Received: 15 August 2022 | Revised: 11 December 2022 | Accepted: 17 January 2023

DOI: 10.1002/smr.2537

RESEARCH ARTICLE - TECHNOLOGY

Journal of Software: Evolution and Process WILEY

A novel combinatorial testing approach with fuzzing strategy

Jinfu Chen[1,2] | Jingyi Chen[1,2] | Saihua Cai[1,2] | Haibo Chen[1,2] | Chi Zhang[1,2]

[1]School of Computer Science and Communication Engineering, Jiangsu University, Zhenjiang, China

[2]Jiangsu Key Laboratory of Security Technology for Industrial Cyberspace, Jiangsu University, Zhenjiang, China

Correspondence
Saihua Cai, School of Computer Science and Communication Engineering, Jiangsu University, Zhenjiang 212013, China.
Email: caisaih@ujs.edu.cn

Funding information
National Natural Science Foundation of China, Grant/Award Numbers: 62172194, 62202206, U1836116; National Key R&D Program of China, Grant/Award Number: 2020YFB1005500; Leading-edge Technology Program of Jiangsu Natural Science Foundation, Grant/Award Number: BK20202001; Natural Science Foundation of Jiangsu Province, Grant/Award Number: BK20220515; China Postdoctoral Science Foundation, Grant/Award Number: 2021M691310; Graduate Research Innovation Project of Jiangsu Province, Grant/Award Numbers: SJCX21_1692, KYCX21_3375, KYCX22_3670; Qinglan Project of Jiangsu Province

Summary

Combinatorial testing (CT) is considered as a practical approach to detect software faults, which has arisen from the interaction between factors affecting the software behavior. However, most of the traditional algorithms on CT generation did not take advantage of the execution results of the earlier test cases, as well as neglect the impact of the nonequilibrium input parameter model (NE-IPM) effect on redundant test cases, which bring a deleterious effect to the detection accuracy of the software faults. To solve these problems, we propose a novel CT approach with fuzzing strategy called CTAF. Based on the idea that fuzzing is performed during execution, CTAF exploits the execution results of earlier tests to provide guidance for subsequent test generation thereby reducing the redundant test cases without compromising the diversity of test cases. And then, we designed three experiments on real subjects of six open source software systems, and the experimental results show that the proposed CTAF approach can effectively improve the NE-IPM effect and enhance the detection accuracy of software faults.

KEYWORDS
combinatorial testing, fuzzing technique, nonequilibrium input parameter model

1 | INTRODUCTION

Modern software is increasing in size and complexity, and thus the factors that can affect software behavior (e.g., configuration options, system inputs, and message events) are becoming more abundant. However, except for individual factors that can cause failure of the System Under Test (SUT), the interactive behaviors between these factors also can trigger faults in the SUT. Therefore, more factors related to software behaviors mean that testing such software becomes more challenging.[1] The testing techniques that handle such software are supposed to explore the following three questions: (1) How to improve the detecting accuracy of software faults; (2) how to enhance the efficiency of testing; and (3) how to balance the detection accuracy and efficiency.

First, if the scope of all possible interactions for all factors is to be covered, this is quite unwise because it will undoubtedly entail a great waste of testing resources. Second, many empirical studies have shown that in real software systems, the target fault detection range is only a small fraction of the entire interaction space.[2,3] Moreover, these interactions between four to six factors generally are capable of trigger a software failure.[2] Combinatorial testing (CT)[4,5] attempts to generate fewer test sets to search as wide a space of interactions as possible for detecting interaction faults in the SUT. As so far, CT has been widely applied in diverse areas where the interaction between certain elements affects the reliability of the software,[6,7] such as Graphical User Interface (GUI) Testing,[8] Security Testing for the Transport Layer Security (TLS) protocol,[9] and Product Lines Testing.[10]

The test set for combinatorial testing relies on the tester to extract a specific model for the SUT. Specifically, the tester needs to identify candidate factors that may affect the behaviors of the software, as well as the connection between these factors, the valid values of each factor, and

the constraints present in these factors to obtain the input parameter model (IPM) of the SUT. And then, the tester sets the test intensity to generate and execute a set of test cases, where each test case is a group of assignments of all factors in this model.

τ-way strategy is a popular choice for test case generation in CT, where τ refers to the interaction strength.[11] τ-way strategy exploits a τ-way coverage array as test suite, which requires that interaction strength τ should be met when testing. As a result, τ-way cover array should contain at least all combinations of the values of τ parameters. The τ-way cover array generation algorithms were summarized or categorized into three main types, including one-test-at-time (OTAT) approach, one-parameter-at-time (OPAT) approach, and a set of test cases one time approach.[12]

Although each of these test case generation algorithms has its own advantages, they fail to focus on a specific class of input parameter models. For example, Table 1 shows the IPM of the *Tcas* program, which derives from the study on pseudo-exhaustive testing.[13] It is clear that two parameters, that is, *Up_Separation and Down_Separation*, have 10 values, and they significantly differ in the number of values from the other 10 parameters, none of which has more than four values. Throughout this paper, we refer to such input parameter model as the nonequilibrium input parameter model (NE-IPM) (it's strictly defined in Section 3.1.). The opposite is the equilibrium input parameter model (EL-IPM). Test cases generated during CT originate from the IPM associated with SUT. That is, the number and values of these cases are largely determined by the number and values of all parameters in the IPM. When the number of critical parameters and the number of their values are small compared with other parameters, there is a situation where the redundancy of valid test cases is severe. Suppose 1-way strategy is adopted to test the program *Tcas*, and only a bug *B* that is triggered by setting the only key parameter *High_Confidence* equal to 0 exists. Consequently, the final generated test set is not unique but must consist of ten test cases. In the most redundant scenario, the parameter *High_Confidence* of nine test cases that trigger *B* and does not detect other vulnerabilities takes the value of 0. Besides, one more test case does not trigger bug *B* because its value corresponding to parameter *High_Confidence* takes the value of 1. In fact, the eight test cases of the nine test cases which trigger *B* are valid but redundant at this point. We believe that the redundancy of test cases in the presence of such nonequilibrium models is more severe than that of balanced parameter models. We refer to the test set redundancy problem relevant to NE-IPM as NE-IPM effect (it's strictly defined in Section 3.1.). Therefore, how to handle the redundancy of test cases and enhance the accuracy of detecting faults for the SUT with NE-IPM is the major challenge in this paper.

Based on this observation, we propose a novel CT approach with fuzzing strategy called CTAF. Fuzzing[14,15] is one of the popular software testing techniques. It improves testing efficiency by testing software based on randomly generated inputs and then guides mutation of test cases driven by the feedback of test results.[16,17] By following the idea that fuzzing is performed in execution, CTAF utilizes the execution results of previous test cases to provide guidance on the generation of subsequent test cases and then handles the above-mentioned redundancy problem for CT used for the SUT with NE-IPM. Specifically, we can acquire all τ-degree schemas contained in the earlier executed test cases. And then, three important factors emerged through combining their execution results, with or without exceptions, that is, τ-degree schemas that have been executed, τ-degree schemas that have caused failures, and τ-degree schemas that have not detected faults. We consider that the more τ-degree schemas that have yet to come up and detect faults a test case owes, the greater the likelihood of covering uncovered interactions and identifying undetected faults.

In addition, through analyzing the causes of NE-IPM effect, we consider that τ-way is not suitable as a criterion for valid test sets with regards to CT generation algorithms for the SUT with NE-IPM. Consequently, we propose a new coverage criterion called τ-sway. First, the generated test set according to τ-sway is smaller than that according to τ-way generally. Note that the size of the two types of test sets becomes progressively

TABLE 1 Input parameter model of *Tcas*.

No	Parameter	Values
1	Cur_Vertical_Sep	[299,300,601]
2	High_Confidence	[0,1]
3	Two_of_Three_Reports_Valid	[0,1]
4	Own_Tracked_Alt	[1,2]
5	Own_Tracked_Alt_Rate	[600,601]
6	Other_Tracked_Alt	[1,2]
7	Alt_Layer_Value	[0,1,2,3]
8	Up_Separation	[0,399,400,499,500,639,640,739,740,840]
9	Down_Separation	[0,399,400,499,500,639,640,739,740,840]
10	Other_RAC	[0,1,2]
11	Other_Capability	[1,2]
12	Climb_Inhibit	[0,1]

more consistent as the test intensity becomes larger. Second, the reduced test cases are actually redundant test cases due to NE-IPM effect. Thus, our new approach is reasonable and efficient in CT.

To evaluate the effectiveness of our approach, we conducted three empirical studies with real subjects from six open source software systems. We compare our strategy with four traditional combinatorial test generation algorithms, and the experimental results show that CTAF achieves better performance than traditional methods for the SUT with NE-IPM.

The main contributions of this paper are as follows:

- We formally analyze the severity of redundancy in nonequilibrium and equilibrium input parameter models.
- We propose a novel CT approach with fuzzing strategy, which is called CTAF to alleviate redundancy present in CT generation for nonequilibrium models, thereby further enhancing the detection accuracy of software faults in this case.
- We conducted three empirical studies, and the experimental results showed that CTAF can help combinatorial testing to achieve better performance in such software applications with nonequilibrium models.

2 | BACKGROUND

This section presents some definitions for formal description of CT and fuzzing.

