

An interleaving approach to combinatorial testing and failure-inducing interaction identification

Xintao Niu,anghai Nie, Member, IEEE, Hareton Leung, Member, IEEE, Jeff Y. Lei, Member, IEEE, Xiaoyin Wang, Member, IEEE, JiaXi Xu and Yan Wang

Abstract—Combinatorial testing(CT) seeks to detect potential faults caused by various interactions of factors that can influence the software systems. When applying CT, it is a common practice to first generate a set of test cases to cover each possible interaction and then to identify the failure-inducing interaction after a failure is detected. Although this conventional procedure is simple and forthright, we conjecture that it is not the ideal choice in practice. This is because 1) testers desire to identify the root cause of failures before all the needed test cases are generated and executed 2) the early identified failure-inducing interactions can guide the remaining test case generation, such that many unnecessary and invalid test cases can be avoided. For these reasons, we propose a novel CT framework that allows both generation and identification process to interact with each other. As a result, both generation and identification stages will be done more effectively and efficiently. We conducted a series of empirical studies on several open-source software, the results of which show that our framework can identify the failure-inducing interactions more quickly than traditional approaches, while requiring fewer test cases.

Index Terms—Software Testing, Combinatorial Testing, Covering Array, Failure-inducing interactions

1 INTRODUCTION

Modern software is becoming more and more complex. To test such software is challenging, as the candidate factors that can influence the system's behaviour, e.g., configuration options, system inputs, message events, are enormous. Even worse, the interactions between these factors can also crash the system, e.g., the incompatibility problems. In consideration of the scale of the industrial software, to test all the possible interactions of all the factors (we call them the interaction space) is not feasible, and even if it is possible, it is resource-inefficient to test all the interactions.

Many empirical studies show that, in real software systems, the effective interaction space, i.e., targeting fault detection, makes up only a small proportion of the overall interaction space [1], [2]. Further, the number of factors involved in these effective interactions is relatively small, of which 4 to 6 is usually the upper bounds [1]. With this observation, applying Combinatorial testing(CT) in practice is appealing, as it is proven to be effective to detect the interaction faults in the system.

CT tests software with an elaborate test suite which checks all the required parameter value combinations. A typical CT life-cycle is shown in Figure 1, which contains four main testing stages. At the very beginning of the testing, engineers should extract the specific model of the software under test (SUT). In detail, they should identify the factors, such as user inputs, and configure options, that could affect the system's behavior. Further effort is required to figure out the constraints and dependencies between each factor and corresponding values for valid testing. After the modeling stage, a set of test cases should be generated and executed to expose the potential faults in the system. In CT, each test case is a set of assignments of all the factors in the test model. Thus, when such a test case is executed, all the interactions contained in the test case are deemed to be checked. The main target of this stage is to design a relatively small set of test cases to achieve some specific coverage. The third testing stage in this cycle is the fault localization, which is responsible for identifying the failure-inducing interactions. To characterize the failure-inducing interactions of corresponding factors and values is important for future bug fixing, as it will reduce the scope of suspicious code to be inspected. The last testing stage of CT is evaluation. In this stage, testers will assess the quality of the previously conducted testing tasks. If the assessment result shows that the previous testing process does not fulfil the testing requirement, some testing stages should be improved, and sometimes, may even need to be re-conducted.

Although this conventional CT framework is simple and straightforward, in terms of the test case generation and fault localization stages, we conjecture that first-generation-then-identification is not the proper choice in practice. The

- Xintao Niu and Changhai Nie are with the State Key Laboratory for Novel Software Technology, Nanjing University, China, 210023.
E-mail: niuxintao@gmail.com, changhainie@nju.edu.cn
- Hareton Leung is with Department of computing, Hong Kong Polytechnic University, Kowloon, Hong Kong.
E-mail: hareton.leung@polyu.edu.hk
- Jeff Y. Lei is with Department of Computer Science and Engineering, The University of Texas at Arlington, Arlington, Texas.
E-mail: ylei@cse.uta.edu
- Xiaoyin Wang is with Department of Computer Science, University of Texas at San Antonio.
E-mail: Xiaoyin.Wang@utsa.edu
- JiaXi Xu and Yan Wang are with School of Information Engineering, Nanjing Xiaozhuang University.
E-mail: xujiaxi@njxzc.edu.cn, wangyan@njxzc.edu.cn

Manuscript received April 19, 2005; revised September 17, 2014.

reasons are twofold. First, it is not realistic for developers to wait for all the needed test cases are generated before they can diagnose and fix the failures that have been detected [3]; Second, and the most important, utilizing the early determined failure-inducing interactions can guide the following test case generations, such that many unnecessary and invalid test cases can be avoided. For this we get the key idea of this paper: *Generation and Fault Localization process should be interleaving*.

Based on the idea, we propose a new CT framework, which instead of dividing the generation and identification into two independent stages, it integrates these two stages together. Specifically, we first execute one or more tests until a failure is observed. Next we immediately turn to the fault localization stage, i.e., identify failure-inducing interactions for that failure. These failure-inducing interactions are used to update the current coverage. In particular, interactions that are related to these failure-inducing interactions do not need to be covered in future executions. Then, we continue to perform regular combinatorial testing.

We remodel the test case generation and failure-inducing interactions identification modules to make them better adapt to this new framework. Specifically, for the generation part of our framework, we augment it by forbidding the appearance of test cases which contain the identified failure-inducing interactions. This is because those test cases containing a failure-inducing interaction will fail as expected, so that it makes no sense for the further failure detection. For the failure-inducing identification module, we augment it by achieving higher coverage. More specifically, we refine the additional test case generation in this module, so that it can not only help to identify the failure-inducing interactions, but also cover as many uncovered interactions as possible. As a result, our new CT framework needs fewer test cases than traditional CT.

Our new framework has strict requirements in accuracy of the identified failure-inducing interactions. This is mainly because it forbids the appearance of test cases which contain the identified interactions. Hence, if these interactions are not failure-inducing, they will never be covered again and an adequate testing will not be reached. To improve the accuracy of the failure-inducing interaction identification results, we propose a novel feed-back checking mechanism which aims at checking whether the interactions identified by our framework are accurate or not. Particularly, if these interactions do not pass the checking process, we will restart the failure-inducing identification module to re-identify other interactions.

We conducted a series of empirical studies on 5 open-source software to evaluate our new framework. These studies consist of two comparisons. The first one is to compare our new framework with the traditional one, which first generates a complete set of test cases and then performs the fault localization. The second one is to compare our framework with the Feedback-driven CT [4], [5], which also adapts an iterative framework to generate test cases and identifying failure-inducing interactions, but to address the problem of inadequate testing. The results show that, in terms of test case generation and failure-inducing interactions identification, our approach can significantly reduce the overall needed test cases and as a result it can more

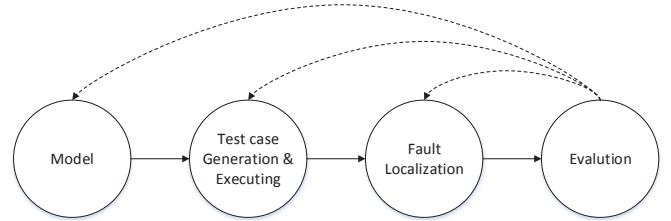


Fig. 1. The life cycle of CT

quickly identify the failure-inducing interactions of the system under test.

The main contributions of this paper are as follows.

- 1) We propose a new CT framework which combines the test case generation and fault localization more closely.
- 2) We augment the traditional CT test case generation and failure-inducing interactions identification process to make them adapt to the new framework.
- 3) We give a novel feed-back checking mechanism which can check whether the interaction identified by our approach is failure-inducing or not, and it significantly improve the accuracy of the results of the failure-inducing interaction identification approach.
- 4) We perform a series of comparisons with traditional CT and Feedback-driven CT. The result of the empirical studies are discussed.

The rest of the paper is organised as follows: Section 2 presents the preliminary background of CT. Section 3 presents an motivating example. Section 4 describes our new framework and a simple case study is also given. Section 5 presents the empirical studies and discusses the results. Section 6 shows the related works. Section 7 concludes the paper and proposes some further work.

2 BACKGROUND

This section presents some definitions and propositions to give a formal model for CT.

Assume that the Software Under Test (SUT) is influenced by n parameters, and each parameter p_i can take the values from the finite set V_i , $|V_i| = a_i$ ($i = 1, 2, \dots, n$). The definitions below are originally defined in [6].

Definition 1. A *test case* of the SUT is a tuple of n values, one for each parameter of the SUT. It is denoted as (v_1, v_2, \dots, v_n) , where $v_1 \in V_1, v_2 \in V_2 \dots v_n \in V_n$.

In practice, these parameters in the test case can represent many factors, such as input variables, run-time options, building options or various combination of them. We need to execute the SUT with these test cases to ensure the correctness of the behaviour of the SUT.

We consider any abnormally executing test case as a *fault*. It can be a thrown exception, a compilation error, an assertion failure, a constraint violation, etc. When faults are triggered by some test cases, it is desired to figure out the cause of these faults.

Definition 2. For the SUT, the n -tuple $(-, v_{n_1}, \dots, v_{n_k}, -)$ is called a k -degree schema ($0 < k \leq n$) when some k parameters

have fixed values and other irrelevant parameters are represented as "-".

In effect a test case itself is a k -degree *schema*, when $k = n$. Furthermore, if a test case contains a *schema*, i.e., every fixed value in the schema is in this test case, we say this test case *contains* the *schema*.

Note that the schema is a formal description of the interaction between parameter values we discussed before.

Definition 3. Let c_l be a l -degree schema, c_m be an m -degree schema in SUT and $l < m$. If all the fixed parameter values in c_l are also in c_m , then c_m *subsumes* c_l . In this case we can also say that c_l is a *sub-schema* of c_m and c_m is a *super-schema* of c_l , which can be denoted as $c_l \prec c_m$.

For example, the 2-degree schema $(-, 4, 4, -)$ is a sub-schema of the 3-degree schema $(-, 4, 4, 5)$, that is, $(-, 4, 4, -) \prec (-, 4, 4, 5)$.

Definition 4. If all test cases that contain a schema, say c , trigger a particular fault, say F , then we call this schema c the *faulty schema* for F . Additionally, if none of sub-schema of c is the *faulty schema* for F , we then call the schema c the *minimal failure-causing schema (MFS)* [6] for F .

Note that MFS is identical to the failure-inducing interaction discussed previously. In this paper, the terms *failure-inducing interactions* and *MFS* are used interchangeably. Figuring the MFS out helps to identify the root cause of a failure and thus facilitate the debugging process.

2.1 CT Test Case Generation

When applying CT, the most important work is to determine whether the SUT suffers from the interaction faults or not, i.e., to detect the existence of the MFS. Rather than impractically executing exhaustive test cases, CT commonly design a relatively small set of test cases to cover all the schemas with the degree no more than a prior fixed number, t . Such a set of test cases is called the *covering array*. If some test cases in the covering array failed in execution, then the interaction faults is regard to be detected. Let us formally define the covering array.

Definition 5. $MCA(N; t, n, (a_1, a_2, \dots, a_n))$ is a t -way *covering array* in the form of $N \times n$ table, where each row represents a *test case* and each column represents a parameter. For any t columns, each possible t -degree interaction of the t parameters (schema) must appear at least once. When $a_1 = a_2 = \dots = a_n = v$, a t -way covering array can be denoted as $CA(N; t, n, v)$.

TABLE 1
A covering array

ID	Test case			
t_1	0	0	0	0
t_2	0	1	1	1
t_3	1	0	1	1
t_4	1	1	0	1
t_5	1	1	1	0

For example, Table 1 shows a 2-way covering array $CA(5; 2, 4, 2)$ for the SUT with 4 boolean parameters. For any two columns, any 2-degree schema is covered. Covering array has proven to be effective in detecting the failures caused by interactions of parameters of the SUT. Many existing algorithms focus on constructing covering arrays such that the number of test cases, i.e., N , can be as small as possible. In general, most of these studies can be classified into three categories according to the construction strategy of the covering array [7]:

1) One test case one time: This strategy repeats generating one test case as one row of the covering array and counting the covered schemas achieved until all schemas are covered [8], [9], [10].

2) A set of test cases one time: This strategy generates a set of test cases at each iteration. Through mutating the values of some parameters of some test cases in this test set, it focuses on optimising the coverage. If the coverage is finally satisfied, it will reduce the size of the set to see if fewer test cases can still fulfil the coverage. Otherwise, it will increase the size of test set to cover all the schemas [11], [12].

3) IPO-like style: This strategy differentiates from the previous two strategies in that it does not firstly generate complete test cases [13]. Instead, it first focuses on assigning values to some part of the factors or parameters to cover the schemas that related to these factors, and then fills up the remaining part to form complete test cases.

In this paper, we focus on the first strategy: One test case one time as it immediately get a complete test case such that the testers can execute and diagnose in the early stage. As we will see later, with respect to the MFS identification, this strategy is the most flexible and efficient one comparing with the other two strategies.

2.2 Identify the failure-inducing interactions

To detect the existence of MFS in the SUT is still far from figuring out the root cause of the failure [14], [15], [16], as we do not know exactly which schemas in the failed test cases should be responsible for the failure. For example, if t_1 in Table 1 failed during testing, there are six 2-degree candidate failure-inducing schemas, which are $(0, 0, -, -)$, $(0, -, 0, -)$, $(0, -, -, 0)$, $(-, 0, 0, -)$, $(-, 0, -, 0)$, $(-, -, 0, 0)$, respectively. Without additional information, it is difficult to figure out the specific schemas in this suspicious set that caused the failure. Considering that the failure can be triggered by schemas with other degrees, e.g., $(0, -, -, -)$ or $(0, 0, 0, -)$, the problem of MFS identification becomes more complicated.

In fact, for a failing test case (v_1, v_2, \dots, v_n) , there can be at most $2^n - 1$ possible schemas for the MFS. Hence, more test cases should be generated to identify the MFS. In CT, the main work in fault localization is to identify the failure-inducing interactions. So in this paper we only focus on the MFS identification. Further works of fault localization such as isolating the specific defective source code will not be discussed.

A typical MFS identification process is shown in Table 2. This example assumes the SUT has 3 parameters, each can take on 2 values, and the test case $(1, 1, 1)$ fails. Then in Table 2, as test case t failed, we mutate one factor of test

TABLE 2
OFOT example

Original test case			Outcome
<i>t</i>	1	1	Fail
Additional test cases			
<i>t</i> ₁	0	1	Pass
<i>t</i> ₂	1	0	Fail
<i>t</i> ₃	1	1	Fail

case *t* one time to generate new test cases: *t*₁ – *t*₃. It turns out that test case *t*₁ passed, which indicates that this test case breaks the MFS in the original test case *t*. So (1,–,–) should be a failure-causing factor, and as other mutating process all failed, which means no other failure-inducing factors were broken, therefore, the MFS in *t* is (1,–,–).

This identification process mutate one factor of the original test case at a time to generate extra test cases. Then according to the outcome of the test cases execution result, it will identify the MFS of the original failing test cases. It is called the OFOT method [6], which is a well-known MFS identification method in CT. In this paper, we will focus on this identification method. It should be noted that the following proposed CT framework can be easily applied to other MFS identification methods.

Note that all the existing MFS identification approaches just give approximation solutions for MFS identification. In fact, to exactly identify the MFS (without any assumptions), it needs exponential number of test cases [17], which is impossible in practice. Hence, all the existing MFS identification approaches, as well as the approach we will proposed in this paper, need additional assumptions or just identify the likely failure-inducing interactions. For example, the OFOT approach is based on the following two assumptions:

Assumption 1. The execution result of a test case is deterministic.

This assumption is a common assumption of CT [17], [18], [19]. It indicates that the outcome of executing a test case is reproducible and will not be affected by some random events.

Assumption 2. Given a failing test case *t*, when we identify the MFS in *t*, any newly generated test case will not introduce new MFS that is not in *t*.

The second assumption is identical to the assumption proposed in [15], [16], [18], which is called the safe value assumption. Based on this assumption, when the additional test case generated by OFOT fails, e.g., *t*₂ in Table 2, we can determine that the additional test case contains the same MFS in the original failing test case, e.g., *t* in Table 2.

3 MOTIVATING EXAMPLE

In this section, a motivating example is presented to show how traditional CT works as well as its limitations. This example is derived from our attempt to test a real-world software—HSQLDB, which is a pure-java relational database engine with large and complex configuration space. To extract and manipulate valid configurations of this highly-configurable system is important, as different configurations can result in significantly different behaviours of the system

[20], [21], [22] (HSQLDB works normally under some proper configurations, but crashes or throws exceptions under some other configurations).

Considering the large configuration space of HSQLDB, we firstly utilized CT to generate a relatively small set of test cases. Each of them is actually a set of specific assignments to those options we cared¹. For each configuration, HSQLDB is tested by sending prepared SQL commands. We recorded the output of each run, but unfortunately, about half of them produced exceptions or warnings. Following the schedule of traditional CT, we started the identification process to isolate the failure-inducing option interactions in those failing configurations. Each failing configuration should be individually handled, in principle, as there may exist distinct failure-inducing option interactions among them. However, this successively identification process, although appealing, was hardly ever followed for this case study. This is because there are too many failing configurations and most of them contain the same failure-inducing option interactions, based on which the MFS identification process is wasteful and inefficient.

