SOFTWARE – PRACTICE AND EXPERIENCE
Softw. Pract. Exper. 2012; **42**:1055–1074
Published online 2 September 2011 in Wiley Online Library (wileyonlinelibrary.com). DOI: 10.1002/spe.1113

Comparison of adaptive random testing and random testing under various testing and debugging scenarios

Huai Liu*,†, Fei-Ching Kuo and Tsong Yueh Chen

Centre for Software Analysis and Testing, Swinburne University of Technology, Hawthorn 3122, Victoria, Australia

SUMMARY

Adaptive random testing is an enhancement of random testing. Previous studies on adaptive random testing assumed that once a failure is detected, testing is terminated and debugging is conducted immediately. It has been shown that adaptive random testing normally uses fewer test cases than random testing for detecting the first software failure. However, under many practical situations, testing should not be withheld after the detection of a failure. Thus, it is important to investigate the effectiveness with respect to the detection of multiple failures. In this paper, we compare adaptive random testing and random testing under various scenarios and examine whether adaptive random testing is still able to use fewer test cases than random testing to detect multiple software failures. Our study delivers some interesting results and highlights a number of promising research projects. Copyright © 2011 John Wiley & Sons, Ltd.

KEY WORDS: software testing; random testing; adaptive random testing; failure-detection effectiveness; testing effectiveness metric

1. INTRODUCTION

Software testing is the primary software engineering activity to assure the software quality. One popular definition of testing is 'the process of executing a program with the intent of finding errors' [1]. When a software contains a *fault* in source code, certain program inputs will trigger the software to behave unexpectedly. This observably unexpected behavior is called *failure*, and the inputs able to reveal failures are called *failure-causing inputs*. Details of failures and failure-causing inputs will be recorded and passed to the debugging team for fixing program faults. Because of limited testing resources, one key challenge of testing is how to effectively find failure-causing inputs from the entire set of inputs (known as the *input domain*). Many testing techniques have been developed to guide the design and selection of *test cases* (program inputs for testing).

Random testing (RT) is a fundamental software-testing technique, whose effectiveness in detecting software failures is often controversial [1, 2]. From the common observation that failure-causing inputs are frequently clustered into contiguous *failure regions* [3–6], Chen *et al.* [7] proposed *adaptive RT* (ART) to enhance the failure-detection effectiveness of RT. In ART, test cases are not only generated in a random manner but also evenly spread all over the whole input domain. Since then, ART has been used for detecting failures in various programs [8–10].

In previous studies on the testing effectiveness of ART, it has always been assumed that after the detection of a failure, testing will be immediately terminated, and the revealed failure will be passed to the debugging team to locate and fix the related fault. With this assumption, the failure-detection

*Correspondence to: Huai Liu, Centre for Software Analysis and Testing, Swinburne University of Technology, Hawthorn 3122, Victoria, Australia.
†E-mail: hliu@swin.edu.au

Contract/grant sponsor: Australian Research Council Discovery Grant; contract/grant number: DP0880295

effectiveness of ART and RT was evaluated and compared with respect to the F-measure, which refers to the expected number of test cases for revealing the first failure. Simulations and empirical studies [11, 12] have shown that ART normally has a smaller F-measure than RT. The F-measure of ART also has a smaller variation than that of RT [13], that is, ART has a more reliable testing performance than RT.

However, the aforementioned simplified scenario of conducting testing and debugging may not always be applicable in reality. In many practical situations, for example, in the early stage of software development, the program under test normally has multiple faults; as a result, testing ought to be continued after one failure is detected. Furthermore, for some programs, it may also be very time consuming to conduct debugging for the identification and removal of the related fault. Hence, it may be economical to continue testing while debugging is being conducted. All these practical situations trigger the following research question: is ART still better than RT in detecting multiple failures?

In this paper, we discuss two typical scenarios of conducting testing and debugging, namely *parallel testing and debugging* (where testing and debugging are simultaneously conducted) and *alternate testing and debugging* (where testing and debugging are conducted in turn). We investigate and compare the effectiveness of ART and RT under these testing and debugging scenarios. Because multiple failures can be revealed under these scenarios, we use a new metric, namely F^n-measure, which is defined as the expected number of test cases required for revealing the first n ($n \geqslant 1$) failures, to evaluate the multiple-failure-detection effectiveness of ART and RT.

The paper is organized as follows. In Section 2, we introduce some background information. In Section 3, we discuss two scenarios of conducting testing and debugging and present the concept of the F^n-measure as well as the theoretical analysis of the F^n-measure of RT. In Sections 4 and 5, we report our simulations and empirical studies where ART and RT were evaluated and compared in terms of detecting multiple failures. In Section 6, we discuss the threats to validity of our study. In Section 7, we conclude the paper and identify some future work.

2. BACKGROUND

2.1. Adaptive random testing

Suppose for a faulty program, the total number of all possible inputs (that is, the size of its input domain) is $\mathcal{D}$ and there are $\mathcal{F}$ failure-causing inputs. The ratio between $\mathcal{F}$ and $\mathcal{D}$ is defined as the *failure-causing ratio* (denoted by θ) of the program, that is, $\theta = \mathcal{F}/\mathcal{D}$ [14, 15]. The geometry of failure-causing inputs as well as their distributions over the input domain constitute the *failure pattern* of the program under test [7]. More specifically, the failure pattern contains information such as the shapes, sizes, orientations, and locations of failure regions. Both the failure-causing ratio and the failure pattern of the program under test are fixed after coding but unknown before testing. Some independent studies [3–6] showed that failure-causing inputs tend to be clustered into some contiguous failure regions.

Given that failure regions are contiguous, the non-failure regions should also be contiguous. If an input under test (*test case*) t is found to be non-failure-causing, it is very likely that its neighbors will not reveal any failures. From such an intuition, Chen *et al.* [7] proposed an enhancement of RT, namely ART, which both randomly selects and evenly spreads test cases. Fixed-size-candidate-set ART [11] (FSCS-ART) is one of the most popular algorithms to implement the 'even spread' notion of ART. In FSCS-ART, two sets of test cases are maintained. One set is the *executed set* $E = \{e_1, e_2, \cdots, e_n\}$, which consists of previously executed test cases; the other set is the *candidate set* $C = \{c_1, c_2, \cdots, c_k\}$, which contains a fixed number (k) of randomly generated candidates. When E is non-empty, a candidate from C will be selected as the next test case if it has the longest distance to its nearest neighbor in E. Figure 1 describes how FSCS-ART selects test cases.

In Figure 1, the termination condition can be satisfied 'when the testing resources are exhausted', 'when a certain number of test cases have been executed', or 'when a certain number of failures have been detected', and so on. Chen *et al.* [11] have pointed out that the F-measure of FSCS-ART

DOI: 10.1002/spe

1. Input an integer k, where $k > 1$.
2. Set $n = 0$ and $E = \{\}$.
3. Randomly generate a test case t from the input domain.
4. **while** (termination condition is not satisfied)
5. Add t into E, and increment n by 1.
6. Randomly generate k candidates $c_1, \cdots, c_k$ from the input domain, and construct C with these candidates.
7. **for** each candidate c_j, where $j = 1, \cdots, k$.
8. Calculate its distance d_j to its nearest neighbor in E.
9. **end_for**
10. Find $c_b \in C$ such that $\forall j = 1, \cdots, k, d_b \geq d_j$.
11. Set $t = c_b$.
12. **end_while**
13. Exit.

Figure 1. Algorithm of selecting test cases in FSCS-ART.

becomes smaller with the increase of k, but a larger k implies a higher computation overhead. Previous studies [11] showed that $k = 10$ is a reasonably fair setting to balance the testing effectiveness and the computation overhead of FSCS-ART.

The basic intuition of ART is the even spread of random test cases across the input domain. Besides FSCS-ART, various ART algorithms have been proposed to achieve the goal of even spread, such as restricted RT [12], ART by dynamic partitioning [16], lattice-based ART [17], and so on. These ART algorithms have different ways of evenly spreading test cases, different failure-detection effectiveness, and different computation overheads. FSCS-ART requires $O(n^2)$ computation time to select n test cases. When the number of test cases is very large, some techniques, such as *forgetting* [18], can be used to reduce the runtime of FSCS-ART. When step 8 in Figure 1 only refers to a fixed number of the executed test cases instead of all already executed test cases by the technique of forgetting, the test case selection time for FSCS-ART can then be independent of n. Previous studies [18] also showed that such a forgetting technique does not significantly affect the failure-detection effectiveness. Because our focus in this paper is on the failure-detection effectiveness rather than the efficiency, we do not use this forgetting technique in this study.

So that the theoretical analysis can be made tractable, it has often been assumed that the test case selection is carried out with replacement, that is, already executed test cases may be selected again. Although ART attempts to prevent test cases from clustering together, it is still likely (although the probability is very low) that ART selects duplicated test cases. The investigations conducted by Chen and Kuo [19] have shown the following: (i) the selection without replacement cannot significantly improve the effectiveness of both ART and RT and (ii) ART (either with or without replacement) is still more effective than RT without replacement. The effect of the selection without replacement will become negligible when the input domain has a large size. Because the test case selection with replacement is more efficient than without replacement and our study will involve large input domains, we use the selection with replacement for ART and RT in this study.

Besides the F-measure, the effectiveness of ART was also evaluated on the basis of the code coverage that it can achieve on the program under test [20]. It was shown that with the same number of test cases, ART not only achieves higher coverage than RT on certain structures of the program under test but also kills more mutants (which refer to the fault-seeded versions of the program under test).

