Hindawi Publishing Corporation
The Scientific World Journal
Volume 2014, Article ID 843248, 16 pages
http://dx.doi.org/10.1155/2014/843248

Research Article

Adaptive Random Testing with Combinatorial Input Domain

Rubing Huang,[1,2] Jinfu Chen,[1] and Yansheng Lu[2]

[1] *School of Computer Science and Telecommunication Engineering, Jiangsu University, 301 Xuefu Road, Zhenjiang, Jiangsu 212013, China*
[2] *School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan, Hubei 430074, China*

Correspondence should be addressed to Rubing Huang; rbhuang@ujs.edu.cn

Received 15 July 2013; Accepted 21 January 2014; Published 19 March 2014

Academic Editors: P. Peris-Lopez and G. A. Trunfio

Random testing (RT) is a fundamental testing technique to assess software reliability, by simply selecting test cases in a random manner from the whole input domain. As an enhancement of RT, adaptive random testing (ART) has better failure–detection capability and has been widely applied in different scenarios, such as numerical programs, some object–oriented programs, and mobile applications. However, not much work has been done on the effectiveness of ART for the programs with combinatorial input domain (i.e., the set of categorical data). To extend the ideas to the testing for combinatorial input domain, we have adopted different similarity measures that are widely used for categorical data in data mining and have proposed two similarity measures based on interaction coverage. Then, we propose a new version named ART–CID as an extension of ART in combinatorial input domain, which selects an element from categorical data as the next test case such that it has the lowest similarity against already generated test cases. Experimental results show that ART–CID generally performs better than RT, with respect to different evaluation metrics.

1. Introduction

Software testing, a major software engineering activity, is widely considered to assure the quality of software under test [1]. Many testing methods have been developed to effectively identify software failures by actively selecting inputs (namely, test cases). Random testing (RT), a basic software testing method, simply chooses test cases at random from the set of all possible program inputs (namely, the input domain) [2, 3]. There are many advantages of using RT in software testing. For example, in addition to simplicity and the efficiency of generating random test cases [2], RT allows statistical quantitative estimation of software's reliability [4]. Due to these advantages, RT has been widely used to detect software failures in different scenarios, such as the testing of UNIX utilities [5, 6], SQL database systems [7, 8], Java JIT compilers [9], and embedded software systems [10]. In spite of the popularity, RT is still criticized by many researchers due to little or no information to guide its test case generation.

Given a faulty program, two basic features are determined by program inputs causing software to exhibit failure behaviors (namely, failure-causing inputs), that is, failure rate θ and failure pattern. Failure rate refers to the ratio between the number of failure-causing inputs and the number of all possible program inputs, while failure pattern refers to the geometry and distribution of failure regions (i.e., the region where failure-causing inputs reside). It has been observed, however, that failure-causing inputs tend to cluster together [11–13]. Given that failure regions are continuous, nonfailure regions should also be contiguous. More specifically, suppose a test case (tc) is not a failure-causing input, test cases that are close to tc (or tc's neighbors) may fail to reveal a failure as well. Therefore, it is intuitively appealing that test cases that spread away from tc may have a higher chance to be failure-causing than tc's neighbors.

Briefly speaking, it is very likely that a more even-spread of random test cases can improve the failure-detection effectiveness of RT. Based on this intuition, Chen et al. [14] have proposed a novel approach, namely, adaptive random testing (ART). Similar to RT, ART also randomly generates test case from the whole input domain. But ART uses additional criteria to guide the test case selection for the purpose of evenly spreading test cases over the input domain. Various ART algorithms have been developed based on different test

case selection criteria, such as ART by distance [15], ART by exclusion [16], ART based on evolutionary search algorithms [17], and ART by perturbation [18]. Essentially, ART achieves test case diversity with the subset of test cases executed at any one time [19].

As an alternative of RT, ART has been successfully applied to different programs, such as numerical programs [15–18], object-oriented programs [20, 21], and mobile application [22]. However, not much work has been done on the effectiveness of ART for programs with combinatorial input domain (or categorical data, i.e., a Cartesian product of finite value domains for each of a finite set of parameter variables). With the popularity of category-partition method [23] and many guidelines to help construct categories and partitions [24–27], combinatorial input domain has been widely applied to different testing scenarios, such as configurable-aware system [28, 29], event-driven software [30], and GUI-based application [31]. In this paper, we propose a new testing strategy called ART-CID as an extension of ART in combinatorial input domain. In order to successfully extend the ART principle into combinatorial input domain, we propose two similarity measures based on interaction coverage and also adopt different well-studied similarity measures that are popularly used for categorical data in data mining [32]. To analyze the effectiveness of ART-CID (mainly FSCS-CID, one version of ART-CID), we compare the effectiveness of FSCS-CID with RT by designing some simulations and the empirical study. Experimental results show that, compared with RT, FSCS-CID can not only use smaller test cases in order to cover all possible combinations of parameter values at a given strength, but also require to generate fewer test cases to identify the first failure in the real-life program.

This paper is organized as follows. Section 2 introduces some preliminaries, including combinatorial input domain, ART, similarity measures used for combinatorial input domain, and the effectiveness measures adopted in our study. Section 3 proposes two similarity measures for combinatorial test cases based on interaction coverage. Section 4 proposes a new algorithm called ART-CID to select test cases from combinatorial input domain. Section 5 reports some experimental studies, which examine the rate of covering value combinations at a given strength and failure-detection effectiveness of our new method. Finally, Section 6 summarizes some discussions and conclusions.

2. Preliminaries

In the following section, some preliminaries of combinatorial input domain, failure patterns, adaptive random testing, similarity and dissimilarity measures for combinatorial input domain, and effectiveness measure are described.

2.1. Combinatorial Input Domain. Suppose that a system under test (SUT) has a set of k parameters (or categories) $P = \{p_1, p_2, \ldots, p_k\}$, which may represent user inputs, configuration parameters, internal events, and so forth. Let V_i be the finite set of discrete valid values (or choices) for p_i $(i = 1, 2, \ldots, k)$, and let C be the set of constraints on parameter value combinations. Without loss of generality, we assume that the order of parameters is fixed; that is, $P = \langle p_1, p_2, \ldots, p_k \rangle$. In the remainder of this paper, we will refer to a *combination of parameters* as a *parameter interaction*, and a *combination of parameter values* or a *parameter value combination* as a *value combination*.

Definition 1. A test profile, denoted as $\mathrm{TP}(k, |V_1||V_2| \cdots |V_k|, C)$, is about the information on a combinatorial input domain of the SUT, including k parameters, $|V_i|$ $(i = 1, 2, \ldots, k)$ values for parameter p_i, and constraints C on value combinations.

In this paper, we assume that all the parameters are independent; that is, no constraint among value combinations is considered ($C = \emptyset$), unless otherwise specified. Therefore, the test profile can be abbreviated as $\mathrm{TP}(k, |V_1||V_2| \cdots |V_k|)$.

To clearly describe some notions and definitions, we present an example of the part of suboptions in an option "View" of the tool PDF shown in Table 1. In this system, there are four configuration parameters, each of which has three values. Therefore, its test profile can be written as $\mathrm{TP}(4, 3^4)$.

Definition 2. Given a $\mathrm{TP}(k, |V_1||V_2| \cdots |V_k|)$, a test case or a test configuration is a k-tuple $(v_1, v_2, \ldots, v_k)$ where $v_i \in V_i$ $(i = 1, 2, \ldots, k)$.

Intuitively speaking, a combinatorial input domain $\mathrm{TP}(k, |V_1||V_2| \cdots |V_k|)$ is a Cartesian product of $\langle V_1, V_2, \ldots, V_k \rangle$ for each of P; that is, $T_{\mathrm{all}} = V_1 \times V_2 \times \cdots \times V_k$. Therefore, the size of all possible test cases is $|V_1| \times |V_2| \times \cdots \times |V_k|$. For example, a 4-tuple tc = (*Full Size*, *Single*, *Clockwise*, *All Pages*) is a test case for the SUT shown in Table 1.

Definition 3. Given a $\mathrm{TP}(k, |V_1||V_2| \cdots |V_k|)$, a τ-wise value combination is a k-tuple $(\widehat{v}_1, \widehat{v}_2, \ldots, \widehat{v}_k)$ involving τ parameters with fixed values (named fixed parameters) and $(k - \tau)$ parameters with arbitrary allowable values (named free parameters), where $0 \le \tau \le k$ and

$$\widehat{v}_i = \begin{cases} v_i \in V_i, & \text{if } p_i \text{ is a fixed parameter;} \\ \text{“}-\text{”}, & \text{if } p_i \text{ is a free parameter.} \end{cases} \quad (1)$$

Generally, τ-wise value combination is also called τ-value schema [33], and τ is called strength. When $\tau = k$, a τ-wise value combination becomes a test case for the SUT as it takes on a specific value for each of its parameters. For ease of description, we define a term $\mathrm{CombSet}_\tau(\mathrm{tc})$ as the set of τ-wise value combinations covered by the test case (tc). Intuitively speaking, a test case (tc) with k parameters contains C_k^τ τ-wise value combinations, that is, $|\mathrm{CombSet}_\tau(\mathrm{tc})| = C_k^\tau$.

For example, considering a test case (tc) = (*Full Size*, *Single*, *Clockwise*, *All Pages*), we can obtain that $\mathrm{CombSet}_1(\mathrm{tc})$ = {(*Full Size*, –, –, –), (–, *Single*, –, –), (–, –, *Clockwise*, –), (–, –, –, *All Pages*)}, while $\mathrm{CombSet}_3(\mathrm{tc})$ = {(*Full Size*, *Single*, *Clockwise*, –), (*Full Size*, *Single*, –, *All Pages*), (*Full Size*, –, *Clockwise*, *All Pages*), (–, *Single*, *Clockwise*, *All Pages*)}.

Table 1: Optional parameters and values of an option "View" in the tool PDF.

Parameters	p_1: Page size	p_2: Page layout	p_3: Rotating view	p_4: Reading
Values	*Full size*	*Single*	*Clockwise*	*Current Page*
	Right width	*Bisected*	*Anticlockwise*	*All pages*
	Right page	*Continuous*	*None*	*None*

Definition 4. The number of parameters required to trigger a failure is referred to as the failure-triggering fault interaction (FTFI) number.

As we know, the faulty model in the combinatorial input domain assumes that failures are caused by parameter interactions. For instance, if the SUT shown in Table 1 fails when p_2 is set to "*Single*", p_3 is set to "*None,*" and p_4 is not equal to "*None,*" this failure is caused by the parameter interaction (p_2, p_3, p_4). Therefore, the FTFI number of this fault is 3.