For the sake of convenience, we provide a highly simplified scenario to illustrate the problems we encountered. Consider four options in HSQLDB – *Server type*, *Scroll Type*, *Parameterised SQL* and *Statement Type*. The possible values each option can take on are shown in Table 3. Based on the report in the bug tracker of HSQLDB², an *incompatible exception* will be triggered if a *parameterised sql* is executed as *prepared statement* by HSQLDB. Hence, when option *parameterisedSQL* is set to be *true* and *Statmetent* to be *preparedStatement*, our testing will crash. Besides this failure, there exists another option value which can also crash this database engine. It is when *Scroll Type* is assigned to *sensitive*, as this feature is not supported by this version of HSQLDB³. Without this knowledge at prior, we need to detect and isolate these two failure-inducing option interactions by CT.

TABLE 3
Highly simplified configuration of HSQLDB

Option	Values
<i>o</i> ₁	Server type
<i>o</i> ₂	Scroll type
<i>o</i> ₃	parameterised SQL
<i>o</i> ₄	Statement Type

Table 4 illustrates the process of traditional CT on this subject. For simplicity of notation, we use consecutive symbols 0, 1, 2 to represent different values of each option (For *parameterisedSQL* and *Statement*, the symbol is up to 1). According to Table 4, traditional CT first generated and executed the 2-way covering array (*t*₁ – *t*₉ in the *generation* part). Note that this covering array covered all the 2-degree schemas for the SUT.

After testing with the 9 test cases (*t*₁ to *t*₉), we found *t*₁, *t*₄, and *t*₇ failed. It is then desired to respectively identify the MFS of these failing test cases. For *t*₁, OFOT method is used to generate four additional test cases (*t*₁₀ – *t*₁₃), and the MFS (–, 0, –, –) of *t*₁ is identified (*Scroll Type*

1. More details in: <http://gist.nju.edu.cn/doc/ict/>

2. For details, see: <http://sourceforge.net/p/hsqldb/bugs/1173/>

3. For details, see: <http://hsqldb.org/doc/guide/html>

is assigned to *sensitive*, respectively). This is because only when changing the second factor of t_1 , the additionally generated test case will pass. Then the same process is applied on t_4 and t_7 . Finally, we found that the MFS of t_4 is $(-, -, -, -)$, indicating that OFOT failed to determine the MFS (this will be discussed later), and the MFS of t_7 is the same as t_1 . Totally, for detecting and identifying the MFS in this example, we generated 12 additional test cases (marked with star).

TABLE 4
Sequential CT process

Generation (Execution)				
	test case		Outcome	
	o_1	o_2	o_3	o_4
t_1	0	0	0	0
t_2	0	1	1	1
t_3	0	2	1	0
t_4	1	0	0	1
t_5	1	1	0	0
t_6	1	2	1	1
t_7	2	0	1	1
t_8	2	1	0	0
t_9	2	2	0	0

Identification				
	test case		Outcome	
	t_{10}^*	t_{11}^*	t_{12}^*	t_{13}^*
t_{10}^*	1	0	0	0
t_{11}^*	0	1	0	0
t_{12}^*	0	0	1	0
t_{13}^*	0	0	0	1
MFS	$(-, 0, -, -)$			
t_{14}^*	2	0	0	1
t_{15}^*	1	1	0	1
t_{16}^*	1	0	1	1
t_{17}^*	1	0	0	0
MFS	$(-, -, -, -)$			
t_{18}^*	0	0	1	1
t_{19}^*	2	1	1	1
t_{20}^*	2	0	0	1
t_{21}^*	2	0	1	0
MFS	$(-, 0, -, -)$			

We refer to such traditional life-cycle as *Sequential CT* (SCT). However, we believe this may not be the best choice in practice. The first reason is that the engineers normally do not want to wait for fault localization after all the test cases are executed. The early bug fixing is appealing and can give the engineers confidence to keep on improving the quality of the software. The second reason, which is also more important, is such life-cycle can generate many redundant and unnecessary test cases, which negatively impacted on both test case generation and MFS identification. The most obvious negative effect in this example is that we did not identify the expected failure-inducing interaction $(-, -, 0, 1)$, which corresponds to option *parameterisedSQL* being set to *true* and *Statmetent* to *preparedStatement*. More shortcomings of the sequential CT are discussed as following:

3.1 Redundant test cases

The first shortcoming of SCT is that it may generate redundant test cases, such that some of them do not cover as many uncovered schemas as possible. As a consequent, SCT may generate more test cases than actually needed. This can be reflected in the following two aspects:

- 1) The test cases generated in the identification stage can also contribute some coverage, i.e., the schemas appear in

the passing test cases in the identification stage may have already been covered in the test case generation stage. For example, when we identify the MFS of t_1 in Table 4, the schema $(0, 1, -, -)$ contained in the extra passing test case $t_{11} - (0, 1, 0, 0)$ has already appeared in the passing test case $t_2 - (0, 1, 1, 1)$. In other word, if we firstly identify the MFS of t_1 , then t_2 is not a good choice as it does not cover as many 2-degree schemas as possible. For example, $(1, 1, 1, 1)$ is better than this test case at contributing more coverage.

2) The identified MFS should not appear in the following generated test cases. This is because according to the definition of MFS, each test case containing this schema will trigger a failure, i.e., to generate and execute more than one test case contained the MFS makes no sense for the failure detection. Taking the example in Table 4, after identifying the MFS $(-, 0, -, -)$ of t_1 , we should not generate the test case t_4 and t_7 . This is because they also contain the identified MFS $(-, 0, -, -)$, which will result in them failing as expected. Since the expected failure caused by MFS $(-, 0, -, -)$ makes t_7 and t_9 superfluous for error-detection, the additional test cases (t_{14} to t_{21}) generated for identifying the MFS in t_4 and t_7 are also not necessary.

3.2 Multiple MFS in the same test case

When there are multiple MFS in the same test case, MFS identification will be negatively affected. Particularly, some MFS identification approaches can not identify a valid schema in this case. For example, there are two MFS in t_4 in Table 4, i.e., $(-, 0, -, -)$ and $(-, -, 0, 1)$ (shown in bold). When we use OFOT method, we found all the additionally generated test cases (t_{14} to t_{17}) failed. These outcomes give OFOT a false indication that all the failure-inducing factors are not broken by mutating those four parameter values. As a result, OFOT cannot determine which schemas are MFS, which is denoted as $(-, -, -, -)$.

The reason why OFOT cannot properly work is that this approach can only break one MFS at a time. If there are multiple MFS in the same test case, the additionally generated test cases will always fail as they contain other non-broken MFS (see bold parts of t_{14} to t_{17}). Some approaches have been proposed to handle this problem, but they either can not handle multiple MFS that have overlapping parts [18], or consume too many additionally generated test cases [17], [23]. So in practice, to make MFS identification more effective and efficient, we need to avoid the appearance of multiple MFS in the same test case.

SCT, however, does not offer much support for this concern. This is mainly because it is essentially a post-analysis framework, i.e., the analysis for MFS comes after the completion of test case generation and execution. As a result, in the generation stage, testers have no knowledge of the possible MFS, and surely it is possible that multiple MFS appear in the same test case.

3.3 Masking effects

Traditional covering array usually offer an inadequate testing due to *Masking effects* [4], [5]. A masking effect [5] is an effect that some failures or exceptions prevent a test case from testing all valid schemas in that test case, which the test case is normally expected to test. For example in Table

t_1 is initially expected to cover six 2-degree schemas, i.e., $(0, 0, - , -)$, $(0, -, 0, -)$, $(0, -, -, 0)$, $(-, 0, 0, -)$, $(-, 0, -, 0)$, and $(-, -, 0, 0)$, respectively. The failure of this test case, however, may prevent the checking of these schemas. This is because, the failing of t_1 (*ScrollType* is set to be *sensitive*) crashed HSQLDB, and as a result, it did not go on executing the remaining test code, which may affect the examination of some interactions of t_1 . Hence, we cannot ensure we have thoroughly exercised all the interactions in this failing test case.

Since traditional covering array cannot reach an adequate testing, as an alternative, *tested t-way interaction criterion* as a more rigorous coverage standard is proposed [5]. According to this criterion, a t -degree schema is covered iff (1) it appears in a passing test case (2)it is identified as MFS or faulty schema. Apparently, this criteria can not be satisfied with traditional covering array alone. Next let us examine whether this criterion can be satisfied with SCT, i.e., the combination of traditional covering array and MFS identification.

One obvious insight is that if there is only **single** MFS in each failing test case, this criterion is satisfied. This conclusion is based on the fact that the MFS identification is actually a process to isolate the failure-inducing interaction among other interactions in the failing test case, and since there is only a single MFS, then other schemas can be determined as non-MFS.

For example in Table 4, t_1 contained a single MFS $(-, 0, -, -)$, and we identified this MFS by generating four extra test cases (t_{10} to t_{13}). As for t_1 , the schema $(-, 0, -, -)$ is determined to be MFS, but since the target of that testing is 2-way coverage, i.e., to cover all the 2-degree schemas, this 1-degree schema do not contribute any more coverage. Based on the fact that $(-, 0, -, -)$ is MFS, all the test cases containing this schema will fail by definition, and surely the super-schemas of $(-, 0, -, -)$ in this test case – $(0, 0, -, -)$, $(-, 0, 0, -)$ and $(-, 0, -, 0)$ are also faulty schemas as all the test cases containing these schemas must contain the MFS $(-, 0, -, -)$, which will fail after execution. The remaining 2-degree schemas $(0, -, 0, -)$, $(0, -, -, 0)$, $(-, -, 0, 0)$ are contained in the additionally generated test case t_{11} $(0, 1, 0, 0)$ (Note that for single MFS, there will be at least one passing additionally generated test case), which are of course non-faulty schemas. After all, all the 2-degree schemas in the failing test case t_1 satisfied the *tested t-way interaction criterion*.

When a failing test case has **multiple** MFS, however, SCT fails to meet that criterion. As discussed previously, SCT cannot properly work on test cases with multiple MFS—and even cannot obtain a valid schema. With this in mind, we cannot determine which schemas in this failing test case are MFS or not, consequently, we cannot ensure we have examined all the t -degree schemas in this failing test case. For example, t_4 has two MFS $(-, 0, -, -)$, $(-, -, 0, 1)$, which can not be identified with OFOT approach (In fact, there is no passing additionally generated test case). As a result, there are two 2-degree schemas $(1, 0, -, -)$ $(-, -, 0, 1)$ in this test case that are neither contained in a passing test case nor determined as MFS or faulty schemas. Hence, *tested t-way interaction criterion* is not satisfied. Since multiple MFS in a test case can introduce masking effects, SCT must be

negatively affected as it lacks mechanisms to avoid the appearance of multiple MFS in failing test cases.

Note that the *masking effects* are actually caused by the *multiple MFS problem* we discussed previously. But these two problems focus on different aspects of combinatorial testing. The *masking effects* mainly focus on the test sufficiency of CT, which can be regarded as a metric to evaluate how many schemas are actually tested [5]. While for *multiple MFS problem*, it mainly focuses on the quality of MFS identification. To be convenient, we separately discuss these two problems later in this paper.

3.4 Augmentation of the SCT

Considering the fact that we do not necessarily to repeatedly identify the same MFS, we can reduce the number of test cases by checking the already identified MFS and removing it from the MFS identification process. For example in Table 4, we do not need to generate 4 additional test cases (t_{18} to t_{21}) to figure out the failure-cause of t_7 is indeed $(-, 0, -, -)$, which has already been identified in previous iteration (t_{10} to t_{13}). Therefore, we only need to check whether there is any MFS other than $(-, 0, -, -)$ in t_7 or not. When applying this augmentation, the overall SCT process for the example in Table 4 will derive to Table 5.

TABLE 5
Augmented Sequential CT process

Generation (Execution)					<i>test case</i>	<i>o₁</i>
		<i>o₂</i>	<i>o₃</i>	<i>o₄</i>		
t_1	0	0	0	0		Fail
t_2	0	1	1	1		Pass
t_3	0	2	1	0		Pass
t_4	1	0	0	1		Fail
t_5	1	1	0	0		Pass
t_6	1	2	1	1		Fail
t_7	2	0	1	1		Pass
t_8	2	1	0	0		Pass
t_9	2	2	0	0		Fail

Identification				
	<i>MFS</i>	$(-, 0, -, -)$		
t_{10}^*	1	0	0	0
t_{11}^*	0	1	0	0
t_{12}^*	0	0	1	0
t_{13}^*	0	0	0	1
MFS		$(-, 0, -, -)$		
t_{14}^*	2	0	0	1
t_{15}^*	1	1	0	1
t_{16}^*	1	0	1	1
t_{17}^*	1	0	0	0
MFS		$(-, -, -, -)$		
t_{18}^*	2	1	1	1

In Table 5, the only difference from Table 4 is that for t_7 , we only generate one test case t_{18} , which mutates the value of the second parameter of t_7 to 1, and hence the already identified MFS $(-, 0, -, -)$ is removed. The pass of t_{18} indicates that there is no other MFS in this test case, and we do no need to generate additional test cases. As a result, we have reduced 3 test cases in total by using the augmented SCT.

Although the augmented SCT can reduce the redundancy of test cases to some extent, there still remain some issues,

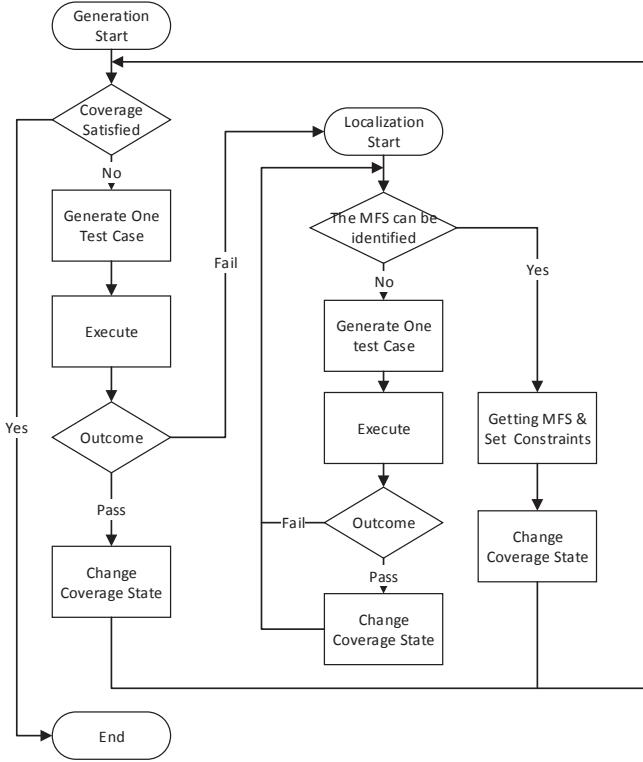


Fig. 2. The Interleaving Framework

e.g., multiple MFS, and masking effects, it cannot handle properly.

4 INTERLEAVING APPROACH

Considering these deficiencies of SCT, we do not need to cover all t -wise interactions before moving to the debugging phase. As an alternative, it is better to make test case generation and MFS identification more closely cooperated with each other. Hence, we propose a new CT generation-identification framework – *Interleaving CT* (ICT). Our new framework aims at enhancing the interaction of generation and identification to reduce the unnecessary and invalid test cases discussed previously. In other word, the ultimate goal of this framework is to better support MFS identification and test case generation, such that both of them can alleviate the three problems we discussed in Section 3.

4.1 Overall framework

The basic outline of our framework is illustrated in Figure 2. Specifically, this new framework works as follows: First, it checks whether all the needed schemas are covered or not. Normally the target of CT is to cover all the t -degree schemas, with t assigned as 2 or 3. If the current coverage is not satisfied, it will generate a new test case to cover as many uncovered schemas as possible. After that, it will execute this test case with the outcome of pass (executed normally, i.e., does not triggered an exception, violate the expected oracle or the like) or fail (on the contrary). When the test case passes, we will update the coverage state, as all the schemas in the passing test case are regarded as error-irrelevant. As a result, the schemas that were not covered

before will be determined to be covered if it is contained in this newly generated test case. Otherwise if the test case fails, then we will start the MFS identification module to identify the MFS in this failing test case. One point to note is that if the test case fails, we will not directly change the coverage, as we can not figure out which schemas are responsible for this failure among all the schemas in this test case until we identify them.

The identification module works in a similar way as traditional independent MFS identification process, i.e., repeats generating and executing additional test cases until it can get enough information to diagnose the MFS in the original failing test case. The difference from traditional MFS identifying process is that we record the coverage that this module has contributed to the overall coverage. In detail, when the additional test case passes, we will label the schemas in these test cases as covered if it has not been covered before. When the MFS is found at the end of this module, we will first set them as forbidden schemas that later generated test cases should not contain (Otherwise the test case must fail and it cannot contribute to more coverage), and second all the t -degree schemas that are related to these MFS as covered. Here the *related* schemas indicate the following three types of t -degree schemas:

First, the MFS **themselves**. Note that we do not change the coverage state after the generated test case fails (both for the generation and identification module), so these MFS will never be covered as they always appear in these failing test cases.

Second, the schemas that are the **super-schemas** of these MFS. By definition of the super-schemas (Definition 3), if the test case contains the super-schemas, it must also contain all its sub-schemas. So every test case that contains the super-schemas of the MFS must fail after execution. As a result, they will never be covered as we do not change the coverage state for failing test cases.