In many previous studies of ART, it was often assumed that the program under test only accepts numeric inputs. Under such an assumption, the Euclidean distance is normally used as the metric for measuring the distance between inputs (line 8 of Figure 1). Some researchers [8, 10, 21, 22] have conducted investigations to apply ART into non-numeric programs.

2.2. *F-measure*

It has been argued that the F-measure is more appropriate than other metrics (such as P-measure and E-measure) in evaluating and comparing the effectiveness of ART and RT [23]. Theoretically,

DOI: 10.1002/spe

the F-measure of RT (denoted by F_{RT}) is equal to $1/\theta$ when test cases are selected with replacement. However, it is extremely difficult to theoretically analyze the F-measure of ART (F_{ART}). Previous studies of ART [11, 17, 24] often measured F_{ART} via simulations and empirical studies, whose basic procedure is introduced as follows.

In empirical studies, the investigation is conducted on real-life programs. Some faults are seeded into a selected program. Each faulty program (called mutant) has a certain failure pattern and a fixed value of θ. A failure is said to be detected if a fault-seeded version behaves differently from the selected program. Normally, empirical studies involve a large number of real-life programs and relevant mutants to produce conclusive results. Such an empirical study is very expensive in both time and labor. On the other hand, simulation studies are conducted in controllable settings and are used to provide comprehensive results under various factors. For simulation studies, so that the testing of faulty programs can be simulated, the θ and the failure pattern are predefined, and failure regions are then randomly placed inside the input domain. If a point inside the simulated failure regions is picked up, a failure is said to be detected.

In both empirical and simulation studies, a testing method (such as ART and RT) is applied to continuously select test cases until a failure is detected. In each run, the number of test cases required to detect the first failure, referred to as the F-count [23], is recorded. Such a process is repeated for a sufficiently large number of times ($\mathcal{S}$) to ensure that the average value of F-counts can be a reliable approximation for F_{ART} within a confidence level of $(1-\alpha)\times 100\%$ and an accuracy range of $\pm r\%$. On the basis of the central limit theorem [25], the value of $\mathcal{S}$ can be calculated as $\mathcal{S} = \left(100\cdot\Phi^{-1}\left((2-\alpha)/2\right)\cdot\sigma/(r\cdot\mu)\right)^2$, where μ and σ are the mean value and the standard deviation of F-counts, respectively, and Φ^{-1} denotes the inverse standard normal distribution function [25]. In previous studies, the default values of confidence level and accuracy range were often set as 95% and $\pm 5\%$, respectively.

3. TWO TESTING AND DEBUGGING SCENARIOS

As discussed in Section 1, previous studies on ART assumed a simplified scenario of testing and debugging, that is, once a failure is detected, testing terminates and debugging immediately starts to locate and remove the related fault. In this study, we investigate the effectiveness of ART under two different testing and debugging scenarios, as presented in the following.

- **Parallel testing and debugging**: Testing and debugging are conducted in parallel regardless of whether the debugging activity has fixed the program faults or not. In other words, after a failure is detected, debugging is conducted to identify and fix the related program fault; simultaneously, testing is continued to reveal further software failures. Such a scenario is particularly applicable when debugging is so time consuming that the testing team should not wait for the updated program. Note that because testing is conducted on the non-updated program, some of its detected failures may be related to the same program fault.
- **Alternate testing and debugging**: Testing and debugging are conducted alternately. After a failure is revealed, testing is temporarily suspended, and the related fault is debugged. After the completion of debugging, testing is then resumed on the updated program to detect further failure, which will in turn be used in the next round of debugging. Under this scenario, each of the detected failures is related to a distinct program fault.

Under both scenarios, multiple failures may be detected. The F-measure only reflects the effectiveness of detecting one software failure. In this study, we propose to use a new testing effectiveness metric, namely F^n-measure, the expected number of test cases required to detect the first n ($n \geqslant 1$) failures. Although the value of F^n-measure can be calculated by cumulating the values of F-measure for n successively detected failures, it is still better to use F^n-measure instead of F-measure in our study because of the following two reasons. Firstly, using the F^n-measure can reflect the effectiveness of ART in a more direct and concise manner than simply giving a list of F-measures and then cumulating them. Secondly, we normally assume the same initial condition

DOI: 10.1002/spe

(for example, testing is conducted from scratch) when comparing the effectiveness of testing methods. However, after the detection of a failure, different testing methods may reach different statuses (for example, different number of test cases have been used by various testing methods so far), and thus, we will not have the identical situations at successive detections of failures. Using the F^n-measure can avoid such a problem. As to be shown later, the F^n-measure can provide some new information on the testing effectiveness, which has never been disclosed by previous studies using the F-measure. In this study, we will not investigate into the characteristics of the F^n-measure, as our focus is on the multiple-failure-detection effectiveness of ART under different testing and debugging scenarios. Such work should be conducted separately, like the previous studies on other testing effectiveness metrics [26].

The F^n-measure of RT, denoted by F^n_{RT}, can be theoretically analyzed as follows. Suppose that a program has m ($m \geqslant 1$) faults and the failure-causing ratio is θ. It is known that $F^1_{\mathrm{RT}} = 1/\theta$. Under the parallel testing and debugging scenario, because no fault is removed, the failure-causing ratio remains to be equal to θ no matter how many failures have been detected. The expected number of test cases to reveal the ith failure after $i-1$ failures have been detected is $1/\theta$. Therefore, $F^n_{\mathrm{RT}} = \sum_{i=1}^{n} 1/\theta = n/\theta$ under the parallel testing and debugging scenario. Under the alternate testing and debugging scenario, assume the following: (i) the debugging activity based on a revealed failure results in the removal of one and only one fault and (ii) the removal of a fault does not introduce a new fault. Suppose that the failure-causing ratio will become θ_j after j faults have been fixed. Obviously, $\theta_0 = \theta$, and $\theta_m = 0$. Therefore, under the alternate testing and debugging scenario, $F^n_{\mathrm{RT}} = \sum_{j=0}^{n-1} 1/\theta_j$, where $1 \leqslant n \leqslant m$. Note that under the alternate testing and debugging scenario, if $n > m$, the F^n-measure for any testing method (including RT and ART) is ∞. In reality, there is no way to determine how many faults are contained in a faulty program; hence, it is possible that the tester tries to detect $n > m$ failures. Under such a situation, the testing activity can only reveal m failures at the best and will be terminated when the resources are exhausted.

4. SIMULATIONS

For convenience of illustration, we assume the numeric input domain in the simulations (this section) and empirical studies (the next section). Some previous studies [8, 10, 21, 22] have been conducted to apply ART into various programs with non-numeric program inputs. Because the measurement of F-measure is totally independent of these non-numeric approaches of ART, it has been used to measure the testing effectiveness without any difficulty in these studies. Because the measurement of F^n-measure is also independent of these approaches, we believe that the F^n-measure can also be easily used to evaluate the effectiveness of these non-numeric ART approaches.

Similar to F_{ART}, the F^n-measure of ART (F^n_{ART}) is extremely difficult, if not impossible, to analytically derive. Therefore, we conducted a series of simulations to evaluate F^n_{ART}. The simulations made use of some simple codes, which were based on authors' previous work [24]. The experimental settings of these simulations are as follows.

4.1. Experimental settings

In the simulations, the input domain is set as a square/cube in two/three-dimensional space. Each coordinate simulates one input parameter, and the program inputs are simulated as the vectors that are composed of two/three real numbers. The size of the input domain (denoted by $\mathcal{D}$) is calculated as the area/volume of the square/cube. The following three settings were used to evaluate F^n_{ART} under various failure patterns and failure-causing ratios.

- Single compact failure region: One single square/cubic failure region is randomly placed inside the input domain. θ is set as 0.05, 0.01, or 0.005. The size of the failure region (that is, the area/volume of the square/cube) is calculated as $\theta\mathcal{D}$.

DOI: 10.1002/spe

- Less compact failure region: One single rectangular/cuboid failure region is randomly placed inside the input domain. The edge length ratios of the failure region are $1{:}\gamma$ and $1{:}\gamma{:}\gamma$ for two-dimensional and three-dimensional input domains, respectively. A larger γ indicates that the failure region is less compact. We set $\theta = 0.01$, and $\gamma = 5, 10, 50, 100$. The size of the failure region (that is, the area/volume of the rectangle/cuboid) is calculated as $\theta\mathcal{D}$.
- Multiple failure regions: A number (η) of equal-sized non-overlapping square/cubic failure regions are randomly placed inside the input domain. We set $\theta = 0.01$, and $\eta = 5, 10, 50, 100$. The size of each failure region (that is, the area/volume of each square/cube) is calculated as $\theta\mathcal{D}/\eta$.

Among these three failure patterns, ART has the smallest F-measure when there is only one single compact failure region. F_{ART} increases with the increase of the dimension of input domain, θ, γ, or η [24].

4.2. Two ways to conduct adaptive random testing after failure detection

In previous simulation studies for evaluating F_{ART}, testing was always terminated once a failure was detected. As such, all elements in the executed set E were always non-failure-causing (note that when a test case t detects a failure, testing will be terminated and t will not be added to E). This ART implementation ensures that every new test case must be far apart from all executed but non-failure-causing test cases. In this study, we attempt to study the ART's effectiveness in detecting multiple failures, so testing will terminate after a certain number (n) of failures have been detected. In other words, in the simulation studies for evaluating F^n_{ART}, some elements in E may be failure causing. In our study, we treat these failure-causing test cases in two different ways.