In [28, 34], Kuhn et al. investigated interaction failures by analyzing the faults reports of several software projects and concluded that failures are always caused by low FTFI numbers.

2.2. Failure Patterns. Given a faulty program, two basic features can be obtained from it. One feature is *failure rate*, denoted by θ, which refers to the ratio of the number of failure-causing inputs to the number of all possible inputs. The other feature is *failure pattern*, which refers to the geometric shapes and the distributions of the failure-causing regions. Both features are fixed but unknown to testers before testing.

In [14], the patterns of failure-causing inputs have been classified into three categories: *point pattern*, *stripe pattern,* and *block pattern*. An illustrative example about three types of failure patterns in a two-dimensional input domain is shown in Figure 1. In this example, suppose the input domain is consisting of parameters x and y where $0 \le x, y \le 10.0$. Point pattern means the tested program will fail when x and y are assigned to particular integers, that is, some specific points in the input domain, while strip pattern may be of the form $0 \le x \le 10.0$, $3.0 \le y \le 3.5$, and block pattern may be of the form $1.5 \le x, y \le 3.0$.

In the combinatorial input domain, failure patterns of any failures belong to the point pattern as all test inputs are discrete. However, from the perspective of functionality and computation of each test input, three failure patterns shown in Figure 1 also exist in the combinatorial input domain. For example, (1) if a failure f_1 in the SUT shown in the Table 1 is caused by "p_2 = *Single* or p_2 = *Continuous*" and "p_3 = *Clockwise* or p_3 = *None*", we believe that the failure pattern of f_1 is a strip pattern and its failure rate is $(2 \times 2)/(3 \times 3) = 0.4444$; (2) if a failure f_2 in the SUT is caused by "$p_1 \neq$ *Full Size*", "$p_2 \neq$ *Single*", "$p_3 \neq$ *None*", and "$p_4 \neq$ *All pages*", we believe that the failure pattern of f_2 is a block pattern and its failure rate is $(2/3)^4 = 0.1975$; and (3) if a failure f_3 is caused by a single test case (*Full Size*, *Single*, *Clockwise*, *None*), we believe that the failure region of f_3 is a point pattern and its failure rate is $1/81 = 0.0123$. According to Kuhn's investigations [28, 34], however, the FTFI numbers are always very low (i.e., the FTFI numbers are smaller than the number of parameters), which means that the strip pattern is the most frequent failure pattern in the combinatorial input domain.

2.3. Adaptive Random Testing (ART). The methodology of adaptive random testing (ART) [14, 15] has been proposed to enhance the failure-detection effectiveness of random testing (RT) by even-spreading test cases across the whole input domain. In ART, test cases are not only randomly generated, but also evenly spread. According to previous ART studies [15–22], ART was shown to reduce the number of test cases required to identify the first fault by as much as 50% over RT.

There are many implementations of ART by different notions. A simple algorithm is the fixed-size-candidate-set ART (FSCS-ART) [15]. FSCS-ART implements the notion of distance as follows. FSCS-ART uses two sets of test cases, namely, the executed set E and the candidate set C. E is a set of test cases that have been executed but without revealing any failure, while C is a set of tests that are randomly selected from the input domain according to the uniform distribution. E is initially empty and the first element is randomly chosen from the input domain and then incrementally updates with the selected elements from C until a failure is exhibited. From C, the element that is farthest away from all test cases in E is chosen as the next test case; that is, the criterion is to choose the element c' from C as the next test case such that

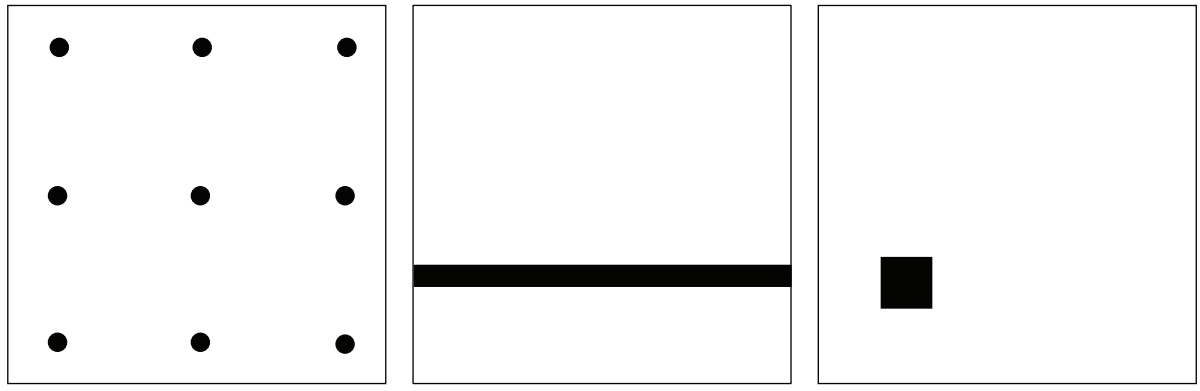
$$(\forall c)\,(c \in C)\left(c \neq c'\right)\left[\min_{e \in E} \operatorname{dist}\left(c', e\right) \geq \min_{e \in E} \operatorname{dist}\left(c, e\right)\right], \tag{2}$$

where dist is defined as the Euclidean distance, that is, in a m-dimensional input domain, for two test inputs, $\mathrm{tc}_1 = (a_1, a_2, \ldots, a_m)$ and $\mathrm{tc}_2 = (b_1, b_2, \ldots, b_m)$,

$$\operatorname{dist}\left(\mathrm{tc}_1, \mathrm{tc}_2\right) = \sqrt{\sum_{i=1}^{m} \left(a_i - b_i\right)^2}. \tag{3}$$

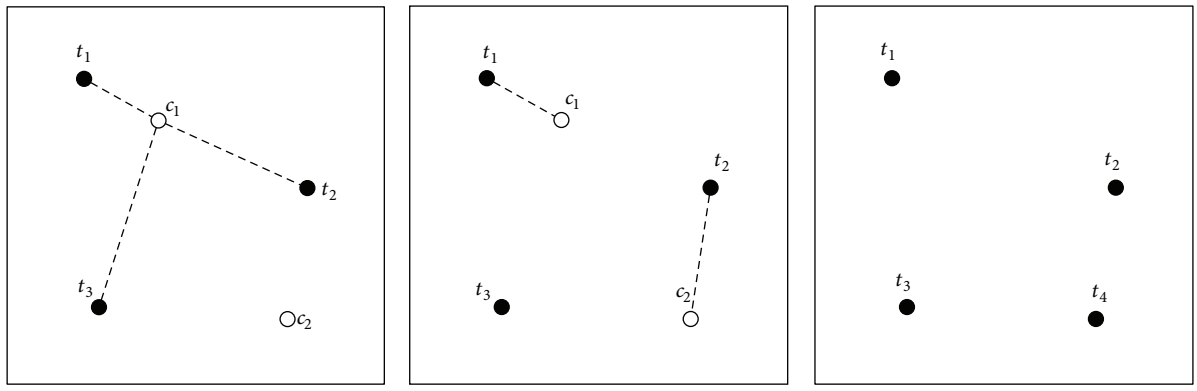
The process is repeated until the desired stopping criterion is satisfied.

Figure 2 gives the illustration of FSCS-ART in a two-dimensional input domain. In Figure 2(a), there are 3 previously executed test cases t_1, t_2, and t_3, and 2 randomly generated candidates c_1 and c_2. To choose among the candidates, the distance of each candidate against each previously executed test case is calculated. Figure 2(b) describes that the closest previously executed test case is determined for each candidate. In Figure 2(c), the candidate c_2 is selected as the next test case (i.e., $t_4 = c_2$), as the distance of c_2 against its nearest previously executed test case is larger than that of the candidate c_1.



(a) Point pattern (b) Strip pattern (c) Block pattern

Figure 1: Three types of failure patterns with two-dimensional input domain.



(a) (b) (c)

Figure 2: Illustration of FSCS-ART in a two-dimensional input domain. Previously executed test cases are denoted as t_1, t_2, and t_3, and randomly generated candidates are denoted as c_1 and c_2, respectively. To select the next test case, (a) multiple candidates are randomly selected, and the nearest previously executed test case to each candidate is determined; (b) these nearest distances are compared among all candidates; and (c) the candidate with the longest such distance is chosen.

In this paper, we emphasize the extension of FSCS-ART as that of ART in combinatorial input domain, unless otherwise specified.

2.4. Similarity and Dissimilarity Measures for Combinatorial Input Domain. Measuring similarity or dissimilarity (distance) between two test inputs is a core requirement for test case selection, evaluation, and generation. Generally speaking, in numerical input domains, Euclidean distance (see (3)) is a mostly used distance measure for continuous data. However, for a combinatorial input domain, since its parameters and corresponding values are finite and discrete, Euclidean distance may not be available and reasonable. Nevertheless, various distance measures (or dissimilarity measures) are popularly used in data mining for evaluating categorical data [32], such as clustering (k-means), classification (KNN, SVM), and distance-based outlier detection. In this subsection, we simply describe the following measures that will be adopted in our paper later.

To illustrate our work clearly, let us define a few terms. Consider a categorical dataset T containing N objects, derived from a TP$(k, |V_1||V_2|\cdots|V_k|)$ for parameters $P_1, P_2, \ldots, P_k$. We also use the following notation.

(i) $f_i(x)$ is the number of times parameter P_i takes the value x in T. Note that if $x \notin V_i$, $f_i(x) = 0$.

(ii) $\widehat{p}_i(x)$ is the sample probability of parameter P_i to take the value x in T. The sample probability is given by

$$\widehat{p}_i(x) = \frac{f_i(x)}{N}. \tag{4}$$

(iii) $p_i^2(x)$ is another probability estimate of parameter P_i to take the value x in T and is given by

$$p_i^2(x) = \frac{f_i(x)\left(f_i(x) - 1\right)}{N(N-1)}. \tag{5}$$

(iv) $S(X, Y)$ is a generalized similarity measure between two data instances denoted as $X = (X_1, X_2, \ldots, X_k)$

and $Y = (Y_1, Y_2, \ldots, Y_k)$ where $X, Y \in T$, and $X_i, Y_i \in V_i$ $(i = 1, 2, \ldots, k)$. Its definition is given as follows:

$$S(X, Y) = \sum_{i=1}^{k} w_i S_i(X_i, Y_i), \quad (6)$$

where $S_i(X_i, Y_i)$ $(i = 1, 2, \ldots, k)$ is the per-parameter similarity between two values for parameter P_i and w_i denotes the weight assigned to the parameter P_i. Therefore, we only require to present the definitions of $S_i(X_i, Y_i)$ and w_i for each similarity measure, unless otherwise specified.

