Identifying minimal failure-causing schemas in the presence of multiple failures

XINTAO NIU and CHANGHAI NIE, State Key Laboratory for Novel Software Technology, Nanjing University
JEFF Y. LEI, Department of Computer Science and Engineering, The University of Texas at Arlington
Hareton Leung and ALVIN CHAN, Department of computing, Hong Kong Polytechnic University

Combinatorial testing (CT) has been proven effective in revealing the failures caused by the interaction of factors that affect the behavior of a system. The theory of Minimal Failure-Causing Schema (MFS) has been proposed. The use of MFS helps to isolate the root cause of a failure after detected by CT. Most algorithms that aim to identify MFS focus on handling a single failure in the System Under Test (SUT). However, we argue that multiple failures are the more common testing scenario, under which masking effects may be triggered so that some failures cannot be observed. The traditional MFS theory, as well as the related identifying algorithms, lack a mechanism to handle such effects; hence, they may incorrectly isolate the MFS in the SUT. To address this problem, we propose a new MFS model that takes into account multiple failures. We first formally analyse the impact of the multiple failures on existing MFS identifying algorithms, especially in situations where masking effects are triggered by multiple failures. We then develop an approach that can assist traditional algorithms to better handle multiple failures testing scenario. Empirical studies were conducted using several kinds of open-source software, which showed that multiple failures with masking effects do negatively affect traditional MFS identifying approaches and that our approach can help to alleviate these effects.

Categories and Subject Descriptors: D.2.5 [**Software Engineering**]: Testing and debugging—*Debugging aids,testing tools*

General Terms: Reliability, Verification

Additional Key Words and Phrases: Software Testing, Combinatorial Testing, Failure-causing schemas, Masking effects

ACM Reference Format:
Xintao Niu, Changhai Nie, Jeff Y. Lei, Hareton Leung and Alvin Chan, 2014.Identifying minimal failure-causing schemas in the presence of multiple failures. *ACM Trans. Embedd. Comput. Syst.* 9, 4, Article 39 (March 2010), 45 pages.
DOI:http://dx.doi.org/10.1145/0000000.0000000

1. INTRODUCTION

With the increasing complexity and size of modern software, many factors, such as input parameters and configuration options, can affect the behaviour of the SUT. The unexpected failures caused by the interaction of these factors can make software testing challenging, especially when the interaction space is large. In the worst case, we need to examine every possible interaction of these factors as each interaction may

---

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

DOI:http://dx.doi.org/10.1145/0000000.0000000

Table I. MS word example

id	*Highlight*	*Status bar*	*Bookmarks*	*Smart tags*	**Outcome**
1	On	On	On	On	PASS
2	Off	Off	On	On	PASS
3	Off	On	Off	Off	Fail
4	On	Off	Off	On	PASS
5	On	Off	On	Off	PASS

cause unique failure [Song et al. 2012]. While exhaustive testing achieves maximal test coverage, it is impractical and uneconomical. One remedy for this problem is combinatorial testing, which systematically samples the interaction space and selects a relatively small set of test cases that cover all valid interactions, with the number of factors involved in each interaction no more than a prior fixed integer, i.e., the *strength* of the interaction. Many works in CT aim to construct the smallest set of test cases [Cohen et al. 1997; Bryce et al. 2005; Cohen et al. 2003; Lei et al. 2008], which is also called *covering array*.

Once failures are detected by a covering array, the failure-inducing interactions in the failing test cases should be isolated. This task is important as it can facilitate debugging efforts by reducing the code scope that needed for inspection [Ghandehari et al. 2012]. However, information from a covering array sometimes is not sufficient to identify the location and number of the failure-inducing interactions [Colbourn and McClary 2008]. Thus, additional information is needed. Consider the following example [Bach and Schroeder 2004], Table I presents a two-way covering array for testing an MS-Word application in which we want to examine various interactions of options for the MS-Word ‘Highlight’, ‘Status Bar’, ‘Bookmarks’ and ‘Smart tags’. Assume the third test case failed. We can get five two-way suspicious interactions that may be responsible for this failure. They are respectively (Highlight: Off, Status Bar: On), (Highlight: Off, Bookmarks: Off), (Highlight: Off, Smart tags: Off), (Status Bar: On, Bookmarks: Off), (Status Bar: On, Smart tags: Off), and (Bookmarks: Off, Smart tags: Off). Without additional information, it is difficult to figure out the specific interactions in this suspicious set caused the failure. In fact, considering that the higher strength interactions could also be failure-inducing interactions, e.g., (Highlight: Off, Status Bar: On, Smart tags: Off), the problem becomes more complicated.

