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

Xintao Niu, Changhai 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 so 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, which could affect the system’s behavior. Further effort is required to figure out the constraints and dependencies among 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 the 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 fulfill 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 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 integrates these two stages together instead of dividing the generation and identification into two independent stages. 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 does not contribute for additional failure detection. For the failure-inducing identification module, we augment it to achieve 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 is efficient at test case generation and MFS identification.

Our new framework has strict requirements in the 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 adequate testing will not be reached. To improve the accuracy of the failure-inducing interaction identification results, we propose a novel feedback 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 and several synthetic software to evaluate our new framework. These studies start with two comparisons. The first one is to compare our new interleaving framework with the traditional sequential framework, 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. We also evaluated the sensitivities of these approaches with respect to the number of the options of the system under test and the number of failure-inducing interactions contained in it. Besides, we discussed the negative influences of non-deterministic failures and the issue of a system with no option value that is irrelevant to any failure-inducing interaction (called the non-safe value issue). The main results of these experiments are summarized as follows:

1) Compared to the other approaches, our new interleaving framework obtained better failure-inducing interaction identification results in most cases (both empirical studies on real software and empirical studies on synthetic software). The new interleaving framework also decreased the number of generated test cases when compared with the traditional sequential framework in most cases, and it obtained a good result at the reduction of masking effects caused by different failure-inducing interactions even when compared to the feedback-driven CT which focuses on the reduction of masking effects.

2) Feedback-driven CT generated the smallest number of test cases in most cases, especially when the number of options of the system under testing is large, it also obtained a good result at the reduction of masking effects. As for traditional sequential framework, its results of these experiments lay in between those of the other two approaches in most cases.

3) The novel feedback checking mechanism benefits our new interleaving framework a lot, especially on the improvement of the accurateness of failure-inducing interaction identification and the coverage of interactions to be checked.

4) Increasing the number of failure-inducing interactions has a negative effect on all these approaches when considering the accurateness of failure-inducing interaction identification.

5) The non-deterministic failures also have a negative effect on these approaches, especially when the possibility of the appearance of failures ranges from 0.3 to 0.8. One potential solution is to increase the redundancy of test case execution.

6) Similar to the issue caused by a large number of failure-inducing interactions, the non-safe value issue also has a negative effect on all these three approaches, but the feedback mechanism can help our new interleaving framework to alleviate this negative effect to some extent.

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 feedback checking mechanism which can check whether the interaction identified by our approach is failure-inducing or not, and it significantly improves 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 results 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 a 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 $V_i$, $|V_i| = a_i$ ($i = 1, 2, \ldots, n$). 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 $(v_1, v_2, \ldots, v_n)$, where $v_1 \in V_1, v_2 \in V_2 \ldots v_n \in V_n$.

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 $(v_1, v_2, \ldots, v_n)$ 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 an $l$-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 $(\cdot, 4, 4, \cdot)$ is a sub-schema of the 3-degree schema $(\cdot, 4, 4, 5)$, that is, $(\cdot, 4, 4, 5) \prec (\cdot, 4, 4, 5)$.

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 designs 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 are considered to be detected. Let us formally define the covering array.

Definition 5. $\text{CA}(N; t, n; (a_1, a_2, \ldots, a_n))$ 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 $a_1 = a_2 = \ldots = a_n = v$, a $t$-way covering array can be denoted as $\text{CA}(N; t, n, v)$.

For example, Table 1 shows a 2-way covering array $\text{CA}(5; 2, 4, 2)$ 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].

![Fig. 1. The life cycle of CT](image-url)
2) A set of test cases one time: This strategy generates a set of test cases at each iteration. By mutating the values of some parameters of some test cases in this test set, it focuses on optimizing the coverage. If the coverage is finally satisfied, it will reduce the size of the set to see if fewer test cases can still fulfill the coverage. Otherwise, it will increase the size of the 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 are 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 gets a complete test case so 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 compared 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 \((0, 0, -, -), (0, -0, 0, -), (0, 0, 0, -), (0, 0, 0, 0), (-0, 0, 0, 0), (-0, 0, 0, 0)\), 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 \((v_1, v_2, ..., v_n)\), 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. 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 of which 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 case \( t \) at one time to generate new test cases: \( t_1 \) and \( t_2 \). It turns out that test case \( t_3 \) passed, which indicates that this test case breaks the MFS in the original test case \( t \). So \((1, -, -)\) should be a failure-causing factor. Besides, since other mutating test cases all failed, there is no any other failure-inducing factor that is broken. Therefore, the MFS in \( t \) is \((1, -, -)\).

This identification process mutates 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 an 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 propose 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.

Note that in practice, these assumptions do not always hold. Hence, the approaches proposed later in this paper actually can only identify approximate MFS instead of the real MFS. We will discuss the impacts of these assumptions on the approaches proposed in this paper in the experiments.

**Additionally, without special emphasis (for example, “the real MFS”), all the sentences contained such as “the MFS identified by some approaches” actually mean that “the approximate MFS obtained by these approaches”.

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 normally works under some proper configurations, but crashes or throws exceptions under some other configurations).

Considering the large configuration space of HSQLDB, we first 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 cared \(^1\). 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 successive 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 HSQLDB \(^2\), an incompatible exception will be triggered if a parameterised SQL is executed as a prepared statement by HSQLDB. Hence, when option Parameterised SQL is set to be true and Statement type 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 HSQLDB \(^3\). Without this knowledge at prior, we need to detect and isolate these two failure-inducing option interactions by CT.

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 Parameterised SQL and Statement type, the symbol is up to 1). According to Table 4, traditional CT first generated and executed the 2-way covering array \(t_1 - t_3\) in the generation part. Note that this covering array covered all the 2-degree schemas for the SUT.

After testing the 9 test cases \((t_1\text{ to }t_9)\), 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 \(t_1\), the OFOT method is used to generate four additional test cases \((t_{10} - t_{13})\), and the MFS \((\_0, 0, 1, -)\) of \(t_1\) is identified \((Scroll\ Type\ is\ assigned\ to\ sensitive,\ respectively)\). 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 to \(t_4\) and \(t_7\). Finally, we found that the MFS of \(t_4\) is \((\_0, 0, 1, -)\), 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 stars).

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 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 \((\_0, 0, 1)\), which corresponds to option Parameterised SQL being set to true and Statement Type being set to preparedStatement. More shortcomings of the sequential CT are discussed in the following subsections.

### 3.1 Redundant test cases

The first shortcoming of SCT is that it may generate redundant test cases so that some of them do not cover as many uncovered schemas as possible. As a consequence, 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 \((0, 1, 0, -)\) contained in the extra passing test case \(t_{11}\)

**TABLE 3**

<table>
<thead>
<tr>
<th>Option</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>(o_1)</td>
<td>Server type</td>
</tr>
<tr>
<td>(o_2)</td>
<td>Scroll type</td>
</tr>
<tr>
<td>(o_3)</td>
<td>Parameterised SQL</td>
</tr>
<tr>
<td>(o_4)</td>
<td>Statement Type</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Option Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>server, web-server, in-process</td>
</tr>
<tr>
<td>sensitive, insensitive, forward-only</td>
</tr>
<tr>
<td>statement, preparedStatement</td>
</tr>
</tbody>
</table>

**TABLE 4**

<table>
<thead>
<tr>
<th>Generation (Execution)</th>
<th>Outcome</th>
</tr>
</thead>
<tbody>
<tr>
<td>(t_1)</td>
<td>Fail</td>
</tr>
<tr>
<td>(t_2)</td>
<td>Pass</td>
</tr>
<tr>
<td>(t_3)</td>
<td>Fail</td>
</tr>
<tr>
<td>(t_4)</td>
<td>Fail</td>
</tr>
<tr>
<td>(t_5)</td>
<td>Fail</td>
</tr>
<tr>
<td>(t_6)</td>
<td>Pass</td>
</tr>
<tr>
<td>(t_7)</td>
<td>Fail</td>
</tr>
<tr>
<td>(t_8)</td>
<td>Pass</td>
</tr>
<tr>
<td>(t_9)</td>
<td>Pass</td>
</tr>
</tbody>
</table>

**Identification**

<table>
<thead>
<tr>
<th>Identification</th>
<th>Outcome</th>
</tr>
</thead>
<tbody>
<tr>
<td>(t_{10})</td>
<td>Fail</td>
</tr>
<tr>
<td>(t_{11})</td>
<td>Pass</td>
</tr>
<tr>
<td>(t_{12})</td>
<td>Fail</td>
</tr>
<tr>
<td>(t_{13})</td>
<td>Fail</td>
</tr>
</tbody>
</table>

**MFS**

<table>
<thead>
<tr>
<th>MFS</th>
<th>Outcome</th>
</tr>
</thead>
<tbody>
<tr>
<td>((_0, 0, -))</td>
<td>Fail</td>
</tr>
<tr>
<td>((1, 0, 0, 1))</td>
<td>Fail</td>
</tr>
</tbody>
</table>

2. For details, see: [http://sourceforge.net/p/hsqldb/bugs/1173/](http://sourceforge.net/p/hsqldb/bugs/1173/)
- (0, 1, 0, 0) has already appeared in the passing test case \( t_2 \) failed. In other words, if we first 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 \( t_6 \) superfluous for error-detection, the additional test cases \((t_{14} \text{ to } t_{21})\) 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 cannot 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 \((t_{14} \text{ to } t_{17})\) 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 \( t_3 \) to \( t_7 \)). Some approaches have been proposed to handle this problem, but they either cannot 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