To directly refer to [32], the measures discussed henceforth will all be in the context of similarity, with dissimilarity or distance measures being converted using the following formula:

$$S(X, Y) = \frac{1}{1 + D(X, Y)}, \quad (7)$$

where $D(X, Y)$ is the dissimilarity measure between X and Y.

Table 2 presents nine similarity measures for categorical parameter values, which are widely used in data mining for categorical data. In Table 2, the last column "Range" represents the range of $S_i(X_i, Y_i)$ for mismatches or matches of parameter values in each measure.

2.5. Effectiveness Measurement. In this paper, we adopt the *F-measure* (i.e., the number of test cases required to detect the first failure) as the measurement of failure-detection effectiveness of testing methods, since previous studies [35] have demonstrated that the F-measure is particularly suitable for adaptive testing strategies such as ART. Intuitively speaking, a smaller F-measure of ART over RT means fewer test cases required by ART to detect the first failure and hence implies a better failure-detection effectiveness of ART than that of RT. For the purpose of clear description, we will use ART *F-ratio* (i.e., the ratio of ART's F-measure (F_{ART}) relative to RT's F-measure (F_{RT})) to indicate the failure-detection effectiveness improvement of ART over RT.

However, it is extremely difficult to theoretically obtain ART's F-measure (F_{ART}). Similar to all other ART studies, F_{ART} is collected via simulations and empirical studies, whose procedure is described as follows. On the one hand, in simulation studies, failure pattern (including its size and sharp) and failure rate θ are predefined for simulating a faulty program. The failure regions are then randomly placed inside the whole input domain. If a point inside one of the failure regions is picked by a testing strategy, a failure is said to be detected. On the other hand, for empirical studies, some faults are seeded into a subject program. Once the subject program behaves differently from its fault-seeded version, it is said that a failure is identified. The number of test cases to find the first failure is regarded as the F_{ART} of that run. Such a process runs $\mathcal{S}$ times repeatedly until a statistically reliable estimate of the F_{ART} (±5% accuracy rate and (1 − 5%) confidence level adopted in our paper) has been obtained. Refer to the value of $\mathcal{S}$; it can be determined dynamically using the same method as shown in [15]. With respect to RT's F-measure (F_{RT}), since test cases are chosen with replacement according to the uniform distribution, F_{RT} is equal to $1/\theta$ theoretically.

Apart from the F-measure used as the measurement, another measurement is also used in our paper, that is, the number of test cases required to first cover all possible value combinations of a given strength τ (denoted N_τ-measure). This measurement is widely used in the combinatorial input domain. Unlike the F-measure, the testing stop condition of N_τ-measure is not that the first failure is detected, but that all possible τ-wise value combinations are first covered. For the purpose of clear description, we use $N_{\mathrm{RT}}(\tau)$ to represent this measurement for RT while $N_{\mathrm{ART}}(\tau)$ for ART.

3. Two Similarity Measures Based on Interaction Coverage

Apart from various similarity measures described in Section 2.4, in this section, we propose another two similarity measures by using interaction coverage: incremental interaction coverage similarity (IICS) and multiple interaction coverage similarity (MICS), in order to apply the characteristics of combinatorial input domain to the selection of test cases. All similarity measures illustrated in Section 2.4 are used to evaluate how similar two test cases are; however, two similarity measures presented in this section are used to evaluate the resemblance of the combinatorial test case against the combinatorial test suite. We will discuss them next.

Before introducing them, we firstly describe a simple similarity measure of the test case against a test suite based on interaction coverage, named normalized covered τ-wise value combinations similarity (or NCVCS$_\tau$) [36], which is widely used in combinatorial input domain.

Definition 5. Given a combinatorial test suite T on TP(k, $|V_1||V_2|\cdots|V_k|$), a combinatorial test case (tc), and the strength τ, normalized covered τ-wise value combinations similarity (NCVCS$_\tau$) of tc against T is defined as the ratio of the number of τ-wise value combinations covered by tc that have already been covered by T to C_k^τ; that is,

$$\mathrm{NCVCS}_\tau(\mathrm{tc}, T) = \frac{\left|\mathrm{CombSet}_\tau(\mathrm{tc}) \bigcap \mathrm{CombSet}_\tau(T)\right|}{C_k^\tau}, \quad (8)$$

where CombSet$_\tau(T)$ can be written as follows:

$$\mathrm{CombSet}_\tau(T) = \bigcup_{\mathrm{tc} \in T} \mathrm{CombSet}_\tau(\mathrm{tc}). \quad (9)$$

Obviously, the NCVCS$_\tau$ is a function that requires to set the strength value τ in advance, and its range is [0, 1.0]. Two properties of the NCVCS$_\tau$ are discussed as follows.

Theorem 6. *If* $NCVCS_\tau(tc, T) = 1.0$, $NCVCS_\lambda(tc, T) = 1.0$, *where* $1 \le \lambda < \tau \le k$.

Proof. When NCVCS$_\tau$(tc, T) = 1.0, it can be noted that T covers all possible τ-wise value combinations covered by tc, that is,

$$\mathrm{CombSet}_\tau(\mathrm{tc}) \subseteq \mathrm{CombSet}_\tau(T). \quad (10)$$

Table 2: Similarity measures for categorical parameter values.

Measure	$S_i(X_i, Y_i)$	ω_i, $i = 1, 2, \ldots, k$	Range
Goodall1	$= \begin{cases} 1 - \sum_{q \in Q} p_i^2(q) & \text{if } X_i = Y_i \\ 0 & \text{otherwise} \end{cases}$	$\frac{1}{k}$	$\left[0, 1 - \frac{2}{N^2}\right]$ $[0, 0]$
Goodall2	$= \begin{cases} 1 - \sum_{q \in Q} p_i^2(q) & \text{if } X_i = Y_i \\ 0 & \text{otherwise} \end{cases}$	$\frac{1}{k}$	$\left[1 - \frac{2}{N^2}, 1\right]$ $[0, 0]$
Goodall3	$= \begin{cases} 1 - \sum p_i^2(X_i) & \text{if } X_i = Y_i \\ 0 & \text{otherwise} \end{cases}$	$\frac{1}{k}$	$\left[0, 1 - \frac{1}{N^2}\right]$ $[0, 0]$
Goodall4	$= \begin{cases} \sum p_i^2(X_i) & \text{if } X_i = Y_i \\ 0 & \text{otherwise} \end{cases}$	$\frac{1}{k}$	$[\frac{1}{N^2}, 1]$ $[0, 0]$
Lin	$= \begin{cases} 2 \log \widehat{p}_i(X_i) & \text{if } X_i = Y_i \\ 2 \log (\widehat{p}_i(X_i) + \widehat{p}_i(Y_i)) & \text{otherwise} \end{cases}$	$\frac{1}{\sum_{i=1}^{k} \log \widehat{p}_i(X_i) + \log \widehat{p}_i(Y_i)}$	$[-2 \log N, 0]$ $\left[-2 \log \frac{N}{2}, 0\right]$
Lin1	$= \begin{cases} \sum_{q \in Q} \log \widehat{p}_k(q) & \text{if } X_i = Y_i \\ 2 \log \sum_{q \in Q} \widehat{p}_k(q) & \text{otherwise} \end{cases}$	$\frac{1}{\sum_{i=1}^{k} \sum_{q \in Q} \log \widehat{p}_i(X_i)}$	$[-N \log N, 0]$ $[-2 \log \frac{N}{2}, 0]$
Overlap	$= \begin{cases} 1 & \text{if } X_i = Y_i \\ 0 & \text{otherwise} \end{cases}$	$\frac{1}{k}$	$[1, 1]$ $[0, 0]$
Eskin	$= \begin{cases} 1 & \text{if } X_i = Y_i \\ \frac{\lvert V_i \rvert^2}{\lvert V_i \rvert^2 + 2} & \text{otherwise} \end{cases}$	$\frac{1}{k}$	$[1, 1]$ $\left[\frac{2}{3}, \frac{N^2}{N^2 + 2}\right]$
OF	$= \begin{cases} 1 & \text{if } X_i = Y_i \\ \frac{1}{1 + \log (N / f_i(X_i)) \times \log (N / f_i(Y_i))} & \text{otherwise} \end{cases}$	$\frac{1}{k}$	$[1, 1]$ $\left[\frac{1}{1 + (\log N)^2}, \frac{1}{1 + (\log 2)^2}\right]$

Note. For measure *Goodall1*, $Q = \{q \mid q \in V_i, p_i(q) \geq p_i(X_i)\}$.
For measure *Goodall2*, $Q = \{q \mid q \in V_i, p_i(q) \leq p_i(X_i)\}$.
For measure *Lin1*, $Q = \{q \mid q \in V_i, \min(\widehat{p}_i(X_i), \widehat{p}_i(Y_i)) \leq \widehat{p}_i(q) \leq \max(\widehat{p}_i(X_i), \widehat{p}_i(Y_i))\}$.

Since

$$\begin{aligned} \text{CombSet}_\lambda(\text{tc}) &= \text{CombSet}_\lambda\left(\text{CombSet}_\tau(\text{tc})\right) \\ &\subseteq \text{CombSet}_\lambda\left(\text{CombSet}_\tau(T)\right) \\ &= \text{CombSet}_\lambda(T) \quad (1 \leq \lambda < \tau \leq k), \end{aligned} \tag{11}$$

T also covers all possible value combinations at strengths lower than τ that are covered by tc. As a consequence, $\text{NCVCS}_\lambda(\text{tc}, T) = 1.0$ where $1 \leq \lambda < \tau \leq k$. □

Theorem 7. *If* $NCVCS_\lambda(tc, T) = 0$, $NCVCS_\tau(tc, T) = 0$ *where* $1 \leq \lambda < \tau \leq k$.

Proof. When $\text{NCVCS}_\lambda(\text{tc}, T) = 0$, it can be noted that each λ-wise value combination covered by tc is not covered by T, indicating that, for $\forall e' \in \text{CombSet}_\lambda(\text{tc})$, $e' \notin \text{CombSet}_\lambda(T)$: that is,

$$\text{CombSet}_\lambda(\text{tc}) \bigcap \text{CombSet}_\lambda(T) = \emptyset. \tag{12}$$

Therefore, the problem converts to demonstrating that $\forall e \in \text{CombSet}_\tau(\text{tc})$, $e \notin \text{CombSet}_\tau(T)$.

