptive random testing. In Michael J. Maher, editor, *Advances in Computer Science - ASIAN 2004. Higher-Level Decision Making*, pages 320–329, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

[6] Tsong Yueh Chen, Fei-Ching Kuo, and Huai Liu. Distributing test cases more evenly in adaptive random testing. *Journal of Systems and Software*, 81(12):2146–2162, 2008.

[7] TY Chen, TH Tse, Zongyuan Yang, et al. An innovative approach to tackling the boundary effect in adaptive random testing. In *2007 40th Annual Hawaii International Conference on System Sciences (HICSS'07)*, pages 262a–262a. IEEE, 2007.

[8] Yueh Tsong Chen, Fei-Ching Kuo, and Huai Liu. Adaptive random testing based on distribution metrics. *Journal of Systems and Software*, pages 1419–1433, 2009.

[9] Yueh Tsong Chen, Hing Leung, and K. I. Mak. Adaptive random testing. *Asian Computing Science Conference*, pages 320–329, 2004.

[10] R. Huang, W. Sun, Y. Xu, H. Chen, D. Towey, and X. Xia. A survey on adaptive random testing. 2020.

[11] Rubing Huang, Haibo Chen, Weifeng Sun, and Dave Towey. Candidate test set reduction for adaptive random testing: An overheads reduction technique. *Science of Computer Programming*, 214:102730, 2022.

[12] Rubing Huang, Weifeng Sun, Haibo Chen, Chenhui Cui, and Ning Yang. A nearest-neighbor divide-and-conquer approach for adaptive random testing. *Science of Computer Programming*, 215:102743, 2022.

[13] Fei-Ching Kuo et al. *On adaptive random testing*. Swinburne University of Technology, Faculty of Information & Communication ..., 2006.

[14] Glenford J Myers, Corey Sandler, and Tom Badgett. *The art of software testing*. John Wiley & Sons, 2011.

[15] Flannery B.P. Teukolsky S.A. Vetterling W.T Press, W.H. *Numerical Recipes*. Cambridge university press, 1986.

[16] András Vargha and Harold D Delaney. A critique and improvement of the cl common language effect size statistics of mcgraw and wong. *Journal of Educational and Behavioral Statistics*, 25(2):101–132, 2000.

[17] J. L. White and I. E. Cohen. A domain strategy for computer program testing. *IEEE Trans. Software Eng.*, pages 247–257, 1980.