When considering a single execution of the test set, traditional covering array usually offer 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. However, the test case is normally expected to test. For example in Table 4, \( t_4 \) is initially expected to cover six 2-degree schemas, i.e., \((0, 0, -, -), (0, 0, 0, -), (0, -, 0, 0), (-, 0, 0, -), (-, 0, -, 0), \) and \((-, -, 0, 0)\). The failure of this test case, however, may prevent the checking of these schemas. This is because the failing of \( t_1 \) (Scroll Type 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 alone cannot reach 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 it appears in a passing test case, or it is identified as MFS or faulty schema. Apparently, this criterion cannot be satisfied with traditional covering array alone (in practice, it is often the case that the test set is rerun until all test cases pass). 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, \( t_4 \) contained a single MFS \((-, 0, -, -)\), and we identified this MFS by generating four extra test cases \((t_{10} \text{ to } t_{13})\). 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 does 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 \(-(0, 0, -, -), (0, 0, 0, -), (0, 0, 0, 0)\) 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 \((0, -, 0, 0), (0, 0, 0, 0), (-, 0, 0, 0)\) are contained in the additionally generated test case \(t_{11}\) \((0, 0, 0, 0)\) (Note that for single MFS, there will be at least one passing additionally generated test case), which are of course non-faulty schemas. In the end, 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\((-, 0, -, -), (-, 0, 0, 1)\), which can not be identified with the OFOT approach (In fact, there is no passing additionally generated test case). As a result, there are two 2-degree schemas \((1, 0, -, -), (-, 0, 0, 1)\) 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 in this running example of this paper, masking effects are actually caused by the multiple MFS problem we discussed previously. However, 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 need 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 (t18 to t21) to figure out the failure-cause of t7 is indeed (τ, 0, τ, -). which has already been identified in previous iteration (t10 to t13). Therefore, we only need to check whether there is any MFS other than (τ, 0, τ, -) in t7 or not. When applying this augmentation, the overall SCT process of the example in Table 4 will evolve into the process shown in Table 5.

### TABLE 5
Augmented Sequential CT process

<table>
<thead>
<tr>
<th>Generation (Execution)</th>
<th>Outcome</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test case</td>
<td>01</td>
</tr>
<tr>
<td>t1</td>
<td>0</td>
</tr>
<tr>
<td>t2</td>
<td>0</td>
</tr>
<tr>
<td>t3</td>
<td>0</td>
</tr>
<tr>
<td>t4</td>
<td>1</td>
</tr>
<tr>
<td>t5</td>
<td>1</td>
</tr>
<tr>
<td>t6</td>
<td>1</td>
</tr>
<tr>
<td>t7</td>
<td>2</td>
</tr>
<tr>
<td>t0</td>
<td>1</td>
</tr>
<tr>
<td>t5</td>
<td>2</td>
</tr>
</tbody>
</table>

In Table 5, the only difference from Table 4 is that for test case t4 and t7, we first checked whether there is any MFS other than the already identified MFS (τ, 0, τ, -). Hence we generated two additional test cases t14 and t19 (highlighted), which exclude the MFS (τ, 0, τ, -) from the original failing test cases t4 and t7. Note that t14 (1, 1, 0, 1) was generated by mutating the value of the second parameter of test case t3 (1, 0, 0, 1) from 0 to 1, but it also can be any value different from the original value 0 of the second parameter in the test case t4, and as a result, it removed the previously identified MFS (τ, 0, τ, -). The same as t14, t19 (2, 1, 1, 1) was also generated by mutating the value of the second parameter of test case t7 (2, 0, 1, 1) from 0 to 1. We then found that t14 still failed after execution, which indicates that t14 contains other different MFS. So, we continued to use OFOT to identify the MFS of t14 and obtained the second MFS (τ, 0, τ, 1). With respect to t19, we found it passed after execution, and hence there is no other MFS in this test case, and we do not need to generate additional test cases. As a result, we have reduced the number of test cases by 2 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, e.g., multiple MFS, and masking effects, that it cannot deal with.

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 cooperate 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 words, the ultimate goal of this framework is to better support MFS identification and test case generation, so that both of them can alleviate the three problems 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 a pass (executed normally, i.e., does not trigger an exception, violate the expected Oracle or the like) or a 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-relevant. 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 must fail after execution. As a result, those test cases that contain the already identified MFS will never be covered in our new CT framework.

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 implicitly forbidden schemas, which was first introduced in [24]. This type of schemas is 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, 1) is the implicitly forbidden schema. This is because, for any test case that contains this schema, it must contain either (1, -1, 1) or (0, 1, -). As a result, (1, 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 first analyse the MFS that 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.

   \[
   t \leftarrow \text{select}(T_{all}, \Omega, \xi) \tag{EQ1}
   \]

   There are three factors that determine the selection of test case \( t \). \( T_{all} \) 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. \( \Omega \) indicates the set of schemas that have not been covered yet. \( \xi \) 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 \( \xi \) is used to help to escape from the local optimum (We will not discuss the specific usage of factor \( \xi \) in this paper, many papers that focus on generating covering array have given the specific implementations).

   As discussed in Section 3, we should make the MFS not appear in the test cases generated afterward, 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 \( T_{MFS} \) indicates the set of test cases that contain the already identified MFS, then the test case selection is augmented as EQ2.

   \[
   t \leftarrow \text{select}(T_{all} - T_{MFS}, \Omega, \xi) \tag{EQ2}
   \]

   In this formula, the only difference from EQ1 is that the candidate test cases that can be selected are changed to \( T_{all} - T_{MFS} \), which excludes \( T_{MFS} \) 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 differences 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 \( t_{\text{failing}} \) be the original failing test case, \( \Delta \) be the mutation parts, \( P \) be the parameters and their values, then the additional test case generation can be formulated as EQ3.

\[
\begin{align*}
\forall t &\leftarrow \text{mutate}(P, t_{\text{failing}}, \Delta) \quad \text{(EQ3)} \\
\end{align*}
\]

EQ3 indicates that the test case \( t \) is generated by mutating the part \( \Delta \) of the original failing test case \( t_{\text{failing}} \). Note that the mutated values may have many choices, as long as they are within the scope of \( P \) and different from those in \( t_{\text{failing}} \). For example, for the original failing test case \( t_1 (0, 0, 0, 0) \) in Table 4, let \( \Delta \) 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 \( T_{\text{candidate}} \), which can be formulated as EQ4.

\[
\begin{align*}
T_{\text{candidate}} &= \{ t \mid t \leftarrow \text{mutate}(P, t_{\text{failing}}, \Delta) \} \quad \text{(EQ4)} \\
\end{align*}
\]

Traditional MFS identification process just selects one test case from \( T_{\text{candidate}} \) 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 to 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 \((T_{\text{candidate}} - T_{\text{MFS}})\) and selects the additional test case which covers the greatest number of uncovered schemas \( (\Omega) \).

\[
\begin{align*}
t &\leftarrow \text{select}(T_{\text{candidate}} - T_{\text{MFS}}, \Omega, \xi) \quad \text{(EQ5)} \\
\end{align*}
\]

(3) Updating uncovered schemas: After the MFS are identified, some related \( t \)-degree schemas, i.e., \( MFS \) themselves, \( super-schemas \) and \( implicitly \ forbidden \) 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.

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 \( \Omega \) (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., \( implicitly \ forbidden \) 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 implicitly forbidden schema (line 12). The checking process involves solving a satisfiability problem. Specifically, if we can not find a test case from the set \((T_{\text{all}} - T_{\text{MFS}})\) (excluding those that contain MFS), such that it contains the schema under checking, then we can determine the schema is the implicitly 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 \( T_{\text{MFS}} \), which will be 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 the accuracy of the identified MFS. This is obvious, because if the schema identified is not an 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 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 \((1, -, -), (1, 1, -), (1, -, 1), (1, 1, 1)\) 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 a 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.

**Algorithm 2 Checking the MFS**

**Input:**
- `candi` → MFS that needs to be checked
- `Repeat` → The number of repeating times
- `Ω` → the schemas that are still uncovered
- `MFS_t` → all the valid test cases that contain MFS
- `candi_t` → all the valid test cases that contain `candi`

**Output:**
- `candi` is MFS or not

```plaintext
1: \( \text{Executed} \leftarrow \emptyset \)
2: while `Repeat` > 0 do
3: \( \text{possible} \leftarrow (MFS_t \cup \text{Executed}) \)
4: \( t_{\text{new}} \leftarrow \text{select\_dissimilar}(\text{possible}, \text{Executed}) \)
5: if `execute`(\( t_{\text{new}} \)) == PASS then
6: \( \text{update}(t_{\text{new}}, \Omega) \)
7: return False
8: end if
9: `Repeat` ← `Repeat` − 1
10: `Executed` append(\( t_{\text{new}} \))
11: end while
12: return True
```

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_{\text{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 new passing test case will contribute to more coverage), and directly return false (line 7). If we cannot find a test case 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 containing this schema), however, the probability that this schema is MFS increases with the increasing of checking strength, i.e., the value of `Repeat` variable. However, 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 a 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 schemas, 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).

It is worth noting that the feedback checking mechanism can also be embedded into SCT. Specifically, we can check the MFS obtained from each failing test case by generating additional test cases. Then, similar to ICT, we need to eliminate those MFS that cannot pass the verification and re-locate the MFS in the corresponding failing test case. However, for SCT, there are two facts that can negatively influence the improvement of the feedback checking mechanism. First, the effects of correcting wrongly identified MFS cannot be further propagated. That is, although we can fix the MFS identification result, it cannot be used in the following cases because the test case generation stage has already finished and some other MFS may never be detected. Second, it costs SCT more for embedding the feedback checking mechanism. This is because SCT needs to identify MFS for more failing test cases than ICT as we have discussed before, and for each failing test case, the feedback checking process needs to run at least one time. Our empirical study also exhibits this point. In fact, even without feedback checking mechanism, SCT still needed more test cases in the MFS identification stage than ICT with feedback checking mechanism (see Table 10 in Section 5.2).