We suppose that $\exists e$ such that $e \in \text{CombSet}_\tau(\text{tc})$ and $e \in \text{CombSet}_\tau(T)$, that is,

$$\begin{aligned} &\text{CombSet}_\lambda\left(\text{CombSet}_\tau(\text{tc})\right) \\ &\bigcap \text{CombSet}_\lambda\left(\text{CombSet}_\tau(T)\right) \neq \emptyset. \end{aligned} \tag{13}$$

Due to $\lambda < \tau$, (13) is equivalent to the equation shown as follows:

$$\text{CombSet}_\lambda(\text{tc}) \bigcap \text{CombSet}_\lambda(T) \neq \emptyset. \tag{14}$$

Obviously, (14) is contradictory to (12). Therefore, $\forall e \in \text{CombSet}_\tau(\text{tc})$, $e \notin \text{CombSet}_\tau(T)$, which means that $\text{NCVCS}_\tau(\text{tc}, T) = 0$ where $1 \leq \lambda < \tau \leq k$. □

As we know, given a $\text{TP}(k, |V_1||V_2| \cdots |V_k|)$ and the strength τ, the number of all possible τ-wise value combinations is fixed; that is, $|\text{CombSet}_\tau(T_{\text{all}})| = \sum_{i_1=1}^{k-\tau} \cdots \sum_{i_\tau = i_{\tau-1}+1}^{k} (|V_{i_1}| \cdots |V_{i_\tau}|)$. In other words, there exists a test case generation method using NCVCS_τ as the criterion, which can generate a certain number of combinatorial test cases denoted

as T ($T \subseteq T_{\text{all}}$) to cover all possible τ-wise value combinations. However, if testing with T fails to reveal any failures due to no failure-causing inputs in T, the next test case generated by this method is, in fact, obtained in a random manner. The main reason is that the NCVCS_τ of each element in T_{all} is equal to 1.0. Therefore, the NCVCS_τ is not particularly suitable for adaptive testing strategies such as ART. To solve this problem, we propose two similarity measures based on interaction coverage in the following subsections.

3.1. Incremental Interaction Coverage Similarity. As discussed in Theorem 6, if all possible τ-wise value combinations are covered by a combinatorial test suite T, all possible value combinations at strengths lower than τ are also covered by T. According to this fact, we present a new similarity measure based on interaction coverage, named incremental interaction coverage similarity (IICS).

Given a combinatorial test suite T on $\text{TP}(k, |V_1||V_2| \cdots |V_k|)$ and a combinatorial test case (tc), the incremental interaction coverage similarity of tc against T is defined as follows:

$$\text{IICS}(\text{tc}, T) = \begin{cases} 1.0, & \text{if tc} \in T; \\ \text{NCVCS}_\tau(\text{tc}, T), & \text{if tc} \notin T, \end{cases} \tag{15}$$

where τ satisfies the following properties: (1) $\text{CombSet}_{\tau-1}(T) = \text{CombSet}_{\tau-1}(T_{\text{all}})$ and (2) $\text{CombSet}_\tau(T) < \text{ComSet}_\tau(T_{\text{all}})$, where $1 \le \tau \le k$ (assume $\text{CombSet}_0(T) \equiv \text{CombSet}_0(T_{\text{all}})$).

It can be noted that if tc $\in T$, the IICS is equal to 1.0 as tc is the same as one of elements in T; if tc $\notin T$, the IICS of tc against T is actually equal to the NCVCS_τ of tc against T where τ is gradually incremented. More specifically, if T covers all possible i-wise value combinations and partial $(i+1)$-wise value combinations occurred in tc, $\tau = i + 1$. Similar to NCVCS_τ, the range of IICS is also [0, 1.0].

Here, we present an example to illustrate IICS. Suppose $T = \{(0,0,0),(0,1,1)\}$ on $\text{TP}(3, 2^3)$, $\text{tc}_1 = (1,0,1)$, and $\text{tc}_2 = (1,1,0)$, $\text{IICS}(\text{tc}_1, T) = \text{NCVCS}_1(\text{tc}_1, T) = 2/C_3^1 = 0.67$ as 1-wise value combinations are not completely covered by T, and hence $\tau = 1$. Let $T = T \bigcup \{\text{tc}_1\}$, $\text{IICS}(\text{tc}_2, T) = \text{NCVCS}_2(\text{tc}_2, T) = 0/C_3^2 = 0$ as T covers all 1-wise value combinations and partial 2-wise value combinations occurred in tc_2, and hence $\tau = 2$.

3.2. Multiple Interaction Coverage Similarity. As shown in Section 3.1, the IICS measure begins at strength $\tau = 1$, and then update the value of τ by $(\tau + 1)$. In other words, it considers different strength values when evaluating the combinatorial test case against the combinatorial test suite. However, the IICS accounts for each strength value at each time rather than simultaneously considering all strength values. As a consequence, we present another similarity measure based on interaction coverage, named multiple interaction coverage similarity (MICS).

Given a combinatorial test suite T on $\text{TP}(k, |V_1||V_2| \cdots |V_k|)$ and a combinatorial test case (tc), the weighted interaction coverage similarity of tc against T is defined as follows:

$$\text{MICS}(\text{tc}, T) = \sum_{\tau=1}^{k} \left(\omega_\tau * \text{NCVCS}_\tau(\text{tc}, T)\right), \tag{16}$$

where $0 \le \omega_\tau \le 1.0$ and $\sum_{\tau=1}^{k} \omega_\tau = 1.0$.

Intuitively speaking, if tc $\in T$, $\text{MICS}(\text{tc}, T) = 1.0$. Similar to IICS, the MICS ranges from 0 to 1.0.

Here, we present an example to explain the definition of MICS. Let $T = \{(0,0,0,0),(0,1,1,1)\}$ on $\text{TP}(4, 2^4)$, $\text{tc}_1 = (1,0,1,0)$, $\text{tc}_2 = (0,1,1,0)$, and $\omega_1 = \omega_2 = \omega_3 = \omega_4$, $\text{MICS}(\text{tc}_1, T) = (1/4) \times \sum_{\tau=1}^{4} \text{NCVCS}_\tau(\text{tc}_1, T) = (1/4) \times (1/C_4^1 + 5/C_4^2 + 3/C_4^3 + 1/C_4^4) = 0.71$, while $\text{MICS}(\text{tc}_2, T) = (1/4) \times \sum_{\tau=1}^{4} \text{CVCS}_\tau(\text{tc}_2, T) = (1/4) \times (0/C_4^1 + 2/C_4^2 + 2/C_4^3 + 1/C_4^4) = 0.46$.

3.3. Properties of Two New Similarity Measures. Some properties of the proposed two similarity measures are discussed in the following subsection.

Theorem 8. *If $\text{CombSet}_{k-1}(T) = \text{CombSet}_{k-1}(T_{all})$, for $\forall \text{tc} \notin T$, $\text{IICS}(\text{tc}, T)$ and $\text{MICS}(\text{tc}, T)$ remain unchanged.*

Proof. On the one hand, if $\text{CombSet}_{k-1}(T) = \text{CombSet}_{k-1}(T_{\text{all}})$ (i.e., T covers all possible $(k-1)$-wise value combinations), for $\forall \text{tc} \notin T$,

$$\begin{aligned} \text{IICS}(\text{tc}, T) &= \text{NCVCS}_k(\text{tc}, T) \\ &= \frac{\left|\text{CombSet}_k(\text{tc}) \bigcap \text{CombSet}_k(T)\right|}{C_k^k} \\ &= \frac{\left|\{\text{tc}\} \bigcap T\right|}{C_k^k} \\ &= 0. \end{aligned} \tag{17}$$

On the other hand, if $\text{CombSet}_{k-1}(T) = \text{CombSet}_{k-1}(T_{\text{all}})$ (i.e., T covers all possible $(k-1)$-wise value combinations), $\forall \text{tc} \in T_{\text{all}}$, $\text{NCVCS}_{k-1}(\text{tc}, T) = 1.0$. According to Theorem 6, it can be concluded that $\text{NCVCS}_\tau(\text{tc}, T) = 1.0$, where $1 \le \tau < k-1$; that is, T covers all possible τ-wise value combinations. In other words, $\text{CombSet}_\tau(T) = \text{CombSet}_\tau(T_{\text{all}})$ $(1 \le \tau < k-1)$. Therefore, for $\forall \text{tc} \notin T$,

$$\begin{aligned} \text{MICS}(\text{tc}, T) &= \sum_{\tau=1}^{k-1} \left(\omega_\tau \times \frac{C_k^\tau}{C_k^\tau}\right) + \omega_k \\ &\quad \times \frac{\left|\{\text{tc}\} \bigcap \text{CombSet}_k(T)\right|}{C_k^k} \\ &= \sum_{\tau=1}^{k-1} \omega_\tau \\ &= 1.0 - \omega_k. \end{aligned} \tag{18}$$

In summary, if $\text{CombSet}_{k-1}(T) = \text{CombSet}_{k-1}(T_{\text{all}})$, for $\forall \text{tc} \notin T$, $\text{IICS}(\text{tc}, T) = 0$ and $\text{MICS}(\text{tc}, T) = 1.0 - \omega_k$. □

According to Theorem 8, a test case generation method using IICS or MICS as the similarity measure becomes a random generation method, when its generated test suite T covers all possible $(k-1)$-wise value combinations. The main reason is that, for any candidates, no matter whether they are included in T or not, the IICS (or MICS) values of all candidates are identical.

Theorem 9. *If* $MICS(tc, T) = 0$, $IICS(tc, T) = 0$.

Proof. If $\text{MICS}(\text{tc}, T) = 0$, $\text{NCVCS}_\tau(\text{tc}, T) = 0$ where $1 \le \tau \le k$ because of $0 \le \omega_\tau \le 1.0$ and $0 \le \text{NCVCS}_\tau(\text{tc}, T) \le 1.0$; that is, all possible τ-wise value combinations covered by tc are not covered by T. According to (15), therefore, $\text{IICS}(\text{tc}, T) = 0$. □

As discussed before, both IICS and MICS consider different interaction coverage when evaluating combinatorial test cases. However, they have some differences. Given a combinatorial test case (tc), its IICS measure is actually calculated by the NCVCS_λ at an appropriate λ value, which means that the IICS measure of tc only considers single interaction coverage, while its MICS measure considers different coverage at the same meanwhile. In other words, tc's calculation time of the IICS measure is less than that of the MICS.

In summary, two new similarity measures based on interaction coverage (IICS and MICS) fundamentally differ from NCVCS due to the following reasons: (1) they do not require setting the strength value in advance, and (2) they are more suitable for adaptive strategies than NCVCS.