To address this problem, prior work [Nie and Leung 2011a] specifically studied the properties of the minimal failure-causing schemas in SUT, based on which additional test cases were generated to identify the MFS. Other approaches to identify the MFS in SUT include building a tree model [Yilmaz et al. 2006], adaptively generating additional test cases according to the outcome of the last test case [Zhang and Zhang 2011], ranking suspicious interactions based on some rules [Ghandehari et al. 2012], using graphic-based deduction [Martínez et al. 2008], among others.

Most existing approaches mainly focus on the ideal scenario in which SUT only contains one failure, under which the outcomes of test cases can be simply categorized into fail or pass. However, in this paper, we argue that SUT with multiple distinguished failures is the more common testing scenario in practice, and moreover, this affects the effectiveness of Failure-inducing Combinations Identifying(FCI) approaches. One main impact of multiple failures on FCI approaches is the masking effect. A masking effect [Dumlu et al. 2011; Yilmaz et al. 2014] occurs when some failures prevent test cases from checking interactions that are supposed to be tested. Take the Linux command *Grep* for example. We noticed that there are two different failures reported in the bug tracker system. The first [1] claims that Grep incorrectly matches unicode

[1]http://savannah.gnu.org/bugs/?29537

patterns with ‘\<\>’, while the second [2] claims an incompatibility between option ‘-c’ and ‘-o’. When we put these two scenarios into one test case, only one failure will be observed, which means the other failure is masked by the observed failure. This effect will prevent test cases from executing normally, leading to incorrect judgment of the correlation between the interactions checked in the test case and the failure that has been masked. This effect was firstly noted by Dumlu and Yilmaz in [Dumlu et al. 2011], in which they found that the masking effects in CT can prevent traditional covering array fail in testing some interactions.

As masking effect can negatively affect the performance of FCI approaches, a natural question is how this effect biases the results of these approaches. In this paper, we formalize the process of identifying the MFS under the circumstances in which masking effects exist in the SUT and try to answer this question. One insight from the formal analysis is that we cannot completely get away from the impact of masking effects even if we do exhaustive testing. Even worse, either ignoring the masking effects or treating multiple failures as one failure is detrimental to the FCI process.

Based on this concern, we propose a strategy to alleviate this impact by adopting the divide and conquer framework, i.e., separately handles each failure in the SUT. For a particular failure under analysis, when applying traditional FCI approaches to identify the failure-inducing interactions, we pick the test cases generated by FCI approaches that trigger any failure other than the failure under analysis and replace them with some newly regenerated test cases. These new test cases should either pass or trigger the same failure under analysis.

The key to our approach is to search for a test case that does not trigger unexpected failures which may introduce the masking effect. To guide the search process, i.e., to reduce the possibility that the newly generated test case will trigger an unexpected failure, a natural idea is to take some characteristics from the existing test cases and make the characteristics of the newly searched test case as different from the test cases which triggered the unexpected failure as possible. To reach this target, we define the *related strength* between the factor and the failure. The higher the *related strength* between a factor and a particular failure, the greater the likelihood that the factor will trigger this failure. We then use the integer linear programming (ILP) technique to find a test case which has the least *related strength* with the unexpected failure.

To evaluate the effectiveness of our approach, we applied our strategy on the FCI approach FIC_BS [Zhang and Zhang 2011]. The subjects used were two open-source software systems found in the developers’ forum in the Source-Forge community. Through studying their bug reports in the bug tracker system as well as their user’s manuals, we built a testing model which can reproduce the reported bugs with given test cases. We then compared the FCI approach augmented with our strategy to the original FCI approach. We further empirically studied the performance of the important component of our strategy – searching satisfied test cases. To conduct this study, we compared our approach with the augmented FCI approach by randomly searching satisfied test cases. We finally compared our approach with the only existing masking handling technique – FDA-CIT [Yilmaz et al. 2014]. Our studies showed that our replacing strategy as well as the searching test case component achieved a better performance than the traditional approaches when the subject suffered multiple failures, especially when these failures can import masking effects.

The main contributions of this paper are:

— We studied the impact of the masking effects caused by multiple failures on the isolation of the failure-inducing interactions in SUT.