### 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 forbid those invalid schemas to 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 forbid them in the following iterations [5]. We adopt the second method for handling constraints. There are two reasons for this choice. First, there are not many constraints in our empirical study such that the dynamic way of identifying them and forbidding them will not affect the efficiency too much. Second, the dynamic process of handling constraints is similar to the way that we identify the MFS, so our framework does not need to be modified a lot for handling constraints.

Specifically, when we execute invalid test cases which cannot be executed or even compiled, we will identify these invalid schemas which trigger this problem. In other words, 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 some possible implicitly illegal schemas to appear in the test cases generated later (through the same way for those identified MFS).

In a more detailed view, those forbidden schemas are formulated into clauses, as introduced in [28].
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 \{in-process, 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 1. By doing 
so, we can use SAT solver [32] to obtain a solution (that 
is, a test case that avoids these forbidden schemas). It is 
noted that, besides these forbidden schemas, there are 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].

There are two key parts in our constraints handling 
techniques. The first part is updating uncovered schemas. 
That is, after one constraint or one MFS is obtained, we will 
update all the schemas that are still needed to be covered. 
This part is done by computing the compatibility between 
the uncovered schemas with those known and discovered 
constraints [28]. After this, all the possible implicated con-
straints (Not known prior, nor explicitly discovered), and hence, our algorithm will not be stuck in the unstoppable 
condition that some schemas cannot be covered. The second part is that, for one test case that is generated by our 
approach, we will compute the satisfiability of the value under 
selected for each parameter. Specifically, for one pending 
vale of one specific parameter, we will first use SAT solver 
to find if there is a solution (one possible test case) that 
contain this value and not violate any of these constraints 
or MFS (including implicated ones). If the solver returns 
true, which means we can find one satisfied test case, then 
this value can be selected as one candidate value for that 
parameter. Otherwise, this value will be discarded.

4.3 Advantages of our framework

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

1) Redundant test cases are eliminated so that the 
overall cost is reduced.

Two facts of our framework support this improvement: (1) The schemas appearing 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 multi-
ple MFS in the same test case obviously improves the 
effectiveness of MFS identification.

3) The Masking effect is reduced, and 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 al-
leviate 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 
better adequate testing than SCT.

4) The quality of MFS identification is improved even 
if Assumption 2 is not satisfied.

As we have discussed in Section 2.2, the MFS iden-
tification approach used in our framework is based 
on the “Safe Value” Assumption (Assumption 2). 
In practice, however, this assumption is not always 
satisfied, which may result in a bad quality of MFS 
identification result. Under such condition, the feed-
back checking mechanism process can alleviate this 
issue and improve the quality of MFS identification.
Specifically, with additional test cases generated in 
the feedback checking mechanism process, we ob-
tain more chances to refine the MFS identification 
result, i.e., we can re-identify the MFS in the failing 
test case if the previous result cannot pass our 
validation. Note that the high quality of the MFS 
identification result is important to our framework 
because the test cases generated later by our frame-
work is heavily based on the previously identified 
MFS.

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.

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 (\( \gamma_0 \), 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 (\( \gamma_0 \), 
-), -) passed the verification, and would be regarded as MFS. 
Note that if either one of these two additional test cases 
passes, we will label (\( \gamma_0 \), 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 \( t_{0_1} t_{1_2} \) are two dissimilar 
test cases. In fact, all the 2-degree schemas that are covered by 
these two test cases are different.

After we determine (\( \gamma_0 \), 0, -) to be MFS, the following 
other test cases \( t_{1_2} t_{1_3} \) will not contain this schema. Corre-
respondingly, all the 2-degree schemas that are related to this 
schema, e.g., (0, 0, -), (\( \gamma_0 \), 0, 1, -), 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., $(0, 1, -, -), (0, 0, -), (0, 0, 0, -), (0, 1, 0, -), (1, 1, 0, -), (1, 0, 0, -)$ and $(-, 0, 0, -)$ respectively, so that it is not necessary to generate more test cases to cover them. We later found that $t_3$ failed, which only contained one MFS as expected, and we easily identified it $(-, 1, 0, -)$ 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 cases are 2 less than that of the traditional sequential CT approach in Table 4 and equal to the augmented sequential CT approach in Table 5. In fact, if we exclude the test cases from the checking process, the interleaving CT approach can reduce even more test cases (6 less than that of the traditional sequential CT approach, and 4 less than the augmented sequential CT approach). However, these additional test cases generated in the checking process will ensure a high quality of MFS identification for interleaving CT approach. In this simple example, both interleaving CT and augmented sequential CT correctly identified all the MFS (better than that of traditional sequential CT), but given more complex subjects with more MFS, we believe interleaving CT can outperform the augmented sequential CT at MFS identification.

Note that in this example, our approach did not wrongly identify the MFS, and hence, this example did not show how ICT handles the circumstance if Algorithm 2 returns false (i.e., if one passing test case is found containing the previously identified MFS in the checking process). Next, we use a simple example to show how ICT works in such condition. Let a SUT have four parameters, of which $p_1$, $p_2$, $p_3$, and $p_4$ are ternary options. There are two MFS in this SUT, which are $(0, 0, 0, -)$ and $(1, 0, 0, -)$, respectively. Now we assume that ICT start with a failing test case $(0, 0, 0, 0)$. Table 7 shows how ICT works in this condition.

In Table 7, we can observe that at the first time, we wrongly identified the MFS. Specifically, after four test cases ($t_1$, $t_2$, $t_3$, and $t_4$) generated by ICT, we identified schema $(-, 0, 0, -)$ as the MFS instead of the real MFS $(0, 0, 0, -)$. The reason why it fails obtaining the real MFS is that $t_1$ introduced the new MFS $(1, 0, 0, -)$. It violated the safe assumption as we discussed in Section 2.2 (Assumption 2 in the last two paragraphs) and hence, it cannot obtain the real MFS. After this, ICT needed to check this schema by generating additional test case $t_5$ $(2, 0, 0, 1)$. It passed during testing, which indicated that we wrongly identified the MFS, i.e., $(-, 0, 0, -)$ is not the real MFS. Then ICT re-started the MFS identification procedure and generated additional four test cases, i.e., $t_7$, $t_8$, $t_9$, and $t_{10}$. Note that in the second MFS identification procedure, ICT needed to generate test cases as different as what has been already generated as possible to cover more un-covered test cases. In the second iteration of the MFS identification, ICT correctly identified the real MFS $(0, 0, 0, -)$. ICT then checked this schema by two test cases $t_5$ and $t_{10}$. Since these two test cases both failed, $(0, 0, 0, -)$ was identified to be the MFS at last. Note that in the second checking procedure, there did not exist other test cases contain the schema $(0, 0, 0, -)$, and hence, we could only use these two already generated test cases to check this schema. In fact, under this condition, all the possible test cases, i.e., $t_1, t_2, t_3,$ and $t_4$, that containing this schema $(0, 0, 0, -)$ were failed. As a result, $(0, 0, 0, -)$ is exactly the MFS according to what MFS is declared (Definition 4).

### 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?

---

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2018.2865772, IEEE Transactions on Software Engineering

0098-5589 (c) 2018 IEEE. Translations and content mining are permitted for academic research only. Personal use is also permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.
Q3: How much does ICT gain from the feedback checking mechanism?

Q4: Does ICT have any advantages over the existed masking effects handling technique — FDA-CIT [5]?

Q5: How well do these approaches perform on software subjects with multiple defects?

Q6: What is the sensibility of our approach to a different number of MFS and a different number of options in SUT?

Q7: How well does our approach perform when the two assumptions listed in Section 2 do not hold?

Q8: How about the static way, i.e., the Error Locating Arrays, of handling combinatorial test generation and fault localization?

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 8. 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.

Table 8: Subject programs

<table>
<thead>
<tr>
<th>Subjects</th>
<th>Version</th>
<th>LOC</th>
<th>Faults</th>
<th>Lan</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tomcat</td>
<td>7.0.40</td>
<td>296138</td>
<td>#59035</td>
<td>java</td>
</tr>
<tr>
<td>Hsqldb</td>
<td>2.0.8</td>
<td>139425</td>
<td>#98149</td>
<td>c</td>
</tr>
<tr>
<td>Gcc</td>
<td>4.7.2</td>
<td>10046</td>
<td>#987</td>
<td>c</td>
</tr>
<tr>
<td>Jflex</td>
<td>1.4.2</td>
<td>173</td>
<td>#98</td>
<td>c</td>
</tr>
</tbody>
</table>

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; Tcas is a module of an aircraft collision avoidance system. We select these programs 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, with a relatively small size (only 173 lines), it has 12 parameters with their values ranging from 2 to 10. As a result, the overall input space for Tcas can reach 460800 [34], [35].

As the main target of our empirical studies is to compare the ability to handle the proposed three issues 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 were caused by the interaction of configuration options. Then for each such bug, we derived 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 a 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 8 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 9.