4. Adaptive Random Testing for Combinatorial Test Inputs

In this section, we propose a new family of methods adopting ART in combinatorial input domain, namely, ART-CID. Similar to previous ART studies, ART-CID can also be implemented according to different notions. In this paper, we present one version of ART-CID by similarity (denoted as FSCS-CID), which uses the strategy of FSCS-ART [15]. Since the similarity measure is used in this paper, the procedure of FSCS-CID may differ from that of FSCS-ART. Detailed information will be given as follows.

4.1. Similarity-Based Test Case Selection in FSCS-CID. FSCS-CID uses two test sets, that is, the candidate set C of fixed size l and the executed set E, each of which has the same definition as FSCS-ART. However, test cases in either C or E are obtained from the combinatorial input domain. For ease of description, let $E = \{e_1, e_2, \ldots, e_m\}$ while $C = \{c_1, c_2, \ldots, c_l\}$. In order to select the next test case c_h from C, the criterion is described as follows:

$$\begin{aligned} &(\forall c_i)\,(c_i \in C) \quad (i = 1, 2, \ldots, l, i \ne h) \\ &\left[\max_{j=1}^{m} S\left(c_h, e_j\right) \le \max_{j=1}^{m} S\left(c_i, e_j\right)\right], \end{aligned} \tag{19}$$

where S is the similarity measure between two combinatorial test inputs. The detailed algorithm of implementing (19) is illustrated as follows (see Algorithm 1).

4.2. Algorithm of FSCS-CID. As discussed before, Algorithm 1 is used to guide the selection of the best test case. In FSCS-CID, the process of Algorithm 1 runs until the stop condition is satisfied. In this paper, we consider two stop conditions: (1) the first software failure is detected (denoted `StopCon1`); and (2) all possible value combinations at strength τ are covered (denoted `StopCon2`). Detailed algorithm of FSCS-CID is shown in Algorithm 2.

Since the frequencies of parameter values are used in some similarity measures such as *Lin*, *OF*, and *Goodall2*, there requires a fixed-size set of test cases in order to count the frequencies. However, the executed set E is incrementally updated with the selected element from the candidate set C until the `StopCon1` (or `StopCon2`) is satisfied. In this paper, we take the following strategy to construct the fixed-size set of test cases when calculating the similarity between test inputs. During the process of choosing the ith ($i > 1$) test input from C as the next test case (i.e., $|E| = i - 1$), each candidate $c_h (c_h \in C)$ requires to be measured against all elements in E according to the similarity measure, and the fixed-size set of test case for c_h is constructed by $E \bigcup \{c_h\}$.

5. Experiment

In this section, some experimental results, including simulations and experiments against real programs, were presented to analyze the effectiveness of FSCS-CID. We mainly compared our method to RT in terms of failure-detection effectiveness (F-measure) and the rate of value combinations coverage at a given strength (N_τ-measure). For ease of describing our work clearly, we used the terms `Goodall1`, `Goodall2`, `Goodall3`, `Goodall4`, `Lin`, `Lin1`, `Overlap`, `Eskin`, `OF`, `IICS`, and `MICS` to, respectively, represent the similarity measure *Goodall1*, *Goodall2*, *Goodall3*, *Goodall4*, *Lin*, *Lin1*, *Overlap*, *Eskin*, *OF*, IICS, and MICS adopted in the FSCS-CID. Additionally, we used the term RT to represent RT.

As shown in (16), a weight is required to be assigned for interaction coverage at each strength value. There are many techniques which conduct on assigning weights; however, in this paper we focus on two distribution styles: (1) equal distribution where each interaction coverage has the same weight, that is, $\omega_1 = \omega_2 = \cdots = \omega_k = 1/k$; and (2) FTFI percentage distribution where according to previous studies [28, 34], for example, in [28], Kuhn et al. investigated several software projects and concluded that the interaction faults are summarized to have 29% to 82% faults as 1-wise faults (i.e., the FTFI number is 1), 6% to 47% of faults as 2-wise faults, 2% to 19% as 3-wise faults, 1% to 7% of faults as 4-wise faults, and even fewer failures beyond 4-wise interactions. As a consequence, we arrange weights as follows: $\omega_1 = \omega$, $\omega_{i+1} = (1/2)\omega_i$, where $i = 1, 2, \ldots, k-1$. For example, if $k = 2$, $\omega_1 = 0.67$ and $\omega_2 = 0.33$; if $k = 3$, $\omega_1 = 0.57$, $\omega_2 = 0.29$, and $\omega_3 = 0.14$. In this paper, therefore, we use the terms `MICS1` and `MICS2` to stand for the `MICS` techniques with the above two weight distribution styles, respectively.

5.1. Simulation. In the following subsection, two simulations were presented to analyze the effectiveness of FSCS-CID

Input: The candidate set $C = \{c_1, c_2, \ldots, c_l\}$, and the executed set $E = \{e_1, e_2, \ldots, e_m\}$.
Output: Best test element $best\_element \in C$.
(1) Set $best\_similarity$ = Max_Value;
(2) **for** (i = 1 to l)
(3) Set $max\_candidate\_similarity$ = Min_Value;
(4) **for** (j = 1 to m)
(5) Calculate the similarity between c_i and e_j, that is, $S(c_i, e_j)$;
(6) **if** ($S(c_i, e_j) > max\_candidate\_similarity$)
(7) $max\_candidate\_similarity = S(c_i, e_j)$;
(8) **end if**
(9) **end for**
(10) **if** ($max\_candidate\_similarity < best\_similarity$)
(11) $best\_similarity = max\_candidate\_similarity$;
(12) $best\_element = c_i$;
(13) **end if**
(14) **end for**
(15) **return** $best\_element$.

Algorithm 1: Select the best test element based on similarity (BTES).

Input: A test profile, TP($k, |V_1||V_2| \cdots |V_k|$).
Output: The size of the executed set E and the first failure-causing input when meeting the `StopCon1` (or the size of E when meeting the `StopCon2`).
(1) Set $E = \{\}$, and $l = 10$;
(2) Randomly generate a test case t_c, according to TP($k, |V_1||V_2| \cdots |V_k|$) by the uniform distribution;
(3) Run the program with t_c, and $E = E \bigcup \{t_c\}$;
(4) **while** (The `StopCon1` (or `StopCon2`) is not satisfied)
(5) Randomly generate l candidates from TP($k, |V_1||V_2| \cdots |V_k|$), according to uniform distribution to form the candidate set $C = \{c_1, c_2, \ldots, c_l\}$;
(6) Set t_c = BTES(C, E);
(7) Run the program with t_c, and $E = E \bigcup \{t_c\}$;
(8) **end while**
(9) **return** $|E|$ and t_c for the `StopCon1` (or $|E|$ for the `StopCon2`).

Algorithm 2: The algorithm of FSCS-CID.

according to the rate of covering τ-wise value combinations (i.e., N_τ-measure). We used two usual test profiles TP($10, 2^{10}$) and TP($10, 2^5 3^5$) that are commonly used in previous studies [37].

5.1.1. Setup. Since the τ was known before testing, in this simulation, we considered the FSCS-CID using NCVCS$_\tau$ [36] as the similarity measure (denoted `NCVCS`). Except the `MICS`, all other methods do not require to be set. As for the `MICS`, different strength values from 1 to k are considered to calculate the MICS measure according to (16). However, due to the known τ, we mainly focused on the strength values from 1 to τ for calculating the MICS measure. As a consequence, (16) becomes as follows:

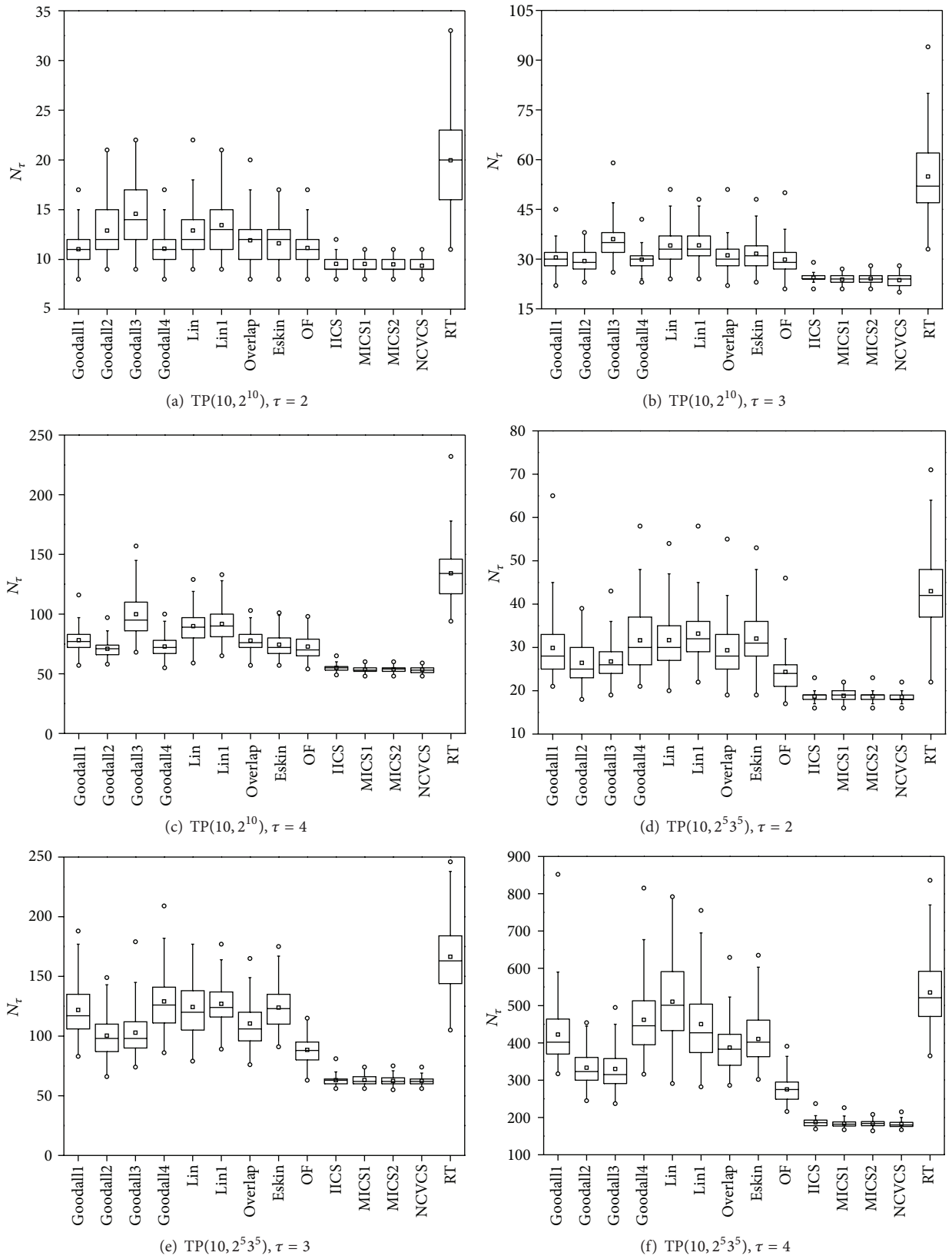
$$\mathrm{MICS}(\mathrm{tc}, T) = \sum_{\lambda=1}^{\tau} \left(\omega_\lambda * \mathrm{NCVCS}_\lambda(\mathrm{tc}, T)\right), \quad (20)$$

where only $\omega_1, \omega_2, \ldots, \omega_\tau$ are considered. Each method runs until the `StopCon2` is satisfied. Additionally, we consider N_τ as the metric to evaluate each method in terms of the rate of covering value combinations at strength τ for each method, where $\tau = 2, 3, 4$.