2.1 | Combinatorial testing

A typical CT lifecycle consists of four main phases[11]: (1) Input parameter modeling: The tester identifies the factors that potentially bring influence the system behavior such as user input and configuration options, and the values corresponding to each parameter, as well as the constraints and dependencies between each factor, need to be identified to avoid ineffective testing; (2) generating test cases: In CT, each test case is an assignment of a set of all factors in the extracted input parameter model; that is, one value is taken out of each parameter to form one element of a combinatorial test case. The primary goal of this phase is to design a relatively small set of test cases to achieve expected interaction coverage; (3) fault location: This phase identifies the minimum failure-causing schema[18] that leads to failures, thereby narrowing the scope of suspect code that needs to be checked for later bug repair. Nevertheless, the masking effect raises difficulties in the identification of the minimum failure-causing schema; (4) evaluation: In this phase, the tester evaluates the quality of the previously performed test tasks. If the evaluation results show that the previous testing process did not meet the testing requirements, certain testing phases need to be improved.

Assume that the operation of an SUT is affected by n parameters, and their real-world significance can be expressed in terms of many factors, such as input variables, run-time options, and build options. Each parameter p_i has a_i discrete values from a finite set V_i. $|V_i| = a_i (i = 1,2,...,n)$. The definitions below are originally defined by Nie et al.[18]

Definition 1 **A test case.** A test case of the SUT is a tuple of n values, and each parameter of the SUT corresponds to a value. The test case is denoted as $(v_1, v_2, ..., v_n)$, where $v_i \in V_i (i = 1,2,...,n)$.

Definition 2 **τ-degree schema.** A τ-degree schema ($0 < \tau \le n$) of the SUT is the τ-tuple $(-, v_{n_1}, ..., v_{n_\tau}, ...)$, where certain τ parameters take on fixed values and other unrelated parameters are denoted as "-." When $\tau = n$, a test case itself is a τ-degree schema.

In addition, if every fixed value in a τ-degree schema is in a test case, we consider this test case contains the τ-degree schema.

Definition 3 **Subschema and super-schema.** Let s_x be an x-degree schema, s_y be an y-degree schema in the SUT, and $x < y$. If all the fixed parameter values in s_x are contained in s_y, then s_y subsumes s_x. Accordingly, s_x can be regarded as a subschema of s_y, and s_y is a super-schema of s_x, that is, $s_x \prec s_y$ or $s_y \succ s_x$.

τ-way strategy tends to be popular in CT test case generation. τ is commonly known as the interaction strength or coverage intensity. This strategy usually produces a relatively small set of test cases that cover all τ-degree schema ($0 < \tau \le n$) of the SUT. Such a test case set is called a covering array in the form of $N \times n$ table. Each row of the table is representative of a test case. Covering array has been shown to be effective in detecting faults caused by interactions between all kinds of factors in the SUT.[11]

Definition 4 **τ-way covering array.** A τ-way covering array in the form of $N \times n$ table can be denoted as $MCA(N; \tau, n, (a_1, a_2, ..., a_n))$, where each row represents a test case and each column represents a parameter. The covering array requires that each potential

τ-degree schema must appear at least once in all τ columns. When $a_i = v$ for any i $(i = 1,2,\ldots,n)$, a τ-way covering array can be denoted as $CA(N;\tau,n,v)$.

In general, these covering arrays construction approaches can be divided into three categories according to construction strategies:

1. OTAT approach[19,20]: The OTAT method generates one test case at a time. Specifically, it starts with an empty initial test case, and then generates a complete test case and calculates the covered interactions. Finally, the process is repeated until all interactions are covered. The OTAT strategies can be divided into two types: Pure computational strategies and search strategies.
2. OPAT approach[21]: Unlike the OTAT method, OPAT focuses on one row of the covering array in each iteration. This approach does not first generate a complete test case but assigns values to some of the factors or parameters, i.e., one column of the covering array. Specifically, the OPAT approach starts with an initial test case that contains the smallest pair of the selected parameters, and then a parameter is iteratively added until all parameters are covered. After obtaining a complete test case, the above process is repeated to generate new test cases, thereby ensuring maximum interaction coverage.
3. A set of test cases one time approach: This approach generates a set of test cases in each iteration. The coverage is optimized by changing the values of some parameters of some test cases in this test set each time. When the desired interaction space is finally fully covered, it tries to remove the test cases to avoid the situation where fewer test cases still satisfy the coverage. Otherwise, it continues to generate test cases until covering all schemas.[22]

2.2 | Fuzzing

Fuzzing ranks as one of the most popular software testing techniques used to evaluate the SUT concerning security and robustness properties.[23–25] Typically, fuzzing detects anomalies by providing malformed input to the SUT and monitoring the execution state, which can be identified by information such as memory and security oracles. According to the dependence degree of program analysis on the program source code, fuzzers can be divided into white-box, gray-box, and black-box.[26] Among them, coverage-guided greybox fuzzing (CGF)[27,28] has been regarded as one of the most successful methods. The process of CGF consists of five main phases: Test case generation phase, test case mutation phase, test case execution phase, execution status monitoring phase, and an anomaly detection phase.

CGF starts with a bunch of initial seeds that are loaded into the seed poor and executed. And then, the test cases from seed poor are mutated in the next stage and the mutated test cases are executed. Fuzzer monitors the execution status of the target program during execution and decides whether to add the mutated test cases to the test queue based on the code coverage. The above process is iterated until predefined conditions are met, for example, upper test time bound and coverage rate. The execution of test cases may trigger abnormal behavior or crashes of the SUT.

3 | FUZZY COMBINATORIAL TESTING APPROACH

In this section, we describe the proposed CTAF in detail. Based on the fact of CT that more different τ-degree schemas imply more interaction space covered, we design the CTAF technique to ensure that each schema in the nonequilibrium model has a fair chance of composing all the desired combinations of options, thereby CTAF efficiently exploring a wide range of interaction space on the factors with regard to the behavior of the target program.

3.1 | Nonequilibrium IPM Effect

A traditional input parameter model can be denoted as a four tuple, namely, $IPM_{SUT}(P,V,R,C)$. P represents the parameters related to the SUT, which can influence the behavior of the software. V represents the values of all parameters from P, which are usually obtained by equivalence partitioning (EP)[29] and boundary value analysis (BVA).[29] R represents the interaction between parameters. C represents the constraints between different parameters from the domain semantics, thereby removing the test cases that don't make sense.

Definition 5 NParameters-NValues Pair. For the IPM of a SUT, namely, $IPM_{SUT}(P,V,R,C)$, NParameters-NValues pair (NP-NVP) represent a pair, where one side is the number of parameters $P_w(0 < w \le n)$ and the other side is the number of values corresponding to the parameter from P_w. In addition, all the number of values of the parameter from P_w are consistent, that is, $|V_1| = \ldots = |V_w| = a_w$. Therefor, an NP-NVP can be denoted as $NP - NVP_{(|P_w|,a_w)}$.

In the case of the *Tcas* program in Table 1, we can figure out four NP-NVPs contained in its IPM, including $NP-NVP_{(7,2)}$, $NP-NVP_{(2,3)}$, $NP-NVP_{(1,4)}$, and $NP-NVP_{(2,10)}$.

Definition 6 **Nonequilibrium IPM.** Based on the definition of traditional IPM, nonequilibrium IPM represents a type of IPM that contains at least two NP-NVPs. In addition, the extent of imbalance is indicated by the association between all NP-NVPs contained in the IPM, rather than the number of NP-NVPs.

For example, assume that there are models $A=2^1\times3^2\times4^2$ and $B=2^9\times8^1$. In this case, although A contains more NP-NVPs, the difference in the number of their parameter values is weak and the parameters with different numbers of values are uniformly distributed. On the contrary, although B has only two NP-NVPs, the difference in the number of parameter values between these two pairs is large, and the parameters with fewer values account for 90% of the total parameters. Therefore, we consider B to be more unbalanced than A.

Definition 7 **Nonequilibrium IPM effect.** Assume that the same CT generation algorithm is used to generate the test sets for the SUT with different input parameters models. The percentage of redundant test cases generated for the SUT with nonequilibrium models is larger than that with equilibrium models. That is, the nonequilibrium model leads to a more severe redundancy in the corresponding test set. Such an effect is called nonequilibrium IPM effect.

Our investigation discovered that the nonequilibrium IPM effect not only exists, but its effect depends on the degree of imbalance. Therefore, taking the nonequilibrium IPM effect into account when generating test cases, we believe that remaking coverage standards as τ-sway is reasonable and efficient. In addition, together with the generation mechanism of CTAF, which decreases the number of times more schema are covered by multiple test cases, our approach helps to alleviate the imbalance model-induced test set redundancy problem.

3.2 | τ-sway covering array

Definition 8 **τ-sway covering array.** Suppose that a SUT has n parameters and $m(0<m\le n)$ NP-NVPs contained in the relevant IPM. Let $p_u(0<u\le m)$ be the number of parameters of uth NP-NVP, $a_u(0<u\le m)$ be the number of values of uth NP-NVP, and assume that the sequence $sortedSeq\_A=(a_{1_1},a_{1_2},...,a_{1_{p_1}},...,a_{u_1},...,a_{u_{p_u}},...,a_{m_1},...,a_{m_{p_m}})$ are arranged from small to large. τ-sway covering array is a covering array where each column refers to elements in v_i(i.e., the set of values of the parameter p_i), while each row refers to a combination of values of n parameters, and the number of rows must be $sortedSeq\_A(0)\times...\times sortedSeq\_A(\tau))$ ($sortedSeq\_A(i)$ represents the ith element in $sortedSeq\_A$).

As mentioned before, with respect to the difference in the number of NP-NVPs, IPM can be divided into nonequilibrium models and equilibrium models. Thus, ignoring the differences between these two models when generating test cases is incorrect for reducing the redundancy of test sets.