Table 9: Inputs model

<table>
<thead>
<tr>
<th>Subjects</th>
<th>Inputs</th>
<th>MFS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tomcat</td>
<td>2 × 41 × 41</td>
<td>1(1) 2(2)</td>
</tr>
<tr>
<td>Hsqldb</td>
<td>2 × 41 × 41</td>
<td>3(3)</td>
</tr>
<tr>
<td>Gcc</td>
<td>2 × 61</td>
<td>3(4)</td>
</tr>
<tr>
<td>Jflex</td>
<td>2 × 2 × 41</td>
<td>2(1)</td>
</tr>
<tr>
<td>Tcas</td>
<td>2 × 3 × 41 × 102</td>
<td>9(16) 10(8) 11(16) 12(8)</td>
</tr>
</tbody>
</table>

In this table, Column “inputs” depicts the input model for each version of the software, presented in the abbreviated form \#values\#number of parameters × ... , e.g., 2 × 32 × 41 indicates the software has 9 parameters that can take on 2 values, 2 parameters taking on 3 values and only one parameter taking 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 someone 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 by ICT 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 focuses on the overall CT process.

With respect to SCT, we used the augmented simulated annealing approach [11], [38] to build covering array. The heuristic search-based algorithm is known to produce smaller covering arrays than the one test case at one time approach. Hence, using this approach is fairer for the approach SCT than using greedy approach (which may result in a larger size of covering array) because it needs to firstly generate a complete covering array.

5.2.1 Study setup
For each software except Teas, a test case was determined to be passing if it ran without any exception; otherwise, it was regarded as failing. For Teas, as the fault is injected, we determined the result of a test case by separately running and comparing the original correct version and the mutated version.

In this experiment, we focused 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 and simulated annealing algorithm. So to avoid the biases of randomness, we conducted each experiment 30 times and then evaluated the results. (Note that the remaining case studies were also based on 30 repeated experiments.) For each run of the experiment, we separately applied SCT approach and our approach to 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 were generated in the CT generation and MFS identification, respectively, were 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 compared to the real MFS. Recall measures how well the real MFS are detected and identified. Their combination is F-measure, defined as

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

5.2.2 Result and discussion
Table 10 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, in most cases, the number of test cases generated by our approach was smaller 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 Teas was smaller, because most of the MFS of Teas are of high degree (t > 6), and the covering arrays (t = 2, 3, 4) rarely detected any of them.). This result indicates that ict was more efficient at both CT generation activity and MFS identification activity.

For Gcc, however, we found 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). However, 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 were 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 subject Gcc was far larger than that of other subjects.

The quality of the identified MFS for other subjects is also listed in Table 11. Based on this table, we found 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 were 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 Teas, at which failures are hard to detect, the f-measure of ict was 0.01 for 4-way coverage, while sct scored 0. This result indicates that ict was far more effective at MFS identification than sct.

Another interesting observation with regard to the MFS identification is that higher t-wise strengths were not always resulting in an improved precision (Take subject Hsqldb for example, the f-measure of ict and sct for 3-way coverage were 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 lower strength covering array is enough to detect the MFS. Specifically, a t-wise covering array can detect all the failures caused by the MFS of t-degree, or less than t-degree. Then, if an MFS is detected, ict and sct can identify them as expected. A higher-wise covering array can certainly detect those low degree MFS too, but compared to the lower strength 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 12 shows the milliseconds consumed by the two approaches on average. The experiment was conducted on Machine HP ProDesk 600 G1 TWR (Intel Core i5, 3.3Hz, 16GB memory). Based on this table, it is obvious that ict cost more time than sct. This is because ict needed to handle the SAT problem (for forbidding the appearance of MFS and constraints), which consumed 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 10

Comparison of the number of test cases

<table>
<thead>
<tr>
<th>Subjects</th>
<th>Method</th>
<th>2-way</th>
<th>3-way</th>
<th>4-way</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Gen</td>
<td>Iden</td>
<td>Total</td>
<td>Gen</td>
</tr>
<tr>
<td>Tomcat</td>
<td>ict</td>
<td>8.3</td>
<td>54.2</td>
<td>60.7</td>
</tr>
<tr>
<td></td>
<td>sct</td>
<td>13.8</td>
<td>55.0</td>
<td>68.3</td>
</tr>
<tr>
<td>Hsqldb</td>
<td>ict</td>
<td>11.7</td>
<td>37.8</td>
<td>49.4</td>
</tr>
<tr>
<td></td>
<td>sct</td>
<td>15.6</td>
<td>32.3</td>
<td>47.9</td>
</tr>
<tr>
<td>Gcc</td>
<td>ict</td>
<td>14.0</td>
<td>28.0</td>
<td>41.4</td>
</tr>
<tr>
<td></td>
<td>sct</td>
<td>14.6</td>
<td>20.1</td>
<td>34.4</td>
</tr>
<tr>
<td>Jflex</td>
<td>ict</td>
<td>14.6</td>
<td>17.0</td>
<td>31.6</td>
</tr>
<tr>
<td></td>
<td>sct</td>
<td>15.9</td>
<td>16.6</td>
<td>32.5</td>
</tr>
<tr>
<td>Tcas</td>
<td>ict</td>
<td>109.1</td>
<td>0.0</td>
<td>109.1</td>
</tr>
<tr>
<td></td>
<td>sct</td>
<td>107.5</td>
<td>0.0</td>
<td>107.5</td>
</tr>
</tbody>
</table>

### Table 11

Comparison of the quality of the identified MFS

<table>
<thead>
<tr>
<th>Subjects</th>
<th>Method</th>
<th>2-way</th>
<th>3-way</th>
<th>4-way</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Gen</td>
<td>Iden</td>
<td>Total</td>
<td>Gen</td>
</tr>
<tr>
<td>Tomcat</td>
<td>ict</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td></td>
<td>sct</td>
<td>1.0</td>
<td>0.77</td>
<td>0.83</td>
</tr>
<tr>
<td>Hsqldb</td>
<td>ict</td>
<td>0.75</td>
<td>1.0</td>
<td>0.86</td>
</tr>
<tr>
<td></td>
<td>sct</td>
<td>0.75</td>
<td>0.77</td>
<td>0.83</td>
</tr>
<tr>
<td>Gcc</td>
<td>ict</td>
<td>0.45</td>
<td>0.28</td>
<td>0.73</td>
</tr>
<tr>
<td></td>
<td>sct</td>
<td>0.45</td>
<td>0.28</td>
<td>0.73</td>
</tr>
<tr>
<td>Jflex</td>
<td>ict</td>
<td>0.13</td>
<td>0.07</td>
<td>0.20</td>
</tr>
<tr>
<td></td>
<td>sct</td>
<td>0.13</td>
<td>0.07</td>
<td>0.20</td>
</tr>
<tr>
<td>Tcas</td>
<td>ict</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td></td>
<td>sct</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
</tbody>
</table>

### Table 12

Time consumes (millisecond)

<table>
<thead>
<tr>
<th>Subject</th>
<th>Method</th>
<th>2-way</th>
<th>3-way</th>
<th>4-way</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tomcat</td>
<td>ict</td>
<td>156.4</td>
<td>270.7</td>
<td>1256.7</td>
</tr>
<tr>
<td></td>
<td>sct</td>
<td>10.0</td>
<td>56.7</td>
<td>305.6</td>
</tr>
<tr>
<td>Hsqldb</td>
<td>ict</td>
<td>345.5</td>
<td>203.6</td>
<td>21918.4</td>
</tr>
<tr>
<td></td>
<td>sct</td>
<td>16.7</td>
<td>151.3</td>
<td>1055.1</td>
</tr>
<tr>
<td>Gcc</td>
<td>ict</td>
<td>180.1</td>
<td>1117.5</td>
<td>5408.5</td>
</tr>
<tr>
<td></td>
<td>sct</td>
<td>8.0</td>
<td>68.0</td>
<td>309.3</td>
</tr>
<tr>
<td>Jflex</td>
<td>ict</td>
<td>187.1</td>
<td>1747.1</td>
<td>11412.4</td>
</tr>
<tr>
<td></td>
<td>sct</td>
<td>75.5</td>
<td>288.8</td>
<td>2491.4</td>
</tr>
<tr>
<td>Tcas</td>
<td>ict</td>
<td>178.0</td>
<td>2914.9</td>
<td>6025.5</td>
</tr>
<tr>
<td></td>
<td>sct</td>
<td>135.6</td>
<td>1750.0</td>
<td>25380.7</td>
</tr>
</tbody>
</table>

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2018.2865772, IEEE Transactions on Software Engineering
Third, we used the tested-t-way coverage criterion [5] to measure the masking effects. Specifically, we re-computed the coverage of the test cases generated by ICT and SCT by counting all the t-degree schemas that were 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 (range from 2 - 4 way). 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. For example in the first sub-figure (2-way for Tomcat), two bars with x-coordinate equal to 1 indicates that ICT approach had 61.5 schemas on average which were covered once and SCT had 1.3 schemas.

As discussed previously, the more schemas are covered with a low-frequency, the less redundant the generated test cases are. Hence it implies effective testing if the number of schemas (y-axis) decreases with the increase of the covered times (x-axis). With respect to Figure 3, it is easy to find that for most of the 15 sub-figures, ICT performed better than SCT. In fact, for ICT, the bars decreased rapidly with the increase of the’ x-axis, while for SCT, the trend was more smooth. See subject tomcat with 2-way coverage, for example, ICT had about 61.5 schemas which were only covered once, about 38.9 schemas covered twice, less than 12 schemas covered more than 6 times. For SCT, however, for most covered times, it had about 10 schemas, which indicates a very low performance.

The interesting exception is subject Tcas, on which ICT and SCT showed 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 detected any of them. Under this condition, since both approaches rarely detected the MFS, the overall process was transferred to be traditional covering array generation (the MFS identification process is omitted).

This result shows that our two modifications of the 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 13, which lists the number of test cases that 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 was limited in a very small number (smaller than 1). For SCT, however, the result was not as good as ICT. In fact, except for subjects Jflex and Tcas, SCT suffered from generating test cases which contain multiple MFS. This is one reason why even though SCT generated many more test cases than ICT, it did not obtain a better MFS identification result than ICT. Two 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 9) 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 14. 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). 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 a sequential way or interleaving way) can make testing more adequate than using covering array alone.