Third, those **implicit forbidden** schemas, which was first introduced in [24]. This type of schemas are caused by the conjunction of multiple MFS. For example, for a SUT with three parameters, and each parameter has two values, i.e., SUT(2, 2, 2). If there are two MFS for this SUT, which are (1, -, 1) and (0, 1, -). Then the schema (-, 1, 1) is the implicit forbidden schema. This is because for any test case that contain this schema, it must contain either (1, -, 1) or (0, 1, -). As a result, (-, 1, 1) will never be covered as all the test cases containing this schema will fail and so we will not change the coverage state. In fact, by Definition 4, they can be deemed as faulty schemas.

The terminating condition of most CT frameworks is to cover all the t -degree schemas. Then since the three types of schemas will never be covered in our new CT framework, we can set them as covered after the execution of the identification module, so that the overall process can stop.

Note that in practice, it may be more effective and efficient if we make more use of the debugging information and bug fixing. That is, before we go on generating test cases, we should firstly analysed the MFS we have already identified and fixed them. After that, we need to re-test the SUT by augmenting the test suites. By doing so, we can further reduce test cases in real software testing scenario.

4.2 Modifications of CT activities

More details of the modifications of CT activities are listed as follows:

(1) *Modified CT Generation* : We adopt the *one test case one time* method as the basic skeleton of the generation process. Originally, the generation of one test case can be formulated as EQ1.

$$t \leftarrow \text{select}(\mathcal{T}_{\text{all}}, \Omega, \xi) \quad (\text{EQ1})$$

There are three factors that determine the selection of test case t . \mathcal{T}_{all} represents all the valid test cases that can be selected to execute. Usually the test cases that have been tested will not be included as they have no more contribution to the coverage. Ω indicates the set of schemas that have not been covered yet. ξ is a random factor. Most CT generation approaches prefer to select a test case that can cover as many uncovered schemas as possible. This greedy selection process does not guarantee an optimal solution, i.e., the final size of the set of test cases is not guaranteed to be minimal. The random factor ξ is used to help to escape from the local optimum.

As discussed in Section 3, we should make the MFS not appear in the test cases generated afterwards, by treating them as the forbidden schemas. In other words, the candidate test cases that can be selected are reduced, because those test cases that contain the already identified MFS should not appear next. Formally, let \mathcal{T}_{MFS} indicates the set of test cases that contain the already identified MFS, then the test case selection is augmented as EQ2.

$$t \leftarrow \text{select}(\mathcal{T}_{\text{all}} - \mathcal{T}_{\text{MFS}}, \Omega, \xi) \quad (\text{EQ2})$$

In this formula, the only difference from EQ1 is that the candidate test cases that can be selected are changed to $\mathcal{T}_{\text{all}} - \mathcal{T}_{\text{MFS}}$, which excludes \mathcal{T}_{MFS} from candidate test cases.

(2) *Modified identification of MFS* : Traditional MFS identification aims at finding the MFS in a failing test case. As discussed before, test cases in the covering array are not enough to identify the MFS. Hence, additional test cases should be generated and executed. Generally, an additional test case is generated based on the original failing test case, so that the failure-inducing parts can be determined by comparing the difference between the additional test cases and the original failing test case. Take the OFOT approach as an example. In Table 4, the additional test case t_{11} is constructed by mutating the second parameter value of the original failing test case t_1 . Then as t_{11} passed the testing, we can determine that the second parameter value (-, 0, -, -) must be a failure-inducing element. Formally, let t_{failing} be the original failing test case, Δ be the mutation parts, \mathcal{P} be the parameters and their values, then the additional test case generation can be formulated as EQ3.

$$t \leftarrow \text{mutate}(\mathcal{P}, t_{\text{failing}}, \Delta) \quad (\text{EQ3})$$

EQ3 indicates that the test case t is generated by mutating the part Δ of the original failing test case t_{failing} . Note that the mutated values may have many choices, as long as they are within the scope of \mathcal{P} and different from those in t_{failing} . For example, for the original failing test case t_1 (0, 0, 0) in Table 4, let Δ be the second parameter value, then test cases (0, 1, 0, 0) and (0, 2, 0, 0) all satisfy EQ3. We refer

to all the test cases that satisfy EQ3 as $\mathcal{T}_{\text{candidate}}$, which can be formulated as EQ4.

$$\mathcal{T}_{\text{candidate}} = \{ t \mid t \leftarrow \text{mutate}(\mathcal{P}, t_{\text{failing}}, \Delta) \} \quad (\text{EQ4})$$

Traditional MFS identification process just selects one test case from $\mathcal{T}_{\text{candidate}}$ randomly. However, to adapt the MFS identification process to the new CT framework, this selection should be refined.

Specifically, there are two points to note. First, the additional test case should not contain the already identified MFS; second, the additional test case is expected to cover as many uncovered schemas as possible. These two goals are similar to CT generation, hence we can directly apply the same selection method on additional test case generation, which can be formulated as EQ5. The same as EQ2, EQ5 excludes the test cases that contain the already identified MFS from the candidate test cases ($\mathcal{T}_{\text{candidate}} - \mathcal{T}_{\text{MFS}}$), and selects the additional test case which covers the greatest number of uncovered schemas (Ω).

$$t \leftarrow \text{select}(\mathcal{T}_{\text{candidate}} - \mathcal{T}_{\text{MFS}}, \Omega, \xi) \quad (\text{EQ5})$$

(3) *Updating uncovered schemas* : After the MFS are identified, some related t -degree schemas, i.e., *MFS themselves*, *super-schemas* and *implicit forbidden schemas*, should be set as covered to enable the termination of the overall CT process. The algorithm that seeks to handle these three types of schemas is listed in Algorithm 1.

Algorithm 1 Changing coverage after identification of MFS

Input: \mathcal{S}_{MFS} ▷ already identified MFS
 Ω ▷ the schemas that are still uncovered
 \mathcal{T}_{all} ▷ all the possible valid test cases
 \mathcal{T}_{MFS} ▷ all the test cases that contain the MFS

Output: Ω ▷ updated schemas that are still uncovered

```

1: for each  $s \in \mathcal{S}_{\text{MFS}}$  do
2:   if  $s$  is  $t$ -degree schema then
3:      $\Omega \leftarrow \Omega \setminus s$ 
4:   end if
5:   for each  $s_p$  is super-schema of  $s$  do
6:     if  $s_p$  is  $t$ -degree schema then
7:        $\Omega \leftarrow \Omega \setminus s_p$ 
8:     end if
9:   end for
10: end for
11: for each  $s \in \Omega$  do
12:   if  $\nexists t \in (\mathcal{T}_{\text{all}} - \mathcal{T}_{\text{MFS}})$ , s.t.,  $t.\text{contain}(s)$  then
13:      $\Omega \leftarrow \Omega \setminus s$ 
14:   end if
15: end for

```

In this algorithm, we firstly check each MFS (line 1) to see if it is a t -degree schema (line 2). We will set those t -degree MFS as covered and remove them from the uncovered schema set Ω (line 3). This is the first type of schemas – *themselves*. For each t -degree super-schema of these MFS, it will also be removed from the uncovered schema set (line 5 - 9), as they are the second type of schemas – *super-schemas*. The last type, i.e., *implicit forbidden schemas*, is the toughest one. To remove them, we need to search through each potential schema in the uncovered schema set (line

11), and check if it is the implicit forbidden schema (line 12). The checking process involves solving a satisfiability problem. Specifically, if we can not find a test case from the set $(\mathcal{T}_{all} - \mathcal{T}_{MFS})$ (excluding those that contain MFS), such that it contains the schema under checking, then we can determine the schema is the implicit forbidden schema and it needs to be removed from the uncovered schema set (line 13). This is because in this case, the schema under checking can appear only in \mathcal{T}_{MFS} , which we will definitely not generate in later iterations. In this paper, a SAT solver will be utilized to do this checking process.

4.2.1 MFS identification approach mutated

To forbid identified MFS in the later generated test cases is efficient for CT, because it will reduce many unnecessary test cases. On the other hand, our framework has strict requirements in accuracy of the identified MFS. This is obvious, because if the schema identified is not a MFS, later generated test cases will forbid a non-MFS schema, which will have two impacts: (1) If this non-MFS schema is the sub-schema of some actual MFS, then the corresponding MFS will never appear and surely we will not detect and identify it. (2) If this non-MFS schema is a sub-schema of some t -degree uncovered schemas, then these schemas will never be covered and an adequate testing will not be reached.

To exactly identify the correct MFS in one failing test case, if possible, however, is not practical due to the cost of testing [6], [14]. This is because for any test case with n parameter values, there are $2^n - 1$ possible schemas which are the candidate MFS. For example, the possible candidate schemas of failing test case $(1, 1, 1)$ are $(1, -, -)$, $(-, 1, -)$, $(-, -, 1)$, $(1, 1, -)$, $(1, -, 1)$, $(-, 1, 1)$ and $(1, 1, 1)$. According to the definition of MFS, we need to individually determine whether these $2^n - 1$ are faulty schemas or not. In fact, even to determine whether a schema is faulty schema or not is not easy, as we must figure out whether all the test cases containing this schema will fail or not. So the complexity to correctly obtain a real MFS is surely exponential. As a result, existing MFS identification approaches actually obtain *approximation* solution through a relatively small size of additionally generated test cases [6], [14], [15], [17], [18], [19], [25].

Based on this insight, to improve the accuracy of the identified MFS, we propose a novel MFS-checking mechanism to assist with MFS identification. It is detailed in Algorithm 2.

In this algorithm, our target is to verify whether the candidate schema $candi$ is MFS or not. The input variable $Repeat$ indicates the checking strength, that is, the number of iterations that schema $candi$ is checked. In each iteration, we will generate a new test case t_{new} which contains this schema $candi$ (line 4) and execute it (line 5). If the newly generated test case fails, which indicates that the probability that the schema $candi$ is MFS increases, we will continue the checking process until the variable $Repeat$ is equal to 0 (line 9, line 2). On the other hand, if the test case passes (line 5), which indicates that the schema $candi$ is not MFS, we will update the uncovered schemas (because the newly passing test case will contribute to more coverage), and directly return false (line 7). If we cannot find a test case

Algorithm 2 Checking the MFS

```

Input:  $candi$                                  $\triangleright$  MFS that needs to be checked
         $Repeat$                                  $\triangleright$  The number of repeating times
         $\Omega$                                      $\triangleright$  the schemas that are still uncovered
         $\mathcal{T}_{MFS}$                              $\triangleright$  all the valid test cases that contain MFS
         $\mathcal{T}_{candi}$                              $\triangleright$  all the valid test cases that contain candi
Output:  $candi$  is MFS or not
1:  $\mathcal{T}_{Executed} \leftarrow \emptyset$ 
2: while  $Repeat > 0$  do
3:    $\mathcal{T}_{possible} \leftarrow (\mathcal{T}_{candi} \setminus \mathcal{T}_{MFS}) \setminus \mathcal{T}_{Executed}$ 
4:    $t_{new} \leftarrow select\_dissimilar(\mathcal{T}_{possible}, \mathcal{T}_{Executed})$ 
5:   if  $execute(t_{new}) == \text{PASS}$  then
6:      $update(t_{new}, \Omega)$ 
7:     return False
8:   end if
9:    $Repeat \leftarrow Repeat - 1$ 
10:   $\mathcal{T}_{Executed}.append(t_{new})$ 
11: end while
12: return True

```

that contains this schema and passes during our checking process, we will return true (line 12).

Note that the output *true* of our checking algorithm does not guarantee this schema $candi$ is 100% MFS (for which we need to generate all the possible test cases contain this schema), however, the probability that this schema is MFS increases with the increasing of checking strength, i.e., the value of $Repeat$ variable. But on the other hand, increasing the value of $Repeat$ also raises our testing cost (we need to generate one more test case if $Repeat$ increases by 1).

With respect to the tradeoff between the quality of MFS identification and testing cost, we need to design an elaborate test set with small number of test cases, while keeping a high probability to check whether the candidate schema under test is indeed MFS or not. Inspired by the idea of generating dissimilar test cases [26], [27], for each iteration, we let the newly generated test case be as different from previously generated test cases as possible (line 3-4). This heuristic idea is based on the fact that there is a small probability that dissimilar tests contain the same fault [26]. As a result, if the checking schema is not MFS, but the test case which contains it fails because of other failure-inducing schema, we may easily verify that it is not the MFS by generating another dissimilar test case (There is a high probability that the newly generated test case does not contain the failure-inducing schema in previous test case, and passes after execution).

4.2.2 Constraints handling

In many systems to be tested, constraints or dependencies exist between parameters. These constraints will render certain test cases invalid [28]. To handle these constraints is important, as we should examine the schemas only with valid test cases [5]. There are two types of method for constraints handling : 1) static method, that is, by knowing the constraints in prior, approaches will forbidden those invalid schemas appear in the generated test cases [20], [28], [29], [30], [31]. 2) dynamic method, that is, it does not initially know which are constraints, but identify them as MFS and forbidden them in the latter iteration [5]. We

adapt the second method for constraints handling, as it can directly applied into our framework.

Specifically, when we execute invalid test cases which cannot be executed or even complied, we will identify these invalid schemas which trigger this problem. In other word, we will regard the incompatibility exception as one type of failure, and identify the illegal schemas as MFS. After this, we will forbid these illegal schemas and also some possible implicit illegal schemas to appear in the latter generation (through the same way for those identified MFS).

In a more detailed view, those forbidden schemas are formulated into clauses, as introduced in [28]. For example, consider the SUT in Table 3. Assume that scroll type *forward-only* is incompatible with *in-process* server type, that is, the forbidden schema is (*in-process*, *forward-only*, -, -). We can formulate it as clause $\{\text{!in-process}, \text{!forward-only}\}$, which means that $\text{!in-process} \& \text{!forward-only} = 1$, where *in-process* and *forward-only* can be 0 or 1 (0 means that this value is not selected, while 1 means this value is selected). This clause limited that only one of them can be set to be 1. By doing so, we can use SAT solver [32] to obtain a solution (that is, a test case that avoid these forbidden schemas). It is noted that, besides these forbidden schemas, there has other conditions a test case must satisfy. For example, in Table 3, each option must be assigned with one, and only one, value. More details of this formulated model can be found in [28], [29].

4.3 Advantages of our framework

In view of the problems listed in Section 3, our new framework has the following advantages:

- 1) *Redundant test cases are eliminated such that the overall cost is reduced:*

Two facts of our framework support this improvement: (1) The schemas appeared in the passing test cases generated for MFS identification are counted towards the overall coverage, so that the test case generation process converges faster, which results in generating a smaller number of test cases. (2) The forbidden of identified MFS. As a result, test cases which contain these MFS will not appear, as well as those additionally generated test cases used to re-identify these MFS.

- 2) *The appearance of multiple MFS in the same test case is limited, improving the effectiveness of MFS identification*

This is mainly because we forbid the appearance of MFS that has been identified. Consequently, following our approach, the number of remaining MFS decreases one by one. Correspondingly, the probability that multiple MFS appear in the same test case will also decrease. Since multiple MFS has a negative effect on MFS identification as discussed in Section 3, the reduction of the appearance of multiple MFS in the same test case obviously improve the effectiveness of MFS identification.

- 3) *The Masking effect is reduced, hence adequate testing is better satisfied*

As discussed in Section 3, SCT suffers from masking effects when there are multiple MFS in one failing

test case. Since our approach theoretically reduces the probability that multiple MFS appear in the same test case, we believe our framework can alleviate the masking effects. In fact, our framework conforms to *tested t-way interaction criterion* because we only update t-way coverage for two types of schemas : (1) *t-degree* schemas in those passing test cases and (2) *t-degree* schemas *related* to MFS. Hence, our *Interleaving CT* framework supports a better adequate testing than SCT.

4.4 Demonstration on an example

Applying the new framework to the scenario of Section 3, we can get the result listed in Table 6.

TABLE 6
Interleaving CT case study

<i>Generation</i>					<i>Identification</i>
t_1	0	0	0	0	
t_2^*	1	0	0	0	Fail
t_3^*	0	1	0	0	Pass
t_4^*	0	0	1	0	Fail
t_5^*	0	0	0	1	Fail
candidate MFS: (-, 0, -, -)					
Checking					
t_6^*	1	0	1	1	Fail
t_7^*	2	0	2	2	Fail
t_8	1	1	1	1	Pass
t_9	0	2	1	1	Pass
t_{10}	0	1	0	1	Fail
t_{11}^*	1	1	0	1	Fail
t_{12}^*	0	2	0	0	Fail
t_{13}^*	0	1	1	0	Pass
t_{14}^*	0	1	0	0	Pass
candidate MFS: (-, -, 0, 1)					
Checking					
t_{15}^*	2	1	0	1	Fail
t_{16}^*	1	2	0	1	Fail
t_{17}	1	2	0	0	Pass
t_{18}	2	1	0	0	Pass
t_{19}	2	2	1	1	Pass

This table consists of two main columns, in which the left column indicates the generation part while the right indicates the identification process. We can find that, after identifying the candidate MFS $(-, 0, -, -)$ for t_1 , we generated two additional test cases (The checking strength, i.e., the *Repeat* value, is 2 in this example) that contain this schema and found both of them failed. It means that the schema $(-, 0, -, -)$ passed the verification, and would be regarded as MFS. Note that if either one of these two additional test cases passes, we will label $(-, 0, -, -)$ as non-MFS, and re-identify the MFS in t_1 . Another point that needs to be noted is that these two additional test cases (t_6, t_7) are two dissimilar test cases. In fact, all the 2-degree schemas that are covered by these two test cases are different.