In the design of CTAF, the maximum size of generated test sets for SUT follows τ-sway coverage. We already know that the imbalance effect is the result of the difference between the NP-NVPs included in the IPM. For example, assume that a SUT has three parameters p_1,p_2,p_3 with two values $(0,1)$ and one parameter p_4 with 10 values $(0,1,2,3,4,5,6,7,8,9)$. And there is one bug in the SUT, that is, $p_1=0$. When we comply with 1-way coverage, the size of test set is 10, and the test set may be $\{\{0,1,1,0\},\{1,1,1,1\},\{1,1,1,2\},\{1,1,1,3\},\{1,1,1,4\},\{1,1,1,5\},\{1,1,1,6\},\{1,1,1,7\},\{1,1,1,8\},\{1,1,1,9\}\}$. However, only the test case $\{0,1,1,0\}$ can detect the bug, while the remaining nine test case are useless. However, if we comply with 1-sway coverage, the size of test set is 2, and the test set may be $\{\{0,1,1,0\},\{1,1,1,1\}\}$. There is only one redundant test case. Therefore, τ-sway is more concerned with the valid but redundant test cases in the nonequilibrium models than τ-way. For example, τ-sway avoids generating valid but redundant test cases that can detect a bug repeatedly, while τ-way may generate a large number of valid test cases to detect the same bug. That is, τ-way only focuses on the validity of test cases and ignores the redundancy. Note that compliance with either of these coverage criteria may fail to produce test cases that can detect faults.

However, although τ-sway coverage can reduce the redundant test cases compared with τ-way coverage, it also reduces the probability of generating test cases that can detect faults. To avoid this shortcoming, our approach uses the feedback results from the previous test to guide the generation of subsequent test cases. Specifically, CTAF decreases the times an already covered schema has been covered by subsequent test cases again through collecting a variety of feedback-driven schema. In this way, test case diversity can be maintained while reducing the redundant test cases, so as to improve accuracy in detecting faults.

3.3 | Workflow of CTAF

3.3.1 | Overview

Figure 1 illustrates the overall design and workflow of CTAF. The main contributing sections are highlighted in gray, and the other processes (including input parameter modeling and initial seed generation) use the traditional CT process. The proposed CTAF utilizes a lightweight framework to collect a variety of feedback-driven schemas to guide the next test case generation, thereby covering more interactions of factors related to the SUT.

The progress of CTAF is also shown in Algorithm 1, which can be summarized as follows: **Step 1:** Load the initial test case set into the test queue (line 1); **Step 2:** Execute all test cases in the queue (line 2); **Step 3:** Generate new test cases by mutation based on the execution results of previous test cases (line 5); **Step 4:** Determine the next test case to be added to the test queue based on the feedback-driven schema and execute the test case (line 6); **Step 5:** Repeat Step 2 to Step 4 until the pre-configured conditions are met (lines 4–9). Such an iterative process, in which the execution results of the earlier test cases provide guidance for the variation and generation of the later test cases, reflects the characteristics of traditional fuzzing. Each component of CTAF is described as follows.

3.3.2 | Initial seeds

The most important aspect of seed generation is diversity and size. For each parameter p_i among the IPM of the SUT, p_i has $a_i(0 < i \le n)$ values. We then store all $a_i(0 < i \le n)$ in an array from small to large. The array can be denoted as $(a_{b_1}, a_{b_2}, ..., a_{b_n})$. Assume that the test intensity in this case is $\tau(0 < \tau \le n)$. Consequently, the upper bound on the size of the seed set can be computed as $a_{b_1} \times ... \times a_{b_\tau}/2$, while the lower bound is initialized to one. The diversity of the initial seeds is measured in terms of the diversity of n-degree schema. That is, the generation of duplicate initial seeds is avoided.

3.3.3 | Mutation

CTAF performs two kinds of mutations to generate new test cases: (1) Mutate the current input by modifying k-degree schemas ($0 < k \le \tau$), that is, subschemas of τ-degree schemas; (2) generate new test cases that must satisfy the condition that at least one k-degree schema ($\tau < k \le n$), that is, superschema of τ-degree schemas, is not included in any executed test case.

3.3.4 | Analysis

Based on the previous execution results, CTAF counts three types of τ-degree schema for the generation of the next test case: (1) The schema that has already appeared; (2) the schema that has not yet found any fault; (3) the schema that has found any fault. The test case that contains most of the first and the second schemas on the target program is the next.

3.3.5 | Execute

This stage executes generated test cases, as well as monitors the execution status and execution results. In general, when τ-sway coverage is met, the program does not continue to run.

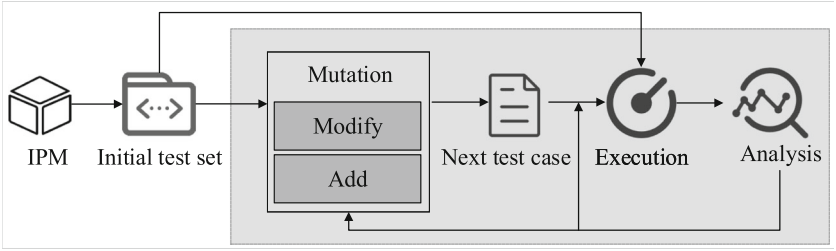

FIGURE 1 The workflow of CTAF

Algorithm 1 CTAF

Input: n parameters $p_1, p_2, \dots, p_n$, all sets of values of n parameters $V_1, V_2, \dots, V_n$, test intensity τ, target program S

Output: A test set Q

1: $Q \leftarrow$random_gen$(\tau, p_1, p_2, \dots, p_n, V_1, V_2, \dots, V_n)$
2: *Execute*(Q)
3: $Vsize \leftarrow (|V_{e_1}|, |V_{e_2}|, \dots, |V_{e_n}|)$, and $|V_{e_1}| < |V_{e_2}| < \dots < |V_{e_n}|$
4: **while do**$|Q| < Vsize[0] \times \dots \times Vsize[\tau]$
5: $M \leftarrow$mutate_gen$(\tau, p_1, p_2, \dots, p_n, V_1, V_2, \dots, V_n)$
6: $tc \leftarrow calculate(M, Q, \tau)$
7: *Execute*(tc)
8: $Q \leftarrow tc$
9: **end while**
10: **return** Q

4 | EMPIRICAL STUDIES

In this section, we conduct several empirical studies to investigate the impact of the types of input parameter models on test case generation in real software testing scenarios and to evaluate the performance of our proposed CTAF approach in dealing with this impact. Each study focused on a particular problem (research question), which are as follows:

1. *RQ1*: Does nonequilibrium IPM effect exist in real software?
2. *RQ2*: How well does our CTAF approach perform compared with traditional approaches on the SUT with NE-IPM?
3. *RQ3*: Does CTAF reduce the impacts of nonequilibrium IPM effect?

In the following experiments, we pay attention to three coverage criteria, namely, $\tau = 3,4,5$. In addition, to prevent randomness bias, we conducted each experiment 30 times and then calculate the average result. All experiments have been conducted on real subjects of six open source software systems, that is, Commons Cli, Commons Lang, Jflex, Jsoup, hsqldb, and Joda Time. All IPMs corresponding subjects were originally built by Nie et al,[30] and they are presented in the abbreviated form $values^{numberofparameters} \times \dots$. For example, $2^6 \times 3^2$ indicates the subject has six parameters that can take on two values, and only two parameters that can take on three values. Note that bug information of all the subjects in the experiments can be found on this website: https://github.com/JyC00/CTAF/blob/main/Buginfo.

4.1 | The existence and characteristics of nonequilibrium IPM effect

Out of rigor for the experiments used to verify whether the nonequilibrium IPM effect is indeed existent in practice, we select two well-established and successful tools in combination testing, namely, PICT and ACTS, to generate test cases for the above subjects. PICT[31] is a well-known CT tool developed by Microsoft based on the OTAT approach. It begins with choosing an uncovered test case, and then iteratively picks the optimal values for the parameters that do not interact in this test case, so as to cover the majority of the uncovered test cases as much as possible. ACTS[32] is another successful combinatorial test generation research tool that implements a series of OPAT-based IPO algorithms, namely, IPOG, IPOG-F, IPOG-F2, IPOG-D, and Base Choice. In this experiment, we use the IPOG version of ACTS. IPOG is a generic version of IPO that supports up to 6-way testing.

4.1.1 | Study setup

Subjects. In the first study, we chose four systems as subjects, that is, Commons Cli, Commons Lang, Jflex, and Jsoup, because these subjects possess different types of input parameter models so that we can draw experimental conclusions by comparing the redundancy of the test sets generated on these models. Table 2 lists information about all subjects, that is, the brief introduction (i.e., *info*), the version corresponding to each subject (i.e., *Version*), the number of lines of code (i.e., *LOC*), the bug's ids of each subject (i.e., *Bug#*), and the IPM of each subject (i.e., *IPM*).