Second, ICT was 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 fell 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 were rarely detected, and all the t-degree schemas appeared in the passing test cases).

In summary, the answer to Q2 is that our approach ICT can alleviate the three problems 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 in reducing the masking effects.

5.4 The benefits of feedback checking mechanism

One important part of the ICT approach is the feedback checking mechanism, which aims at judging whether the schemas identified by ICT is real MFS or not by additionally generating test cases containing the schemas under check. It is interesting to evaluate how valuable is this feedback checking mechanism, i.e., how much improvement ICT gained from this mechanism.
### 5.4.1 Study setup

For this, we created a mutation version of ict by removing the feedback checking mechanism from the original ict approach. We later call this mutation approach the ict-nonfb. Then, we applied this approach to test the five subjects listed in Table 8 and identified the MFS contained in them. At last, we evaluated the benefits of the feedback checking mechanism by comparing the results obtained by ict-nonfb and ict.

### 5.4.2 Result and discussion

We list the results of the number of test cases generated by ict-nonfb in Table 15, the f-measure of MFS identification in Table 16, the average number of test cases containing multiple MFS in Table 17, and the tested-t-way coverage in Table 18. Additionally, we attached the gaps between ict-nonfb with ict in the parentheses. The value with a negative sign indicates the reduction in the corresponding metric (e.g., number of test cases, the f-measure, the number of test cases containing multiple MFS, the tested-t-way coverage) made by ict-nonfb when compared with ict, while non-negative sign indicates the increase in that corresponding metric.

The following could be observed:

1) ict-nonfb generated a smaller amount of test cases than ict. Specifically, except for the tcas program subject, ict-nonfb reduced the number of test cases by about 1.9 to 23.3. This is as expected because the feedback checking mechanism needs to generate additional test cases to check whether the schemas identified by ict is real MFS or not.
2) The quality of the MFS identification of ict-nonfb decreased a lot. In fact, except for the 2-way coverage of Gcc, ict either obtained higher f-measures or performed equally well on all the remaining subjects of all the t-ways (2, 3, and 4-way coverage). Additionally, the gaps between them ranged from 9% to 55.7%, which is not trivial.

3) There were no distinct gaps between ict-nonfb and ict at the number of test cases that containing multiple MFS. In fact, for all the subjects except for Gcc, ict-nonfb and ict both generated 0 test case that containing multiple MFS. For Gcc, ict-nonfb performed better at 2-way (but the gap is only 0.1) and 3-way coverage, while ict performed better at 4-way coverage.

4) The tested-t-way coverage of ict-nonfb also decreased. In fact, besides those subjects that ict-nonfb and ict performed equally well, ict-nonfb reduced the tested-t-way coverage by about 0.02% (0.2 tested-2-way schemas) to 4.94% (296.4 tested-4-way schemas).

To summarize, the answer to Q3 is that: Without feedback checking mechanism, the number of test cases generated by ict reduced, but the quality of MFS identification and tested-t-way coverage decreased significantly. It indicates that the additional test cases generated in feedback checking mechanism is worthwhile, and it is beneficial to adopt feedback checking mechanism in the CT process (in order to obtain a better MFS identification result and a higher tested-t-way coverage).

### 5.5 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 appear 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 the following 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 are 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 criterion is not the same. fda-cit directly uses the tested-t-way coverage to guide their process. This supports 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.
5.5.1 Study setup

The design of this case study is similar to the previous two. For each subject in Table 8, we applied \textit{fda-cit} to generate test cases and identify the MFS. After that, we gathered the overall test cases generated (\textit{fda-cit} does not need additional test cases to identify the MFS), MFS identification results (including recall, precision, and \textit{F}-measure), and the other three metrics, i.e., covered times of schemas, the number test cases which contain multiple MFS, and the tested-t-coverage. The same as previous experiments, we repeated each experiment 30 times for different coverage (2, 3, and 4 way), and then gathered and analysed the average data. Note that the MFS identification approach in the \textit{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]. Another point needs to be noted is that we also used the augmented simulated annealing approach [11], [38] to build covering array for the FDA-CIT.

5.5.2 Result and discussion

1) Total number of test cases. The total number of test cases generated by \textit{fda-cit} for each subject is shown in Table 19. To better evaluate the performance of \textit{fda-cit}, we list the gaps between FDA-CIT with \textit{ict} and \textit{sct} respectively in the parentheses (the first number is for \textit{ict}, the second one is \textit{sct}). The value with a negative sign indicates the reduction in the test cases between \textit{fda-cit} and other two approaches, while the value without negative sign indicates the number of test cases which \textit{fda-cit} generated more than the other two approaches.

From this table, one observation is that \textit{fda-cit} was better than \textit{sct} in almost all cases. Combining the results of previous studies for \textit{sct} and \textit{ict}, we can conclude that \textit{sct} was the most inefficient approach at test case generation. Second, for \textit{ict} and \textit{fda-cit}, there were ups and downs on both sides. In detail, \textit{fda-cit} needed fewer test cases at lower coverage (2-way and 3-way coverage), while \textit{ict} performed better at higher coverage (4-way).

This result is reasonable. First, \textit{fda-cit} did not need additional test cases to identify the MFS, which would reduce some cost when compared with \textit{ict}, especially when the coverage is low (For low coverage, the test cases generated by \textit{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 would make \textit{fda-cit} generate some unnecessary test cases.

2) \textit{F}-measure of MFS identification. The results of the quality of MFS identification by \textit{fda-cit} is listed in Table 20. Same as the previous metric, the comparison between \textit{fda-cit} with \textit{ict} and \textit{sct} is also attached (the first number is for \textit{ict}, the second one is \textit{sct}).

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

This result suggests that the classification tree approach used by \textit{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 \textit{fda-cit}'s primary concern is to avoid masking effects and to give every \textit{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 [39]. 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 \textit{fda-cit} tending to be super-schema of the real MFS.

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 \textit{ict} and \textit{sct}, we attach the data for \textit{ict} and \textit{sct}, with the solid line and dotted line, respectively.

From this figure, we can see the trend of the bars of \textit{fda-cit} matches pretty well with the curve representing \textit{ict}, which has a significant advantage over the curve of \textit{sct}. This result implies that the test case redundancy of \textit{ict} is similar to that of \textit{fda-cit}, which is not severe when compared with \textit{sct}.

4) Test cases containing multiple MFS. Table 21 shows the number of test cases that contain multiple MFS on average for \textit{fda-cit}. The same as before, we also list the gaps between \textit{fda-cit} with \textit{ict} and \textit{sct} respectively in the parentheses (the first number is for \textit{ict}, the second one is for \textit{sct}). From this table, we can easily find that \textit{fda-cit} did almost the same as \textit{sct} at restricting the appearance of test cases that contain multiple MFS. However, both of them did not as well as \textit{ict}. In fact, except for subject \textit{Jflex} (which contains single MFS) and \textit{Tcas} (which contains high-degree MFS that are rarely detected), \textit{fda-cit} generated more test cases that contain multiple MFS than \textit{ict}, and the gap between them increased with the increase of test coverage (\textit{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 \textit{ict} was 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 \textit{ict} obtained a higher-quality of MFS identification.

5) Masking effects. The result is listed in Table 22, which shows the results of tested-t-way coverage. The gaps between \textit{fda-cit} with \textit{ict} and \textit{sct} are listed in the parentheses, respectively (the first one is \textit{ict}, the second one is \textit{sct}).

With regard to tested-t-way coverage, we can find that our approach \textit{ict} was still the best approach at reducing the masking effects, even though when compared with approach \textit{fda-cit}. In fact, among the 15 cases listed in Table 22, there are 13 cases on which \textit{ict} performed equal to or better than \textit{fda-cit} (The only two exceptions are \textit{Gcc} for 3-way and 4-way coverage). Note that in some particular cases, the gaps between \textit{ict} and \textit{fda-cit} are not trivial, e.g., \textit{ict} obtained 10 more percent of tested-t-way coverage than \textit{fda-cit} at 2-way coverage for subject \textit{Tomcat}).
TABLE 19
Number of test cases generated by *fda-cit*

<table>
<thead>
<tr>
<th></th>
<th>2-way</th>
<th>3-way</th>
<th>4-way</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tomcat</td>
<td>28.4(-32.3,-39.9)</td>
<td>65.5(-44.4,-34.2)</td>
<td>147.4(-17.2,-39.9)</td>
</tr>
<tr>
<td>Hsqldb</td>
<td>29.9(-19.5,-18.0)</td>
<td>83.5(-48.9,-29.8)</td>
<td>201.2(34.9,-35.3)</td>
</tr>
<tr>
<td>Gcc</td>
<td>21.7(-19.7,-12.7)</td>
<td>63.4(-25.6,-16.8)</td>
<td>120.7(24.0,-19.4)</td>
</tr>
<tr>
<td>Jflex</td>
<td>19.8(-11.8,-12.7)</td>
<td>64.5(-11.1,-9.5)</td>
<td>179.5(28.8, 11.8)</td>
</tr>
<tr>
<td>Tcas</td>
<td>109.9(0.8, 2.4)</td>
<td>416.6(-1.1,-1.7)</td>
<td>1544.7(-8.1,-14.0)</td>
</tr>
</tbody>
</table>

TABLE 20
The F-measure of MFS identification for *fda-cit*

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

Above all, this result suggests that our approach *ict* does reach the same or better level when compared with *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 accurate is more important.