After we determining $(-, 0, -, -)$ to be MFS, the following test cases (t_8 to t_{19}) will not contain this schema. Correspondingly, all the 2-degree schemas that are related to this schema, e.g. $(0, 0, -, -)$, $(-, 0, 1, -)$, etc, will also not appear in the following test cases. Additionally, the passing test case t_3 generated in the identification process cover six 2-degree schemas, i.e., $(0, 1, -, -)$, $(0, -, 0, -)$, $(0, -, -, 0)$, $(-, 1, 0, -)$, $(-, 1, -, 0)$, and $(-, -, 0, 0)$ respectively, so that it is not necessary to generate more test cases to cover them. We later found

that t_8 failed, which only contain one MFS as expected, and we easily identified it $(-, -, 0, 1)$ with four extra-generated test cases (t_{11} to t_{14}) and two checking test cases (t_{15} to t_{16}). This schema is 2-degree MFS, which will be forbidden in the following test cases and set to be covered.

Above all, when using the interleaving CT approach, the overall generated test case are 2 less than that of the traditional sequential CT approach in Table 4, and only one more test case than the augmented sequential CT approach in Table 5. Additionally, the new approach identified the expected MFS $(-, -, 0, 1)$ which both traditional SCT and augmented SCT fail to identify.

5 EMPIRICAL STUDIES

To evaluate the effectiveness and efficiency of the interleaving CT approach, we conducted a series of empirical studies on several open-source software subjects. Each of these studies aims at addressing one of the following research questions:

Q1: Does *ICT* perform better than augmented *SCT* at the overall cost and the accuracy of MFS identification?

Q2: Does *ICT* alleviate the three problems proposed in Section 3. Specifically, (1) does *ICT* reduce generating redundant and useless test cases, (2) does *ICT* reduce the appearance of test cases which contain multiple MFS, and (3) does *ICT* reduce the impacts of masking effects?

Q3: Does *ICT* have any advantages over the existed masking effects handling technique — *FDA-CIT* [5]?

Q4: What is the sensibility of our approach to different number of MFS and different number of options in SUT?

Q5: How well does our approach perform when the two assumptions listed in Section 2 do not hold?

Note that we will refer to *SCT* as the *augmented SCT* approach in the remaining part of this paper (Augmented *SCT* performs more effective and efficient than traditional *SCT*).

5.1 Subject programs

The five subject programs used in our experiments are listed in Table 7. Column “Subjects” indicates the specific software. Column “Version” indicates the specific version that is used in the following experiments. Column “LOC” shows the number of source code lines for each software. Column “Faults” presents the fault ID, which is used as the index to fetch the original fault description from the bug tracker for that software. Column “Lan” shows the programming language for each software (For subjects written in more than one programming language, only the main programming language is shown).

TABLE 7
Subject programs

Subjects	Version	LOC	Faults	Lan
Tomcat	7.0.40	296138	#55905	java
Hsqldb	2.0rc8	139425	#981	Java
Gcc	4.7.2	2231564	#55459	c
Jflex	1.4.2	10040	#87	Java
Tcas	<i>v</i> ₁	173	#Seed	c

Among these subjects, Tomcat is a web server for java servlet; Hsqldb is a pure-java relational database engine;

Gcc is a programming language compiler; Jflex is a lexical analyzer generator; and Tcas is a module of an aircraft collision avoidance system. We select these software as subjects because their behaviours are influenced by various combinations of configuration options or inputs. For example, the component *connector* of Tomcat is influenced by more than 151 attributes [33]. For program Tcas, although with a relatively small size (only 173 lines), it has 12 parameters with their values ranged from 2 to 10. As a result, the overall input space for Tcas can reach 460800 [34], [35].

As the target of our empirical studies is to compare the ability of fault detection between our approach with traditional ones, we firstly must know these faults and their corresponding MFS in prior, so that we can determine whether the schemas identified by those approaches are accurate or not. For this, we looked through the bug tracker of each software and focused on the bugs which are caused by the interaction of configuration options. Then for each such bug, we derive its MFS by analysing the bug description report and the associated test file which can reproduce the bug. For Tcas, as it does not contain any fault for the original source file, we took an mutation version for that file with injected fault. The mutation was the same as that in [35], which is used as an experimental object for the fault detection studies.

5.1.1 Specific inputs models

To apply CT on the selected software, we need to firstly model their input parameters. As discussed before, the whole configuration options are extremely large so that we cannot include all of them in our model in consideration of the experimental time and computing resource. Instead, a moderate small set of these configuration options is selected. It includes the options that cause the specific faults in Table 7, so that the test cases generated by CT can detect these faults. Additional options are also included to create some noise for the MFS identification approach. These options are selected randomly. Details of the specific options and their corresponding values of each software are posted at <http://gist.nju.edu.cn/doc/ict/>. A brief overview of the inputs models as well as the corresponding MFS (degree) is shown in Table 8.

TABLE 8
Inputs model

Subjects	Inputs	MFS
Tomcat	$2^8 \times 3^1 \times 4^1$	1(1) 2(2)
Hsqldb	$2^9 \times 3^2 \times 4^1$	3(2)
Gcc	$2^9 \times 6^1$	3(4)
Jflex	$2^{10} \times 3^2 \times 4^1$	2(1)
Tcas	$2^7 \times 3^2 \times 4^1 \times 10^2$	9(16) 10(8) 11(16) 12(8)

In this table, Column “inputs” depicts the input model for each version of the software, presented in the abbreviated form $\#values\#\#number\ of\ parameters \times \dots$, e.g., $2^9 \times 3^2 \times 4^1$ indicates the software has 9 parameters that can take on 2 values, 2 parameters can take on 3 values, and only one parameter that can take on 4 values. Column “MFS” shows the degrees of each MFS and the number of MFS (in the parentheses) with that corresponding degree.

Note that these inputs just indicate the combinations of configuration options. To conduct the experiments, some

other files are also needed. For example, besides the XML configuration file, we need a prepared HTML web page and a java program to control the startup of the tomcat to see whether exceptions will be triggered. Other subjects also need some corresponding auxiliary files (e.g., c source files for GCC, SQL commands for Hsqldb, and some text for Jflex). Additionally, there are two constraints among the subjects. The first constraint is from Tomcat, of which the error page location must not be empty. The second one is from Hsqldb, of which you can only process with the “*next()*” method in a non-scrollable result set.

5.2 Comparing ICT with SCT

The covering array generating algorithm used in the experiment is AETG [8], as it is the most common one-test-case-one-time generation algorithm. Another reason for choosing AETG, which is also the most important, is that the mutation of this algorithm, i.e., AETG_SAT [28], [29] is a rather popular approach to handle constraints in covering array generation, which is the key to our framework. The MFS identifying algorithm is OFOT [6] as discussed before. The constraints handling solver (integrated into AETG_SAT) is a java SAT solver – SAT4j [36]. Note that all the three algorithms or techniques can be easily replaced with other similar approaches. For example, we can use other one-test-one-time covering array generation algorithms, like DDA [9], or other MFS identification techniques [17], [18], or other popular SAT solvers [37]. However, to select specific algorithms for the three components of combinatorial testing is not the key concern of this paper; instead, our work focus on the overall CT process.

5.2.1 Study setup

For each software except *Tcas*, a test case is determined to be passing if it ran without any exceptions; otherwise it is regarded as failing. For *Tcas*, as the fault is injected, we determine the result of a test case by separately running it with the original correct version and the mutation version. The test case will be labeled as passing if their results are the same; otherwise, it is deemed as failing.

In this experiment, we focus on three coverage criteria, i.e., 2-way , 3-way and 4-way, respectively. It is known that the generated test cases vary for different runs of AETG algorithm. So to avoid the biases of randomness, we conduct each experiment 30 times and then evaluate the results. (Note that the remaining case studies are also based on 30 repeated experiments.) For each run of the experiment, we separately applied SCT approach and our approach on the prepared subject to detect and identify the MFS.

To evaluate the results of the two approaches, one metric is the cost, i.e., the number of test cases that each approach needs. Specifically, the test cases that are generated in the CT generation and MFS identification, respectively, are recorded and compared for these two approaches. Apart from this, another important metric is the quality of their identified MFS. For this, we used standard metrics: *precision* and *recall*, which are defined as follows:

$$\text{precision} = \frac{\#\text{the num of correctly identified MFS}}{\#\text{the num of all the identified schemas}}$$

and

$$\text{recall} = \frac{\#\text{the num of correctly identified MFS}}{\#\text{the num of all the real MFS}}$$

Precision shows the degree of accuracy of the identified schemas when comparing to the real MFS. *Recall* measures how well the real MFS are detected and identified. Their combination is F-measure, defined as

$$F - \text{measure} = \frac{2 \times \text{precision} \times \text{recall}}{\text{precision} + \text{recall}}$$

5.2.2 Result and discussion

Table 9 presents the results for the number of test cases. In Column ‘Method’, *ict* indicates the interleaving CT approach and *sct* indicates the sequential CT approach. The results of three covering criteria, i.e., 2-way, 3-way, and 4-way are shown in three main columns. In each of them, the number of test cases that are generated in *CT generation activity* (Column ‘Gen’), in *MFS identification activity* (Column ‘Iden’), and the total number of test cases (Column ‘Total’) are listed.

One observation from this table is that the number of test cases generated by our approach is far less than that of the *sct* approach. In fact, except for subject *Gcc*, our approach reduced about dozens of test cases on average when compared to approach *sct* (The improvement for subject *Tcas* is smaller, because most of the MFS of *Tcas* are of high degree ($t > 6$), and the covering arrays ($t = 2, 3, 4$) rarely detect any of them.). This result indicates that *ict* is more efficient at both *CT generation activity* and *MFS identification activity*.

For *Gcc*, however, we find that *ict* generated a bit more test cases at *MFS identification activity* (Note that even for this subject, *ict* still generated fewer test cases at *CT generation activity*). But when considering the fact that *ict* obtained a higher quality of the identified MFS, we believe this cost was worth it for *Gcc*. In fact, the f-measures of *ict* are 0.34, 0.7, and 0.78, respectively, for subject *Gcc*, while *sct* only scored 0.1, 0.08, and 0.11, respectively. This gap between *ict* and *sct* for *Gcc* was far larger than that of other subjects.

The quality of the identified MFS for other subjects are also listed in Table 10. Based on this table, we find that *ict* performed better than *sct*. In fact, except for subject *Jflex*, of which both *ict* and *sct* perfectly identified the MFS (the MFS of *Jflex* is a single 2-degree schema and easy to identify), *ict* obtained a higher score at *f-measure* than *sct* for all the subjects. For example, the *f-measures* of *ict* are 0.83, 1.0, and 0.99, respectively for subject *Hsqldb*, while *sct* only scored 0.5, 0.49, and 0.43, respectively. Even for subject *Tcas*, at which failures are hard to detect, the *f-measure* of *ict* is 0.01 for 4-way coverage, while *sct* scored 0. This result indicates that *ict* is far more effective at MFS identification than *sct*.

Another interesting observation with regard to the MFS identification is that higher t-wise strengths are not always resulting an improved precision (Take subject *Hsqldb* for example, the *f-measure* of *ict* and *sct* for 3-way coverage are 1.0 and 0.49, respectively; while 0.99 and 0.43 for 4-way coverage). This is because the effectiveness of MFS identification is related to the degree of MFS (i.e., the number of parameter values in the MFS) contained in the SUT. That is, if all the MFS in the SUT is of low degree, a low-wise covering array is enough to detect the MFS. This is because

TABLE 9
Comparison of the number of test cases

Subjects	Method	2-way			3-way			4-way		
		Gen	Iden	Total	Gen	Iden	Total	Gen	Iden	Total
Tomcat	ict	8.3	54.2	60.7	31.1	50.3	79.9	78.9	53.0	130.2
	sct	13.8	55.0	68.3	38.9	61.0	99.7	92.8	95.5	187.3
Hsqldb	ict	11.7	37.8	49.4	40.7	47.7	88.3	113.0	53.5	166.3
	sct	15.6	32.3	47.9	48.4	65.1	113.3	123.0	114.0	236.5
Gcc	ict	14.0	28.0	41.4	41.6	47.5	89.0	94.3	50.4	144.7
	sct	14.6	20.1	34.4	52.9	27.8	80.2	101.9	38.8	140.1
Jflex	ict	14.6	17.0	31.6	48.6	17.0	65.6	133.7	17.0	150.7
	sct	15.9	16.6	32.5	49.9	24.1	74.0	133.2	44.5	177.7
Tcas	ict	109.1	0.0	109.1	414.7	3.0	417.7	1545.4	7.4	1552.8
	sct	107.5	0.0	107.5	418.3	0.0	418.3	1556.1	2.6	1558.7

TABLE 10
Comparison of the quality of the identified MFS

Subjects	Method	2-way			3-way			4-way		
		Precision	Recall	F-measure	Precision	Recall	F-measure	Precision	Recall	F-measure
Tomcat	ict	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
	sct	0.75	1.0	0.86	0.88	1.0	0.93	0.88	1.0	0.93
Hsqldb	ict	1.0	0.77	0.83	1.0	1.0	1.0	0.97	1.0	0.99
	sct	0.7	0.4	0.5	0.53	0.47	0.49	0.45	0.43	0.43
Gcc	ict	0.45	0.28	0.34	0.77	0.65	0.7	0.83	0.75	0.79
	sct	0.13	0.07	0.1	0.09	0.07	0.08	0.12	0.1	0.11
Jflex	ict	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
	sct	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
Tcas	ict	0.0	0.0	0.0	0.0	0.0	0.0	0.15	0.0	0.01
	sct	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

a t-wise covering array can detect all the failures caused by the MFS of t-degree, or less than t-degree. Then, if a MFS is detected, *ict* and *sct* can identify them as expected. It is surely that a higher-wise can also detect those low degree MFS. But compared to the low-wise covering array, it generates much more test cases. As a result, many failing test cases may contain the same MFS, and worse, it increases the chance that a failing test case contains multiple MFS. This surely decreases the accuracy of MFS identification (See Section 3.2).

Additionally, Table 11 shows the milliseconds consumed by the two approaches on average. The experiment is conducted on Machine HP ProDesk 600 G1 TWR (Intel Core i5, 3.3Hz, 16GB memory). Based on this table, it is obviously that *ict* costed more time than *sct*. This is because *ict* needs to handle the SAT problem (for forbidding the appearance of MFS and constraints), which needs additional computing resources than *sct*. Considering the long test case execution time of large software projects, however, this extra test case generation time of *ict* is trivial in most cases.

TABLE 11
Time consumes (millisecond)

Subject	Method	2-way	3-way	4-way
Tomcat	ict	556.4	2703.7	12367.7
	sct	10.0	56.7	305.6
Hsqldb	ict	345.5	2093.6	21918.4
	sct	16.7	151.3	1055.1
Gcc	ict	180.1	1117.5	5408.5
	sct	8.0	68.0	309.3
Jflex	ict	187.1	1747.1	11412.4
	sct	75.5	288.8	2491.4
Tcas	ict	178.0	2914.9	60725.5
	sct	135.6	1750.0	25380.7

In summary, the answer to Q1 is: **Our approach *ict* needs fewer test cases than the augmented sequential CT approach, and the quality of MFS identification of *ict* is higher than *sct*.**

5.3 Alleviation of the three problems

Section 3 shows three problems that impact the performance of CT process, which are *redundant test case generation*, *multiple MFS in the same test case* and *masking effects*, respectively. To learn if *ict* can alleviate these problems, we re-use the experiment in the first study, i.e., let *sct* and *ict* generate test cases to identify the MFS in the five program subjects. Then, we respectively investigate the extent to which ICT and SCT are affected by those issues.

5.3.1 Study setup

We design three metrics for each of the three problems. First, to measure the *redundant test case generation*, we gather the number of times that each schema is covered. This metric directly indicates the redundancy of generated test cases, as it is obvious that if there are too many schemas that are repeatedly being covered by different test cases, then the CT process is inefficient as if one schema is covered and tested, it is unnecessary to check them again with other test cases. Note that this metric is closely related to the number of test cases discussed in the previous study, more test cases surely make schemas being covered more times. However, there exist one difference, i.e., test cases can evenly cover many schemas for a relatively few times, or alternatively, some schemas are covered many times, but others not.

Second, to measure *multiple MFS in the same test case* is simple, we just directly search for each generated test case and examine if it contains more than one MFS.

Third, we use the *tested-t-way* coverage criteria [5] to measure the masking effects. Specifically, we re-compute the coverage of the test cases generated by ICT and SCT by counting all the t -degree schemas that is either covered in a passing test case or identified as MFS or faulty schema. For ICT and SCT, the higher is the *tested-t-way* coverage, the more adequate is the testing and hence the less masking effects.

5.3.2 Result and discussion

1) Redundant test cases.