TABLE 2 Subject programs of *RQ1*

Subjects	Info	Version	LOC	Bug#	IPM
Commons CLI	Parsing command line options passed to programs	1.3.1	6433	#265,#255	2^8
Commons Lang	Provides extra methods for manipulation of core classes.	3.2	61596	#1087,#1081	$2^6 \times 3^2$
Jflex	Lexical analyser generator	1.4.2	10745	#93,#98	$2^{11} \times 3^2 \times 4^1$
Jsoup	Java HTML Parser	1.9.1	10489	#611,#652	$2^6 \times 6^1$

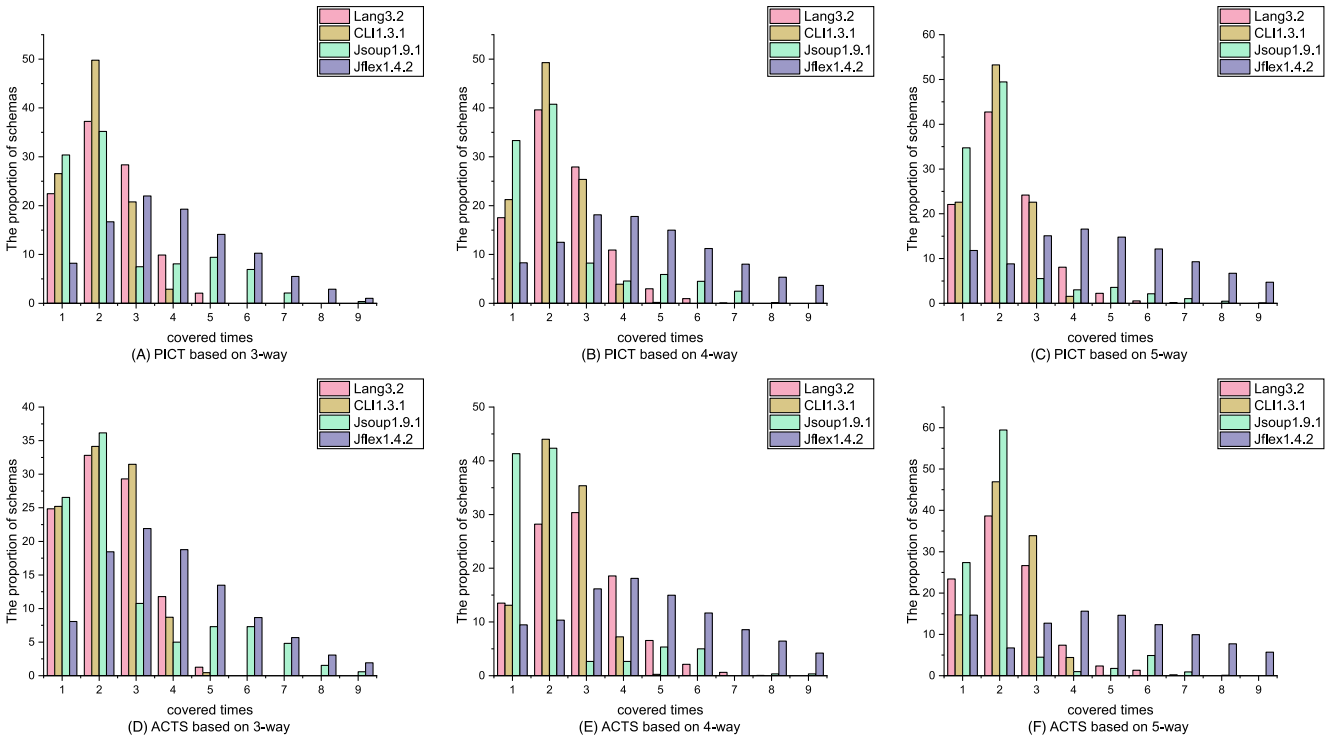

FIGURE 2 The redundancy of test cases of *RQ1*

Metric. To evaluate the level of redundancy in test case generation, we captured the times that each schema is covered and then calculated the proportion of the number of schemas to the sum of schemas under the same covered times. This metric provides a direct indication of the redundancy of the generated test cases, since it is evident that it is inefficient for the CT process to proceed if too many schemas are repeatedly covered by different test cases. That is, there is no need to examine a schema again with other test cases if they are covered and tested. However, since more test cases definitely result in more schemas being covered, thus, we calculate the percentage of schema under the consistent number of covered times to reflect the degree of redundancy in test case generation.

4.1.2 | Results

The experimental results are presented in Figure 2. The figure consists of six subplots (i.e., a–f), where subplots *a,b,c* are the experimental results of the tool PICT under the test criterion 3-way to 5-way, and subplots *d,e,f* belong to the experimental results of ACTS under the same test criterion as PICT. For each subplot, the *x*-axis indicates the total times that a schema is covered, and the *y*-axis indicates the proportion of schemas covered. For example, in Figure 2(A), the four bars with horizontal coordinate equal to 1 indicate that the proportion of schemas covered once in the test set that is generated by PICT under 3-way is 22.45% in Lang 3.2, 26.56% in CLI 1.3.1, 30.38% in Jsoup 1.9.1, and 8.22% in Jflex 1.4.2, respectively.

As previously mentioned, the more schemas covered with low frequency, the less redundancy in the generated test cases. It can be easily found from Figure 2 that the Lang and CLI outperform Jsoup and Jflex in general, where the maximum of the relevant schemas of Lang and CLI

are repeatedly covered up to 8 times, while the relevant schemas of Jsoup and Jflex can be repeatedly covered up to nine times. In addition, the upper bound of the covered times of relevant schemas of CLI in two groups (shown in Figure 2(A,C)) is 4, while the upper bound of the remaining four groups is 5. The maximal covered times about Lang in two groups (shown in Figure 2(A,D)) is 5, in three groups (shown in Figure 2(B,C,F)) is 7, and in the remaining group (shown in Figure 2(E)) is 8. For Jsoup, the maximal covered times of Figure 2(B,F) are 8 while the maximal covered times is 9 in the other four subplots. For Jflex, the maximal covered times is 9 in all subplots. Thus, it can be seen that the likelihood of redundancy in the test set from low to high is CLI, Lang, Jsoup, and Jflex.

This result is consistent with the nonequilibrium IPM effect we found. According to the definition of NE-IPM, only the CLI 2^6 is the balanced model among these four subjects in Table 2; thus, the test set associated with CLI contains the least number of redundant test cases. It is clear that the model of Jflex $2^{11} \times 3^2 \times 4^1$ is more unbalanced than that of Lang $2^6 \times 3^2$. In addition, 25% parameters have three values and 75% parameters have two values in the model of Lang, while approximately 14.3% parameters have six values and 85.7% parameters have two values in the model of Jsoup $2^6 \times 6^1$; thus, the model of Jsoup is more unbalanced than that of Lang. For Jflex, 14.3% parameters have three values, 7.1% parameters have four values, and 78.6% have two values. Compared with Jsoup, Jflex has the same percent (14.3%) of parameters with three values, while the remaining 85.7% parameters in Jsoup have two values and 7.1% parameters have four values. Therefore, the sequence of the four subjects is CLI, Lang, Jsoup, and Jflex according to their imbalance degree from small to large.

Answer to *RQ1*: Nonequilibrium IPM effect exists in real software, and its impact on the generation of redundant test cases is limited by the imbalance degree of the model.

4.2 | Comparing our approach with traditional approaches

The second experiment is designed to compare the performance of our CTAF approach with four traditional approaches in terms of the size of the test set and the accuracy of detecting faults. To perform this study, we need to apply CTAF approach and four traditional CT approaches (without the interaction between generation and execution) based on four traditional generation algorithms and traditional CT process to generate test cases on four software (nine versions in total), and then evaluate their experimental results.

Since OPAT and OTAT are two most classic CT generation algorithms, hence, we select two OPAT-based approaches (IPOG, IPOF)[21] and two OTAT-based approaches (PICT,[31] T-reduction[33]) for the traditional CT process to compare with our CTAF approach. In the compared approaches, IPOG and IPOF are derived from the successful OPAT-based tool ACTS. PICT is one of the popular OTAT-based CT generation tools. And Test Vector Generator (TVG) strategy is also an OTAT-based τ-way testing tool; it extends an OTAT algorithm called AETG that uses incremental SAT solving to handle constraints. TVG supports three algorithms: T-reduced, Plus-one, and Random sets. Because a report[34] claims that the T-reduction algorithm often produces optimal results compared with the other two algorithms, therefore, we use the T-reduction version of TVG in the experiment. In this experiment, we focus on the problem of redundant test sets relevant to the SUT with NE-IPM effect; thus, the traditional CT process and our proposed CTAF approach do not introduce the phase of minimal failure-causing schema identification and evaluation.

4.2.1 | Study setup

Subjects. In the second study, we choose nine subjects from different versions of four systems, that is, hsqldb, Jflex, Joda Time, and Jsoup. All of these IPMs of nine subjects not only belong to NE-IPM but also have different degrees of disequilibrium. Table 3 lists the information about all

TABLE 3 Subject programs of *RQ2*.

Subjects	Info	Version	LOC	Bug#	IPM
hsqldb	Database management software written in pure Java	2.0rc8	139425	#1005,#981	$2^9 \times 3^2 \times 4^1$
hsqldb		2.2.5	156066	#1173,#1179	$2^8 \times 3^3$
hsqldb		2.2.9	162784	#1280,#1286	$2^8 \times 3^3$
Jflex	Lexical analyser generator	1.4.1	10040	#80,#87	$2^{10} \times 3^2 \times 4^1$
Jflex		1.4.2	10745	#93,#98	$2^{11} \times 3^2 \times 4^1$
Joda	De facto standard date and time library for Java	2.3	82158	#77,#86	$2^6 \times 3^2 \times 4^1$
Joda		2.8.2	85026	#296,#297	$2^5 \times 3^2 \times 5^1$
Joda		2.9.1	85512	#347,#361	$2^3 \times 3^3 \times 4^1$
Jsoup	Java HTML Parser	1.9.1	10489	#611,#652	$2^6 \times 6^1$

subjects, that is, the brief introduction (i.e., *info*), the version corresponding to each subject (i.e., *Version*), the number of lines of code (i.e., *LOC*), the bug's ids of each subject (i.e., *Bug#*), and the IPM of each subject (i.e., *IPM*).