To summarize, the answer to Q4 is that when compared to the adaptive CT approach *fda-cit*, *ict* did better at MFS identification, reduction of masking effects, and reduction of test cases containing multiple MFS in most cases, while *fda-cit* generated a smaller number of test cases.

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 configu-
ration. 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, ict 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. In this case, fda-cit needs a smaller amount of test cases [5].

5.6 Multiple defects

Since there is only one defect in each software subject used in the previous experiments, it is interesting to observe how well these approaches work on programs with multiple defects. To identify the MFS in the programs with multiple defects is more complex than in the software with a single defect. One problem is that one defect may crash the system under test so that other defects will not have the chance to be triggered. Even worse, some defects may have interference with each other [40], e.g., constructive and destructive interference [41], making fault localization more difficult. For all these reasons, it is important to conduct experiments on multiple defects.

5.6.1 Study setup

The software subjects with multiple bugs used in this experiment are listed in Table 23. In this table, we listed corresponding versions of each software, lines of code, number of classes, the bug IDs, their corresponding input model, and MFS information.

Note that in this study we only selected 2 out of 5 subject applications to experiment with multiple defects. The reason for this is that it is very hard and time-consuming to obtain reproducible testing scenarios that contain multiple option-related defects. In order to simplify the process, we adopted the strategy to select a small number of subject applications, but for each subject application, we obtained more different versions of that application. By doing so, we can also reuse the test scripts we have built for a subject application. As a result, for the experiments of multiple defects, we have built five different versions of subject applications, of which the number is equal to the number of the subject applications used in the experiment for a single defect. This is also why we only use a limited number of bugs for these applications. Also, we believe it is common that the test cases under execution have limited number of multiple bugs. This is because in practice if there are too many distinct defects is the SUT, it needs human re-examination rather than just doing an automatic diagnosis [18].

For each version of the subjects, we applied the previous three approaches, i.e., ict, sct, and fda-cit, on generating test cases and fault diagnosis. It is noted that, for sct and ict, we need to distinguish different faults for them. In our experiments, we simply took the one-bug-at-a-time strategy [40]. More specifically, when identifying the MFS for one particular defect, we only labeled the test cases failed with this specific defect as fail, and labeled other test cases (either passed after execution or failed with other defects) as pass.

5.6.2 Result and discussion

We list the results of the number of test cases generated in this experiment in Table 24, the f-measure of MFS identification in Table 25, the average number of test cases containing multiple MFS in Table 26, and the tested-t-way coverage in Table 27.

There are several observations in the experiments with multiple defects:

1) The results of the number of test cases satisfied the following relationship: fda-cit generated the smallest number of test cases in most cases, and the second-best was ict, while the last one was sct. Specifically, fda-cit reduced the number of test cases by 20.36 on average at 2-way coverage when compared with the approach sct, and 19.2 at 3-way coverage, and 65.7 at 4-way coverage. fda-cit also reduced the number of test cases by about 20.6 when compared with ict at 2-way coverage, but generated slightly more test cases than ict at 3-way coverage and 4-way coverage (increased of 2.7 and 0.6, respectively). With respect to ict, it reduced the number of test cases by about 21.9 and 66.4 at 3-way and 4-way coverage, respectively, when compared with sct. These two approaches generated almost the same number of test cases at 2-way coverage.

2) With respect to the quality of MFS identification, these three approaches satisfied the following relationship: ict obtained the highest score at MFS identification, followed by sct and fda-cit. In fact, except for the 2-way coverage at which ict and sct obtained almost the same f-measure on average, ict increased the f-measure at least by 30% and 32% on average, respectively, at 3-way and 4-way coverage when compared with other two approaches.

3) The results that related to the number of test cases containing multiple MFS satisfied the following relationship: ict generated the smallest number of test cases that containing multiple MFS in most cases, and the second-best approach was fda-cit, while the last one was sct. Specifically, ict reduced the number of test cases containing multiple MFS by about 1.0 at the 2-way coverage when compared with fda-cit, 3.84 at the 3-way coverage, and 10.04 at the 4-way coverage. For fda-cit, it reduced the number of test cases by about 0.2 at the 3-way coverage when compared with sct, and 2.52 at the 4-way coverage. (These two approaches generated a similar number of test cases that containing multiple schemas at 2-way coverage).
4) Concerning the tested-t-way coverage, these three approaches satisfied the following relationship: \textit{sct} covered the most number of tested-t-way schemas, followed by approaches \textit{fda-cit} and \textit{ict}, respectively. In fact, except for the 2-way coverage at which \textit{sct} and \textit{ict} covered almost the same number of tested-t-way schemas, \textit{sct} outperformed the other two approaches at 3-way and 4-way coverage. Specifically, \textit{sct} increased the tested-3-way coverage by 20% and 16%, when compared with approaches \textit{ict} and \textit{fda-cit}, respectively, and increased the tested-4-way coverage by 38% and 16%, respectively.

The reason why \textit{ict} was clearly outperformed by \textit{sct} at reducing masking effects under multiple defects (this is the only different conclusion when compared with the results of Section 5.5) is that the decreasing of the tested-t-way coverage of \textit{ict} was caused by the reduction of passing test cases. This is because due to the one-bug-at-one-time strategy for handling multiple defects, \textit{ict} labeled the test cases which failed with the defects other than the defect under analysis as passing test cases. As a consequence, it can normally identify the MFS for the defect, but these test cases which failed with other defects cannot contribute to any tested-t-way coverage. Therefore, the tested-t-way coverage obtained by \textit{ict} decreased.

Above all, the answer to Q5 is:

Except for the masking effects, other results matched well with the results obtained from the experiments of a single defect. Specifically, \textit{ict} obtained the best MFS identification results and generated the least number of test cases containing multiple MFS, \textit{sct} obtained the most tested-t-way coverage, and \textit{fda-cit} generated the smallest number of test cases.

5.7 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 numbers of MFS in the SUT and different numbers of options in the SUT on three approaches, i.e., \textit{ict}, \textit{sct}, and \textit{fda-cit}.

5.7.1 Study setup

To vary parameters of interest in a controlled setting in this study, we used synthetic subjects instead of real programs (the real program typically represents 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 numbers of
MFS, we used the subject with 11 parameters, and each parameter had 5 values, i.e., the inputs model is \((5^{11})\). Then we considered the following possible numbers of 2-degree MFS: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 20, 30, 40, 50, 60, 70, 80 and 90. The detailed information of each synthetic subject application is shown in Table 31 in Appendix B. Then for each run of the experiment, we first injected the corresponding number of MFS into the synthetic subject and then ran all the three approaches on the subject. At last, the results of each approach were collected and analysed.

The second study is to evaluate the performance of approaches under various numbers of options. Hence we used synthetic subjects with the following numbers of options \((8, 9, 10, 12, 16, 20, 30, 40, 50, 60, 70, 80, 90, 100)\). The detailed information of each synthetic subject application is shown in Table 32 in Appendix B. Each option had two values, and each subject had three 2-degree MFS. Then for each subject, we applied the three approaches and compared their performance.

### 5.7.2 Number of MFS

The results for the sensitivity of the number of MFS are shown in Figure 6, 7 and 8, of which the first figure focuses on the quality of MFS identification, the second figure shows the cost, and the last one shows the results of the masking effects.

![Fig. 6. F-measure for various numbers of MFS](image)

One observation from Figure 6 is that, with the increasing of the number of MFS, the f-measures of all three approaches decreased rapidly. In fact, when the number of MFS was greater than 60, the f-measures of all three approaches were near 0. This is mainly because if there were too many MFS, it was hard to get a passing test case, and hence, it was challenging to distinguish MFS from those schemas which were not related to the failures.

Another observation is that for most cases, ict performed the best, then followed by sct, and the last was 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 numbers of MFS](image)
Regarding the masking effects, one observation is that, with the increase in the number of MFS, the tested-t-way coverage of all these three approaches decreased. This is because, with the increase in the number of MFS, the number of passing test cases decreased, i.e., test cases are more likely to fail with these MFS. Worse, since the MFS quality also decreased with the increase in the number of MFS, these approaches can hardly find any schema that satisfies the tested-t-way coverage criteria. Another observation is that when the number of MFS is relatively high, \textit{fda-cit} obtained a slightly higher score at the value of tested-t-way coverage when compared to the other two approaches. We believe it is because that the test cases generated by \textit{fda-cit} contain more passing test cases. Approaches \textit{sct} and \textit{ict}, on the contrary, generated more failing test cases (in the MFS identification stage, \textit{sct} and \textit{ict} generate test cases that are similar to the original failing test case with only one value mutation. As a consequence, these test cases are more likely to fail, especially at the condition of there are many MFS in the SUT).

![Fig. 8. The tested-t-way coverage for various numbers of MFS](image)

Considering that approaches \textit{ict} and \textit{sct} need to identify the MFS in each of the failing test cases which may contain single MFS or multiple MFS, it is very interesting to observe the performance for these two approaches on the test cases that containing multiple MFS only. Hence, we filtered the results obtained from those failing test cases that only contain single MFS, and focused on those test cases that contain multiple MFS. The MFS identification results (multiple MFS) are listed in Figure 9. Additionally, we attached the decrease of f-measure of these two approaches when compared with the results on the test cases that are not distinguished by containing single MFS and multiple MFS in Figure 10. Note that there is no data at 1 on the x-axis because there is no test case containing multiple MFS in this condition (the SUT only contains one MFS).