Our result is shown in Figure 3. This figure consists of 15 sub-figures, one for each subject with specific testing coverage (ranged from 2 - 4 way). For each sub-figure, the x-axis represents the number of times a schema is covered in total, the y-axis represents the number of schemas. For example in the first sub-figure (2-way for Tomcat), two bars with x-coordinate equal to 1 indicates that *ict* approach have 61.5 schemas in average which are covered once and *SCT* have 1.3 schemas.

As discussed previously, the more schemas that are covered with a low-frequency, the less redundant the generated test cases are. Hence it implies an effective testing if the number of schemas (y-axis) decrease with the increase of the covered times (x-axis). Most of the 15 sub-figures indicate that *ict* performs better than *sct*. In fact, for *ict*, the bars decrease rapidly with the increase of x-axis, while for *sct*, the trend is more smooth. See subject tomcat with 2-way coverage for example, *ict* have about 61.5 schemas which are only covered once, about 38.9 schemas covered twice, less than 12 schemas with the covered times more than 6. For *sct*, however, for most covered times, it has about 10 schemas, which indicates a very low performance.

The interesting exception is subject *Tcas*, on which *ict* and *sct* show a similar trend. This is because all the MFS of *Tcas* are of high degree ($t > 6$), and the covering arrays ($t = 2, 3, 4$) rarely detect any of them. Under this condition, since both approaches rarely detected the MFS, the overall process will be transferred to be traditional covering array generation (the MFS identification process is omitted).

This result shows that our two modifications of traditional approach, i.e., taking account of the covered schemas by test cases generated in MFS identification and forbidding the appearance of existing MFS to reduce the test cases that are used to identify the same MFS, are useful, especially when the MFS are detected and identified.

2) Multiple MFS.

The result is shown in Table 12, which lists the number of test cases (on average for the repeated 30 experiments) that contain more than one MFS.

From this table, one observation is that *ict* obtained a better result than *sct* at limiting the test cases which contain multiple MFS. For all the subjects except *Gcc*, *ict* nearly eliminated all the test cases which contain multiple MFS. Even for *Gcc*, the size of test cases which contain multiple MFS are limited in a very small number (less than 1). For *sct*, however, the result is not as good as *ict*. In fact except subjects *Jflex* and *Tcas*, *sct* suffers from generating test cases which contain multiple MFS. This is one reason why even though *sct* generate much more test cases than *ict*, it did not obtain a better MFS identification result than *ict*. Two

TABLE 12
Number of test cases that contain multiple MFS

Subject	Method	2-way	3-way	4-way
Tomcat	<i>ict</i>	0.0	0.0	0.0
	<i>sct</i>	1.2	4.6	10.8
Hsqldb	<i>ict</i>	0.0	0.0	0.0
	<i>sct</i>	0.2	1.6	4.1
<i>Gcc</i>	<i>ict</i>	0.5	0.8	0.6
	<i>sct</i>	0.2	1.8	3.4
<i>Jflex</i>	<i>ict</i>	0.0	0.0	0.0
	<i>sct</i>	0.0	0.0	0.0
<i>Tcas</i>	<i>ict</i>	0.0	0.0	0.0
	<i>sct</i>	0.0	0.0	0.0

exceptions are subjects *Jflex* and *Tcas*, on which both *ict* and *sct* did not generate test cases containing multiple MFS. The reason is that *Jflex* has only one MFS (see Table 8) and the MFS of *Tcas* are all high degrees which are hardly detected.

3) Masking effects.

The results of masking effect for each approach is shown in Table 13. There are two parts in this table, in which the left part shows the *tested-t-way* coverage for each approach. Specifically, the number of t -degree ($t = 2, 3, 4$) schemas which are *tested* (in the passing test cases or identified as faulty schemas) are gathered, as well as the percentage of the total t -degree schemas (in the parentheses followed). While the right part shows the number of test cases that ignores the masking effects (i.e., the test cases with no multiple MFS). Several observations can be obtained from this result:

First, the extent to which *sct* and *ict* suffered from masking effects is not severe. Actually, the lowest tested-t-way coverage of *ict* is 99.17% (4-way for *Gcc*) and *sct* is 97.69% (4-way for *Gcc*). This result shows that combining MFS identification with covering array (either in the sequential way or interleaving way) can make testing more adequate than using covering array alone. Additionally, the number of test cases that ignores the masking effects accounts for a great proportion of the test cases in *CT generation*, indicating that both approaches do not suffer too much from the masking effects.

Second, *ict* is more effective than *sct* at handling the masking effects. With respect to *tested-t-way* coverage, *ict* covered almost all the tested-t-way schemas for all the subjects (except for *Gcc*, but for which *ict* still covered more tested-t-way schemas than *sct*). On the other hand, *sct* was not as good as *ict*. In fact, *sct* fall behind *ict* for almost all the subjects except *Tcas*. For subject *Tcas*, both *ict* and *sct* covered all the tested-t-way schemas (failures of *Tcas* are rarely detected, and all the t-degree schemas appeared in the passing test cases). With respect to the test cases that did not be masked by failures, we found the number of non-masked test cases generated by *ict* was smaller than that of *sct*, but this is mainly because the total number of test cases generated *ict* was smaller than that of *sct*. Based on these results, *ict* is a better approach at avoiding the masking effects. We believe this is due to the higher quality of MFS identification of *ict*, because it can better distinguish non-failure-inducing schemas from those MFS, and hence give those schemas more chances to be tested without the appearance of MFS.

In summary, the answer to Q2 is that our approach *ict*

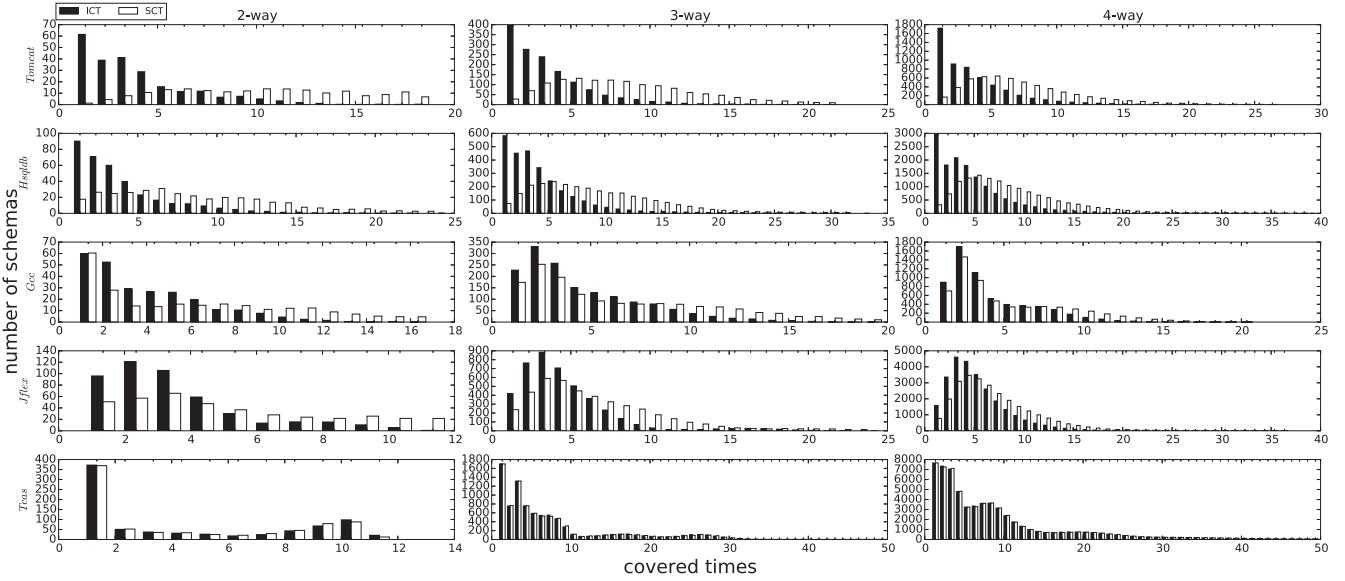


Fig. 3. The redundancy of test cases

TABLE 13
Masking effects results

Subjects	Method	Tested-t-way coverage			non-masked test cases		
		2-way	3-way	4-way	2-way	3-way	4-way
Tomcat	ict	236.0(100.00%)	1424.0(100.00%)	5600.0(100.00%)	8.3	31.1	78.9
	sct	233.8(99.07%)	1397.2(98.12%)	5501.7(98.24%)	12.6	34.3	82.0
Hsqliedb	ict	357.0(100.00%)	2742.0(100.00%)	14135.5(100.00%)	11.7	40.7	113.0
	sct	352.1(98.63%)	2713.7(98.97%)	13984.5(98.93%)	15.4	46.8	118.9
Gcc	ict	251.4(99.76%)	1526.4(99.38%)	5997.6(99.17%)	13.5	40.8	93.7
	sct	250.0(99.21%)	1519.4(98.92%)	5908.2(97.69%)	14.4	51.1	98.5
Jflex	ict	473.0(100.00%)	4282.0(100.00%)	26532.0(100.00%)	14.6	48.6	133.7
	sct	468.8(99.11%)	4216.6(98.47%)	26177.5(98.66%)	15.9	49.9	133.2
Tcas	ict	837.0(100.00%)	9158.0(100.00%)	64696.0(100.00%)	109.1	414.7	1545.4
	sct	837.0(100.00%)	9158.0(100.00%)	64696.0(100.00%)	107.5	418.3	1556.1

can alleviate the three problem discussed in Section 3, and when compared to *sct*, *ict* is a better approach to resolve these issues. Additionally, both *ict* and *sct* have a good performance on reducing the masking effects.

5.4 Comparison with FDA-CIT

fda-cit [5] is a feedback framework that can augment the traditional covering array to iteratively identify the MFS, and can handle the masking effects. The overall process can be illustrated in Figure 4. Specifically, it will first generate a t -way covering array and execute all the test cases in it. After that it will utilize the classification tree method to identify the MFS. Then it will forbid the identified MFS to be appeared and compute the tested- t -way coverage. If the tested- t -way coverage is not satisfied, it will repeat the previous process, i.e., generating additional test cases and identifying MFS. Like our *ict* approach, FDA-CIT is also an adaptive approach which iteratively generates test cases and identifies the MFS. Besides these commonalities, there are several important differences between our approach and *fda-cit* (shaded in Figure 4):

First, the **granularity** of adaptation. Instead of handling one test case one time as *ict*, *fda-cit* tries to generate a batch of test cases at each iteration (A complete covering array will

be generated at the first iteration, and more test cases will be supplemented to cover those t -degree schemas which are masked at latter iterations). To generate a batch of test cases may improve the degree of parallelism of testing, but this coarser granularity may also introduce some problems, e.g., some test cases generated at one iteration may fail with the same MFS, which is a potential waste because it is better to use one failing test case to reveal one particular MFS.

Second, the **MFS identification** approach is different. *fda-cit* uses the classification tree on existing executed test cases to characterize the MFS. Different from our OFOT approach, this post-analysis technique does not need additional test cases, but as a side effect it cannot precisely find the MFS. Worse, the effectiveness of this post-analysis approach depends greatly on the covering array, e.g., if there is a large number of failing tests, and a small size test suite, there is little information to exclude the particular MFS [18].

Third, the **coverage criteria** is not the same. *fda-cit* directly uses the tested- t -way coverage to guide their process. This supports a better adequate testing and reduces the impacts of masking effects. As we will see later in our experiments, however, the incorrect MFS identification may prevent *fda-cit* from reaching this type of coverage.

Next, we will design experiments to measure the perfor-

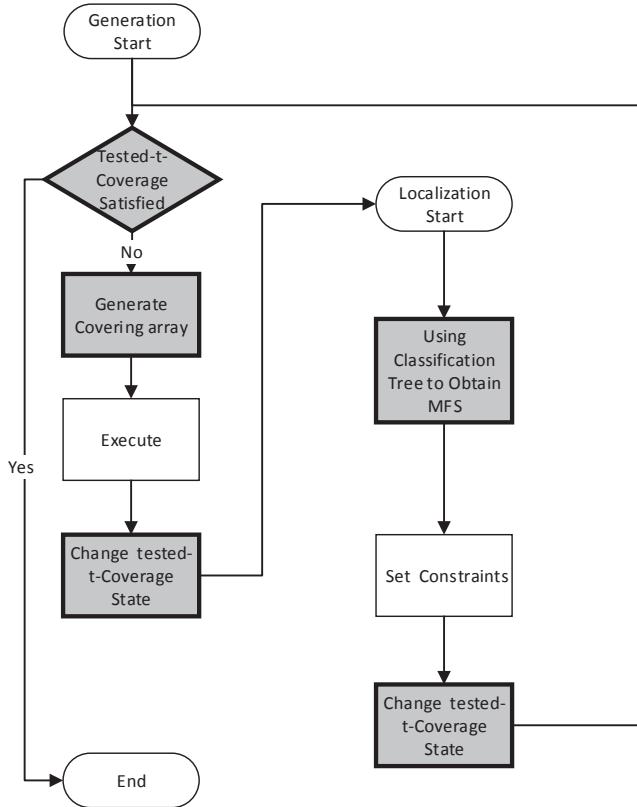


Fig. 4. The Framework of fda-cit

mance of *fda-cit*.

5.4.1 Study setup

The design of this case study is similar to the previous two. For each subject in Table 7, we apply *fda-cit* to generate test cases and identify the MFS. After that, we gather the overall test cases generated (*fda-cit* does not need additional test cases to identify the MFS), MFS identification results(including recall, precision and f-measure), and the other two metrics, i.e., covered times of schemas, the test cases which contain multiple MFS, and the t-tested-coverage. The same as previous experiments, we will repeat each experiment 30 times for different coverage (2, 3, and 4 way), and then gather and analyse the average data. Note that the MFS identification approach in the *fda-cit* has two versions, i.e., ternary-class and multiple-class. In this paper, we use the multiple-class version for comparison, as it performs better than the former [5].

5.4.2 Result and discussion

1) Total number of test cases The total number of test cases generated by *fda-cit* for each subject is shown in Table 14. To better evaluate the performance of *fda-cit*, we list the gaps between FDA-CIT with *ict* and *sct* respectively in the parentheses (the first number is for *ict*, the second one is *sct*). The value with negative sign indicates the reduction in the test cases between *fda-cit* and other two approaches, while the value without negative sign indicates the number of test cases which *fda-cit* generated more than the other two approaches.

From this table, one observation is that *fda-cit* is better than *sct* at almost all cases. Combining the results of previous studies for *sct* and *ict*, we can conclude that *sct* is the most inefficient approach at test case generation. Second, for *ict* and *fda-cit*, there are ups and downs on both sides. In detail, *fda-cit* needs less test cases at lower coverage (2-way and 3-way coverage), while *ict* performs better at higher coverage (4-way).

This result is reasonable. First, *fda-cit* does not need additional test cases to identify the MFS, which will reduce some cost when comparing with *ict*, especially when the coverage is low (For low coverage, the test cases generated by *ict* in the MFS identification stage account for a considerable proportion of the overall test cases). On the other hand, as noted earlier, the coarse grained generation will make *fda-cit* generate some unnecessary test cases.

2) F-measure of MFS identification The results of the quality of MFS identification by *fda-cit* is listed in Table 15. Same as previous metric, the comparison between *fda-cit* with *ict* and *sct* are also attached (the first number is for *ict*, the second one is *sct*).

This table shows a discernible disparity between *fda-cit* with the other two approaches. In fact, besides subject *Jflex* of which all three approaches accurately identified the single low-degree MFS (with F-measure equal to 1), and subject *Tcas* of which all three approaches can hardly detect failures (with F-measure equal to 0), *ict* leads over *fda-cit* by about 26% to 77%, which is not trivial. The result is similar when comparing *sct* with *fda-cit*.

This result suggests that the classification tree approach used by *fda-cit*, although very resource-saving (does not need additional test cases), is ineffective to accurately identify MFS, especially when there are multiple MFS with high degrees.

Note that *fda-cit*'s primary concern is to avoid masking effects and to give every *t*-degree schema a fair chance to be tested, not to perform fault characterization. On the other hand for the classification tree method, when only a very small set of test cases fail, it will result in the input data for classification tree to be highly unbalanced [38]. Another point is that all the MFS identified by the classification tree method should contain the same parameter value on the root, which will result in the schemas identified by *fda-cit* tend to be super-schema of the real MFS. Although this leads to the f-measure of MFS identification lower than that of *ict*, it does not have much negative influence on the masking effects reduction. This because to forbid the appearance of super-schema of real MFS will not reduce the number of schemas that is not MFS to be tested. As a result, the *tested-t-way* coverage is not changed (This coincides well with the experimental results).

3) Redundant test cases. The result is listed in Figure 5. The same as Figure 3, for each sub-figure, the x-axis represents the number of times a schema is covered in total, and the y-axis represents the number of schemas. To enable an intuitive comparison with *ict* and *sct*, we attach the data for *ict* and *sct*, with solid line and dotted line, respectively.