Metric. In combinatorial testing, the diversity of test cases guarantees an adequate search of the interaction space that affects software behavior; however, the diversity is often accompanied by a larger size of the test set. An efficient CT process requires the test set as small in size as possible without losing diversity; therefore, the quality of a combinatorial test set should be measured in two dimensions, that is, the size of the test set and the ratio of faults detection. The ratio of faults detection refers to the percentage of test cases that detect software defects in the test set.

Note that we consider any exceptional execution of test cases to be a fault, which may be a thrown exception, a compilation error, an assertion failure, a constraint violation, and so forth.

4.2.2 | Results

Tables 4–6 show the size of generated test sets (i.e., *Set*), the ratio of faults detection (i.e., *Bug%*) for each approach on all subjects under different coverage criteria (3-way, 4-way, 5-way for IPOG, IPOF, PICT, and TVG and 3-sway, 4-sway, 5-sway for CTAF), respectively. The bold values in Tables 4–6 represent the optimal performanc among five algorithms under the same metric.

Size of test set. From a horizontal view, when the test intensity is 3, the average size of the test set on all nine subjects generated by the four traditional algorithms is approximately 5.26 times that of CTAF, which is similar when the test intensity is 5. And when the test intensity is 4, the average size of the test set generated by four traditional algorithms on these nine subjects is approximately 6.46 times that of CTAF. From a vertical perspective, it is known that the size of the test set increases with the intensity of the test. In the case of hsqldb 2.0rc8, when the test intensity increases from 3 to 4, the test cases generated by IPOF grow almost 2.62 times. When the test intensity increases from 3 to 4, the average

TABLE 4 The execution result under 3-way and 3-sway.

	IPOF		IPOG		PICT		TVG		CTAF	
Subject	Set	Bug%	Set	Bug%	Set	Bug%	Set	Bug%	Set	Bug%
hsqldb2.0rc8	45	35.6%	45	35.6%	45	31.1%	47	31.3%	**8**	**37.5%**
hsqldb225	35	48.6%	36	50.0%	35	48.6%	35	47.2%	**8**	**54.2%**
hsqldb229	35	48.6%	36	47.2%	35	42.9%	36	45.9%	**8**	**50.0%**
Jflex1.4.1	36	**77.8%**	40	67.5%	38	73.7%	38	69.2%	**8**	73.9%
Jflex1.4.2	45	46.7%	47	**59.6%**	46	54.3%	48	44.9%	**8**	58.3%
Joda2.3	42	26.2%	43	32.6%	43	32.6%	43	**34.1%**	**8**	33.3%
Joda2.8	50	46.0%	49	53.1%	49	46.9%	55	55.4%	**8**	**58.3%**
Joda2.9	41	36.6%	45	31.1%	46	34.8%	47	33.3%	**8**	**45.8%**
Jsoup1.9	39	64.1%	42	71.4%	39	74.4%	38	69.2%	**8**	**75.0%**

TABLE 5 The execution result under 4-way and 4-sway.

	IPOF		IPOG		PICT		TVG		CTAF	
Subject	Set	Bug%	Set	Bug%	Set	Bug%	Set	Bug%	Set	Bug%
hsqldb2.0rc8	118	32.2%	124	33.1%	119	34.5%	119	33.6%	**16**	**35.4%**
hsqldb225	91	51.6%	94	50.0%	93	50.5%	90	47.8%	**16**	**58.3%**
hsqldb229	91	47.3%	94	54.3%	91	51.6%	89	49.4%	**16**	**58.3%**
Jflex1.4.1	98	75.5%	117	72.6%	104	76.0%	101	70.3%	**16**	**79.2%**
Jflex1.4.2	125	50.4%	132	43.2%	125	**52.0%**	128	44.5%	**16**	51.1%
Joda2.3	111	**37.8%**	100	33.0%	109	31.2%	110	36.4%	**16**	37.5%
Joda2.8	128	50.8%	120	50.0%	129	51.9%	135	51.9%	**16**	**52.1%**
Joda2.9	118	33.9%	112	33.0%	116	33.6%	136	31.6%	**24**	**34.7%**
Jsoup1.9	80	76.3%	72	75.0%	80	76.3%	82	75.6%	**16**	**81.3%**

TABLE 6 The execution result under 5-way and 5-sway.

	IPOF		IPOG		PICT		TVG		CTAF	
Subject	**Set**	**Bug%**	**Set**	**Bug%**	**Set**	**Bug%**	**Set**	**Bug%**	**Set**	**Bug%**
hsqldb2.0rc8	288	34.7%	334	33.5%	305	32.8%	293	34.5%	**32**	**37.5%**
hsqldb225	220	48.6%	239	**50.2%**	221	48.4%	218	48.6%	**32**	49.8%
hsqldb229	220	48.6%	239	**51.0%**	217	48.8%	219	48.4%	**32**	49.0%
Jflex1.4.1	241	75.5%	258	74.0%	249	75.9%	245	74.3%	**32**	**77.1%**
Jflex1.4.2	317	51.4%	356	49.4%	326	47.9%	321	50.5%	**32**	**53.1%**
Joda2.3	247	34.4%	283	33.2%	243	34.2%	253	32.8%	**32**	**39.6%**
Joda2.8	284	49.6%	310	48.1%	290	50.7%	291	48.8%	**32**	**51.0%**
Joda2.9	246	33.3%	220	33.6%	283	33.6%	287	31.7%	**72**	**36.6%**
Jsoup1.9	170	75.3%	168	73.2%	159	73.6%	165	**77.0%**	**32**	76.0%

TABLE 7 Subject Programs of *RQ3*.

Subjects	Info	Version	LOC	Bug#	IPM
hsqldb	Database management software written in pure Java	2.0rc8	139425	#1005,#981	$2^9 \times 3^2 \times 4^1$
Jflex	Lexical analyser generator	1.4.2	10745	#93,#98	$2^{11} \times 3^2 \times 4^1$
Joda	De facto standard date and time library for Java	2.8.2	85026	#296,#297	$2^5 \times 3^2 \times 5^1$
Jsoup	Java HTML Parser	1.9.1	10489	#611,#652	$2^6 \times 6^1$

growth of test sets generated by IPOF for nine subjects is 2.61 times, 2.52 times for IPOG, 2.57 times for PICT and 2.55 times for TVG on average. When the test intensity increases from 4 to 5, the average growth of test sets generated by IPOF for nine subjects is 2.32 times, 2.49 times for IPOG, 2.36 times for PICT and 2.32 times for TVG on average. However, for CTAF, the test sets on nine subjects increase by 2.11 times on average, whether the test intensity grows from 3 to 4 or from 4 to 5.

Ratio of fault detection. When the test intensity is 3 and 5, CTAF has a higher detection rate of faults on six subjects than that of the other four algorithms. When the test intensity is 4, CTAF has a higher detection rate of faults on seven subjects than that of the other four algorithms. In the remaining eight scenes, when the rate of defect detection of CTAF trails the traditional algorithms, the ratio of defect detection of CTAF is slightly below the best algorithm under eight scenes, that is, ranks second.

Answer to *RQ2*: Our approach CTAF requires fewer test cases than four typical approaches (IPOG, IPOF, PICT, TVG), and achieves better detection accuracy of faults.

4.3 | Nonequilibrium IPM Effect on CTAF

The former experiment has demonstrated that nonequilibrium IPM effect does occur in practice, as well as CTAF outperforms four other conventional approaches in terms of the size of the test set and the detection accuracy of faults. However, the most prominent task for CTAF is to mitigate the nonequilibrium effect. Accordingly, this experiment is performed to examine the effectiveness of CTAF in processing nonequilibrium IPM effect under various disequilibrium input parameter models.

Since both this experiment and the first experiment are associated with yielding redundancy in the test set, thus, we take the same metrics in this experiment as the first experiment, i.e., the proportion of schemas under different covered times. And then, our experimental results are compared with each of the four traditional algorithms (IPOG, IPOF, PICT, TVG) to evaluate whether CTAF solves the problem explored in *RQ3*.

4.3.1 | Study setup

Subjects. The experiment relevant to *RQ2* has proved that our approach CTAF requires fewer test cases than four typical approaches (IPOG, IPOF, PICT, TVG), and achieves better detection accuracy. Therefore, we choose one version of each of the four software in *RQ2* as four subjects in the experiment relevant to *RQ3*. In Table 7, we list the brief introduction (i.e., *info*), the version corresponding to each subject (i.e., *Version*), the number of lines of code (i.e., *LOC*), the bug's ids of each subject (i.e., *Bug#*), and the IPM of each subject (i.e., *IPM*).