One way is called *ART with comprehensive memory* (abbreviated as ART-c), which means that ART will memorize all executed test cases in the executed set E, regardless of whether these test cases are failure causing or not. ART-c ensures that test cases are far apart from each other. In other words, ART-c attempts to achieve an even spread of the whole set of executed test cases.

The other way to treat the failure-causing test cases is called *ART with selective memory* (abbreviated as ART-s), which means that ART will only memorize the non-failure-causing test cases, that is, excluding all failure-causing test cases from E. ART-s ensures that each new test case is away from all already executed but non-failure-causing test cases. Intuitively speaking, given that the failure regions are contiguous, if a test case is failure causing, it is very likely that its neighbors are also failure causing. By 'forgetting' the failure-causing test cases, ART-s increases the probability of failure detection.

Obviously, ART-c and ART-s are effectively identical before the detection of the first failure. In other words, they have the same F-measures. On the other hand, as to be shown later, their F^n-measures are quite different because of their different ways of treating failure-causing test cases.

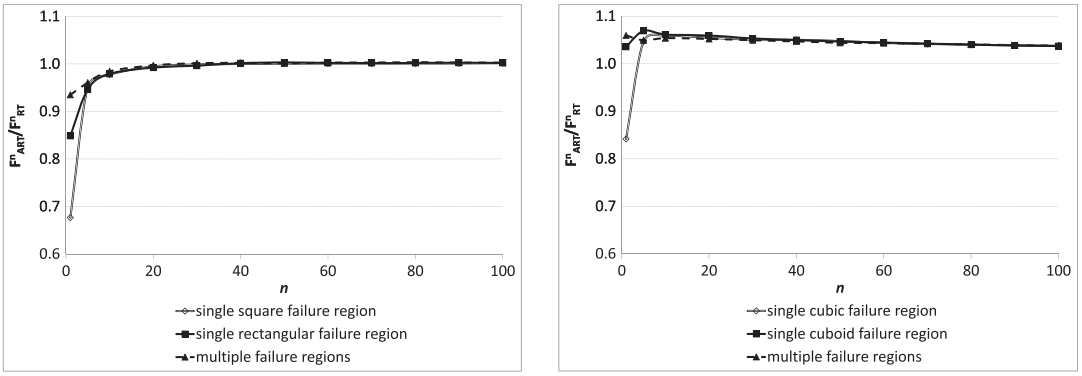
Note that we can also implement ART from scratch after a failure detection, that is, we forget all executed test cases once a failure is detected and rerun ART with an empty E. However, this is not a good way because it ignores all the useful information provided by the previously executed test cases. Therefore, we do not investigate further along this approach in this paper.

4.3. Comparison of F^n_{ART} and F^n_{RT} under the parallel testing and debugging scenario

Simulations have been conducted to evaluate the F^n-measures for both ART-c and ART-s (denoted by $F^n_{\text{ART-c}}$ and $F^n_{\text{ART-s}}$, respectively) under the parallel testing and debugging scenario. In the simulations, because θ has been predefined, we can theoretically derive $F^n_{\text{RT}} = n/\theta$ (as discussed in Section 3). Hence, we do not collect the simulated values for F^n_{RT}. The theoretically derived values for F^n_{RT} will be used instead. Furthermore, we only report the ratio between F^n_{ART} and F^n_{RT} because we are interested in how much ART outperforms RT in terms of the F^n-measure. Obviously, $F^n_{\text{ART}}/F^n_{\text{RT}} < 1$ implies that ART is better than RT with respect to the F^n-measure, and the smaller the ratio is, the more significantly ART outperforms RT.

The simulation results on the F^n-measures of ART-c are reported in Figure 2, where the x-axis denotes n ($n = 1, 5, 10, 20, \cdots, 100$), the number of detected failures, and the y-axis denotes the

DOI: 10.1002/spe



(a) Two-dimension (b) Three-dimension

Figure 2. Comparison of $F^n_{\text{ART-c}}$ and F^n_{RT} under the parallel testing and debugging scenario.

$F^n_{\text{ART-c}}/F^n_{\text{RT}}$. Note that although the maximum number (that is, 100) of the detected failures seems large, such a broad range of n can help us get a full picture of the F^n-measure. In the simulations, we observed that ART-c has similar F^n-measures for different values of θ, γ, and η. Therefore, in Figure 2, we only plot the data of $F^n_{\text{ART-c}}$ for single square/cubic failure region with $\theta = 0.01$, single rectangular/cuboid failure region with $\gamma = 10$, and multiple failure regions with $\eta = 10$.

From the simulation results, we can observe the following:

- With the increase of n, $F^n_{\text{ART-c}}$ is quickly approaching to F^n_{RT}.
- Similar to F_{ART}, $F^n_{\text{ART-c}}$ also becomes higher when the dimension of input domain is higher, the failure region is less compact, or the number of failure regions is larger.

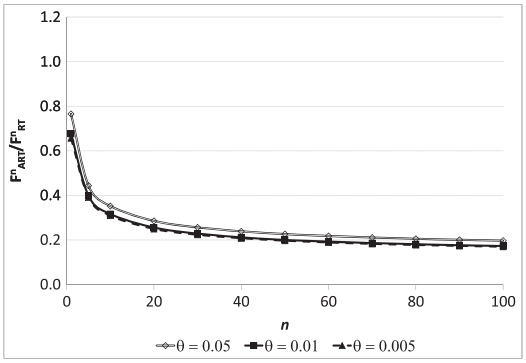
Figure 3 gives the simulation results on the F^n-measures of ART-s. We have the following observations:

- Under most situations, $F^n_{\text{ART-s}}/F^n_{\text{RT}} < 1$, that is, ART-s normally has a smaller F^n-measure than RT.
- The larger the n is, the smaller is $F^n_{\text{ART-s}}/F^n_{\text{RT}}$.
- $F^n_{\text{ART-s}}/F^n_{\text{RT}}$ increases with the increase of the dimension, θ, γ, and η.
- Even under the most unfavorable condition for ART (that is, there are a large number of equal-sized failure regions [24]), ART-s has much smaller F^n-measures than RT (on the contrary, ART and RT have similar F-measures under such a condition).

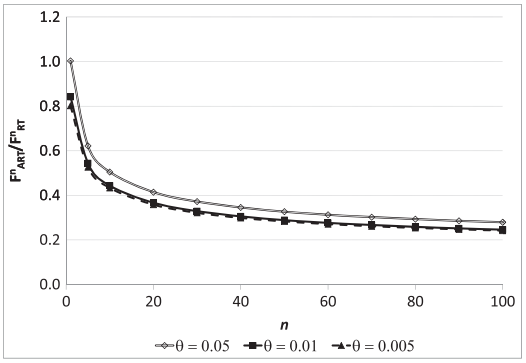
From our simulation results, it can be easily concluded that under the parallel testing and debugging scenario, ART-s is much more effective than ART-c in terms of the F^n-measure. ART is based on the intuition that adjacent inputs cause similar program behaviors. ART-s ensures that each selected test case is far apart from all already executed but non-failure-causing test cases, which is consistent with the basic intuition of ART. On the other hand, in ART-c, some test cases may be enforced far apart from some failure-causing inputs; as a result, these test cases may have a low chance to detect a failure. Therefore, it is intuitively expected that ART-s can detect failures more quickly than ART-c.

It might be argued that the small F^n-measure of ART-s may be an illusion of high testing effectiveness, as it is possible that ART-s might repeatedly select test cases from the same failure region. However, it should be noted that under the parallel testing and debugging scenario, we do not have the assumption that a failure region is related to one and only one program fault. One fault may cause multiple failure regions, and one failure region may be related to multiple faults. We may need to select a certain number of test cases from one failure region in order to reveal (and hence fix) all the related program faults.

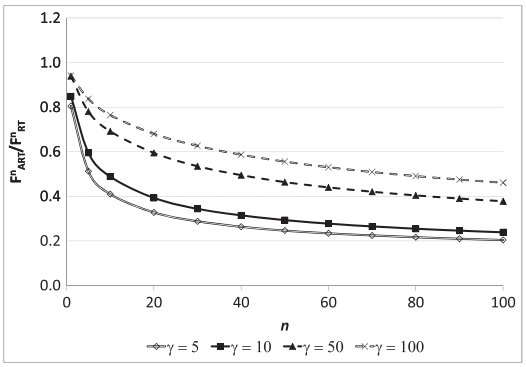
DOI: 10.1002/spe



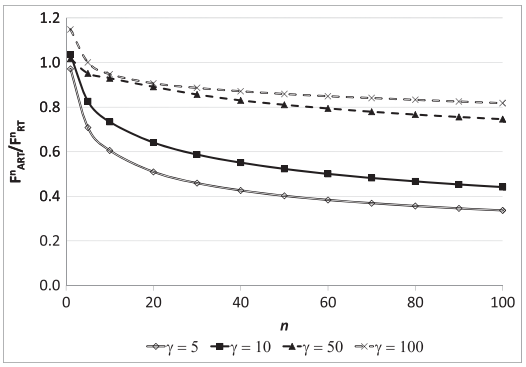
(a) Two-dimension, single square failure region