From this figure, we can see the trend of the bars of *fda-cit* matches pretty well with the curve representing *ict*, which has a significant advantage over the curve of *sct*. This result

TABLE 14
Number of test cases generated by *fda-cit*

	2-way	3-way	4-way
Tomcat	28.4(-32.3,-39.9)	65.5(-14.4,-34.2)	147.4(17.2,-39.9)
Hsqldb	29.9(-19.5,-18.0)	83.5(-4.8,-29.8)	201.2(34.9,-35.3)
Gcc	21.7(-19.7,-12.7)	63.4(-25.6,-16.8)	120.7(-24.0,-19.4)
Jflex	19.8(-11.8,-12.7)	64.5(-1.1,-9.5)	179.5(28.8,1.8)
Tcas	109.9(0.8,2.4)	416.6(-1.1,-1.7)	1544.7(-8.1,-14.0)

TABLE 15
The F-measure of MFS identification for *fda-cit*

	2-way	3-way	4-way
Tomcat	0.22(-77.57%,-63.28%)	0.31(-69.09%,-61.94%)	0.33(-66.67%,-59.52%)
Hsqldb	0.32(-51.26%,-18.26%)	0.29(-71.12%,-20.45%)	0.32(-66.19%,-10.76%)
Gcc	0.07(-26.48%,-2.19%)	0.4(-30.28%,31.50%)	0.49(-29.57%,38.29%)
Jflex	1.0(0.00%,0.00%)	1.0(0.00%,0.00%)	1.0(0.00%,0.00%)
Tcas	0.0(0.00%,0.00%)	0.0(0.00%,0.00%)	0.0(-0.81%,0.00%)

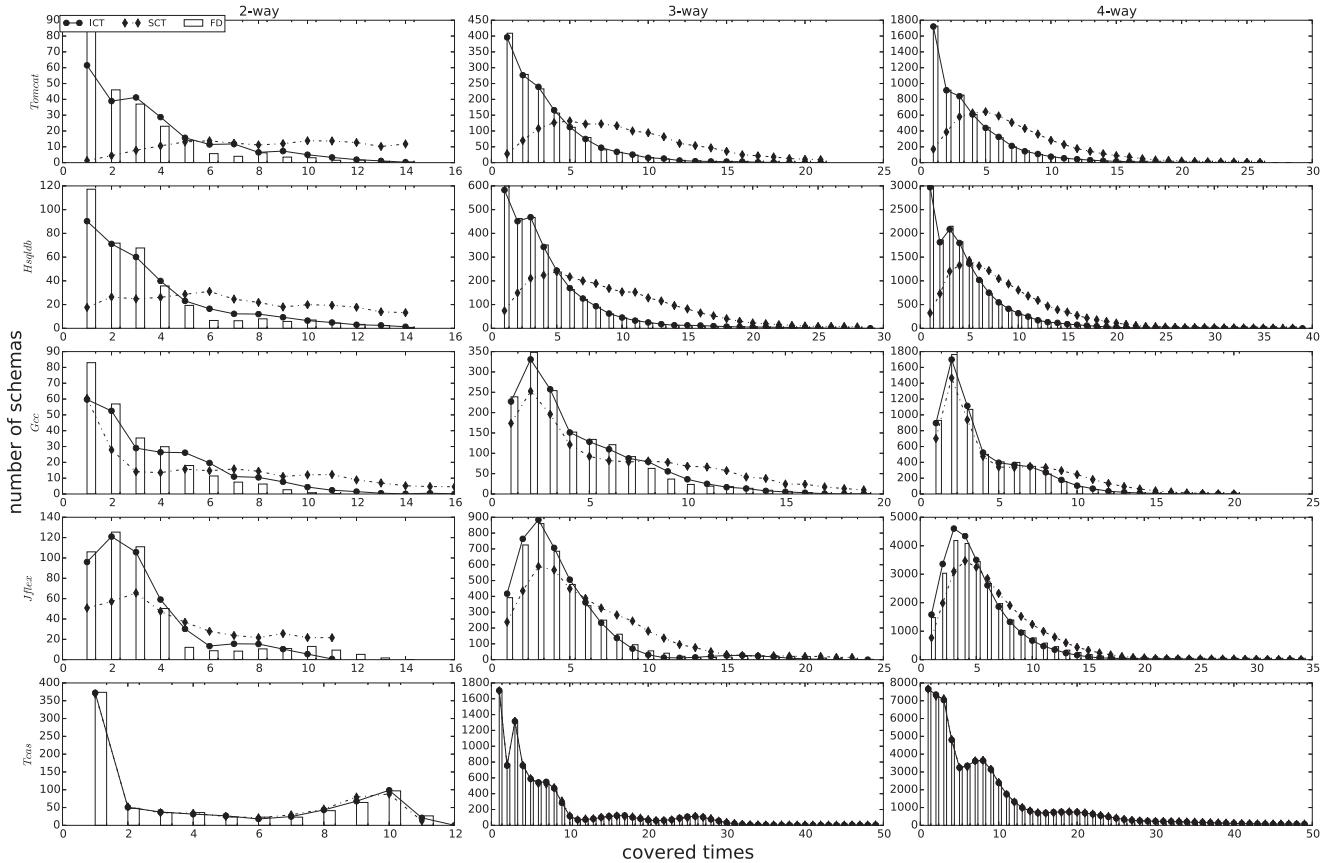


Fig. 5. The redundancy of test cases generated by *fda-cit*

implies that the test case redundancy between *ict* and *fda-cit* are similar, which is not severe when comparing with *sct*.

4) Test Cases containing multiple MFS

Table 16 shows the number of test cases contain multiple MFS on average for *fda-cit*. The same as before, we also list the gaps between *fda-cit* with *ict* and *sct* respectively in the parentheses (the first number is for *ict*, the second one is for *sct*). From this table, we can easily find that *fda-cit* did almost the same as *sct* at restricting the appearance of test cases that contain multiple MFS. But both of them did not as well as *ict*. In fact, except for subject *Jflex* (which contains single MFS) and *Tcas* (which contains high-degree MFS that are rarely

detected), *fda-cit* generated more test cases that contain multiple MFS than *ict*, and the gap between them increased with the increasing of test coverage (*fda-cit* generated about 1 more test cases for 2-way coverage, 2 more test cases for 3-way coverage, and 5 more test cases for 4-way coverage). This result shows that *ict* is the best approach among them to reduce the appearance of test cases that contain multiple MFS, and we also believe this is one reason why *ict* can obtain a high-quality of MFS identification.

5) Masking effects. The result is listed in Table 17 and Table 18, of which the first shows the results of *tested-t-way* coverage and the second shows the number of test cases

TABLE 16
Number of test cases contain multiple MFS for *fda-cit*

	2-way	3-way	4-way
Tomcat	0.9(0.9,-0.3)	4.5(4.5,-0.1)	9.8(9.8,-1.0)
Hsqldb	0.4(0.4,0.2)	1.5(1.5,-0.1)	4.9(4.9,0.8)
Gcc	2.4(1.9,2.2)	2.5(1.7,0.7)	4.1(3.5,0.7)
Jflex	0.0(0.0,0.0)	0.0(0.0,0.0)	0.0(0.0,0.0)
Tcas	0.0(0.0,0.0)	0.0(0.0,0.0)	0.0(0.0,0.0)

ignores the masking effects. For the two tables, the gaps between *fda-cit* with *ict* and *sct* are listed in the parentheses, respectively (the first one is *ict*, the second one is *sct*).

With regard to *tested-t-way* coverage, we can find that our approach *ict* is still the best approach to reduce the masking effects, even though when compared to approach *fda-cit*. In fact, among the 15 cases listed in Table 17, there are 13 cases on which *ict* performed equal to or better than *fda-cit* (The only two exceptions are *Gcc* for 3-way and 4-way coverage). Note that on some particular cases, the gaps between *ict* and *fda-cit* are not trivial, e.g., *ict* obtained 10 more percent of tested-t-way coverage than *fda-cit* at 2-way coverage for subject *Tomcat*.

With regard to *the number of test cases which ignore the masking effects*, we find that for almost all the cases (except *tested-3-way* coverage of *Tcas*), *fda-cit* performs better than both *ict* and *sct*. This is because for approaches *ict* and *sct*, the proportion of test cases for identifying the MFS is not trivial, and the test cases that aim to identifying MFS are more likely to be affected by masking effects (many of them will contain the MFS and fail during testing). Considering there is little difference between the overall size of test cases generated by *fda-cit* and *ict*, it is easily to learn that the number of test cases generated for *CT generation of fda-cit* is larger than *ict* and *sct*. As a consequence, the number of test cases generated by *ict* that can be counted as ignoring the masking effects is smaller than that of *fda-cit*. In other words, if we take the test cases for identifying the MFS into account for *ict*, *the number of test cases which ignore the masking effects of ict* is actually similar to that of *fda-cit*.

Above all, this result suggests that our approach *ict* does reach the same or better level when compared to *fda-cit* at reducing the masking effects. The conclusion also implies that, to limit the masking effects, only using an adaptive framework to separately identify the MFS is not enough; making MFS identification accurately is more important.

To summarize, the answer to Q3 is that **when compared to the adaptive CT approach *fda-cit*, our approach *ict* performed more effectively, especially at the quality of MFS identification and reduction of masking effects, while there is no apparent differences between them at the cost (the number of test cases). This also indicates that *ict* is a more efficient approach.**

Note that one reason that *ict* did not generate more test cases than *fda-cit* is that all the subjects we used in the experiments have just one test file for each test configuration. Here test configuration equals to the test case we discussed throughout the paper. *fda-cit* is designed to work better for subjects of which one configuration has multiple test files. Under the scenario of multiple test files, *cit* should separately handle each of them, because each test file may

contain distinct MFS. As a result, the number of additional configurations needed will grow linearly with the number of failing test files. Under this case, *fda-cit* needs smaller amount of test cases [5].

5.5 Sensitivity of the approaches

In order to reduce the bias of the choice of subjects, and to obtain a more general conclusion, we conducted several experiments on the subjects with various characteristics in this section. More specifically, we considered the impacts of different number of MFS in the SUT and different number of options in the SUT on three approaches, i.e., *ict*, *sct* and *fda-cit*.

5.5.1 Study setup

To vary parameters of interest in a controlled setting in this study, we use synthetic subjects instead of real programs (which typically represent only one particular parameter setting, and hence it is hard to get software with the expected number of options or MFS).

Specifically, for the first study, that is, evaluating the performance of approaches under different number of MFS, we use the subject with 11 parameters, and each parameter has 5 values, i.e., the inputs model is (5^{11}) . Then we consider the following possible number of 2-degree MFS: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 20, 30, 40, 50, 60, 70, 80 and 90. Then for each run of the experiment, we will first inject the corresponding number of MFS into the synthetic subject, and then run all the three approaches on the subject. At last, the results (MFS identification quality and cost) of each approach for each run is collected and analysed.

The second study is to evaluate the performance of approaches under different number of options. Hence we use synthetic subjects with the following number of options (8, 9, 10, 12, 16, 20, 30, 40, 50, 60, 70, 80, 90, 100). Each option has two values, and each subject has three 2-degree MFS. Then for each subject, we will apply the three approaches and compare their performance.

5.5.2 Number of MFS

The result for the sensitivity to the number of MFS is shown in Figure 6 and 7, of which the first figure focuses on the quality of MFS identification, and the second figure shows the cost.

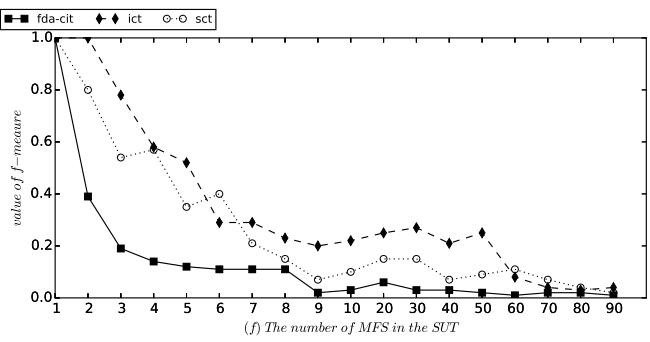


Fig. 6. F-measure for various number of MFS

One observation from Figure 6 is that, as the number of MFS increases, the f-measures of all three approaches

TABLE 17
The tested-t-way coverage for *fda-cit*

	2-way	3-way	4-way
Tomcat	212.1(-10.13%, -9.28%)	1390.2(-2.37%, -0.50%)	5378.0(-3.96%, -2.25%)
Hsqldb	345.1(-3.33%, -1.99%)	2725.1(-0.62%, 0.42%)	14096.7(-0.27%, 0.80%)
Gcc	250.2(-0.48%, 0.08%)	1530.8(0.29%, 0.74%)	6017.5(0.33%, 1.82%)
Jflex	473.0(0.00%, 0.89%)	4282.0(0.00%, 1.53%)	26532.0(0.00%, 1.34%)
Tcas	837.0(0.00%, 0.00%)	9158.0(0.00%, 0.00%)	64695.9(-0.00%, -0.00%)

TABLE 18
Non-masked test cases for *fda-cit*

	2-way	3-way	4-way
Tomcat	27.5(19.2, 14.9)	61.0(29.9, 26.7)	137.6(58.7, 55.6)
Hsqldb	29.5(17.8, 14.1)	82.0(41.3, 35.2)	196.3(83.3, 77.4)
Gcc	19.3(5.8, 4.9)	60.9(20.1, 9.8)	116.6(22.9, 18.1)
Jflex	19.8(5.2, 3.9)	64.5(15.9, 14.6)	179.5(45.8, 46.3)
Tcas	109.9(0.8, 2.4)	416.6(1.9, -1.7)	1544.7(-0.7, -11.4)

decrease rapidly. In fact, when the number of MFS is greater than 60, the f-measures of all three approaches are near 0. This is mainly because when there are too many MFS, it is hard to get a passing test case, and hence, it is challenging to distinguish MFS from those schemas which are not related to the failures.

Another observation is that for most cases, *ict* performs the best, then follows *sct*, and the last is *fda-cit*. It is clear that *ict* can work well under the condition of multiple MFS, when compared with the other two approaches.

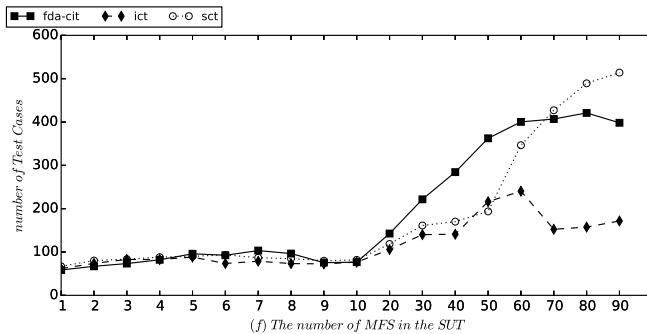


Fig. 7. Test Cases for various number of MFS

With regard to the cost, one observation is that with the increase of the number of MFS, all three approaches need more test cases to identify the MFS. The reason is also obvious – a high number of MFS trigger more failing test cases, which needs more additional test cases for MFS identification. For *fda-cit*, even though it does not need additional test cases for MFS identification, a high number of MFS will lead to slower convergence. This is because it is harder to fulfil the *tested-t-way* coverage when there are too many failing test cases, and the slower convergence will surely result in more test cases generated.

Another observation about the cost is that *ict* generated the smallest size of test cases when compared to the other two approaches. In fact, when the number of MFS is greater than 20, the cost of *sct* and *fda-cit* increased rapidly (reached to about 500 test cases), which far exceeded that of *ict*.

Above all, with the increase of number of MFS in the SUT, the performance of all three approaches decreases, but

ict still performs better than the other two approaches.

5.5.3 Number of options

The result for the sensitivity of number of options in the SUT is shown in Fig. 8 and Fig. 9, which depicts the quality of MFS identification and the number of test cases needed, respectively, with the increase of the number of options.

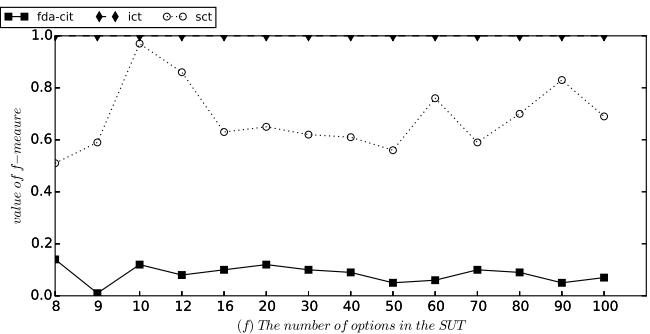


Fig. 8. F-measure for various number of options

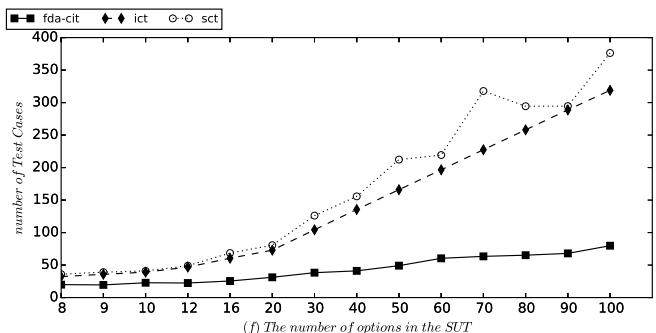


Fig. 9. Test Cases for various number of options

With regard to the quality of MFS identification, it is clear that *ict* performs the best, then follows *sct*, and the last is *fda-cit*. In fact, for all the subjects, *ict* scored 1.0 of f-measure, which indicates that *ict* accurately identified all the MFS. On the other hand, *sct* scored around 0.5 to 0.9, and *fda-cit* only scored around 0.1. This result is consistent to the previous

study, indicating that *ict* can accurately identify the MFS, even though when the number of options is large.