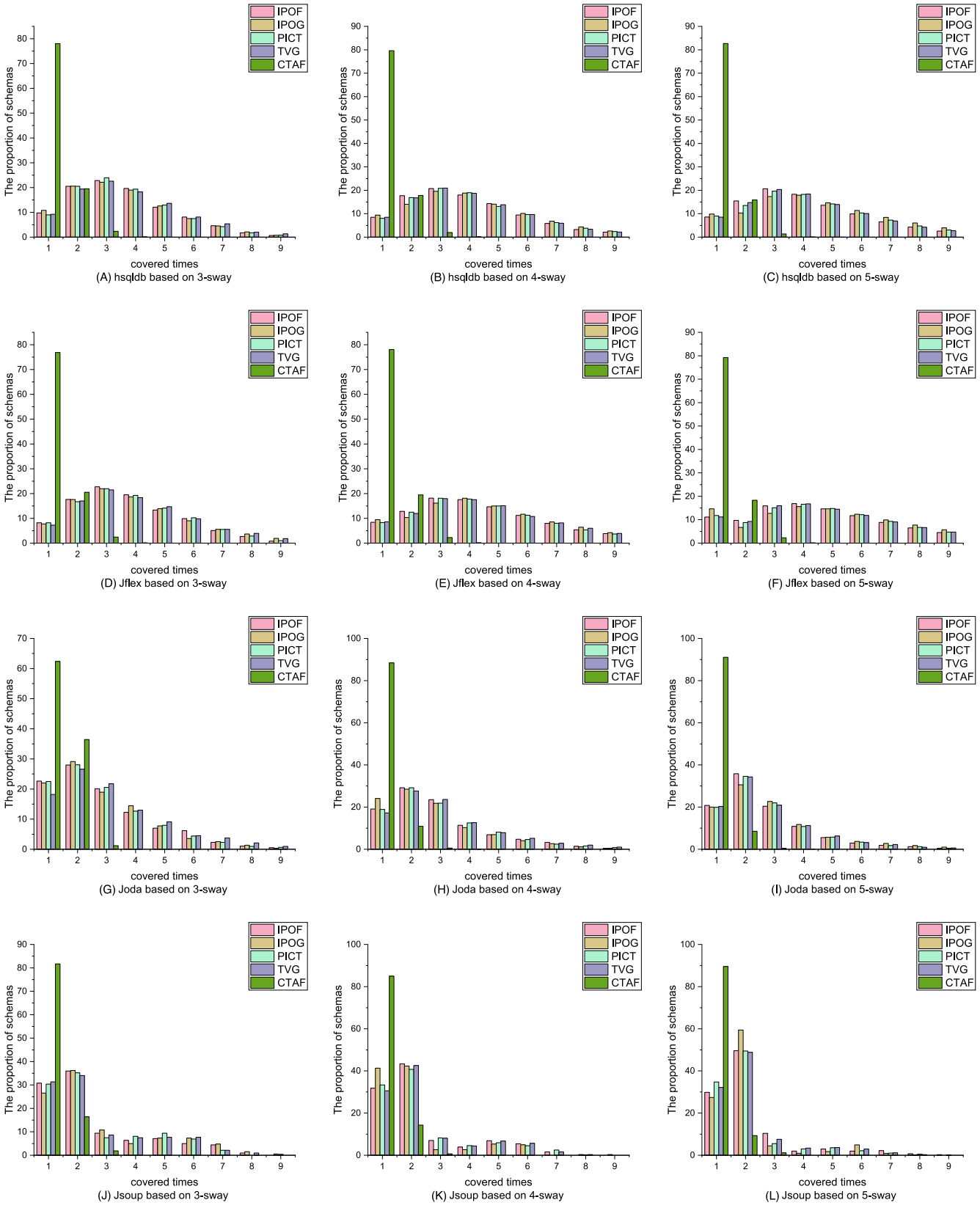

FIGURE 3 The redundancy of test cases of *RQ3*

4.3.2 | Results

The experimental results are shown in Figure 3. The graph consists of 12 subplots, where each row of the subplot corresponds to the experimental results for each subject under the different test criteria (3-sway, 4-sway, and 5-sway). For each subplot, the *x*-axis indicates the total covered times of a schema and the *y*-axis indicates the proportion of schemas under these covered times.

We have already known that more schemas with lower frequency coverage mean that the test set contains less redundancy. That is, if this is an efficient test, the proportion of schemas (*y*-axis) decreases as the times of a schema is covered (*x*-axis) increases. It is quite easy to discover from Figure 3 that the performance of CTAF exceeds other four approaches in all 12 groups. In fact, the bars of CTAF approach decreases rapidly as the x-axis increases, while the trend gradually decreases from rising and then leveling off for the other four methods. Specifically, in the first group, for instance, the schema percentage of CTAF decreases as the number of coverage gradually increases, which are 78.02%, 19.53%, 2.36%, 0.09%, and 0.00%. The percentage of schemas of IPOF increases from 9.74% to 22.83% and then decreases to 0.66%. The percentage of schemas of IPOG increases from 10.80% to 22.17% and then decreases to 0.77%. The percentage of schemas of PICT increases from 8.97% to 23.92% and then decreases to 0.80%. The percentage of schemas of TVG increases from 9.23% to 22.58% and then decreases to 1.35%. Moreover, for these four subjects, at least 62.36% of the schemas in CTAF approach are covered only once on average under arbitrary coverage criteria. Therefore, for the SUT with nonequilibrium IPM, CTAF significantly outperforms other four approaches in terms of generating less redundant test cases.

It is worth mentioning that our CTAF approach has reached 62.36% of the schemas that are covered only once in all 12 scenarios, with the highest reaching 91.04%. In other words, most schemas generated by CTAF are covered once. The maximal percentage of schemas which are generated by other four traditional approaches for hsqldb and covered only once is 10.8% (shown in Figure 3(A)), 14.66% for Jflex (shown in Figure 3(F)), 24.07% for Joda (shown in Figure 3(G)), 41.33% for Jsoup (shown in Figure 3(K)). In addition, for the four traditional approaches, the largest percentage of all schemas is the one that is covered twice. This also shows that CTAF can better handle redundant test cases.

Answer to *RQ3*: CTAF is able to mitigate nonequilibrium IPM effect. In addition, CTAF also can efficiently generate diverse test cases for SUT with nonequilibrium IPM.

4.4 | Discussion

First, a rigorous experiment was implemented to demonstrate the nonequilibrium effect we observed. Second, to alleviate the test set redundancy problem caused by the nonequilibrium effect, we propose a new CT approach and experimentally demonstrate that it can mitigate this effect and improve the accuracy of the defect detection rate. However, our experiments still exists some validity threats. For example, one threat is the selection of traditional generation algorithms. Our experiments mainly involve purely computational-based OTAT strategies and OPAT strategies, search-based OTAT strategies such as Harmony search strategy (HSS)[34] to build coverage arrays are not experimented. Besides, CTAF does not perform well in terms of time overhead. This is because both CTAF and fuzzy tests are performed in actual execution, so CTAF has the same drawbacks as fuzzy tests with low efficiency. But at the same time, they also improve the accuracy of detecting defects. In actual testing, the harm of vulnerabilities is much greater than the consumption of testing resources, especially when the Internet is developing rapidly nowadays, and whether the application of artificial intelligence is safe is of great concern to society. It is worthwhile to trade time for a potential vulnerability. Therefore, we believe that for an efficient and complete testing process, the advantages of CTAF technology, i.e., improved defect detection accuracy, are more important than its disadvantages.

5 | RELATED WORK

In this section, we review both the existing CT work and fuzzing work and briefly review the existing work on test set reduction out of the consideration that our work aims to avoid redundancy in the test sets in CT.

5.1 | Combinatorial testing

A great number of researches[12,35,36] have been conducted on τ-way generation strategies; they can be classified into τ-way test generation for single input model and τ-way test generation for multiple input models according to different application scenarios. The effectiveness of CT depends on the coverage of all combinations of values among the parameters. τ-way strategies for single-input models aim to find solutions that efficiently construct a test case set that satisfies the coverage and constraints with minimum size.

Mathematical methods[37,38] usually extend existing mathematical structures or use recursive methods to construct the optimal test sets in theoretical under suitable circumstances. Mathematical methods impose specific requirements on the input parameter model and usually fail to handle constraints, thus, they are unappreciated in practical applications. The computational methods are made up of greedy algorithms and search algorithms. Greedy algorithms generate the desired τ-way covering array step by step by constructing the covering array one row or column at a time, namely one-test-at-a-time framework or one-parameter-at-a- time framework. This category of methods is usually simpler to implement and faster to compute, but it is very difficult to generate a covering array that is very advantageous in terms of scale. Under the one-test-at-a-time framework, each test should be able to cover as many combinations of uncovered values as possible, for instance, Automatic Efficient Test Generator (AETG),[39] the Deterministic Density Algorithm (DDA),[20] Pairwise Independent Combinatorial Testing tool (PICT),[31] the BDD (Binary Decision Diagram)-based approach,[40] the MDD (Multivalued Decision Diagram)-based approach.[41] The idea of one-parameter-at-a-time framework starts by generating a sub covering array for a portion of the parameters that meets the coverage requirements and then continuously adds new parameters until the final covering array is completed. The typical representative of this category of algorithm is In-Parameter-Order-General (IPOG),[21] and then many variants of IPOG have been derived, for example, IPOG-F and IPOG-D. Search-based methods[42,43] usually randomly construct a test set at first, followed by some heuristic or evolutionary optimization search strategies to find a τ-way test set as small as possible. This kind of methods can be applied to any test model as well as generate a covering array in small size, but the time consumption is very long.

5.2 | Fuzzing

Fuzzers can be classified as generation-based[44–46] and mutation-based.[27,28,47]

Generation-based fuzzers tend to make use of models or grammars to describe the desired format of the input,[48] so they can efficiently generate test cases that are grammatically checked by a syntax parser with a greater possibility of testing the deep code of the target program. However, if the target program is closed source with no public documentation, it can be a daunting task to analyze the file format. For this reason, a recent work has been proposed to impute input structures over an initial seed corpus by static analysis or machine learning.[49,50] Given the infinite input space, researchers have suggested leveraging execution feedback to help test case generation prevent the blind generation of inefficient test cases. For example, Apollo[51] measures the difference in execution time to support generated SQL queries.