5.1.2. Results. Figure 3 summarizes the number of test cases required to cover all possible τ-wise value combinations (i.e., N_τ) generated by each method for the above two designed test profiles. Based on the experimental data, we have the following observations.

(1) For each test profile, the N_τ ($\tau = 2, 3, 4$) metric values of all FSCS-CID methods using different similarity measures are smaller than those of RT. In other words, the FSCS-CID methods require the smaller number of test cases for covering all τ-wise value combinations than RT, which means that the FSCS-CID methods have the higher rates of covering value combinations than RT.

(2) Among all the FSCS-CID methods, the `NCVCS` is the most effective technique. The results show that



(a) TP$(10, 2^{10})$, $\tau = 2$

(b) TP$(10, 2^{10})$, $\tau = 3$

(c) TP$(10, 2^{10})$, $\tau = 4$

(d) TP$(10, 2^5 3^5)$, $\tau = 2$

(e) TP$(10, 2^5 3^5)$, $\tau = 3$

(f) TP$(10, 2^5 3^5)$, $\tau = 4$

Figure 3: The N_τ metric for different test case generation techniques.

Table 3: The detailed information about 6 programs with 9 versions.

Program	Test profile	LOC	#S.	#D.	#U.	θ_S	θ_L
Count	TP$(6, 2^1 3^5)$	42	15	12	10	0.111111	0.444444
Series	TP$(3, 5^2 7^1)$	288	23	22	14	0.034286	0.462857
Tokens	TP$(8, 2^4 3^4)$	192	21	15	5	0.111111	0.388889
Ntree	TP$(4, 4^4)$	307	32	24	12	0.023438	0.492188
Nametbl	TP$(5, 2^1 3^2 5^2)$	329	51	44	19	0.133333	0.444444
Flex-v1	TP$(7, 2^4 3^1 6^1 16^1)$	8426	19	16	11	0.031250	0.427083
Flex-v2		9932	20	11	7	0.019097	0.375000
Flex-v3		9965	17	6	6	0.020833	0.118056
Flex-v4		10055	16	11	9	0.135417	0.427083

the N_τ values of the NCVCS are about 30%~50% of those of the RT. The IICS has the second best N_τ metric values, followed by the OF. For TP$(10, 2^{10})$, the Goodall3 is least effective, while for TP$(10, 2^5 3^5)$, the Lin performs least.

(3) From the perspective of the similarity category, the FSCS-CID methods using the interaction-coverage-based similarity measures (including IICS, MICS, and NCVCS) perform best, while the FSCS-CID methods using the information-theoretic similarity measures (including Lin and Lin1) perform worst.

5.1.3. Analysis. Here, we briefly analyze the above observations. The observation (1) is explained as follows. The FSCS-CID methods using different similarity measures select the next test case that has the smallest similarity value against already generated test cases, while RT simply generates teat cases at random from combinatorial input domain. As a consequence, the FSCS-CID methods achieve test cases more diversely than RT over the combinatorial input domain.

As for the observations (1) and (2), they are easy to be explained. On the one hand, since the N_τ metric is related to τ-wise value combinations, the NCVCS performs best because it selects the next test case that covers of uncovered τ-wise value combinations as much as possible. In other words, it may have the fastest rate of covering all τ-wise value combinations. On the other hand, another two interaction-coverage-based methods, such as IICS and MICS, consider different strength values for generating test cases; however, both of them take the strength τ as an indispensable part. In detail, the IICS calculates the test candidate from the strength 1 to τ, while the MICS considers different strengths from 1 to τ at the same time. Hence, it is reasonable that, compared to other categories, the FSCS-CID methods using interaction-coverage-based similarity measures perform best according to the N_τ metric.

5.2. An Empirical Study. In this section, an empirical study was conducted to compare the performance between FSCS-CID and RT in practical situations, using the F-measure as the effectiveness metric. To describe data clearly, we used ART F-ratio, which is defined as the F-measure ratio between FSCS-CID and RT, that is, $F_{\text{FSCS-CID}}/F_{\text{RT}}$. Intuitively speaking, the smaller ART F-ratio value implies the higher improvement of FSCS-CID over RT, and $(1 - F_{\text{FSCS-CID}}/F_{\text{RT}})$ is the F-measure improvement of FSCS-CID over RT.

In this empirical study, we use a set of six fault-seeded C programs with 9 versions. The five subject programs, including count, series, tokens, ntree, and nametbl, are downloaded from Chris Lott's website (http://www.maultech.com/chrislott/work/exp/), which have been widely used in the research of combinatorial space such as comparison of defect revealing mechanisms [38], evaluation of different combination strategies for test case selection [39], and fault diagnosis [40, 41]. The remainder subject programs are a series of flex programs (the model used in this paper is unconstrained, which has some limitations: "We note that in a real test environment an unconstrained TSL would most likely be prohibitive in size and would not be used" [42].), downloaded from Software Infrastructure Repository (SIR) [43], which are popularly used in combinatorial test suite construction [44] and combinatorial interaction regression testing [42].

Table 3 presents detailed information about these subject programs, from which the third column "LOC" represents the number of lines of executable code in these programs, and "#S." is the number of seeded faults in each subject program, while "#D." is the number of faults that can be detected by some test cases derived from the accompanying test profiles, which are not guaranteed to be able to detect all faults. However, in our study, we only use a portion of detectable faults, of which the size is shown as "#U.". The main reason is due to the fact that faults in the set of detectable faults but not in the set of used faults have high failure rates that exceed 0.5. As we know, if the failure rate θ of a fault is larger than 0.5, the F-measure of random testing is theoretically less than $1/\theta = 2$. As a consequence, the F-measure of FSCS-CID depends on the first randomly selected test case. In other words, if the first test case cannot detect a failure, the $F_{\text{FSCS-CID}}$ is larger than or equal to 2. Therefore, the F-measure of FSCS-CID is dependent on random testing.

For the purpose of clear description, we order i used faults in each subject program in a descend order according to failure rate and abbreviate them as $m_1, m_2, \ldots, m_i$. The range of failure rates in each program, as shown in Table 3, is from θ_S to θ_L.

We used all twelve FSCS-CID versions using different similarity measures to test these fault-seeded programs. The results of the empirical study are given in Figure 4, where x-axis represents each seeded fault in the subject program, while

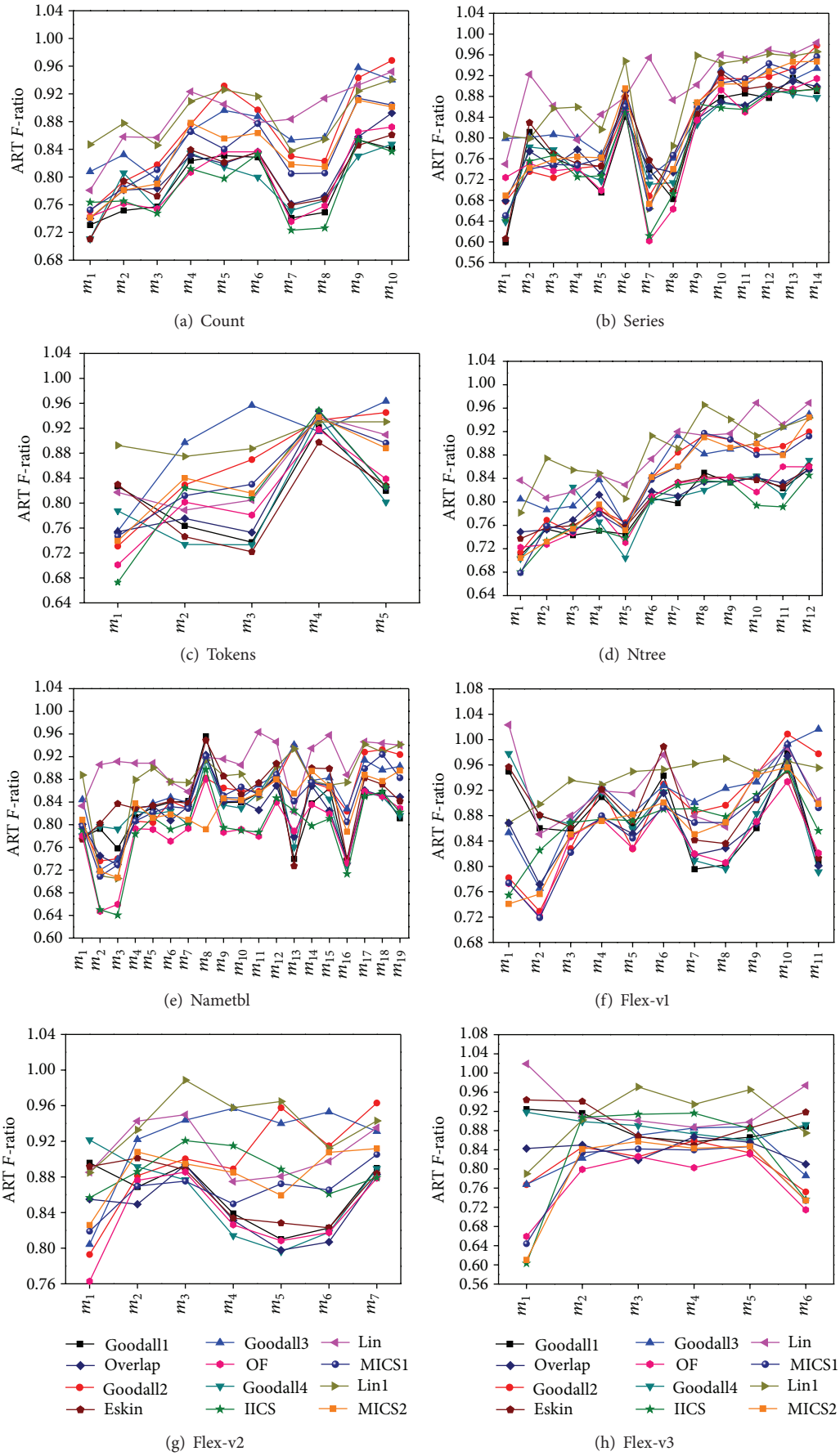


Figure 4: Continued.