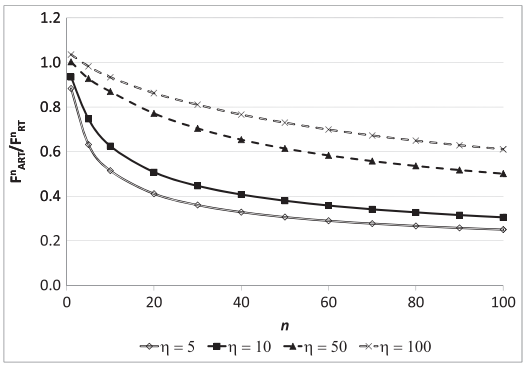
(b) Three-dimension, single cubic failure region



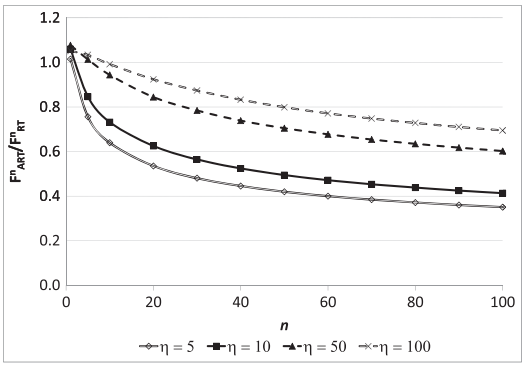
(c) Two-dimension, single rectangular failure region



(d) Three-dimension, single cuboid failure region



(e) Two-dimension, multiple failure regions



(f) Three-dimension, multiple failure regions

Figure 3. Comparison of $F^n_{\text{ART-s}}$ and F^n_{RT} under the parallel testing and debugging scenario.

4.4. *Comparison of F^n_{ART} and F^n_{RT} under the alternate testing and debugging scenario*

We also conducted a series of simulations to study and compare F^n-measures of ART and RT under the alternate testing and debugging scenario. In order to simulate different faults and the activity of debugging, we only investigated the 'multiple failure regions' situation. In these simulations, we have made two additional assumptions. One assumption is that each failure region is related to one and only one program fault, that is, η failure regions imply $m = \eta$ distinct faults. Note that such an assumption may not be valid in practice, but we need it to simplify our simulation settings, and this assumption is no longer required in our empirical studies (Section 5). Another assumption is that the removal of a fault does not introduce new faults. Under these two assumption, the debugging

DOI: 10.1002/spe

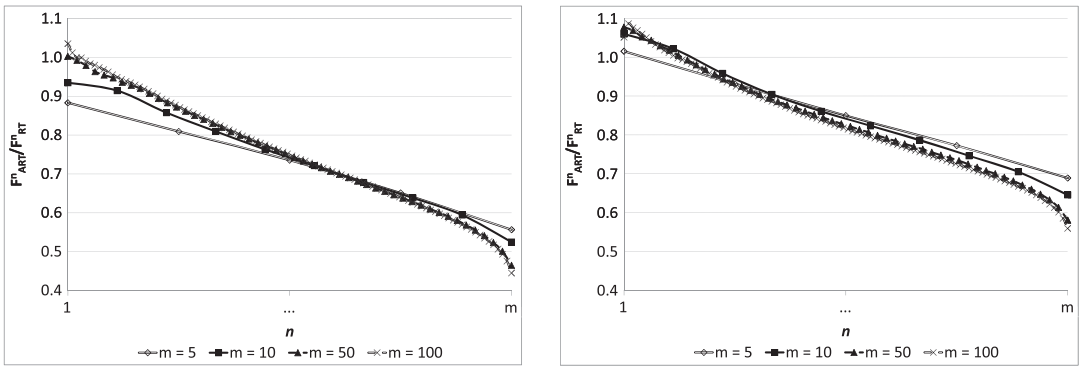
activity is simulated as follows. After a failure is detected, the region containing the failure-causing test case will be regarded as a non-failure region (that is, all points from this region will no longer be regarded as failure causing).

In these simulations (where there are $m = \eta$ equal-sized failure regions), given that the original failure-causing ratio is θ, after j ($j = 0, 1, \cdots, m$) failures have been detected, the failure-causing ratio becomes $((m-j)/m)\theta$. Therefore, F_{RT}^n is $\sum_{j=0}^{n-1} m/((m-j)\theta)$ (where $1 \leqslant n \leqslant m$) or ∞ (where $n > m$). Because the concrete values of F_{RT}^n can be theoretically derived, we do not collect the simulated values for F_{RT}^n but use the theoretically derived values for F_{RT}^n in the analysis. Furthermore, we only report $F_{\mathrm{ART}}^n / F_{\mathrm{RT}}^n$ for ease of comparison in Figures 4 and 5. In these figures, $n = 1, 2, \cdots, m$, and the number of simulated faults m is equal to the number of failure regions η.

The observations made from Figures 4 and 5 are listed as follows:

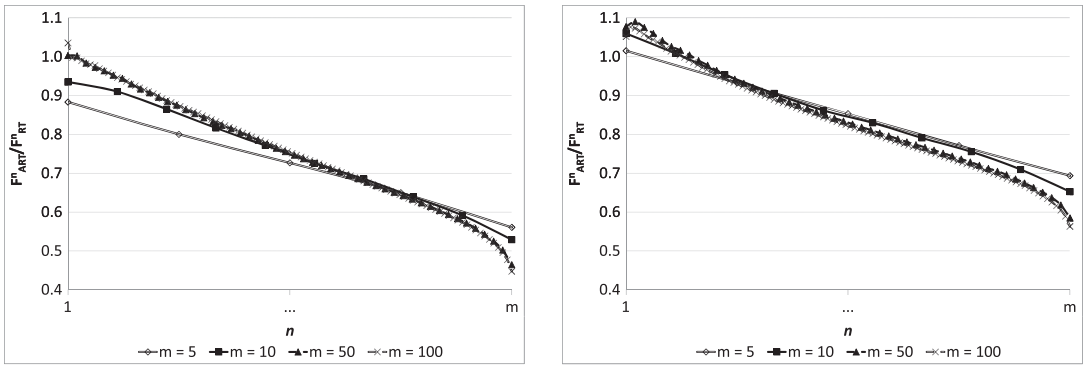
- Both $F_{\mathrm{ART\text{-}c}}^n / F_{\mathrm{RT}}^n$ and $F_{\mathrm{ART\text{-}s}}^n / F_{\mathrm{RT}}^n$ decrease towards $1/2$ as n is approaching m (that is, the number of fixed faults is approaching the total number of original faults).
- There is no apparently significant difference between the performances of ART-c and ART-s.

Previous studies [24] have shown that when there are a large number of equal-sized square/cubic failure regions (which is the most unfavorable condition for ART), ART and RT have similar performance in terms of the F-measure (the effectiveness of revealing one failure/fault). Our results show that under the alternate testing and debugging scenario, even if the initial pattern in the simulations



(a) Two-dimension (b) Three-dimension

Figure 4. Comparison of $F_{\mathrm{ART\text{-}c}}^n$ and F_{RT}^n under the alternate testing and debugging scenario (where $m = \eta$).



(a) Two-dimension (b) Three-dimension

Figure 5. Comparison of $F_{\mathrm{ART\text{-}s}}^n$ and F_{RT}^n under the alternate testing and debugging scenario (where $m = \eta$).

DOI: 10.1002/spe

Table I. Input domains of subject programs.

Program	Dimension	From	To
`bessj0`	1	(-500)	(500)
`bessj`	2	$(2, -500)$	$(102, 500)$
`plgndr`	3	$(-1, 10, -1.1)$	$(12, 100, 1.1)$
`select`	4	$(-500, -500, -500, -500)$	$(500, 500, 500, 500)$

is the most unfavorable condition for ART, ART significantly outperforms RT in terms of the F^n-measure (the effectiveness of revealing multiple failures/faults). This is understandable because with the increase of n, the number of undetected failure regions decreases, and thus the failure pattern becomes more and more favorable for ART. Although it was already known that the performance of ART becomes better when the number of failure regions is smaller [24], our study is still worthwhile as it quantitatively (not only intuitively and qualitatively) shows to what extent ART can outperform RT with respect to F^n-measure.

5. EMPIRICAL STUDIES

We selected four C programs, namely `bessj0`, `bessj`, `plgndr`, and `select`, from Numerical Recipes [27], as the subject programs in our empirical analysis. We defined hyperrectangular input domains for these programs, as summarized in Table I. Among these programs, `bessj0`, `bessj`, and `plgndr`, which compute various special functions, accept one, two, and three input parameters, respectively. The program `select` attempts to find the qth smallest element from an array containing p real numbers. In this study, we set $p = 4$ and $q = 2$, that is, a four-dimensional input domain is used for `select` in our study.