[2] http://savannah.gnu.org/bugs/?33080

```
public float foo(int a, int b, int c, int d){
  //step 1 will cause an exception when b == c
  float x = (float)a / (b - c);

  //step 2 will cause an exception when c < d
  float y = Math.sqrt(c - d);

  return x+y;
  }
```

Fig. 1. A simple program *foo* with four input parameters

Table II. test cases and their corresponding result

id	test case	result	id	test case	result
1	(7, 2, 4, 3)	PASS	13	(11, 2, 4, 3)	PASS
2	(7, 2, 4, 5)	Ex 2	14	(11, 2, 4, 5)	Ex 2
3	(7, 2, 6, 3)	PASS	15	(11, 2, 6, 3)	PASS
4	(7, 2, 6, 5)	PASS	16	(11, 2, 6, 5)	PASS
5	(7, 4, 4, 3)	Ex 1	17	(11, 4, 4, 3)	Ex 1
6	(7, 4, 4, 5)	Ex 1	18	(11, 4, 4, 5)	Ex 1
7	(7, 4, 6, 3)	PASS	19	(11, 4, 6, 3)	PASS
8	(7, 4, 6, 5)	PASS	20	(11, 4, 6, 5)	PASS
9	(7, 5, 4, 3)	PASS	21	(11, 5, 4, 3)	PASS
10	(7, 5, 4, 5)	Ex 2	22	(11, 5, 4, 5)	Ex 2
11	(7, 5, 6, 3)	PASS	23	(11, 5, 6, 3)	PASS
12	(7, 5, 6, 5)	PASS	24	(11, 5, 6, 5)	PASS

— We proposed a divide and conquer strategy for selecting test cases to alleviate the impact of these effects.
— We designed an efficient test case searching method which can find a test case that does not trigger an unexpected failure.
— We conducted several empirical studies and showed that our strategy can assist FCI approaches to achieve better performance in identifying failure-inducing interactions in SUT with masking effects.

2. MOTIVATING EXAMPLE

We constructed a small example to illustrate the motivation of our approach. Assume a method *foo* has four input parameters: *a, b, c,* and *d*. The four parameter types are all integers and their values are: $v_a = \{7, 11\}, v_b = \{2, 4, 5\}, v_c = \{4, 6\}, v_d = \{3, 5\}$. The code of *foo* is shown in Figure 1.

There are two potential failures of *foo*: first, in step 1 we can get an *Arithmetic Exception* when b is equal to c, i.e., $b = 4$ and $c = 4$, that causes a division by zero. Second, another *Arithmetic Exception* will be triggered in step 2 when $c < d$, i.e., $c = 4$ and $d = 5$, taking square root of a negative number. So the expected failure-inducing interactions in this example should be (-, 4, 4, -) and (-, -, 4, 5).

Traditional FCI algorithms do not consider the code detail; instead, they apply black-box testing, i.e., feed inputs to the programs and execute them to observe the result. The basic justification behind those approaches is that the failure-inducing interactions for a particular failure can only appear in those test cases that trigger this failure. Traditional FCI approaches aim at using as few test cases as possible to get the same (or approximate) result as exhaustive testing, so the results derived from an exhaustive testing set are the best that these FCI approaches can achieve. Next, we will show how exhaustive testing works to identify the failure-inducing interactions for the program.

Table III. Identified failure-inducing interactions and their corresponding Exception

Failure-inducing interaction	Exception
(-, 4, 4, -)	Ex 1
(-, 2, 4, 5)	Ex 2
(-, 5, 4, 5)	Ex 2

We first generate every possible test case listed in the column "test case" of Table II. The execution results are listed in the result column of Table II. In this column, *PASS* means that the program runs without any exception. *Ex 1* indicates that the program triggered an exception corresponding to step 1 and *Ex 2* indicates the program triggered an exception corresponding to step 2. From the data listed in Table II, we can determine that (-, 4 , 4, -) must be the failure-inducing interactions of Ex 1 as all the test cases that triggered Ex 1 contain this interactions. Similarly, interactions (-, 2, 4, 5) and (-, 5, 4, 5) must be the failure-inducing interactions of Ex 2. We list these interactions and the corresponding exceptions in Table III.

Note that in this example we did not get the expected result with traditional FCI approaches. The failure-inducing interactions for Ex 2 are (-,2,4,5) and (-,5,4,5), respectively, instead of the expected interaction (-,-,4,5). So why did we fail to get the (-,-,4,5)? The reason lies in *test case 6* (7,4,4,5) and *test case 18* (11,4,4,5). These two test cases contain the interaction (-,-,4,5), but they did not trigger Ex 2; instead, Ex 1 was triggered.

Now consider the source code of *foo*. We can find that if Ex 1 is triggered, it will stop executing the remaining code and report the exception result. In other word, Ex 1 may mask Ex 2. Let us re-examine the interaction (-,-,4,5). If we suppose that *test case 6* and *test case 18* should trigger Ex 2 if they did not trigger Ex 1, then we can conclude that (-,-,4,5) should be the failure-inducing interaction of Ex 2, which is identical to the expected one.