Another observation about to the MFS identification is that there is no clear correlation between the MFS quality and the number of options. In fact, there is no clear regularities for the curves representing the f-measures of *sct* and *fda-cit* with the increasing of number of options. It shows that the number of options in the SUT does not have influence on the quality of MFS identification.

With regard to the number of test cases, there is a clear trend that *fda-cit* performs the best, of which the number of needed test cases grows slowly. This is mainly because it does not generate additional test cases for MFS identification. The second best is *ict*, as the number of test cases increases linearly with the number of options in the SUT. This is due to the mechanism of the MFS identification approach applied in the *ict* framework, i.e., we must always generate the same number of test cases as the number of options in the SUT to identify the MFS. The complexity can be further reduced ($O(\log N)$) [17], [18], [25]). The last one is *sct*, of which the number of test cases is always larger than that of *ict*.

Therefore, the number of options in the SUT does not influence the quality of MFS identification; and although generating more test cases than *fda-cit*, *ict* is still a better choice when considering the precision and recall in MFS identification.

In summary, the answer to Q4 is: **Large number of MFS has a negative impact on the quality of the MFS identification of all the three approaches, while the number of options does not. Additionally, *ict* performs best among the three approaches for various number of MFS and options.**

5.6 The ability of handling assumptions

The last study is designed to evaluate the performances of the three approaches when the two assumptions proposed on Section 2.2, i.e., deterministic failures and existence of safe values, do not hold.

5.6.1 Study setup

The same as previous study, to make the characteristics of the SUT under control, we decided to use synthetic subjects instead of real programs in this case study. Particularly, synthetic subjects can be injected with various types of faults, e.g., the nondeterministic failures with various probabilities that can be triggered during testing, such that it helps us to evaluate the performance of these approaches for various extent to which these two assumptions do not hold. As a result, we can obtain a more general conclusion instead of those results based on some specific programs.

Specifically, for the first assumption, we decided to inject two MFS, of which one is deterministic (the test case which contains it always fails after execution), and the other is nondeterministic (the test case which contains it may fail or may not after execution). Then we considered the following possible probabilities that the nondeterministic MFS may be triggered (The probability that the test case which contains it fails after execution): 0.01, 0.05, 0.1, 0.15, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.80, 0.9, and 0.98. We repeated the experiment

30 times for each probability to avoid the random affects. For each run of the experiment, we applied all the three approaches on the subject, and recorded their the results (MFS identification quality and cost).

The second study is to evaluate the performance of approaches when the safe value assumption do not hold. In fact, in our previous studies, the safe value assumption were not always hold. For example, in the first study, we did not list any safe value in prior for our approach *ict*, instead, we just generated additional test cases contain the schemas under test. As a result, we did not always reach to the 100% f-measure of MFS identification. For this study, we decided to evaluate these approaches on the condition that there is no safe value, i.e., every parameter value is contained in at least one MFS. We used synthetic subjects with the following number of options: 8, 10, 16, 20, 30, 40, 50, 60, 70, 80, and 90. Then we applied the three approaches on each subject and compared their performance under the condition that there is non-safe value in each subject.

5.6.2 Nondeterministic failures

The results of evaluating the approaches on nondeterministic failures are shown in Fig. 10 and Fig. 9, of which the first figure depicts the quality of MFS identification with various probabilities that the nondeterministic failures are triggered, while the second figure shows the number of test cases.

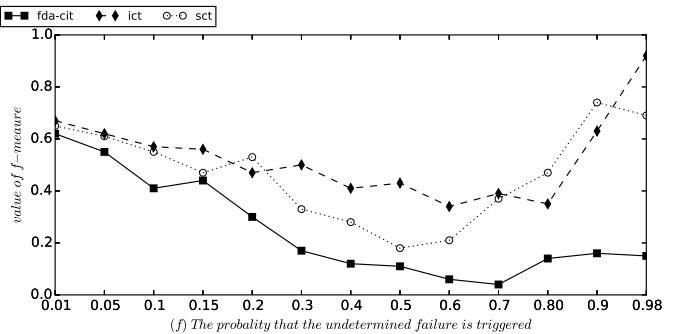


Fig. 10. F-measure for various probability of un-determining failure

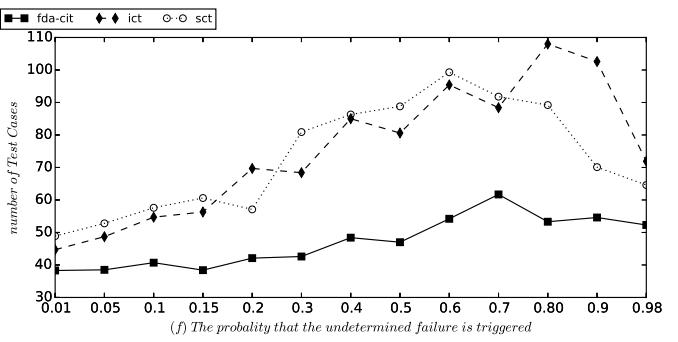


Fig. 11. Test Cases for various probability of un-determining failure

With regard to MFS identification, there are two observations. First, if the probability is too small (little than 0.15) or too high (large than 0.9), all the three approaches performed relatively well. This is reasonable, because if the probability of triggering MFS is too small, we can hardly detect the

failure. As a result, the MFS is similar to a non-failure inducing schema, and hence it has little influence on the MFS identification. On the other hand, if the probability is too high, which indicates that we can easily detect it. Under this condition, the failure is similar to a deterministic failure, which also has little influence on the MFS identification. The worst condition is in between (the probability is around 0.3 to 0.8). Under such condition, the failure may appear at one testing, but disappears at the next time. These two statuses exchanged frequently, and result in a negative influence on the MFS identification.

The second observation is that *ict* still performed best among the three approaches (there are 10 among 13 cases that *ict* performed better than *sct*, and both *ict* and *sct* obtained higher f-measures than *fda-cit* on all the conditions). This result indicates that *ict* can handle the non-deterministic failures properly.

With regarded to the test cases, our conclusion is similar to previous study, that is, *fda-cit* needs the smallest amount of test cases, then follows the *ict*, and the last is *sct*.

Above all, we can learn that the non-deterministic failures do effects the MFS quality of the three approaches, but *ict* can handle it well, and perform the best among three approaches.

5.6.3 Non Safe values

The results of evaluating the approaches on non-safe values are shown in Fig. 12 and Fig. 13.

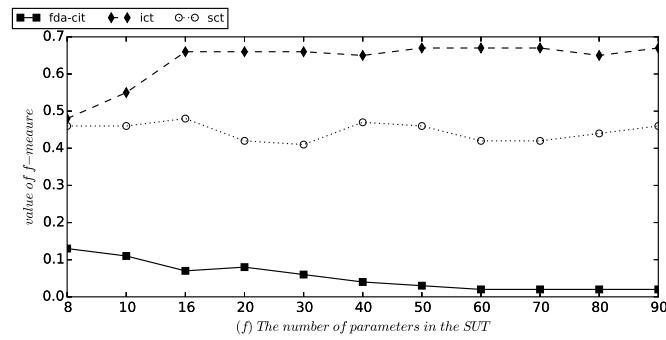


Fig. 12. F-measure for various number of un-safe values

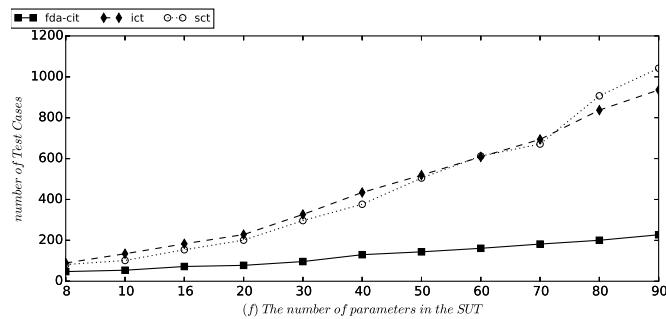


Fig. 13. Test Cases for various number of un-safe values

Similar to previous study, non-safe value do affect to approaches. And for the non-safe value, our approach *ict* has a more clear advantage than the three approaches.

Above, the answer to the question 5 is:

Without assumptions, do affects the results of three approaches, but our approach works the best among them.

5.7 Threats to validity

There are several threats to validity in our empirical studies. First, our experiments are based on only 5 open-source software. More subject programs are desired to make the results more general, such that the conclusion of our experiment can reduce the impact caused by specific input space and specific degree or location of the MFS.

Second, there are many generation algorithms and MFS identification algorithms. In our empirical studies, we just used AETG [8] as the test case generation strategy, and OFOT [6] as the MFS identification strategy. As different generation and identification algorithms may affect the performance our proposed CT framework, especially on the number of test cases, some studies using different test case generation and MFS identification approaches are desired.

6 RELATED WORKS

Combinatorial testing has been widely applied in practice [39], especially on domains like configuration testing [40], [41], [42] and software inputs testing [8], [43], [44]. A recent survey [7] comprehensively studied existing works in CT and classified them into eight categories according to the testing procedure. Based on this study, we learn that test case generation and MFS identification are two most important parts in CT studies.

Many works have been proposed for covering array generation, which can be mainly classified into the following four categories [7]: 1) greedy methods [8], [9], [13], [45], which are very fast and effective, but may consume too many test cases. 2) mathematical methods [46], [47], [48], [49], which can also be extremely fast and can produce optimal test sets in some special cases, but they impose many restrictions. 3) Heuristic search techniques [11], [50], [51], [52], [53], [54], which can generate very small size of test cases, but may cost much computation time and 4) random methods [55], [56], which are extremely fast, but generate more test cases than greedy approaches.

The MFS identification problem also attracts many interests in CT. These approaches for identifying MFS can be partitioned into two categories [14] according to how the additional test cases are generated: *adaptive*—additional test cases are chosen based on the outcomes of the executed tests [6], [17], [18], [19], [23], [25], [34], [57] or *nonadaptive*—additional test cases are chosen independently and can be executed in parallel [14], [15], [16], [38], [40].

Although CT has been proven to be effective at detecting and identifying the interaction failures in SUT, however, to directly apply them in practice can be inefficient and some times even does not work at all. Some problems, e.g., constraints of parameters values in SUT [28], [29], masking effects of multiple failures [4], [5], dynamic requirement for the strength of covering array [42], will cause difficulty to the CT process. To overcome these problems, some works try to make CT more adaptive and flexible.

JieLi [25] augmented the MFS identifying algorithm by selecting one previous passing test case for comparison,

such that it can reduce some extra test cases when compared to another efficient MFS identifying algorithm [18].

S.Fouché et al., [42] introduced the notion of incremental covering array. Different from traditional covering array, it does not need a fixed strength to guide the generation; instead, it can dynamically generate high-way covering array based on existing low-way covering array, which can support a flexible tradeoff between the covering array strength and testing resources. Cohen [28], [29] studied the impacts of constraints on CT, and proposed an SAT-based approach that can handle those constraints. Bryce and Colbourn [58] proposed an one-test-case-one-time greedy technique to not only generate test cases to cover all the t -degree interactions, but also prioritize them according to their importance. E. Dumlu et al., [4] developed a feedback driven combinatorial testing approach that can assist traditional approaches in avoiding the masking effects between multiple failures. Yilmaz [5] extended that work by refining the MFS diagnosing method. Specifically, this feedback driven approach firstly generates a t -way covering array, and after executing them, the MFS will be identified by utilizing a classification tree method. It then forbids these MFS and generates additional test cases to cover the interactions that are masked by the MFS. This process continues until all the interactions are covered. Additionally, Nie [59] constructed an adaptive combinatorial testing framework, which can dynamically adjust the inputs model, strength t of the covering array, and the generation strategy during CT process.

Our work differs from the above studies mainly in that we proposed a highly interactive framework for test case generation and MFS identification. Specifically, we do not generate a complete t -way covering array at first; instead, when a failure is triggered by a test case, we immediately terminate test case generation and turn to MFS identification. After the MFS is identified, the coverage will be updated and the test case generation process continues.

Besides the works on fault localization in combinatorial testing, some code-based fault localization studies also show some similarities with our work. Existing code-based fault localization can be mainly classified into two categories [60]: First, *statistical* approaches [61], [62], [63]. These approaches utilize the coverage of statements or other program entities in the execution traces of failed and passed tests to compute suspiciousness of each statement or other program entities. Then they will rank these program entities according to their likelihood of containing the defect, i.e., the computed suspiciousness scores. These approaches are effective, but may need sufficient test cases execution results. Second, *experimental* approaches [64], [65], [66]. By altering some inputs, code, or some other entities, these approaches can generate additional test cases. By comparing these test cases, as well as the testing outcomes, the failure-inducing parts of the test cases will be isolated. In fact, two MFS identification approaches are directly inspired by the delta debugging ideas [18], [25]. Additionally, a study [67] initially combines the MFS identification approach with code-based localization techniques to obtain a better fault isolation result.

From these works, the idea in BugEx [60] is quite similar to our approach, although they are applied different contexts. Specifically, the main task of BugEx is to automatically run tests and experiments to systematically narrow

down the failure causes. Unlike traditional fault localization approaches, this work also generates additional test cases. BugEx uses feedback from test outcomes to guide test generation, and also leverages test case generation for debugging purposes. We believe that this work can guide our work to further understand the MFS and failure-causing code.

Another work which shares similar ideas comes from the Software Product Lines (SPL) testing field [26], [68], [69], [70]. Many techniques in CT have been applied on SPL testing [70], among which Henard C, et al. [26] considered both test case generation and prioritizing (by selecting dissimilar tests). Also, our framework can be deemed as a solution to the test case generation and prioritization problem, which aims at fault localization as well as fault detection. As a result, it is appealing to apply our framework on the SPL testing problem. On the other hand, the idea of selecting dissimilar tests may be one potential solution to avoid multiple MFS appearing in one test case, which may improve the effectiveness of our framework.

7 CONCLUSION AND FUTURE WORKS

Combinatorial testing is an effective testing technique at detecting and diagnosis of the failure-inducing interactions in the SUT. Traditional CT separately studies test case generation and MFS identification. In this paper, we proposed a new CT framework, i.e., *interleaving CT*, that integrates these two important stages, which allows for both generation and identification better share each other's information. As a result, the interleaving CT approach can provide a more efficient testing than traditional sequential CT.

Empirical studies were conducted on five open-source software subjects. The results show that with our new CT framework, there is a significant reduction on the number of generated test cases when compared to the traditional sequential CT approach, while there is no decline in the quality of the identified MFS. Further, when comparing with another adaptive CT framework *fda-cit* [4], [5], our approach also performed better, especially with better quality of the MFS identification.

As a future work, we plan to extend our interleaving CT approach with more test case generation and MFS identification algorithms, to see the extent on which our new CT framework can enhance those different CT-based algorithms. Another interesting work is to combine the interleaving CT approach with the masking effects technique *fda-cit* [5]. By this, we believe the impacts of masking effects can be further reduced and it can support a better quality of MFS identification.

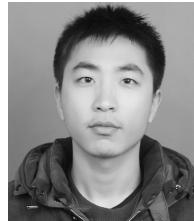
ACKNOWLEDGMENTS

This work was supported by the National Natural Science Foundation of China (No. 61272079), the Research Fund for the Doctoral Program of Higher Education of China (No.20130091110032), the Science Fund for Creative Research Groups of the National Natural Science Foundation of China(No. 61321491), the Major Program of National Natural Science Foundation of China (No. 91318301), and National Science Foundation Award CCF-1464425.