For each subject program, we applied the mutation analysis technique [28] to generate various mutants, each of which is related to a single fault. We used a C program mutation tool, namely *Csaw* [29]. Csaw can inject many distinct faults into a single line of code. For example, the eighth line of program `bessj0` is a simple assignment statement ('y = x * x'). Csaw can seed 27 different faults into it, including operator replacement (e.g., 'y += x * x', 'y = x / x"), variable replacement (e.g., 'z = x * x'), and so on. These 27 faults, in turn, result in 27 different mutants. Therefore, even for small-sized programs, Csaw can generate a large amount of mutants. Some statistical data in the mutation analysis are given in Table II. In the table, N_t denotes the total number of mutants generated by Csaw for each subject program. Among all these mutants, there are a number (N_i) of mutants that are syntactically incorrect and thus cannot be compiled. Among the remaining mutants, some mutants show certain problems (such as overflow, infinite loop) when executed. The number of such mutants is denoted by N_a. N_e denotes the number of *equivalent mutants*, which show exactly the same behavior as the program under test for any possible program input. Because it is an undecidable problem to automatically identify all the equivalent mutants, in this study, we considered a mutant as equivalent to the program under test if it cannot be killed by any of 100,000 randomly generated test cases.[‡] For the remaining mutants, we divided them into two groups, the mutants whose failure-causing ratio (θ) is larger than or equal to 0.1 and those with $\theta < 0.1$.[§] The numbers of mutants in these two groups are denoted by N_l and N_s. Note that $N_t = N_i + N_a + N_e + N_l + N_s$.

Consider mutants $\mathcal{M}_1$ and $\mathcal{M}_2$. Suppose that for any test case t, if t kills $\mathcal{M}_1$, it also kills $\mathcal{M}_2$; on the other hand, if $\mathcal{M}_1$ passes the test on t, $\mathcal{M}_2$ also passes the test on t. In other words, $\mathcal{M}_1$ and $\mathcal{M}_2$ have identical failure-causing inputs. Therefore, $\mathcal{M}_1$ and $\mathcal{M}_2$ will have the 'same' failure pattern,

[‡]Note that except for the first input parameter of `bessj` and the first and second input parameters of `plgndr`, all other input parameters of the subject programs are real numbers. Hence, each subject program has at least one real number parameter. As a consequence, each program has an extremely large number of possible inputs. It is infeasible to test each mutant against all possible program inputs.

[§]The value of θ for each mutant was experimentally derived. We used RT to test each mutant and got the F-measure. θ is calculated as $1/F_{\mathrm{RT}}$.

DOI: 10.1002/spe

Table II. Mutant statistics of subject programs.

Program	N_t	N_i	N_a	N_e	N_l	N_s	N_d
`bessj0`	832	276	0	103	221	232	141
`bessj`	1180	345	39	135	308	353	107
`plgndr`	556	120	12	221	9	194	10
`select`	1322	469	62	305	464	22	9

although they may be related to different faults (and thus may return different program outputs).[¶] Among the generated mutants, we found that many mutants show the same failure pattern.

In our study, we only used the faults seeded in the mutants with $\theta < 0.1$. Among these mutants, we first found the mutants with the same failure pattern, from which we randomly selected one of them and discarded all others. Finally, we get N_d (whose values are given in Table II) mutants, which are 'distinct' to one another in terms of their failure patterns. For programs `bessj0` and `bessj`, we randomly selected 10 mutants from their 141 and 107 distinct mutants, respectively. Because programs `plgndr` and `select` only have 10 and 9 distinct mutants, respectively, all of them were selected for experimentation. The faults associated with the selected mutants were altogether seeded into each subject program to generate four multiple-fault mutants, which then effectively contain 10, 10, 10, and 9 faults for `bessj0`, `bessj`, `plgndr`, and `select`, respectively. The multiple-fault mutants were then used to evaluate and compare the F^n-measures of ART and RT. Note that the failure patterns in our subject programs are different from the ideal settings in the simulations (Section 4). Distinct faults in the subject programs cause different values of θ and various failure patterns (such as different sizes and shapes of failure regions). In addition, some faults may have common failure-causing inputs, that is, parts of their failure regions are overlapped by each other.

5.1. Comparison of F^n_{ART} and F^n_{RT} under the parallel testing and debugging scenario

We used ART-c, ART-s, and RT to test the multiple-fault-seeded programs. After a failure is detected, ART-s will forget the test case that reveals the failure, whereas ART-c will memorize all executed test cases, regardless of whether they are failure causing or not. Testing will terminate when 20 failures are detected. The experimental results are shown in Tables III–VI and Figure 6. Note that as compared with the simulations, it is often very difficult to analyze the value of θ for each mutant, so we do not provide the theoretical values of F^n_{RT} in the empirical study. In Tables III–VI, we give all the empirical values of F^n-measures for RT, ART-c, and ART-s. Figure 6 reports the ratios $F^n_{\text{ART-c}}/F^n_{\text{RT}}$ and $F^n_{\text{ART-s}}/F^n_{\text{RT}}$.

From Tables III–VI and Figure 6, we can have the following observations:

- The F^n-measure of ART-c is smaller than that of RT when n is small. However, $F^n_{\text{ART-c}}/F^n_{\text{RT}}$ is approaching 1 with the increase of n.
- ART-s always has a smaller F^n-measure than RT, and $F^n_{\text{ART-s}}/F^n_{\text{RT}}$ becomes smaller when n becomes larger.
- $F^n_{\text{ART-s}}$ is always smaller than $F^n_{\text{ART-c}}$.

The results of our empirical studies under the parallel testing and debugging scenario are consistent with the simulation results in Section 4.3. In terms of the F^n-measure, the effectiveness of ART-c and RT is similar when n is very large, whereas ART-s always outperforms RT and the improvement of performance becomes more significant with the increase of n. In summary, ART-s can detect the same number of failures with fewer test cases than RT and ART-c under the parallel testing and debugging scenario.

As compared with the simulation results in Section 4.3, Tables III–VI do not show the impact of some factors (such as dimension, θ, γ, and η) on F^n_{ART}. In the simulations, we can change just one factor while keeping all others unmodified. In this way, we are able to know how a factor is

[¶]One example is that the faults of $\mathcal{M}_1$ and $\mathcal{M}_2$ are seeded in the same statement. In this situation, it is likely that $\mathcal{M}_1$ and $\mathcal{M}_2$ have identical failure-causing inputs and thus the same failure pattern.

DOI: 10.1002/spe

Table III. F^n-measures on `bessj0` under the parallel testing and debugging scenario.

n	F^n_{RT}	$F^n_{\text{ART-c}}$	$F^n_{\text{ART-s}}$
1	12.07	9.63	9.63
2	23.84	21.61	16.62
3	35.81	33.46	22.38
4	47.53	45.86	27.38
5	60.04	58.00	32.01
6	72.13	70.14	36.23
7	84.35	82.41	40.29
8	96.23	94.52	44.15
9	108.15	106.64	47.93
10	119.95	118.75	51.89
11	131.81	130.89	55.43
12	143.78	143.00	59.05
13	155.43	154.50	62.56
14	167.83	166.42	65.90
15	180.07	178.56	69.33
16	192.72	190.86	72.74
17	205.12	202.65	76.15
18	216.82	214.82	79.52
19	228.83	226.70	82.81
20	241.18	238.69	86.22

Table IV. F^n-measures on `bessj` under the parallel testing and debugging scenario.

n	F^n_{RT}	$F^n_{\text{ART-c}}$	$F^n_{\text{ART-s}}$
1	9.52	6.51	6.51
2	19.17	16.87	11.12
3	28.67	26.23	14.60
4	38.40	36.07	17.85
5	48.00	45.95	21.00
6	57.43	55.77	23.88
7	67.29	65.25	26.55
8	77.23	75.24	29.12
9	86.71	85.43	31.67
10	96.62	95.70	34.07
11	106.18	105.39	36.45
12	116.32	115.30	38.78
13	126.12	124.76	41.05
14	135.49	134.70	43.32
15	145.40	144.70	45.52
16	154.97	154.65	47.70
17	164.34	164.62	49.88
18	173.81	173.99	52.01
19	183.37	183.69	54.10
20	192.95	193.10	56.20

correlated with F^n_{ART}. However, in the empirical studies, we lost the flexibility of controlling various factors individually. Once the subject programs were selected and faults were injected, failure patterns are fixed, and thus all factors could not be changed. Furthermore, it is very difficult, if not impossible, to control a specific factor through the design of a seeded fault.

In order to validate whether the improvement of ART-c or ART-s over RT is statistically significant, we conducted binomial tests [30] for the data shown in Tables III–VI. The significant level is set as 0.05, and the null hypothesis (H_0) is that ART-s/ART-c does not outperform RT under the parallel testing and debugging scenario. The results of the statistical tests are given in Tables VII and VIII, which also report the p-value on each subject program. It can be observed that all p-values

DOI: 10.1002/spe

Table V. F^n-measures on `plgndr` under the parallel testing and debugging scenario.

n	F^n_{RT}	$F^n_{\text{ART-c}}$	$F^n_{\text{ART-s}}$
1	395.47	146.86	146.86
2	798.52	474.53	250.37
3	1172.46	794.22	340.98
4	1562.02	1129.05	428.72
5	1949.51	1468.95	513.60
6	2323.60	1808.80	599.33
7	2728.78	2145.28	674.84
8	3114.15	2493.14	748.55
9	3499.32	2846.09	823.59
10	3886.99	3188.22	894.12
11	4269.16	3549.04	965.54
12	4674.50	3889.53	1032.92
13	5051.07	4234.58	1105.93
14	5450.24	4581.24	1178.81
15	5829.39	4927.29	1248.50
16	6216.40	5282.07	1317.86
17	6581.47	5640.55	1386.27
18	6959.63	5988.94	1454.12
19	7353.79	6340.34	1525.34
20	7731.94	6716.12	1589.72

Table VI. F^n-measures on `select` under the parallel testing and debugging scenario.

n	F^n_{RT}	$F^n_{\text{ART-c}}$	$F^n_{\text{ART-s}}$
1	2.18	1.87	1.87
2	4.35	4.03	3.69
3	6.52	6.15	5.41
4	8.61	8.28	7.01
5	10.73	10.38	8.57
6	12.86	12.51	10.06
7	15.04	14.57	11.53
8	17.26	16.70	12.96
9	19.40	18.87	14.40
10	21.46	20.92	15.84
11	23.62	23.07	17.23
12	25.76	25.20	18.61
13	27.83	27.30	20.04
14	29.98	29.43	21.38
15	32.11	31.52	22.72
16	34.35	33.60	24.11
17	36.45	35.78	25.47
18	38.55	37.91	26.84
19	40.72	40.04	28.22
20	43.00	42.28	29.53

Table VII. Binomial tests on $F^n_{\text{ART-c}}$ and F^n_{RT} under the parallel testing and debugging scenario.