Mutation-based fuzzers generate test cases by mutation of the initial input and the test cases generated during the fuzzing process. If a test case triggers a new execution path, it is deemed useful and kept for further mutations. In this way, the fuzzers can quickly dig deeper into the deep logic and explore more program states so that it may trigger bugs hidden in certain code regions.[17,24,52] AFL[53] uses coverage feedback as a guide and performs random bit-flip mutations. Since test cases generated by simple bit-flip mutations are hard to survive through complex checks such as phantom numbers, existing fuzzers[27,54,55] overcome this problem with symbolic execution or taint analysis. Driller[56] performs selective concolic execution, while QSYM[47] combines symbolic simulation with native execution. However, the above mentioned mutation-based fuzzers take the input structure into consideration. When highly structured inputs are required, their effectiveness decreases significantly because of incorrectly formatted inputs. To alleviate the above problem, higher level mutations than bits and bytes have also been proposed, for example, Fuzzill[57] designs custom intermediate representations (IRs) for mutation and semantic analysis in JavaScript.

5.3 | Test set reduction

The development of software functionalities and demands leads to more complicacy of software, which further triggers the growth tendency of the size of test suites and thereby resulting in the problem of the high cost of software testing.[58–60] As a result, numerous approaches have been proposed to reduce the test set. Test set reduction is conventionally performed upon a set of test demands, which is equivalent to an entity that must be covered, such as a statement, branch, or interaction. In essence, the original test set is substituted by a reduced test set to conduct the tests, while the reduced test set covers the same testing requirements as the original test set. The existing approaches used to reduce the test set can be classified into two main groups.

One group reduces the test sets via greedy algorithms.[61,62] Whenever the next test case that needs to be added to the reduced test set is selected, this test case must satisfy that it would cover the test requirements that are minimally covered by the current test set. In addition, some heuristics techniques are developed to optimize the framework, for example, covering the test requirements that are hard to cover.

Another group can be referred to post-optimization.[63] It is built on the principle that if a test case is dropped while the covered test demands remain unchanged, i.e., the set of covered test demands is the same as the original test set, then the test case can be concluded to be redundant in the original test set. Therefore, instead of adding test cases to a reduced test set, this category of algorithms aims to gradually remove the redundant test cases. The above two categories of approaches carry out the test set reduction in the test case generation phase or after the test set generation is completed. In our proposed approach, we analyze the input parameter modeling before the generation of the test set to stem the redundancy in the test set under the condition that it does not degrade the testing efficiency.

6 | CONCLUSION AND FUTURE WORK

Combinatorial testing is a widely used software testing technology. It places emphasis on the factors that affect software and the interaction between these factors. In this paper, We present an interesting phenomenon, namely, the nonequilibrium IPM effect. The effect refers to the fact that differences in the parameters and corresponding values included in the IPMs of SUT, i.e., different degrees of equilibrium IPM, tend to cause different degrees of redundancy in the combined test set. To solve the problem, we develop a new CT approach that is called CTAF, which handles nonequilibrium model effects efficiently with the feedback provided by the execution information of early test cases for subsequent test generation. In addition, we propose a new combinatorial coverage criterion τ-sway in this approach. Based on this coverage criterion, our approach generates significantly fewer redundant test cases while maintaining their diversity.

First, an empirical study on whether the nonequilibrium IPM effect is real is conducted for a total of four open source software. We analyze the characteristics of each software's IPM, and we find that the experimental results indicate that the degree of redundancy in the combinatorial test sets of the four types of software was consistent with the degree of disequilibrium in their respective IPMs. In the second experiment, an empirical study on the performance of CTAF and four other CT techniques based on traditional CT generation methods and CT processes is conducted on a total of six open source software containing nine subjects. The experimental results show that CTAF outperforms the comparison methods because it can achieve comparable, even higher, detection ratio of faults and use fewer test sets. Finally, an empirical study is conducted on four open source software on whether CTAF deals with nonequilibrium IPM effects. The experimental results show that CTAF can effectively mitigate the redundancy of the combined test cases due to the nonequilibrium IPM of SUT.

In the future, we intend to investigate the performance difference of our CTAF method on SUT with different sizes of IPMs using more software to see if our new CT approach can further reduce the time overhead and improve efficiency. Another interesting work is to extend the CTAF method to combine with masking effect techniques so that it can support a better quality of the minimum failure-causing schema identification.

ACKNOWLEDGMENTS

The project was supported by the National Natural Science Foundation of China (NSFC) (Grant nos. 62172194, 62202206, and U1836116), the National Key R&D Program of China (Grant no. 2020YFB1005500), the Leading-edge Technology Program of Jiangsu Natural Science Foundation (Grant no. BK20202001), the Natural Science Foundation of Jiangsu Province (Grant no. BK20220515), the China Postdoctoral Science Foundation (Grant no. 2021M691310), the Graduate Research Innovation Project of Jiangsu Province (Grant nos. SJCX21_1692, KYCX21_3375, and KYCX22_3670), and the Qinglan Project of Jiangsu Province.

DATA AVAILABILITY STATEMENT

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

ORCID

Jinfu Chen https://orcid.org/0000-0002-3124-5452
Jingyi Chen https://orcid.org/0000-0003-2668-6592
Saihua Cai https://orcid.org/0000-0003-0743-1156

REFERENCES

1. Perera A, Aleti A, Böhme M, Turhan B. Defect prediction guided search-based software testing. In: 2020 35th IEEE/ACM International Conference on Automated Software Engineering (ASE). IEEE; 2020:448-460.
2. Kuhn DR, Reilly MJ. An investigation of the applicability of design of experiments to software testing. In: 27th Annual NASA Goddard/IEEE Software Engineering Workshop, 2002. Proceedings. IEEE; 2002:91-95.
3. Kuhn DR, Wallace DR, Gallo AM. Software fault interactions and implications for software testing. *IEEE Trans Software Eng*. 2004;30(6):418-421.
4. Rao C, Li N, Lei Y, Guo J, Zhang Y, Kacker RN, Kuhn DR. Combinatorial test generation for multiple input models with shared parameters. *IEEE Trans Software Eng*. 2022;48(7):2606-2628.
5. Bryce RC, Colbourn CJ, Cohen MB. A framework of greedy methods for constructing interaction test suites. In: 27th International Conference on Software Engineering (ICSE 2005). ACM; 2005:146-155.
6. Wu H, Xu L, Niu X, Nie C. Combinatorial testing of RESTful APIs. In: ACM/IEEE International Conference on Software Engineering (ICSE). ASE; 2022:426-437.
7. Maehren M, Nieting P, Hebrok S, Merget R, Somorovsky J, Schwenk J. {TLS-Anvil}: adapting combinatorial testing for {TLS} libraries. In: 31st USENIX Security Symposium (USENIX Security 22) USENIX Association; 2022:215-232.
8. Adamo D, Nurmuradov D, Piparia S, Bryce RC. Combinatorial-based event sequence testing of android applications. *Inf Softw Technol*. 2018;99:98-117.
9. Simos DE, Bozic J, Garn B, Leithner M, Duan F, Kleine K, Lei Y, Wotawa F. Testing TLS using planning-based combinatorial methods and execution framework. *Softw Qual J*. 2019;27(2):703-729.
10. Henard C, Papadakis M, Perrouin G, Klein J, Heymans P, Traon YL. Bypassing the combinatorial explosion: using similarity to generate and prioritize t-wise test configurations for software product lines. *IEEE Trans Software Eng*. 2014;40(7):650-670.