![Fig. 9. F-measure (multiple MFS in one test case) for various numbers of MFS](image)

We can first observe that approach \textit{ict} outperformed \textit{sct} on MFS identification on test cases that containing multiple MFS. In fact, for all the cases listed in Figure 9, \textit{ict} obtained higher scores of f-measure than \textit{sct} (note that for all the cases, the f-measure of \textit{sct} is under 0.1). The gaps between them ranged from 0.05 to 0.69, which was not trivial. Second, the condition that multiple MFS appear in one test case has significant negative effects on \textit{sct}, while only has a relatively slight influence on \textit{ict}. Specifically, the decrease of f-measure of \textit{sct} (when compared with the f-measure obtained by \textit{sct} on test cases that are not distinguished by single MFS and multiple MFS) ranged from 0.01 to 0.73, while the decrease of f-measure of \textit{ict} was no more than 0.32. In fact, there are three cases (x-axis of 6, 80, and 90) on which \textit{ict} even performed better than before.

Above all, with the increasing of the number of MFS in the SUT, the performance of all three approaches decreased, but \textit{ict} still performed better than the other two approaches.

### 5.7.3 Number of options

The results for the sensitivity of the number of options are shown in Fig. 11, Fig. 12 and Fig. 13, which depicts the quality of MFS identification, the number of generated test cases, and the results of masking effects, respectively.

![Fig. 11. F-measure for various numbers of MFS](image)

With regard to the quality of MFS identification, it is clear that \textit{ict} performed the best, then followed by \textit{sct}, and the
parameter options of the SUT. Hence, the complexity of the MFS identification algorithm (OFOT) used in our approach is \( O(N) \), while the complexities of others are \( O(\log N) \). In our paper, we did not use other algorithms proposed in [17], [18], [25]. Hence, it is possible that the number of test cases generated in this paper would be further reduced. The last one was \( \text{sct} \), of which the number of test cases was always larger than that of \( \text{ict} \).

In regard to the masking effects, we can observe that the number of options had no influence on the results of \( \text{ict} \). Specifically, \( \text{ict} \) always covered all the tested-t-way schemas no matter what was the number of the options. For the other two approaches, i.e., \( \text{sct} \) and \( \text{fda-cit} \), their tested-t-way coverage increased with the increase in the number of options. This is because with the increase in the number of options, the number of all the schemas that need to be covered increased, but the number of MFS deceased. Hence, the tested-t-way coverage of these two approaches increased.

Therefore, the number of options in the SUT did not have much influence on the quality of MFS identification; and although generating more test cases than \( \text{fda-cit} \), \( \text{ict} \) was still a better choice when considering the quality of MFS identification and the reduction of masking effects.

In summary, the answer to Q6 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, concerning various numbers of MFS, \( \text{ict} \) obtained the best MFS identification results and generated the smallest number of test cases in most cases, \( \text{fda-cit} \) obtained the highest covered-t-coverage. As for various numbers of options, \( \text{ict} \) still did the best at MFS identification, and it also obtained the highest covered-t-coverage, while \( \text{fda-cit} \) generated the smallest number of test cases. Besides, in these two conditions, the results obtained by \( \text{sct} \) lay in between those of the other two approaches in most cases.

5.8 The ability of handling assumptions

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

5.8.1 Study setup

The same as the previous study, in order 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 non-deterministic failures with various probabilities that can be triggered during testing, such that it helps us to evaluate the performance of these approaches for various extents 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 the MFS that is non-deterministic (the test case which
contains it may fail or may not after execution). Then we considered the following possible probabilities that the non-deterministic 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, respectively. The detailed information of each synthetic subject application is shown in Table 33 in Appendix B. We repeated the experiment 30 times for each probability to avoid the random effects. For each run of the experiment, we applied all the three approaches on the subject and recorded their results (MFS identification quality and cost).

The second study is to evaluate the performance of approaches when the safe value assumption does not hold. In fact, in our previous studies, the safe value assumption was also not always hold. For example, in the first study, we did not give any safe value to our approach ict. Instead, we just generated additional test cases containing the schemas under test. As a result, we did not always reach 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 input model, and the information of MFS are listed in Table 28. The same as Table 9, input model is presented in the abbreviated form \#values\#number of parameters × ..., and Column “MFS” shows the degrees of each MFS and the number of MFS (in the parentheses) with that corresponding degree. The detailed information of each synthetic subject application is shown in Table 34 in Appendix B. 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.

<table>
<thead>
<tr>
<th>Subjects</th>
<th>Inputs</th>
<th>MPS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Syn1</td>
<td>3</td>
<td>(6)</td>
</tr>
<tr>
<td>Syn2</td>
<td>2(10)</td>
<td>(4)</td>
</tr>
<tr>
<td>Syn3</td>
<td>4(12)</td>
<td>(7)</td>
</tr>
<tr>
<td>Syn4</td>
<td>4(16)</td>
<td>(5)</td>
</tr>
<tr>
<td>Syn5</td>
<td>2(10)</td>
<td>(4)</td>
</tr>
<tr>
<td>Syn6</td>
<td>2(10)</td>
<td>(4)</td>
</tr>
<tr>
<td>Syn7</td>
<td>2(6)</td>
<td>(4)</td>
</tr>
<tr>
<td>Syn8</td>
<td>2(4)</td>
<td>(4)</td>
</tr>
<tr>
<td>Syn9</td>
<td>2(4)</td>
<td>(4)</td>
</tr>
<tr>
<td>Syn10</td>
<td>2(6)</td>
<td>(4)</td>
</tr>
</tbody>
</table>

5.8.2 Non-deterministic failures

The results of evaluating how well these approaches handle non-deterministic failures are shown in Fig. 14, Fig. 12, and Fig. 16, of which the first figure depicts the quality of MFS identification with various probabilities that the non-deterministic failures are triggered, the second figure shows the number of test cases, and the last one shows the results of the masking effects.

With regard to MFS identification, there are two observations. First, if the probability was below 0.5, all the three approaches did not identify any MFS at all (with f-measure of 0). We believe there are two possible reasons for the low f-measure of all the three approaches. The first one is that if the probability of triggering MFS was too small (below 0.2), approaches could hardly detect the failure, and hence could not identify the MFS. Another one is that if the probability of triggering MFS is around 0.5, then the failure may appear at one testing, but disappear at the next time. These two statuses exchanged frequently and resulted in a negative influence on the MFS identification.

The second observation is that when the probability of triggering MFS was larger than 0.5, the f-measure of all the three approaches increased. In fact, when the probability of triggering MFS was larger than 0.9, the f-measure of all the three approaches was more than 0.4. We believe if the probability was relatively high, then all the three approaches could easily detect it. Under this condition, the failure was similar to a deterministic failure, which also had little influence on the MFS identification.

With regard to the test cases, our conclusion is similar
to the previous study, that is, \textit{fda-cit} needed the smallest amount of test cases, then followed by \textit{ict} and \textit{sct}.

With respect to the masking effects, we can make the following observation. If the probability was too small (below 0.15) or too high (above 0.9), all the three approaches performed relatively well. This is reasonable because if the probability of triggering MFS was too small, approaches could hardly detect the failure. As a result, there are more passing test cases generated by these three approaches, and more tested-t-way schemas can be checked. On the other hand, if the probability was too high, all these three approaches could easily identify the MFS accurately. Under this condition, the results of the tested-t-way coverage obtained by these three approaches were similar to the condition for handling deterministic failure. The worst condition was in between (the probability was around 0.3 to 0.8). Under such condition, all these three approaches can neither identify the MFS accurately nor generate a large number of passing test cases.

Since the non-deterministic failures have negative effects on MFS identification, it is desirable to alleviate the effects. In this paper, we consider the redundancy of test case execution, i.e., we repeatedly run one test case to check whether it fails or not instead of just one time. We conducted an additional experiment to evaluate the performance of this strategy. Specifically, all the experimental set-ups are as the same as the previous experiment on non-deterministic failures, except that we set the redundancy of test execution to be 5 (run 5 times for each test case).

Figure 17 shows the results. From this figure, we can easily learn that for all these three approaches, there was a significant improvement in the quality of MFS identification. In fact, all these three approaches start to identify at least one MFS among 30 times even the probability of triggering MFS was as low as 0.2. Additionally, when the probability was larger than 0.4, the f-measure of all the three approaches was larger than 0.4. What’s more, the f-measure of all the three approaches was close to 1 when the probability was larger than 0.5. These results indicated that the redundancy of test case execution is one potential approach to handle the non-deterministic failures problem.

5.8.3 Non-Safe values
The results of evaluating how well these approaches handle non-safe values are shown in Fig. 18, Fig. 19, and Fig. 20.

There are several observations about these figures. First, the non-safe values did affect the MFS identification quality of all the approaches. In fact, the f-measures of all the three approaches listed in Fig. 19 were lower than those 0.7. Specifically, \textit{ict}’s f-measure ranged from 0.52 to 0.67, \textit{sct}’s ranged from 0.42 to 0.64, and \textit{fda-cit}’s ranged from 0.01 to about 0.08. Based on these data, we can conclude our second observation, that is, \textit{ict} also performed best under the condition that there are no safe values. We believe this is due to our feedback checking process, which significantly improves the quality of MFS identification, and reduces the negative effects caused by non-safe values. At last, with regard to the number of test cases, \textit{fda-cit} still generated the fewest, but this also led to the low f-measure of MFS identification.

Concerning the masking effects, these three approaches satisfied the following relationship: \textit{fda-cit} obtained the highest score at the tested-t-way coverage, followed by \textit{ict}, and
sct was the last one. In fact, this condition is similar to the condition of there is a large number of MFS in the SUT (there are many MFS so that there are no safe values in our experiments). As discussed in Section 5.7.2, when the number of MFS is relatively high, fda-cit generated more passing test cases than approaches sct and ict. Hence, fda-cit performed the best among the three approaches. The reason that ict performed better than sct is that ict identified more MFS than sct. As a result, ict obtained more schemas that satisfied the tested-t-way coverage.