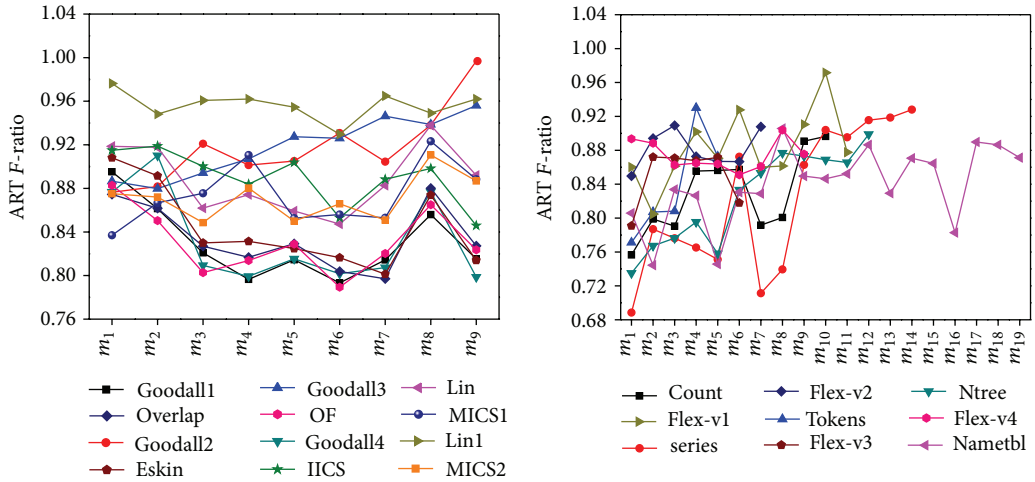


Figure 4: ART F-ratio of FSCS-CID methods using different similarity measures.

y-axis represents the ART F-ratio. As shown in Figures 4(a)–4(i), each figure corresponds to a particular subject program, while Figure 4(j) represents the average ART F-ratio of all FSCS-CID versions for each subject program.

From Figures 4(a)–4(j), we can observe the following conclusions.

(1) According to ART F-ratio, all twelve FSCS-CID versions, including `Goodall1`, `Goodall2`, `Goodall3`, `Goodall4`, `Lin`, `Lin1`, `Overlap`, `Eskin`, `OF`, `IICS`, `MICS1`, and `MICS2`, perform better than `RT`. In the best case, the improvement of FSCS-CID over `RT` is about 40% (i.e., ART F-ratio is 60%).

(2) With the increase of failure rate θ, the ART F-ratio of each FSCS-CID version increases as well in most programs. In other words, when θ is larger, the improvement of each FSCS-CID version over `RT` is smaller.

(3) The failure-detection capability of FSCS-CID depends on some factors, such as the program (or test profile) and failure type (including failure rate θ and failure pattern). For example, in program `count` faults m_3 and m_4 have the same failure rate; however, the ART F-ratio of each FSCS-CID version when detecting m_3 is very different from that when detecting m_4.

(4) Figure 4(j) describes the average ART F-ratio of all FSCS-CID versions when detecting each fault for each subject program. It can be clearly seen that the ART F-ratio of the FSCS-CID algorithm generally fluctuates from 0.75 to 0.90 among all faults for each program, which means FSCS-CID can improve about 10%~25% of F-measure over RT in the average.

(5) Among all FSCS-CID versions, no method performs best for all programs, and no method performs worst. In order to compare the failure-detection capabilities of different FSCS-CID versions, Table 4 shows the average ART F-ratio of each FSCS-CID version for each subject program. According to data shown in Table 4, it is obvious that in general one of FSCS-CID version `OF` performs best, followed by `IICS`, while `Lin` and `Lin1` generally perform worst. In addition, `Eskin` performs best for the program `tokens` and `Goodall1` has the best performance for the program `flexv4`.

In summary, our simulation results (Section 5.1) have shown that our FSCS-CID algorithm (irrespective of used similarity measure) has higher rates of covering value combinations at different strength values than those of random testing. Besides, the empirical study has shown that the FSCS-CID algorithm performs better than RT in terms of the number of test cases required to detect the first failure (i.e., F-measure).

5.3. Threats to Validity. The experimental results suffer from some threats to validity; in this section, we outline the major threats. In the simulation study, two widely used, but limited, test profiles were employed. In the empirical study, many real-life programs were used, which have been popularly investigated by different researches. However, the faults seeded in each subject program have high failure rates. To address these potential threats, additional studies using a great number of test profiles and a great number of subject programs with low failure rates will be investigated in the future.

In addition, although two metrics (N_τ-measure and F-measure) were employed in our experiment, we recognize that there may be other metrics which are more pertinent to the study.

6. Discussion and Conclusion

Adaptive random testing (ART) [15] has been proposed to enhance the failure-detection capability of random testing

Table 4: The average ART F-ratio of each FSCS-CID version for each subject program (%).

Method	Count	Series	Tokens	Ntree	Nametbl	Flex-v1	Flex-v2	Flex-v3	Flex-v4
Goodall1	**79.07**[a]	79.75	81.38	79.24	82.76	87.49	85.99	88.65	**82.97**
Goodall2	86.31	81.84	86.16	83.59	85.17	88.47	89.99	81.38	91.80
Goodall3	86.95	84.17	89.74	85.72	85.54	90.86	92.16	83.66	91.80
Goodall4	79.15	79.58	79.86	79.60	82.66	87.84	85.81	88.86	83.28
Lin	*88.84*[b]	*90.07*	85.18	88.57	*90.97*	92.21	90.94	*93.10*	88.79
Lin1	88.79	87.65	*90.31*	*88.81*	87.57	*93.95*	*94.06*	90.63	*95.64*
Overlap	80.76	81.01	81.11	80.56	82.36	85.64	84.53	84.06	83.53
Eskin	80.07	81.32	**80.47**	80.39	84.65	89.35	86.42	90.09	84.35
OF	79.70	78.84	80.80	79.75	**79.09**	**83.57**	**83.64**	**77.20**	83.07
IICS	78.59	**78.72**	81.59	**78.27**	79.29	87.06	88.67	82.69	88.97
MICS1	83.58	81.77	84.43	82.63	84.21	86.02	86.53	79.00	87.37
MICS2	83.55	82.30	84.42	83.05	83.12	86.56	88.47	78.89	87.11

[a]The bold datum is minimum in each column.
[b]The italic datum is maximum in each column.

(RT) by evenly spreading test cases all over the input domain and has been widely applied in various applications such as numerical programs, Java programs, and object-oriented programs. In this paper, we broaden the principle of ART in a new type of input domain that has not yet been investigated, that is, combinatorial input domain. Due to special characteristics of combinatorial input domain, the test case similarity (or dissimilarity) measures previously used in ART may not be suitable for combinatorial input domain. By adopting some well-known similarity measures used in data mining and proposing two new similarity measures based on interaction coverage, we proposed a new approach to apply original ART into combinatorial input domain, named ART-CID. We conducted some experiments including simulations and the empirical study to analyze the effectiveness of one version of ART-CID (FSCS-CID, which is based on fixed-size-candidate-set ART). Compared with RT, FSCS-CID not only brings higher rates in covering all possible combinations at any given strengths, but also requires fewer combinatorial test cases to detect the first failure in the seeded program.

Combinatorial interaction testing (CIT) [33] is a black-box testing method and has been widely used in combinatorial input domain. It aims at constructing an effective test suite to identify interaction faults caused by parameter interactions. Some greedy CIT algorithms, such as AETG [45], TCG [46], and DDA [47], may have similar mechanism as FSCS-CID. Taking AETG for example, similar to AETG, FSCS-CID also first constructs some candidates, and then from which the "*best*" element would be chosen as the next test case according to some criteria. However, there are some fundamental differences between AETG and FSCS-CID, which are mainly summarized as follows.

(1) Different construction strategies of candidates: FSCS-CID constructs candidates in a random manner, while AETG first orders all parameters and then assigns a value to each parameter, such that all assigned parameter values can cover the largest number of value combinations at a given strength.

(2) Different test case selection criteria: AETG selects an element from candidates as the next test case such that it covers the largest number of value combinations at a given strength, while FSCS-CID chooses the next test case according to its used similarity measure.

(3) Different goals achieved: AETG aims at covering all possible value combinations of a given strength with fewer test cases, which means that the unique stopping condition of AETG is that all value combinations of a given strength are covered by generated test cases, while FSCS-CID is an adaptive strategy, which means that the stopping condition of FSCS-CID is not limited to covering all value combinations of a give strength, for example, detecting a first failure in the SUT.

In this paper, constraints among value combinations have not been considered; however, they often exist in real-life programs. For example, as shown in Table 1, there may exist a constraint among "*Full Size*" of p_1 and "*Single*" of p_2, that is, when $p_1 = \textit{Full Size}$, $p_2 \neq \textit{Single}$ (i.e., "*Full Size*" and "*Single*" cannot occur in a combinatorial test cases). In this case, the method FSCS-CID proposed in this paper can still be successfully executed only by judging that each selected test case violates constraints among value combinations or not. Generally speaking, this judgment process can be implemented in the following phases: (1) when constructing the candidate set and (2) when adding the latest test case into the executed set. However, how to deal with constraints among value combinations should be further studied.

In the future, we plan to further investigate how to improve the effectiveness of the approach by adopting other similarity measures that may be available in combinatorial input domain or by considering additional factors to guide test case generation. In addition, how to extend other original

ART algorithms into combinatorial input domain is also expected.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

Acknowledgments

The authors would like to thank C. M. Lott for sending them the failure reports of five subject programs (count, series, tokens, ntree, and nametbl) and the Software-artifact Infrastructure Repository (SIR) [43] which provided the source code and fault data for the program flex. They also would like to thank T. Y. Chen for the many helpful discussions and comments. This work is in part supported by the National Natural Science Foundation of China (Grant no. 61202110) and Natural Science Foundation of Jiangsu Province (Grant no. BK2012284).

References

[1] B. Hailpern and P. Santhanam, "Software debugging, testing, and verification," *IBM Systems Journal*, vol. 41, no. 1, pp. 4–12, 2002.