11. Niu X, Nie C, Leung H, Lei Y, Wang X, Xu J, Wang Y. An interleaving approach to combinatorial testing and failure-inducing interaction identification. *IEEE Trans Software Eng*. 2020;46(6):584-615.
12. Nie C, Leung H. A survey of combinatorial testing. *ACM Comput Surv*. 2011;43(2):11:1-11:29.
13. Kuhn DR, Okun V. Pseudo-exhaustive testing for software. In: 30th Annual IEEE / NASA Software Engineering Workshop (SEW-30 2006). IEEE Computer Society; 2006:153-158.
14. Pang Q, Yuan Y, Wang S. MDPFuzz: testing models solving Markov decision processes. In: Proceedings of the 31st acm sigsoft international symposium on software testing and analysis ASE; 2022:378-390.
15. Lyu C, Liang H, Ji S, Zhang X, Zhao B, Han M, Li Y, Wang Z, Wang W, Beyah R. Slime: program-sensitive energy allocation for fuzzing. In: Proceedings of the 31st ACM Sigsoft International Symposium on Software Testing and Analysis ASE; 2022:365-377.
16. Huang H, Guo Y, Shi Q, Yao P, Wu R, Zhang C. BEACON: directed grey-box fuzzing with provable path pruning. In: 43rd IEEE Symposium on Security and Privacy, SP 2022 IEEE; 2022:36-50.
17. Nguyen HL, Grunske L. BEDIVFUZZ: integrating behavioral diversity into generator-based fuzzing. In: 44th IEEE/ACM 44th International Conference on Software Engineering, ICSE 2022 ACM; 2022:249-261.
18. Nie C, Leung H. The minimal failure-causing schema of combinatorial testing. *ACM Trans Softw Eng Methodol*. 2011;20(4):15:1-15:38.
19. Tung Y-W, Aldiwan WS. Automating test case generation for the new generation mission software system. In: 2000 IEEE Aerospace Conference. Proceedings (Cat. No. 00TH8484), Vol. 1. IEEE; 2000:431-437.
20. Bryce RC, Colbourn CJ. The density algorithm for pairwise interaction testing. *Softw Test Verification Reliab*. 2007;17(3):159-182.
21. Lei Y, Kacker R, Kuhn DR, Okun V, Lawrence J. IPOG/IPOG-D: efficient test generation for multi-way combinatorial testing. *Softw Test Verification Reliab*. 2008;18(3):125-148.
22. Nurmela KJ. Upper bounds for covering arrays by tabu search. *Discret Appl Math*. 2004;138(1-2):143-152.
23. Böhme M, Cadar C, Roychoudhury A. Fuzzing: challenges and reflections. *IEEE Softw*. 2021;38(3):79-86.
24. Wu M, Jiang L, Xiang J, Huang Y, Cui H, Zhang L, Zhang Y. One fuzzing strategy to rule them all. In: 44th IEEE/ACM 44th International Conference on Software Engineering, ICSE 2022 ACM; 2022:1634-1645.
25. Liu Y, Wang Y, Su P, Yu Y, Jia X. Instruguard: find and fix instrumentation errors for coverage-based greybox fuzzing. In: 36th IEEE/ACM International Conference on Automated Software Engineering, ASE 2021 IEEE; 2021:568-580.
26. Godefroid P. Fuzzing: hack, art, and science. *Commun ACM*. 2020;63(2):70-76.
27. Aschermann C, Schumilo S, Blazytko T, Gawlik R, Holz T. REDQUEEN: fuzzing with input-to-state correspondence. In: 26th Annual Network and Distributed System Security Symposium, NDSS 2019 The Internet Society; 2019:1-15.
28. Gan S, Zhang C, Chen P, Zhao B, Qin X, Wu D, Chen Z. GREYONE: data flow sensitive fuzzing. In: 29th USENIX Security Symposium, USENIX Security 2020 USENIX Association; 2020:2577-2594.
29. Myers GJ, Sandler C, Badgett T. *The Art of Software Testing*. John Wiley & Sons; 2011.
30. Niu X, Nie C, Lei JY, Leung H, Wang X. Identifying failure-causing schemas in the presence of multiple faults. *IEEE Trans Software Eng*. 2020;46(2):141-162.
31. Czerwonka J. Pairwise testing in real world. In: 24th Pacific Northwest Software Quality Conference, Vol. 200. Citeseer; 2006:1-12.
32. Borazjany MN, Yu L, Lei Y, Kacker R, Kuhn R. Combinatorial testing of ACTS: a case study. In: Fifth IEEE International Conference on Software Testing, Verification and Validation, ICST 2012 IEEE Computer Society; 2012:591-600.
33. Blackburn MR, Busser RD. T-VEC: a tool for developing critical systems. In: Proceedings of 11th Annual Conference on Computer Assurance. COMPASS'96 IEEE; 1996:237-249.
34. Alsewari ARA, Zamli KZ. Design and implementation of a harmony-search-based variable-strength t-way testing strategy with constraints support. *Inf Softw Technol*. 2012;54(6):553-568.
35. Khalsa SK, Labiche Y. An orchestrated survey of available algorithms and tools for combinatorial testing. In: 25th IEEE International Symposium on Software Reliability Engineering, ISSRE 2014 IEEE Computer Society; 2014:323-334.
36. Wu H, Nie C, Petke J, Jia Y, Harman M. A survey of constrained combinatorial testing. CoRR abs/1908.02480; 2019.
37. Hartman A, Raskin L. Problems and algorithms for covering arrays. *Discret Math*. 2004;284(1-3):149-156.
38. Moura L, Raaphorst S, Stevens B. Upper bounds on the sizes of variable strength covering arrays using the Lovász local lemma. *Theor Comput Sci*. 2019;800:146-154.
39. Cohen DM, Dalal SR, Fredman ML, Patton GC. The AETG system: an approach to testing based on combinatorial design. *IEEE Trans Software Eng*. 1997;23(7):437-444.
40. Segall I, Tzoref-Brill R, Farchi E. Using binary decision diagrams for combinatorial test design. In: Proceedings of the 20th International Symposium on Software Testing and Analysis, ISSTA 2011 ACM; 2011:254-264.
41. Gargantini A, Vavassori P. Efficient combinatorial test generation based on multivalued decision diagrams. In: Hardware and software: Verification and testing - 10th International Haifa Verification Conference, HVC 2014, Lecture Notes in Computer Science, vol. 8855 Springer; 2014:220-235.
42. Torres-Jimenez J, Avila-George H, Izquierdo-Marquez I. A two-stage algorithm for combinatorial testing. *Optim Lett*. 2017;11(3):457-469.
43. Esfandyari S, Rafe V. A tuned version of genetic algorithm for efficient test suite generation in interactive t-way testing strategy. *Inf Softw Technol*. 2018;94:165-185.
44. Jiang B, Liu Y, Chan WK. Contractfuzzer: fuzzing smart contracts for vulnerability detection. In: Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering, ASE 2018 ACM; 2018:259-269.
45. Pham V-T, Böhme M, Roychoudhury A. Model-based whitebox fuzzing for program binaries. In: Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering, ASE 2016 ACM; 2016:543-553.
46. Veggalam S, Rawat S, Haller I, Bos H. Ifuzzer: An evolutionary interpreter fuzzer using genetic programming. In: Computer security - ESORICS 2016 - 21st European Symposium on Research in Computer Security, Lecture Notes in Computer Science, vol. 9878 Springer; 2016:581-601.
47. Yun I, Lee S, Xu M, Jang Y, Kim T. QSYM: a practical concolic execution engine tailored for hybrid fuzzing. In: 27th USENIX Security Symposium, USENIX Security 2018 USENIX Association; 2018:745-761.
48. He X, Xie X, Li Y, Sun J, Li F, Zou W, Liu Y, Yu L, Zhou J, Shi W, Huo W. SoFi: reflection-augmented fuzzing for JavaScript engines. In: CCS '21: 2021 ACM SIGSAC Conference on Computer and Communications Security ACM; 2021:2229-2242.

49. Xie D, Li Y, Kim M, Pham HV, Tan L, Zhang X, Godfrey MW. DocTer: documentation-guided fuzzing for testing deep learning API functions. In: ISSTA '22: 31st ACM SIGSOFT International Symposium on Software Testing and Analysis ACM; 2022:176-188.
50. Bastani O, Sharma R, Aiken A, Liang P. Synthesizing program input grammars. *ACM SIGPLAN Notices*. 2017;52(6):95-110.
51. Jung J, Hu H, Arulraj J, Kim T, Kang W-H. APOLLO: automatic detection and diagnosis of performance regressions in database systems. *Proc VLDB Endow*. 2019;13(1):57-70.
52. Liang J, Wang M, Zhou C, Wu Z, Jiang Y, Liu J, Liu Z, Sun J. PATA: fuzzing with path aware taint analysis. In: 43rd IEEE Symposium on Security and Privacy, SP 2022 IEEE; 2022:1-17.
53. Zalewski M. AFL (American Fuzzy Lop). https://github.com/google/AFL. Accessed June 13, 2022; 2013.
54. Chen P, Chen H. Angora:efficient fuzzing by principled search. In: 2018 IEEE Symposium on Security and Privacy, SP 2018 IEEE Computer Society; 2018:711-725.
55. Rawat S, Jain V, Kumar A, Cojocar L, Giuffrida C, Bos H. VUzzer: application-aware evolutionary fuzzing. In: 24th Annual Network and Distributed System Security Symposium, NDSS 2017 The Internet Society; 2017.
56. Stephens N, Grosen J, Salls C, Dutcher A, Wang R, Corbetta J, Shoshitaishvili Y, Kruegel C, Vigna G. Driller: augmenting fuzzing through selective symbolic execution. In: 23rd Annual Network and Distributed System Security Symposium, NDSS 2016 The Internet Society; 2016.
57. Groß S. Fuzzil: Coverage guided fuzzing for JavaScript engines. Department of Informatics, Karlsruhe Institute of Technology; 2018.
58. Chen J, Wang X, Cai S, et al. A software defect prediction method with metric compensation based on feature selection and transfer learning. *Front Inf Technol Electron Eng*. 2022;23(5):715-731.
59. Chen J, Chen H, Guo Y, et al. A novel test case generation approach for adaptive random testing of object-oriented software using k-means clustering technique. *IEEE Trans Emerg Top Comput Intell*. 2021;6(4):969-981.
60. Omari M, Chen J, French-Baidoo R, et al. A proactive approach to test case selection—an efficient implementation of adaptive random testing. *Int J Softw Eng Knowl Eng*. 2020;30(8):1169-1198.
61. Harrold MJ, Gupta R, Soffa ML. A methodology for controlling the size of a test suite. *ACM Trans Softw Eng Methodol (TOSEM)*. 1993;2(3):270-285.
62. Blue D, Segall I, Tzoref-Brill R, Zlotnick A. Interaction-based test-suite minimization. In: 35th International Conference on Software Engineering, ICSE '13 IEEE Computer Society; 2013:182-191.
63. Nguyen CD, Marchetto A, Tonella P. Combining model-based and combinatorial testing for effective test case generation. In: International Symposium on Software Testing and Analysis, ISSTA 2012 ACM; 2012:100-110.

How to cite this article: Chen J, Chen J, Cai S, Chen H, Zhang C. A novel combinatorial testing approach with fuzzing strategy. *J Softw Evol Proc*. 2023;35(12):e2537. doi:10.1002/smr.2537