	`bessj0`	`bessj`	`plgndr`	`select`
p-value	0.000002	0.011818	0.000002	0.000002
Conclusion	Reject H_0	Reject H_0	Reject H_0	Reject H_0

Table VIII. Binomial tests on $F^n_{\text{ART-s}}$ and F^n_{RT} under the parallel testing and debugging scenario.

	`bessj0`	`bessj`	`plgndr`	`select`
p-value	0.000002	0.000002	0.000002	0.000002
Conclusion	Reject H_0	Reject H_0	Reject H_0	Reject H_0

DOI: 10.1002/spe

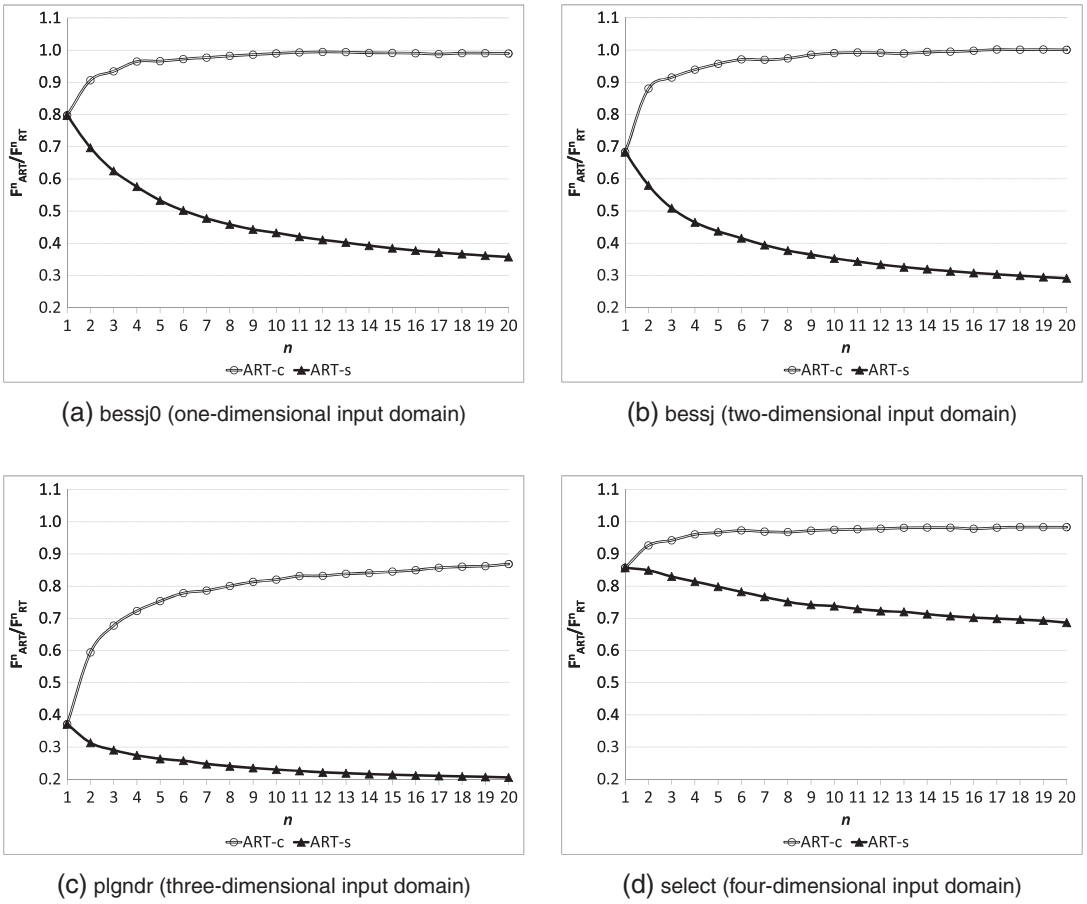


Figure 6. Comparison of F^n_{ART} and F^n_{RT} on real-life programs under the 'parallel testing and debugging' scenario.

are smaller than 0.05. As a consequence, all the null hypotheses are rejected. Therefore, it is statistically significant that ART-c or ART-s outperforms RT in terms of the F^n-measure (where $n \leqslant 20$) under the parallel testing and debugging scenario.

5.2. *Comparison of F^n_{ART} and F^n_{RT} under the alternate testing and debugging scenario*

We also tested the multiple-fault mutants by ART-c, ART-s, and RT under the alternate testing and debugging scenario. When a failure is detected, the related program fault will be fixed prior to further testing. In these empirical studies, we also have the assumption that the removal of a fault does not introduce new faults. Because we already know the faults, the debugging activity can be simulated in the following way. When a failure is revealed, we run all m original single-fault mutants, from which the corresponding multiple-fault mutants are constructed, against the failure-causing test case (the number of faults $m = 10, 10, 10$, and 9 for programs `bessj0`, `bessj`, `plgndr`, and `select`, respectively). Then, the fault(s) injected in the killed mutant(s) will be regarded as relevant to the revealed failure. Note that different from the simulations studies in Section 4.4, a detected failure may be related to more than one single-fault mutant. If such a situation occurs, we randomly select one of the killed mutants and treat its fault relevant to the revealed failure and remove it. The empirical results are shown in Tables IX–XII and Figure 7, where the number of detected failures is $n = 1, 2, \cdots, m$.

From Tables IX–XII and Figure 7, we observed the following:

- Both ART-c and ART-s always outperform RT in terms of the F^n-measure.
- ART-s always has smaller F^n-measures than ART-c.

DOI: 10.1002/spe

Table IX. F^n-measures on `bessj0` ($m = 10$) under the alternate testing and debugging scenario.

n	F^n_{RT}	$F^n_{\text{ART-c}}$	$F^n_{\text{ART-s}}$
1	12.07	9.63	9.63
2	29.82	23.93	19.02
3	54.34	43.27	27.05
4	83.05	71.18	34.71
5	116.90	102.20	43.46
6	158.23	137.96	53.88
7	209.27	180.94	71.23
8	276.86	237.38	105.25
9	467.83	397.73	237.42
10	1420.44	1305.82	1013.26

Table X. F^n-measures on `bessj` ($m = 10$) under the alternate testing and debugging scenario.

n	F^n_{RT}	$F^n_{\text{ART-c}}$	$F^n_{\text{ART-s}}$
1	9.52	6.51	6.51
2	20.21	18.07	13.22
3	31.14	29.05	19.35
4	42.92	40.92	24.96
5	54.76	52.47	30.05
6	67.03	64.47	34.93
7	78.89	76.61	39.57
8	90.93	89.39	44.12
9	108.16	106.67	50.83
10	143.93	141.30	68.69

Table XI. F^n-measures on `plgndr` ($m = 10$) under the alternate testing and debugging scenario.

n	F^n_{RT}	$F^n_{\text{ART-c}}$	$F^n_{\text{ART-s}}$
1	395.47	146.86	146.86
2	770.12	480.56	241.32
3	1162.79	797.15	338.40
4	1564.89	1116.20	427.78
5	1945.92	1478.19	509.90
6	2372.34	1861.15	605.97
7	2895.87	2293.06	733.53
8	3655.31	2846.64	968.99
9	5985.38	3894.45	1736.51
10	13,584.70	7137.34	4491.72

Our first observation is consistent with a simulation result in Section 4.4 that ART has smaller F^n-measure than RT under the alternate testing and debugging scenario. As compared with the simulation results of ART-s and ART-c reported in Section 4.4, our second observation indicates that there is a significant difference between the F^n-measures of ART-s and ART-c. Such a discrepancy between our simulations and empirical studies is explained as follows. We have investigated the faults seeded in the subject programs. Many failure regions associated with these seeded faults are adjacent to each other or are even overlapping. As discussed in Section 4.3, ART-s ensures that the next test case is only far apart from the executed and non-failure-causing test cases, whereas ART-c enforces the next test case far apart from all executed test cases (including those that are failure causing). Therefore, ART-s has a higher chance than ART-c to reveal the faults whose failure regions are overlapping with or adjacent to those of the already fixed faults. As a consequence, a performance difference between ART-c and ART-s is expected. On the other hand, in our simulations, the simulated failure regions are randomly placed and are designed not to have any overlapping. In

DOI: 10.1002/spe

Table XII. F^n-measures on `select` ($m = 9$) under the alternate testing and debugging scenario.