Besides these observations, it is also important to figure out how many times that the effects of non-safe values were triggered. More specifically, for the approaches ict and sct which need to identify the MFS for each of the failing test cases, we need to figure out how many times when these MFS actually caused failures during the MFS identification for one specific failing test case. We listed the results in Figure 21 and Figure 22, in which Figure 21 recorded the number of total times that the non-safe MFS are triggered for each software subject, while Figure 22 recorded average number of times that the non-safe MFS are triggered for each time of MFS identification.

![Fig. 21. The times of non-safe MFS are triggered in total for each subject](image1)

![Fig. 22. The times of non-safe MFS are triggered for each time of MFS identification](image2)

Based on these two figures (Figure 21 and 22), we can learn that for both of two approaches, the number of times that non-safe MFS was triggered was not trivial. In fact, except for the first subject, the number that non-safe MFS was triggered by these two approaches was both beyond 73.6 times for the remaining subjects (the maximal non-safe MFS’s triggered times of sct was up to 572.8, and for ict, this number was up to 385.8). Additionally, the number that non-safe MFS was triggered for each time that MFS identification proceeded was still not small. Specifically, for each time of MFS identification, these non-safe MFS were triggered by about 22 times at average for ict, and 7.3 times for sct.

Above all, the answer to the Q7 is:

The non-deterministic failures and non-safe values do negatively affect the results of all the three approaches. Besides, concerning the condition of non-safe values, ict obtained the best MFS identification results and performed better than sct at the reduction of test cases and masking effects, while fda-cit generated the smallest number of test cases and obtained the highest covered-t-coverage. As for non-deterministic failures, ict still obtained the best MFS identification results and the second highest covered-t-coverage in most cases, fda-cit did the best at the reduction of test cases, while sct obtained the highest covered-t-coverage in most cases. Moreover, one potential solution for handling non-deterministic failures is the redundancy of test case execution.

### 5.9 Comparison with static error locating arrays

Considering that all these approaches evaluated in our previous experiments are all dynamic approaches (generating test cases), it is interesting to observe how well does the alternative way, i.e., static error locating arrays, perform on the CT problems.

Error locating array [14], [16] is a well-designed set of test cases that can support not only failure detection but also the identification of the MFS of the failure. It is known that only with a covering array sometimes is not sufficient to identify the MFS; thus additional test cases are needed. Martínez et al. [15] have proved that a \((t + d)\)-way covering array can identify all the MFS with the number of them no more than \(d\) and degree no more than \(t\). After executing all the test cases in the \((t + d)\)-way covering array, the MFS can be obtained by keeping those \(t\)-degree or less than \(t\)-degree schemas that only appear in the failing test cases. So with the number \(d\) and degree \(t\) known in prior, a \((t + d)\)-way covering array is an Error Locating Array (ELA).

To compare our approach with this Error Locating Array is meaningful, as both approaches have the same target. The relationship between our approach with the Error Locating Array can be deemed as the dynamic vs. static. In detail, our approach dynamically detects and identifies the MFS in the SUT, i.e., the test cases generated by our approach are changed according to the specific MFS. On the contrary, ELA just generates a static covering array, and it can support MFS identification if the number and degree of these schemas are known in prior.

#### 5.9.1 Study setup

In this section, we will apply ELA to identify the MFS of the 5 subjects in Table 8. It is noted that the conclusion that a \((t + d)\)-way covering array is an ELA is based on that there must exist safe values for each parameter of the SUT. A safe value is the parameter value that is not in any part of these MFS. In our experiment, all the five subject programs satisfy this condition. Based on this, we then applied ELA to generate appropriate covering arrays for
each subject program and recorded the MFS identified as well as the overall test cases generated. The covering array generation algorithm we adopted in this experiment is also augmented simulated annealing approach [11], [38], and similar to previous experiments, this experiment is repeated 30 times.

5.9.2 Result and discussion
First, we list the subject information (the number of MFS \(d\) and maximal MFS degree \(t\)), together with the covering arrays that ELA needs to build, in Table 29.

<table>
<thead>
<tr>
<th>Subject</th>
<th>(t)</th>
<th>(d)</th>
<th>(t+d)</th>
<th>ELA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tomcat</td>
<td>3</td>
<td>5</td>
<td>8</td>
<td>CA(N,5,10,((2^3 \times 3^3 \times 4^4)))</td>
</tr>
<tr>
<td>Hsldb</td>
<td>3</td>
<td>6</td>
<td>10</td>
<td>CA(N,6,12,((2^3 \times 3^3 \times 4^4)))</td>
</tr>
<tr>
<td>Gcc</td>
<td>3</td>
<td>7</td>
<td>13</td>
<td>CA(N,7,10,((2^3 \times 6^4)))</td>
</tr>
<tr>
<td>Jflex</td>
<td>1</td>
<td>3</td>
<td>5</td>
<td>CA(N,3,13,((2^{10} \times 3^3 \times 4^4)))</td>
</tr>
<tr>
<td>Tecas</td>
<td>48</td>
<td>60</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

From this table, we can learn that for most subjects, ELA needs to build high-way covering arrays. In fact, for tecas, the CA that is needed to be built is 60-way, which is far beyond the number of parameters of TCAS (12) and hence, the corresponding covering array cannot be generated. One exception is Jflex, for which ela needs to build a 3-way covering array. It indicates that the 4-way covering arrays generated for Jflex in the previous experiments are unnecessary. From this point of view, ELA can provide an upper-bound on the strength of the covering arrays that need to be generated for one software subject.

The number of overall test cases, the quality of the identified MFS, and the results of masking effects are listed in Table 30. We can firstly observe that this approach needs more test cases than the approaches discussed before. This is as expected because this approach needs to generate a higher-way covering array than previous approaches. Apart from the high cost, this approach correctly identified all the real MFS. The accuracy has been proved in [15], [16]. Note that this perfect MFS identification result is based on the fact that it knows the number and degree of the MFS at first, which is usually not available in practice. With respect to the masking effects, ELA nearly obtained the 100% tested-t-way coverage of most subjects. They do so at the cost of using significantly more tests. Additionally, in the case when the MFS is of low degrees, ELA obtained a low percentage of the tested-t-way coverage when \(t\) is relatively high (See Tested-3-way and Tested-4-way of Jflex in Table 30).

In summary, the answer to the Q8 is:
ELA gets the best quality of the MFS identification and reduction of masking effects, but it needs much more cases than all the other dynamic approaches. It also needs to know the number and degree of the MFS in prior, which limits its application in practice.

5.10 Threats to validity
There are two threats to validity in our empirical studies. First, our experiments are based on 5 open-source software. However, these software programs only represented some specific inputs space and specific degree or location of the MFS. To make the conclusion more general, we need to observe how these approaches performed on other inputs models with different characteristics. For this, we created a batch of synthetic input models. More specifically, for the synthetic models used in Section 5.7, we consider how approaches performed under different numbers of parameters (8 to 100) instead of some specific number of parameters obtained from previous experiments. Additionally, we also consider the different numbers of MFS (1 to 90) that contained in these synthetic models instead of only 1 or 2. In Section 5.8, these models are created by considering the various probabilities of triggering MFS (1% to 98%) instead of 100% or only some specific probabilities. We also considered the number of times (1 to 47) that unsafe values are introduced each time the MFS identification proceeded in the experiments. Another point that is related to the subject application is that the number of test cases needed by the proposed approach would grow linearly with the number of configuration options of the subject application under testing.

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 of 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 [42], especially on domains like configuration testing [43], [44], [45] and software inputs testing [8], [46], [47]. 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], [48], which are very fast and effective, but may consume too many test cases. 2) mathematical methods [49], [50], [51], [52], 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], [53], [54], [55], [56], [57], which can generate very small size of test cases, but may cost much computation time and 4) random methods [58], [59], 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], [60] or nonadaptive–additional test cases are chosen independently and can be executed in parallel [14], [15], [16], [39], [43].
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 sometimes 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 requirements for the strength of covering array [45], 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., [45] 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 [61] proposed a 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 [62] 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 [63]:

First, statistical approaches [64], [65], [66]. 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 [67], [68], [69]. 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 [70] 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 [63] is quite similar to our approach, although they are applied to 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], [71], [72], [73]. Many techniques in CT have been applied on SPL testing [73], 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 to 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 for 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, which 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 more efficient testing than augmented sequential CT.
Empirical studies were conducted on five open-source software subjects and several other synthetic software. The results showed that when compared to the other approaches, icl obtained better MFS identification results in most cases (both empirical studies on real software and empirical studies on synthetic software). icl also decreased the number of generated test cases when compared with sct, and it obtained a good result at the reduction of masking effects between different MFS even when compared to fdacit. As for fdacit, it generated the smallest number of test cases in most cases, especially when the number of options is large. It also obtained a good result when handling masking effects. The results obtained by sct of these experiments lay in between those of the other two approaches in most cases. Additionally, we learned that there are several factors that may have negative effects on these approaches, which are the large number of MFS, the non-deterministic failures (especially when the possibility of the appearance of failures ranged from 0.3 to 0.8), and the non-safe values, respectively. The feedback checking mechanism and redundancy of test case execution may help to alleviate these negative effects to some extent.

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 fdacit [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

We would like to thank anonymous referees for their suggestions and comments on this paper. This work was supported by the National Key Research and Development Plan (No. 2018YFB1003800), US National Science Foundation Award CNS-1748109, US Department of Homeland Security (DHS-14-ST-062-001), and US National Institute of Standards and Technologies Award(70NANB15H199).

REFERENCES


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.