REFERENCES

- [1] D. R. Kuhn and M. J. Reilly, "An investigation of the applicability of design of experiments to software testing," in *Software Engineering Workshop, 2002. Proceedings. 27th Annual NASA Goddard/IEEE*. IEEE, 2002, pp. 91–95.
- [2] D. R. Kuhn, D. R. Wallace, and J. AM Gallo, "Software fault interactions and implications for software testing," *Software Engineering, IEEE Transactions on*, vol. 30, no. 6, pp. 418–421, 2004.
- [3] S. Yoo, M. Harman, and D. Clark, "Fault localization prioritization: Comparing information-theoretic and coverage-based approaches," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 22, no. 3, p. 19, 2013.
- [4] E. Dumlu, C. Yilmaz, M. B. Cohen, and A. Porter, "Feedback driven adaptive combinatorial testing," in *Proceedings of the 2011 International Symposium on Software Testing and Analysis*. ACM, 2011, pp. 243–253.
- [5] C. Yilmaz, E. Dumlu, M. Cohen, and A. Porter, "Reducing masking effects in combinatorial interaction testing: A feedback driven-adaptive approach," *Software Engineering, IEEE Transactions on*, vol. 40, no. 1, pp. 43–66, Jan 2014.
- [6] C. Nie and H. Leung, "The minimal failure-causing schema of combinatorial testing," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 20, no. 4, p. 15, 2011.
- [7] C. Nie and H. Leung, "A survey of combinatorial testing," *ACM Computing Surveys (CSUR)*, vol. 43, no. 2, p. 11, 2011.
- [8] D. M. Cohen, S. R. Dalal, M. L. Fredman, and G. C. Patton, "The aetg system: An approach to testing based on combinatorial design," *Software Engineering, IEEE Transactions on*, vol. 23, no. 7, pp. 437–444, 1997.
- [9] R. C. Bryce and C. J. Colbourn, "The density algorithm for pairwise interaction testing," *Software Testing, Verification and Reliability*, vol. 17, no. 3, pp. 159–182, 2007.
- [10] Y.-W. Tung and W. S. Aldiwan, "Automating test case generation for the new generation mission software system," in *Aerospace Conference Proceedings, 2000 IEEE*, vol. 1. IEEE, 2000, pp. 431–437.
- [11] M. B. Cohen, C. J. Colbourn, and A. C. Ling, "Augmenting simulated annealing to build interaction test suites," in *Software Reliability Engineering, 2003. ISSRE 2003. 14th International Symposium on*. IEEE, 2003, pp. 394–405.
- [12] K. J. Nurmiela, "Upper bounds for covering arrays by tabu search," *Discrete applied mathematics*, vol. 138, no. 1, pp. 143–152, 2004.
- [13] Y. Lei, R. Kacker, D. R. Kuhn, V. Okun, and J. Lawrence, "Ipog/ipog-d: efficient test generation for multi-way combinatorial testing," *Software Testing, Verification and Reliability*, vol. 18, no. 3, pp. 125–148, 2008.
- [14] C. J. Colbourn and D. W. McClary, "Locating and detecting arrays for interaction faults," *Journal of combinatorial optimization*, vol. 15, no. 1, pp. 17–48, 2008.
- [15] C. Martínez, L. Moura, D. Panario, and B. Stevens, "Algorithms to locate errors using covering arrays," in *LATIN 2008: Theoretical Informatics*. Springer, 2008, pp. 504–519.
- [16] C. Martínez, L. Moura, D. Panario, and B. Stevens, "Locating errors using elas, covering arrays, and adaptive testing algorithms," *SIAM Journal on Discrete Mathematics*, vol. 23, no. 4, pp. 1776–1799, 2009.
- [17] X. Niu, C. Nie, Y. Lei, and A. T. Chan, "Identifying failure-inducing combinations using tuple relationship," in *Software Testing, Verification and Validation Workshops (ICSTW), 2013 IEEE Sixth International Conference on*. IEEE, 2013, pp. 271–280.
- [18] Z. Zhang and J. Zhang, "Characterizing failure-causing parameter interactions by adaptive testing," in *Proceedings of the 2011 International Symposium on Software Testing and Analysis*. ACM, 2011, pp. 331–341.
- [19] L. S. G. Ghandehari, Y. Lei, T. Xie, R. Kuhn, and R. Kacker, "Identifying failure-inducing combinations in a combinatorial test set," in *Software Testing, Verification and Validation (ICST), 2012 IEEE Fifth International Conference on*. IEEE, 2012, pp. 370–379.
- [20] D. Jin, X. Qu, M. B. Cohen, and B. Robinson, "Configurations everywhere: Implications for testing and debugging in practice," in *Companion Proceedings of the 36th International Conference on Software Engineering*. ACM, 2014, pp. 215–224.
- [21] X. Qu, M. B. Cohen, and G. Rothermel, "Configuration-aware regression testing: an empirical study of sampling and prioritization," in *Proceedings of the 2008 international symposium on Software testing and analysis*. ACM, 2008, pp. 75–86.
- [22] C. Song, A. Porter, and J. S. Foster, "itree: Efficiently discovering high-coverage configurations using interaction trees," in *Proceedings of the 2012 International Conference on Software Engineering*. IEEE Press, 2012, pp. 903–913.
- [23] Z. Wang, B. Xu, L. Chen, and L. Xu, "Adaptive interaction fault location based on combinatorial testing," in *Quality Software (QSIC), 2010 10th International Conference on*. IEEE, 2010, pp. 495–502.
- [24] M. B. Cohen, M. B. Dwyer, and J. Shi, "Interaction testing of highly-configurable systems in the presence of constraints," in *Proceedings of the 2007 international symposium on Software testing and analysis*. ACM, 2007, pp. 129–139.
- [25] J. Li, C. Nie, and Y. Lei, "Improved delta debugging based on combinatorial testing," in *Quality Software (QSIC), 2012 12th International Conference on*. IEEE, 2012, pp. 102–105.
- [26] C. Henard, M. Papadakis, G. Perrouin, J. Klein, P. Heymans, and Y. Le Traon, "Bypassing the combinatorial explosion: Using similarity to generate and prioritize t-wise test configurations for software product lines," *Software Engineering, IEEE Transactions on*, vol. 40, no. 7, pp. 650–670, 2014.
- [27] C. Henard, M. Papadakis, M. Harman, Y. Jia, and Y. Le Traon, "Comparing white-box and black-box test prioritization," in *Proceedings of the 38th International Conference on Software Engineering*. ACM, 2016, pp. 523–534.
- [28] M. B. Cohen, M. B. Dwyer, and J. Shi, "Constructing interaction test suites for highly-configurable systems in the presence of constraints: A greedy approach," *Software Engineering, IEEE Transactions on*, vol. 34, no. 5, pp. 633–650, 2008.
- [29] M. B. Cohen, M. B. Dwyer, and J. Shi, "Exploiting constraint solving history to construct interaction test suites," in *Testing: Academic and Industrial Conference Practice and Research Techniques-MUTATION*, 2007. IEEE, 2007, pp. 121–132.
- [30] M. Grindal, J. Offutt, and J. Mellin, "Handling constraints in the input space when using combination strategies for software testing," 2006.
- [31] L. Yu, F. Duan, Y. Lei, R. N. Kacker, and D. R. Kuhn, "Constraint handling in combinatorial test generation using forbidden tuples," in *Software Testing, Verification and Validation Workshops (ICSTW), 2015 IEEE Eighth International Conference on*. IEEE, 2015, pp. 1–9.
- [32] M. Berkelaar, K. Eikland, and P. Notebaert, "Ip_solve 5.5, open source (mixed-integer) linear programming system," Software, May 1 2004, last accessed Dec, 18 2009. [Online]. Available: <http://ipsolve.sourceforge.net/5.5/>
- [33] K. Kolinko, "The HTTP Connector," <http://tomcat.apache.org/tomcat-7.0-doc/config/http.html>, 2014, [Online; accessed 3-Nov-2014].
- [34] K. Shakya, T. Xie, N. Li, Y. Lei, R. Kacker, and R. Kuhn, "Isolating failure-inducing combinations in combinatorial testing using test augmentation and classification," in *Software Testing, Verification and Validation (ICST), 2012 IEEE Fifth International Conference on*. IEEE, 2012, pp. 620–623.
- [35] D. R. Kuhn and V. Okun, "Pseudo-exhaustive testing for software," in *Software Engineering Workshop, 2006. SEW'06. 30th Annual IEEE/NASA*. IEEE, 2006, pp. 153–158.
- [36] D. Le Berre, A. Parrain et al., "The sat4j library, release 2.2, system description," *Journal on Satisfiability, Boolean Modeling and Computation*, vol. 7, pp. 59–64, 2010.
- [37] N. Eén and N. Sörensson, "An extensible sat-solver," in *Theory and applications of satisfiability testing*. Springer, 2004, pp. 502–518.
- [38] J. Zhang, F. Ma, and Z. Zhang, "Faulty interaction identification via constraint solving and optimization," in *Theory and Applications of Satisfiability Testing-SAT 2012*. Springer, 2012, pp. 186–199.
- [39] D. R. Kuhn, R. N. Kacker, and Y. Lei, "Practical combinatorial testing," *NIST Special Publication*, vol. 800, p. 142, 2010.
- [40] C. Yilmaz, M. B. Cohen, and A. A. Porter, "Covering arrays for efficient fault characterization in complex configuration spaces," *Software Engineering, IEEE Transactions on*, vol. 32, no. 1, pp. 20–34, 2006.
- [41] M. B. Cohen, J. Snyder, and G. Rothermel, "Testing across configurations: implications for combinatorial testing," *ACM SIGSOFT Software Engineering Notes*, vol. 31, no. 6, pp. 1–9, 2006.
- [42] S. Fouché, M. B. Cohen, and A. Porter, "Incremental covering array failure characterization in large configuration spaces," in *Proceedings of the eighteenth international symposium on Software testing and analysis*. ACM, 2009, pp. 177–188.
- [43] M. N. Borazjany, L. Yu, Y. Lei, R. Kacker, and R. Kuhn, "Combinatorial testing of acts: A case study," in *Software Testing, Verification*

- and Validation (ICST), 2012 IEEE Fifth International Conference on.* IEEE, 2012, pp. 591–600.
- [44] B. Garn and D. E. Simos, “Eris: A tool for combinatorial testing of the linux system call interface,” in *Software Testing, Verification and Validation Workshops (ICSTW), 2014 IEEE Seventh International Conference on.* IEEE, 2014, pp. 58–67.
- [45] R. C. Bryce, C. J. Colbourn, and M. B. Cohen, “A framework of greedy methods for constructing interaction test suites,” in *Proceedings of the 27th international conference on Software engineering.* ACM, 2005, pp. 146–155.
- [46] A. Hartman, “Software and hardware testing using combinatorial covering suites,” in *Graph theory, combinatorics and algorithms.* Springer, 2005, pp. 237–266.
- [47] N. Kobayashi, “Design and evaluation of automatic test generation strategies for functional testing of software,” *Osaka, Japan, Osaka Univ*, 2002.
- [48] A. W. Williams, “Determination of test configurations for pairwise interaction coverage,” in *Testing of Communicating Systems.* Springer, 2000, pp. 59–74.
- [49] A. W. Williams and R. L. Probert, “Formulation of the interaction test coverage problem as an integer program,” in *Testing of Communicating Systems XIV.* Springer, 2002, pp. 283–298.
- [50] R. C. Bryce and C. J. Colbourn, “One-test-at-a-time heuristic search for interaction test suites,” in *Proceedings of the 9th annual conference on Genetic and evolutionary computation.* ACM, 2007, pp. 1082–1089.
- [51] M. B. Cohen, P. B. Gibbons, W. B. Mugridge, and C. J. Colbourn, “Constructing test suites for interaction testing,” in *Software Engineering, 2003. Proceedings. 25th International Conference on.* IEEE, 2003, pp. 38–48.
- [52] S. A. Ghazi and M. A. Ahmed, “Pair-wise test coverage using genetic algorithms,” in *Evolutionary Computation, 2003. CEC’03. The 2003 Congress on*, vol. 2. IEEE, 2003, pp. 1420–1424.
- [53] T. Shiba, T. Tsuchiya, and T. Kikuno, “Using artificial life techniques to generate test cases for combinatorial testing,” in *Computer Software and Applications Conference, 2004. COMPSAC 2004. Proceedings of the 28th Annual International.* IEEE, 2004, pp. 72–77.
- [54] H. Wu, C. Nie, F.-C. Kuo, H. Leung, and C. J. Colbourn, “A discrete particle swarm optimization for covering array generation,” *Evolutionary Computation, IEEE Transactions on*, vol. 19, no. 4, pp. 575–591, 2015.
- [55] P. Schroeder, P. Bolaki, and V. Gopu, “Comparing the fault detection effectiveness of n-way and random test suites,” in *Empirical Software Engineering, 2004. ISESE’04. Proceedings. 2004 International Symposium on.* IEEE, 2004, pp. 49–59.
- [56] C. Nie, H. Wu, X. Niu, F.-C. Kuo, H. Leung, and C. J. Colbourn, “Combinatorial testing, random testing, and adaptive random testing for detecting interaction triggered failures,” *Information and Software Technology*, vol. 62, pp. 198–213, 2015.
- [57] L. Shi, C. Nie, and B. Xu, “A software debugging method based on pairwise testing,” in *Computational Science—ICCS 2005.* Springer, 2005, pp. 1088–1091.
- [58] R. C. Bryce and C. J. Colbourn, “Prioritized interaction testing for pair-wise coverage with seeding and constraints,” *Information and Software Technology*, vol. 48, no. 10, pp. 960–970, 2006.
- [59] C. Nie, H. Leung, and K.-Y. Cai, “Adaptive combinatorial testing,” in *Quality Software (QSIC), 2013 13th International Conference on.* IEEE, 2013, pp. 284–287.
- [60] J. Rößler, G. Fraser, A. Zeller, and A. Orso, “Isolating failure causes through test case generation,” in *Proceedings of the 2012 International Symposium on Software Testing and Analysis.* ACM, 2012, pp. 309–319.
- [61] J. A. Jones, M. J. Harrold, and J. Stasko, “Visualization of test information to assist fault localization,” in *Proceedings of the 24th international conference on Software engineering.* ACM, 2002, pp. 467–477.
- [62] B. Liblit, A. Aiken, A. X. Zheng, and M. I. Jordan, “Bug isolation via remote program sampling,” *ACM SIGPLAN Notices*, vol. 38, no. 5, pp. 141–154, 2003.
- [63] C. Liu, X. Yan, L. Fei, J. Han, and S. P. Midkiff, “Sober: statistical model-based bug localization,” in *ACM SIGSOFT Software Engineering Notes*, vol. 30, no. 5. ACM, 2005, pp. 286–295.
- [64] A. Zeller and R. Hildebrandt, “Simplifying and isolating failure-inducing input,” *Software Engineering, IEEE Transactions on*, vol. 28, no. 2, pp. 183–200, 2002.
- [65] A. Zeller, “Yesterday, my program worked. today, it does not. why?” in *Software Engineering/IEEE/FSE99.* Springer, 1999, pp. 253–267.
- [66] G. Mishergi and Z. Su, “Hdd: hierarchical delta debugging,” in *Proceedings of the 28th international conference on Software engineering.* ACM, 2006, pp. 142–151.
- [67] L. S. Ghandehari, Y. Lei, D. Kung, R. Kacker, and R. Kuhn, “Fault localization based on failure-inducing combinations,” in *Software Reliability Engineering (ISSRE), 2013 IEEE 24th International Symposium on.* IEEE, 2013, pp. 168–177.
- [68] G. Perrouin, S. Sen, J. Klein, B. Baudry, and Y. Le Traon, “Automated and scalable t-wise test case generation strategies for software product lines,” in *Software Testing, Verification and Validation (ICST), 2010 Third International Conference on.* IEEE, 2010, pp. 459–468.
- [69] R. E. Lopez-Herrejon, J. Javier Ferrer, F. Chicano, E. N. Haslinger, A. Egyed, and E. Alba, “A parallel evolutionary algorithm for prioritized pairwise testing of software product lines,” in *Proceedings of the 2014 conference on Genetic and evolutionary computation.* ACM, 2014, pp. 1255–1262.
- [70] R. E. Lopez-Herrejon, S. Fischer, R. Ramler, and A. Egyed, “A first systematic mapping study on combinatorial interaction testing for software product lines,” in *Software Testing, Verification and Validation Workshops (ICSTW), 2015 IEEE Eighth International Conference on.* IEEE, 2015, pp. 1–10.



Xintao Niu born in 1988, received his B.S degree from Nanjing University of Science and Technology. He is currently working toward the PhD degree in the Department of Computer Science and Technology at Nanjing University. His Research interest is software testing, especially on combinatorial testing and fault diagnosis. His work is supervised by Dr. Nie.



Changhai Nie A Professor of Software Engineering in National Key Laboratory for Novel Software Technology and Department of Computer Science and Technology at Nanjing University. His research interest is software testing and search base software engineering, especially in combinatorial testing, search based software testing, software testing methods comparison and combination and et al.



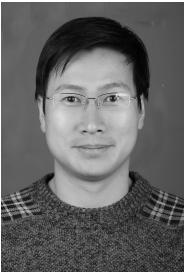
Hareton Leung received the PhD degree in computer science from University of Alberta. He is an associate professor and the director at the Laboratory for Software Development and Management in the Department of Computing, the Hong Kong Polytechnic University. He currently serves on the editorial board of Software Quality Journal and Journal of the Association for Software Testing. His research interests include software testing, software maintenance, quality and process improvement, and software metrics



Jeff Y. Lei is a full professor in Department of Computer Science and Engineering at the University of Texas, Arlington. He received his Bachelor's degree from Wuhan University (Special Class for Gifted Young), his Master's degree from Institute of Software, Chinese Academy of Sciences, and his PhD degree from North Carolina State University. He was a Member of Technical Staff in Fujitsu Network Communications, Inc. for about three years. His research is in the area of automated software analysis, testing and verification, with a special interest in software security assurance at the implementation level.



Xiaoyin Wang born in Harbin, Heilongjiang Province, China in 1984. From September 2006 to January 2012, he was a Ph.D. candidate in the Software Engineering Institute (SEI) of Peking University. His advisor is Prof. Hong Mei, and he also did research under the supervision of Prof. Lu Zhang and Prof. Tao Xie. From Oct 2008 to Sept 2009, he visited Singapore Management University as a research fellow, where he worked with Prof. David Lo. In Jan. 2012, he began to work with Prof. Dawn Song, as a PostDoc in UC Berkeley. In August 2013, he joined the computer science department of University of Texas at San Antonio.



Jiaxi Xu born in 1972, Senior Engineer in School of Information Engineering of Nanjing Xiaozhuang University. His research interest is software testing, especially embedded software testing and open source software testing.



Yan Wang received the MS degree in Control Theory and Control Engineering from University of Electronic Science and Technology of China. She is currently a lecturer in the School of Information Engineering at Nanjing Xiaozhuang University. Her research interests include development and testing of embedded software, and combinatorial interaction testing.