n	F^n_{RT}	$F^n_{\text{ART-c}}$	$F^n_{\text{ART-s}}$
1	2.18	1.87	1.87
2	4.42	4.18	3.90
3	7.02	6.36	6.01
4	10.07	8.93	8.27
5	13.50	11.87	10.81
6	17.69	15.29	13.68
7	23.16	19.59	17.27
8	30.66	25.79	22.17
9	44.83	37.57	31.57

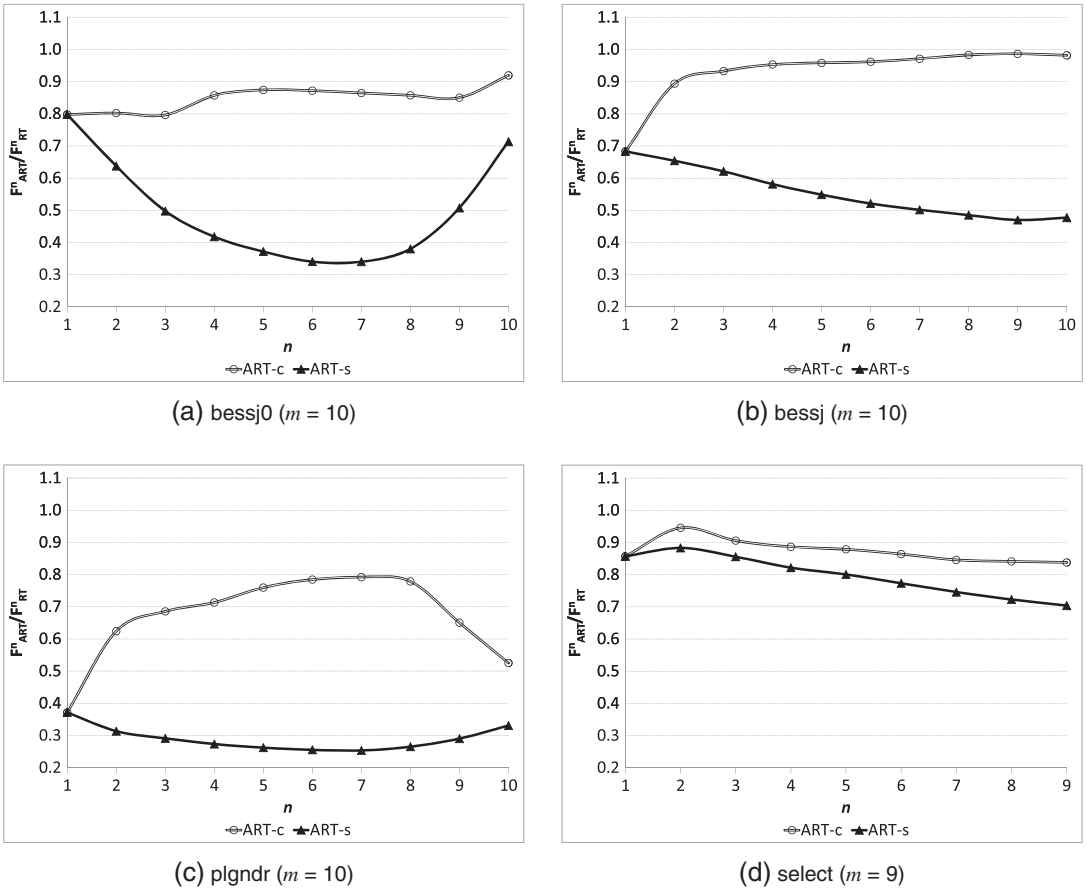


Figure 7. Comparison of F^n_{ART} and F^n_{RT} on real-life programs under the 'alternate testing and debugging' scenario.

such a case, failure regions may be adjacent to or far apart from each other; therefore, the impact of keeping or forgetting failure-causing test cases will then be less significant.

There is another difference between the results of the simulations (Section 4.4) and the empirical studies (this section). For the simulations (refer to Figures 4 and 5), both $F^n_{\text{ART-c}}/F^n_{\text{RT}}$ and $F^n_{\text{ART-s}}/F^n_{\text{RT}}$ monotonically decrease with the increase of n. However, such monotonicity is rarely observed in the empirical studies (refer to Figure 7). The reason behind such a discrepancy can be explained as follows. For RT, its F^n-measure is solely dependent on the value of θ. However, the performance of ART not only depends on θ but also on the failure pattern [24]. In the simulations, all the simulated failure regions have the same size and shape, so ART has consistent performance in detecting each

DOI: 10.1002/spe

failure region. However, in the empirical studies, different faults result in different failure patterns (failure regions with different sizes, shapes, orientations, and locations), and thus ART has different effectiveness in revealing distinct faults. For example, on programs `bessj0` and `plgndr`, $F^n_{\text{ART-s}}/F^n_{\text{RT}}$ first decreases as n increases, but after reaching a certain value ($n = 7$), it increases with the increase of n. Such a relatively large value of $F^n_{\text{ART-s}}/F^n_{\text{RT}}$ does not necessarily imply that the performance of ART-s deteriorates in terms of the F^n-measure along with the increase of n; as a matter of fact, it only implies that the improvement of ART-s over RT for some faults is not as significant as that for other faults. In our empirical studies, $F^n_{\text{ART-s}}/F^n_{\text{RT}}$ is always smaller than 1 for any $n \leqslant m$ (note that when $n > m$ both $F^n_{\text{ART-s}}$ and F^n_{RT} are ∞), that is, ART-s is always more effective than RT, despite the fact that its improvement over RT varies for different numbers of faults.

Tables XIII and XIV summarize the results of binomial tests for the F^n-measures under the alternate testing and debugging scenario for ART-c and ART-s, respectively. All p-values are smaller than the significant level of 0.05, and thus, all the null hypotheses (H_0, ART-c/ART-s does not outperform RT under the alternate testing and debugging scenario) are rejected. In conclusion, it is statistically significant that ART-c/ART-s is more effective than RT in terms of the F^n-measure under the alternate testing and debugging scenario.

6. THREATS TO VALIDITY

There are several threats to validity in our study, as discussed in the following.

The major threat to internal validity is about the implementations of ART-c and ART-s. All the programming work for implementing ART-c and ART-s was conducted on the basis of the source code in our previous studies [11, 24]. Only a small part of the code was modified, such as changing the termination condition, eliminating the failure-causing test cases from the executed set, and so on. All the code has also been checked by independent programmers. We are confident that ART-c and ART-s were correctly implemented.

There are two main threats to external validity. One is about the experimental settings of simulations. Although we have considered various situations, such as different dimensions and various failure patterns, it is very difficult, if not impossible, to fully simulate real-life situations. The other threat to external validity is the selection of subject programs and the seeded faults. We selected four programs and injected a number of faults that are associated with various failure patterns. However, as discussed in Section 5.1, these fault-seeded programs are still insufficient to provide a comprehensive picture on how F^n-measures of ART-c and ART-s are correlated with various factors. Even if we used both simulations and empirical studies, we cannot say that ART-c and ART-s will have similar performances on other real-life programs.

The threat to construct validity involves the measurement. In our study, we used the F^n-measure to evaluate and compare the testing effectiveness of ART-c, ART-s, and RT. As discussed in Section 3, the F^n-measure is more appropriate than the F-measure for our study. There also exist other testing effectiveness metrics, such as E-measure (the expected number of failures detected

Table XIII. Binomial tests on $F^n_{\text{ART-c}}$ and F^n_{RT} under the alternate testing and debugging scenario.

	`bessj0`	`bessj`	`plgndr`	`select`
p-value	0.001953	0.001953	0.001953	0.003906
Conclusion	Reject H_0	Reject H_0	Reject H_0	Reject H_0

Table XIV. Binomial tests on $F^n_{\text{ART-s}}$ and F^n_{RT} under the alternate testing and debugging scenario.

	`bessj0`	`bessj`	`plgndr`	`select`
p-value	0.001953	0.001953	0.001953	0.003906
Conclusion	Reject H_0	Reject H_0	Reject H_0	Reject H_0

DOI: 10.1002/spe

by a set of test cases) and P-measure (the probability of at least one failure being detected by a set of test cases). Immediately from their definitions, neither P-measure nor E-measure depends on how test cases are sequenced, whereas the ordering of test cases has an impact on the value of F^n-measure (as well as F-measure). In other words, like the F-measure, the F^n-measure is much more suitable than P-measure and E-measure to reflect the impact of various test sequences on the failure-detection effectiveness of *adaptive testing strategies* and thus more appropriate to evaluate and compare the effectiveness of ART and RT. With that said, it is still worthwhile to conduct the research on other testing effectiveness metrics, which however is out of the scope of the paper and should be part of the future work.

There is little threat to conclusion validity in our study. A large number of experimental trails have been conducted for each simulation run as well as each subject program. These trials have resulted in sufficiently huge experimental data, which helped us obtain statistically reliable results on the F^n-measures of ART-c and ART-s. The binomial tests conducted in Section 5 also confirmed that the improvement of ART-c/ART-s over RT in terms of the F^n-measure is statistically significant.