[2] P.-S. Loo and W.-K. Tsai, "Random testing revisited," *Information and Software Technology*, vol. 30, no. 7, pp. 402–417, 1988.

[3] R. Hamlet, "Randomg testing," in *Encyclopedia of Software Engineering*, J. J. Marciniak, Ed., pp. 970–978, John Wiley & Sons, New York, NY, USA, 2002.

[4] P. G. Frankl, R. G. Hamlet, B. Littlewood, and L. Strigini, "Evaluating testing methods by delivered reliability," *IEEE Transactions on Software Engineering*, vol. 24, no. 8, pp. 586–601, 1998.

[5] B. P. Miller, L. Fredriksen, and B. So, "An empirical study of the reliability of UNIX utilities," *Communications of the ACM*, vol. 33, no. 12, pp. 32–44, 1990.

[6] B. P. Miller, D. Koski, C. Lee et al., "Fuzz revisited: a re-examination of the reliability of UNIX utilities and services," Tech. Rep. CS-TR-1995-1268, University of Wisconsin, Madison, Wis, USA, 1995.

[7] D. Slutz, "Massive stochastic testing of SQL," in *Proceedings of the 24th International Conference on Very Large Data Bases (VLDB '98)*, pp. 618–622, New York, NY, USA, August 1998.

[8] H. Bati, L. Giakoumakis, S. Herbert, and A. Surna, "A genetic approach for random testing of database systems," in *Proceedings of the 33rd Interanational Conference on Very Large Data Bases (VLDB '07)*, pp. 1243–1251, Vienna, Austria, September 2007.

[9] T. Yoshikawa, K. Shimura, and T. Ozawa, "Random program generator for Java JIT compiler test system," in *Proceedings of the 3rd International Conference on Quality Software (QSIC '03)*, pp. 20–23, Dallas, Tex, USA, November 2003.

[10] J. Regehr, "Random testing of interrupt-driven software," in *Proceedings of the 5th ACM International Conference on Embedded Software (EMSOFT '05)*, pp. 290–298, Jersey City, NJ, USA, September 2005.

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

[12] G. B. Finelli, "NASA software failure characterization experiments," *Reliability Engineering and System Safety*, vol. 32, no. 1-2, pp. 155–169, 1991.

[13] P. G. Bishop, "The variation of software survival time for different operational input profiles," in *Proceedings of the 23rd International Symposium on Fault-Tolerant Computing (FTCS '23)*, pp. 98–107, Toulouse, France, June 1993.

[14] T. Y. Chen, T. H. Tse, and Y. T. Yu, "Proportional sampling strategy: a compendium and some insights," *Journal of Systems and Software*, vol. 58, no. 1, pp. 65–81, 2001.

[15] T. Y. Chen, H. Leung, and I. K. Mak, "Adaptive random testing," in *Proceedings of the 9th Asian Computing Science Conference (ASIAN '04)*, pp. 320–329, Chiang Mai, Thailand, December 2004.

[16] K. P. Chan, T. Y. Chen, and D. Towey, "Restricted random testing: adaptive random testing by exclusion," *International Journal of Software Engineering and Knowledge Engineering*, vol. 16, no. 4, pp. 553–584, 2006.

[17] A. F. Tappenden and J. Miller, "A novel evolutionary approach for adaptive random testing," *IEEE Transactions on Reliability*, vol. 58, no. 4, pp. 619–633, 2009.

[18] J. Mayer, "Lattice-based adaptive random testing," in *Proceedings of the 20th IEEE/ACM International Conference on Automated Software Engineering (ASE '05)*, pp. 333–336, Long Beach, Calif, USA, November 2005.

[19] T. Y. Chen, F.-C. Kuo, R. G. Merkel, and T. H. Tse, "Adaptive random testing: the ART of test case diversity," *Journal of Systems and Software*, vol. 83, no. 1, pp. 60–66, 2010.

[20] I. Ciupa, A. Leitner, M. Oriol, and B. Meyer, "ARTOO: adaptive random testing for object-oriented software," in *Proceedings of the 30th International Conference on Software Engineering (ICSE '08)*, pp. 71–80, Leipzig, Germany, May 2008.

[21] Y. Lin, X. Tang, Y. Chen, and J. Zhao, "A divergence-oriented approach to adaptive random testing of Java programs," in *Proceedings of the 24th IEEE/ACM International Conference on Automated Software Engineering (ASE '09)*, pp. 221–232, Washington, DC, USA, November 2009.

[22] Z. Liu, X. Gao, and X. Long, "Adaptive random testing of mobile application," in *Proceedings of the 2nd International Conference on Computer Engineering and Technology (ICCET '10)*, pp. 297–301, Chengdu, China, April 2010.

[23] T. J. Ostrand and M. J. Balcer, "The category-partition method for specifying and generating fuctional tests," *Communications of the ACM*, vol. 31, no. 6, pp. 676–686, 1988.

[24] T. Y. Chen, P.-L. Poon, and T. H. Tse, "A choice relation framework for supporting category-partition test case generation," *IEEE Transactions on Software Engineering*, vol. 29, no. 7, pp. 577–593, 2003.

[25] T. Y. Chen, P.-L. Poon, S.-F. Tang, and T. H. Tse, "On the identification of categories and choices for specification-based test case generation," *Information and Software Technology*, vol. 46, no. 13, pp. 887–898, 2004.

[26] P.-L. Poon, S.-F. Tang, T. H. Tse, and T. Y. Chen, "CHOC'LATE: a framework for specification-based testing," *Communications of the ACM*, vol. 53, no. 4, pp. 113–118, 2010.

[27] T. Y. Chen, P. L. Poon, S. F. Tang, and T. H. Tse, "DESSERT: a divide-and-conquer methodology for identifying categories, choices, and choice relations for test case generation," *IEEE Transactions on Software Engineering*, vol. 38, no. 4, pp. 794–809, 2012.

[28] D. R. Kuhn, D. R. Wallace, and A. M. Gallo Jr., "Software fault interactions and implications for software testing," *IEEE Transactions on Software Engineering*, vol. 30, no. 6, pp. 418–421, 2004.

[29] C. Yilmaz, M. B. Cohen, and A. A. Porter, "Covering arrays for efficient fault characterization in complex configuration spaces," *IEEE Transactions on Software Engineering*, vol. 32, no. 1, pp. 20–34, 2006.

[30] R. C. Bryce, S. Sampath, and A. M. Memon, "Developing a single model and test prioritization strategies for event-driven software," *IEEE Transactions on Software Engineering*, vol. 37, no. 1, pp. 48–64, 2011.

[31] X. Yuan, M. B. Cohen, and A. M. Memon, "GUI interaction testing: incorporating event context," *IEEE Transactions on Software Engineering*, vol. 37, no. 4, pp. 559–574, 2011.

[32] S. Boriah, V. Chandola, and V. Kumar, "Similarity measures for categorical data: a comparative evaluation," in *Proceedings of the 8th SIAM International Conference on Data Mining (SDM '08)*, pp. 243–254, Atlanta, Ga, USA, April 2008.

[33] C. Nie and H. Leung, "A survey of combinatorial testing," *ACM Computing Surveys*, vol. 43, no. 2, article 11, 2011.

[34] D. R. Kuhn and M. J. Reilly, "An investigation of the applicability of design of experiments to software testing," in *Proceedings of the 27th Annual NASA Goddard/IEEE Software Engineering Workshop (SEW '02)*, pp. 91–95, Greenbelt, Maryland, December 2002.

[35] T. Y. Chen, F.-C. Kuo, and R. Merkel, "On the statistical properties of testing effectiveness measures," *Journal of Systems and Software*, vol. 79, no. 5, pp. 591–601, 2006.

[36] R. Huang, X. Xie, T. Y. Chen, and Y. Lu, "Adaptive random test case generation for combinatorial testing," in *Proceedings of the 36th Annual International Computer Software and Applications Conference (COMPSAC '12)*, pp. 52–61, Izmir, Turkey, July 2012.

[37] R. C. Bryce, C. J. Colbourn, and D. R. Kuhn, "Finding interaction faults adaptively using distance-based strategies," in *Proceedings of the 18th IEEE International Conference and Workshops on Engineering of Computer-Based Systems (ECBS '11)*, pp. 4–13, Las Vegas, Nev, USA, April 2011.

[38] C. M. Lott and H. D. Rombach, "Repeatable software engineering experiments for comparing defect-detection techniques," *Empirical Software Engineering*, vol. 1, no. 3, pp. 241–277, 1996.

[39] M. Grindal, B. Lindström, J. Offutt, and S. F. Andler, "An evaluation of combination strategies for test case selection," *Empirical Software Engineering*, vol. 11, no. 4, pp. 583–611, 2006.

[40] Z. Zhang and J. Zhang, "Characterizing failure-causing parameter interactions by adaptive testing," in *Proceedings of the 20th International Symposium on Software Testing and Analysis (ISSTA '11)*, pp. 331–341, Toronto, Canada, July 2011.

[41] L. S. G. Ghandehari, Y. Lei, T. Xie, D. R. Kuhn, and R. Kacker, "Identifiying failureinducing combinations in a combinatorial test set," in *Proceedings of the 5th IEEE International Conference on Software Testing, Verication and Validation (ICST '12)*, pp. 370–379, Montreal, Canada, January 2012.

[42] X. Qu, M. B. Cohen, and K. M. Woolf, "Combinatorial interaction regression testing: a study of test case generation and prioritization," in *Proceedings of the 23rd International Conference on Software Maintenance (ICSM '07)*, pp. 255–264, Paris, France, October 2007.

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

[44] B. S. Ahmed, K. Z. Zamli, and C. P. Lim, "Constructing a t-way interaction test suite using the particle swarmoptimization approach," *International Journal of Innovative Computing, Information and Control A*, vol. 8, no. 1, pp. 431–451, 2012.

[45] D. M. Cohen, S. R. Dalal, M. L. Fredman, and G. C. Patton, "The AETG system: an approach to testing based on combinatorial design," *IEEE Transactions on Software Engineering*, vol. 23, no. 7, pp. 437–444, 1997.

[46] Y.-W. Tung and W. S. Aldiwan, "Automating test case generation for the new generation mission software system," in *Proceedings of the IEEE Aerospace Conference*, vol. 1, pp. 431–437, Big Sky, Mont, USA, March 2000.

[47] R. C. Bryce and C. J. Colbourn, "A density-based greedy algorithm for higher strength covering arrays," *Software Testing Verification and Reliability*, vol. 19, no. 1, pp. 37–53, 2009.