7. CONCLUSION AND FUTURE WORK

ART is an enhancement of RT. Previous studies always examined and compared the testing effectiveness of ART and RT under a particular testing and debugging scenario that once a failure is detected, testing is terminated and debugging is conducted immediately. Under this scenario, it is therefore appropriate to measure the failure-detection effectiveness of ART by means of the expected number of test cases to detect the first failure (F-measure). However, in many practical situations, especially in the early stage of software development, there is normally more than one fault, and thus testing should not be terminated after the detection of one failure. Moreover, on some occasions, debugging can be very expensive in time and labor; hence, it may be more economical to conduct testing and debugging in parallel. Therefore, in this paper, we investigated the performance of ART under various testing and debugging scenarios. More specifically, we evaluate and compare the testing effectiveness of ART and RT under the parallel testing and debugging scenario and the alternate testing and debugging scenario. Multiple failures are assumed under these scenarios. Because the F-measure cannot directly reflect the testing effectiveness of ART under these scenarios, we used a new metric, namely F^n-measure, which refers to the expected number of test cases to detect the first n ($n \geqslant 1$) failures. Through a series of experiments, we have answered the research question stated at the beginning of the paper that ART normally uses fewer test cases than RT to detect the same number of failures under various scenarios of testing and debugging.

In this paper, we have also investigated various strategies of implementing ART in the context of detection of multiple failures. Our investigations showed that ART-c does not significantly outperform RT under the parallel testing and debugging scenario, but ART-c significantly outperforms RT under the alternate testing and debugging scenario. On the other hand, ART-s is more effective than RT in terms of the F^n-measure under both scenarios. Under the alternate testing and debugging scenario, ART-s and ART-c have similar F^n-measures if failure regions are randomly distributed inside the input domain; however, if failure regions tend to be overlapping or adjacent to each other, ART-s outperforms ART-c. Such results imply that it is better to use ART-s rather than ART-c, that is, it is better to forget the failure-causing test cases when selecting the next test case in ART. Such a conclusion is consistent with the intuition of ART and the contiguity of failure regions

Our experimental results highlight a number of potential projects. The difference between the effectiveness of ART-c and ART-s triggers the investigation on how to make use of failure-causing and non-failure-causing test cases in further testing (not only for ART but also for all adaptive testing strategies). Interesting and useful results have been obtained about the relationship between the testing effectiveness of ART and the relative locations of different failure regions. Chen and Merkel [23] have proven that the upper bound of the F-measure of a testing method is 50% of F_{RT}. It is worthwhile to further investigate the upper bound of the F^n-measure of a testing method. As a pilot study, we only analyzed the multiple-failure-detection effectiveness of one particular ART algorithm, namely FSCS-ART. There are many ART algorithms in the literature [12, 16, 17], and their

DOI: 10.1002/spe

F-measures are quite different under various situations. Further work can be conducted to examine and compare the F^n-measures of these algorithms. There is another technique of 'forgetting' some already executed test cases [18], which is designed to reduce the computation overhead of ART. It should be noted that the intuition of such a forgetting technique is different from the intuition of selective memory in ART-s. The investigation on how to integrate these two techniques will help us develop a new and more cost-effective ART algorithm. Another important future work is to evaluate and compare the F^n-measures of ART-c, ART-s, and RT on various types of programs, especially those with non-numeric program inputs. There are studies [31–34] on how to reduce the F-measure of ART when the dimension of input domain is high. Although solving the high-dimension problem of ART is out of the scope of our paper, it is worthwhile to further investigate the F^n-measures of various ART algorithms for high-dimensional cases.

REFERENCES

1. Myers GJ. *The Art of Software Testing*, 2nd edn. John Wiley and Sons: Hoboken, 2004. Revised and Updated by T. Badgett and T. M. Thomas with C. Sandler.
2. Menzies T, Cukic B. When to test less. *IEEE Software* 2000; **17**(5):107–112.
3. Ammann PE, Knight JC. Data diversity: An approach to software fault tolerance. *IEEE Transactions on Computers* 1988; **37**(4):418–425.
4. Bishop PG. The variation of software survival times for different operational input profiles. *Proceedings of the 23rd International Symposium on Fault-Tolerant Computing (FTCS-23)*, 1993; 98–107.
5. van der Meulen M, Bishop P, Revilla M. An exploration of software faults and failure behaviour in a large population of programs. *Proceedings of the 15th International Symposium on Software Reliability Engineering (ISSRE 2004)*, 2004; 101–112.
6. White LJ, Cohen EI. A domain strategy for computer program testing. *IEEE Transactions on Software Engineering* 1980; **6**(3):247–257.
7. Chen TY, Tse TH, Yu YT. Proportional sampling strategy: A compendium and some insights. *The Journal of Systems and Software* 2001; **58**(1):65–81.
8. Ciupa I, Leitner A, Oriol M, Meyer B. ARTOO: Adaptive random testing for object-oriented software. *Proceedings of the 30th International Conference on Software Engineering (ICSE 2008)*, 2008; 71–80.
9. Jiang B, Zhang Z, Chan W, Tse T. Adaptive random test case prioritization. *Proceedings of the 24th International Conference on Automated Software Engineering (ASE 2009)*, 2009; 233–244.
10. Lin Y, Tang X, Chen Y, Zhao J. A divergence-oriented approach to adaptive random testing of Java programs. *Proceedings of the 24th International Conference on Automated Software Engineering (ASE 2009)*, 2009; 221–232.
11. Chen TY, Leung H, Mak IK. Adaptive random testing. *Proceedings of the 9th Asian Computing Science Conference, Lecture Notes in Computer Science,* vol. 3321, 2004; 320–329.
12. Chan KP, Chen TY, Towey D. Restricted random testing: Adaptive random testing by exclusion. *International Journal of Software Engineering and Knowledge Engineering* 2006; **16**(4):553–584.
13. Liu Y, Zhu H. An experimental evaluation of the reliability of adaptive random testing methods. *Proceedings of the 2nd IEEE International Conference on Secure System Integration and Reliability Improvement (SSIRI 2008)*, 2008; 24–31.
14. Weyuker EJ, Jeng B. Analyzing partition testing strategies. *IEEE Transactions on Software Engineering* 1991; **17**(7):703–711.
15. Gutjahr WJ. Partition testing vs. random testing: The influence of uncertainty. *IEEE Transactions on Software Engineering* 1999; **25**(5):661–674.
16. Chen TY, Eddy G, Merkel RG, Wong PK. Adaptive random testing through dynamic partitioning. *Proceedings of the 4th International Conference on Quality Software (QSIC 2004)*, 2004; 79–86.
17. Mayer J. Lattice-based adaptive random testing. *Proceedings of the 20th International Conference on Automated Software Engineering (ASE 2005)*, 2005; 333–336.
18. Chan KP, Chen TY, Towey D. Forgetting test cases. *Proceedings of the 30th Annual International Computer Software and Applications Conference (COMPSAC 2006)*, 2006; 485–492.
19. Chen TY, Kuo FC. Is adaptive random testing really better than random testing. *Proceedings of the 1st International Workshop on Random Testing (RT 2006)*, 2006; 64–69.
20. Chen TY, Kuo FC, Liu H, Wong WE. Does adaptive random testing deliver a higher confidence than random testing? *Proceedings of the 8th International Conference on Quality Software (QSIC 2008)*, 2008; 145–154.
21. Kuo FC. On adaptive random testing. *PhD Thesis*, Faculty of Information and Communications Technologies, Swinburne University of Technology, 2006.
22. Merkel RG. Analysis and enhancements of adaptive random testing. *PhD Thesis*, School of Information Technology, Swinburne University of Technology, 2005.
23. Chen TY, Merkel R. An upper bound on software testing effectiveness. *ACM Transactions on Software Engineering and Methodology* 2008; **17**(3):16:1–16:27.

DOI: 10.1002/spe

24. Chen TY, Kuo FC, Zhou ZQ. On favorable conditions for adaptive random testing. *International Journal of Software Engineering and Knowledge Engineering* 2007; **17**(6):805–825.
25. Tijms H. *Understanding Probability: Chance Rules in Everyday Life*. Cambridge University Press: Cambridge, 2004.
26. Chen TY, Kuo FC, Merkel R. On the statistical properties of testing effectiveness measures. *Journal of Systems and Software* 2006; **79**(5):591–601.
27. Press WH, Flannery BP, Teukolsky SA, Vetterling WT. *Numerical Recipes in C: The Art of Scientific Computing*. Cambridge University Press: Cambridge, 1992.
28. DeMillo RA, Lipton RJ, Sayward FG. Hints on test data selection: Help for the practicing programmer. *IEEE Computer* 1978; **11**(4):31–41.
29. Ellims M, Ince D, Petre M. The Csaw C mutation tool: Initial results. *Proceedings of the Testing: Academic and Industrial Conference Practice and Research Techniques—MUTATION (TAICPART-MUTATION 2007)*, 2007; 185–192.
30. Motulsky H. *Intuitive Biostatistics*. Oxford University Press: Oxford, 1995.
31. Kuo FC, Chen TY, Liu H, Chan WK. Enhancing adaptive random testing for programs with high dimensional input domains or failure-unrelated parameters. *Software Quality Journal* 2008; **16**(3):303–327.
32. Kuo FC, Sim KY, Sun C, Tang SF, Zhou ZQ. Enhanced random testing for programs with high dimensional input domains. *Proceedings of the 19th International Conference on Software Engineering and Knowledge Engineering (SEKE 2007)*, 2007; 135–140.
33. Mayer J. Adaptive random testing with randomly translated failure region. *Proceedings of the 1st International Workshop on Random Testing (RT 2006)*, 2006; 70–77.
34. Mayer J, Schneckenburger C. Adaptive random testing with enlarged input domain. *Proceedings of the 6th International Conference on Quality Software (QSIC 2006)*, 2006; 251–258.

DOI: 10.1002/spe