Unless we fix the code that triggers Ex 1 and re-execute all the test cases, we cannot validate the supposition that *test case 6* and *test case 18* should trigger Ex 2 in case they did not trigger Ex 1. So in practice, when we lack resources to execute all the test cases repeatedly or can only do black-box testing, a more economical and efficient approach to alleviate the masking effect on FCI approaches is desired.

3. FORMAL MODEL OF MINIMAL FAILURE-CAUSING SCHEMA

This section presents some definitions and propositions for a formal model to solve the FCI problem.

3.1. Failure-causing Schemas in CT

Assume that the behaviour of SUT is influenced by *k* parameters, and each parameter p_i has a_i discrete values from the finite set V_i, i.e., a_i = $|V_i|$ (i = 1,2,..k). In practice, these parameters can represent many factors, such as input variables, run-time options, building options, or various interactions of them. Next we will give some formal definitions, some of which (Definitions 3.1, 3.3, 3.4) below were originally defined in [Nie and Leung 2011b].

Definition 3.1. A *test case* of SUT is an array of *k* values, one for each parameter of the SUT, which is denoted as a *k*-tuple (v_1, v_2,...,v_k), where $v_1 \in V_1$, $v_2 \in V_2$ ... $v_k \in V_k$.

For example in Section 2, ($a = 7$, $b = 2$, $c = 4$, $d = 3$) is a test case, which is actually a group of values being assigned to each input parameter.

Definition 3.2. A *failure* is the abnormal execution of a test case.

In CT, such a *failure* can be a thrown exception, compilation error, assertion failure or constraint violation. In this paper, we focus on studying the impact of multiple *failures* on failure-inducing interactions identification. To facilitate our discussion, we introduce the following assumptions that will be used throughout this paper:

ASSUMPTION 1. *The execution result of a given test case is deterministic.*

This assumption is the most common assumption of CT fault diagnosis. It indicates that the outcome of executing a test case is reproducible and will not be affected by some unexpected random events.

ASSUMPTION 2. *Different failures in the SUT can be distinguished by various information such as exception traces, state conditions, or the like.*

This assumption indicates that the testers can detect different failures during testing. As different failures will complicate the testing task, distinguishing them is the first step to resolve them.

Now let us consider the condition that some failures are triggered by some test cases. It is then desirable to determine the cause of these failures and hence some parameter values of these failing test cases must be analysed.

Definition 3.3. For the SUT, the *t*-tuple (-,v_{k_1},...,v_{k_t},...) is called a *t*-degree *schema* ($0 < t \leq k$) when some t parameters have fixed values and the others can take on their respective allowable values, represented as "-".

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

For example, (-, 4, 4, -) in Table III is a two-degree schema. And the test case (7, 4 ,4, 3) contains this schema.

Definition 3.4. Let c_l be a *l*-degree schema, c_m be an *m*-degree schema in SUT, and $l < m$. If all the fixed parameter values in c_l are also in c_m, then c_m *subsumes* c_l. In this case, we can also say that c_l is a *sub-schema* of c_m, and c_m is a *super-schema* of c_l, denoted as $c_l \prec c_m$.

For example, in the motivation example, the two-degree schema (-, 4, 4, -) is a sub-schema of the three-degree schema (-, 4, 4, 5), that is, (-,4,4,-) $\prec$ (-,4,4,5).

Definition 3.5. If all test cases contain a schema, say c, and trigger a particular failure, say F, then we call c the *failure-causing schema* for F. Additionally, if none of the sub-schema of c is the *failure-causing schema* for F, we then call c the *Minimal Failure-causing Schema*, i.e., the MFS for F.

In fact, MFS is identical to the failure-inducing interactions discussed previously. Identifying MFS helps to focus on the root cause of a failure and thus facilitate the debugging efforts.

Some notations used later are listed below for convenient reference:

— k : The number of parameters that influence the SUT.
— V_i : The set of discrete values that the ith parameter of SUT can take.
— T^* : The exhaustive set of test cases for the SUT. For a SUT with k parameters, and each parameter can take $|V_i|$ values, the number of test cases in T^* is $\prod_{i=1}^{i<=k} |V_i|$. Note that some test cases may be invalid if there exists constraints among the parameters.
— $A \backslash B$: the set of elements that belong to set A but not to B. For example $T_i \backslash T_j$ indicates the set of test cases that belong to set T_i, but not to T_j.
— L : The number of failures contained in the SUT.

Table IV. Example of Proposition 3.6

c	c
(0, 0, - , -)	(0, -, - , -)
$\mathcal{T}(c)$	$\mathcal{T}(c)$
(0, 0, 0 ,0)	(0, 0, 0 ,0)
(0, 0, 0 ,1)	(0, 0, 0 ,1)
(0, 0, 1 ,0)	(0, 0, 1 ,0)
(0, 0, 1 ,1)	(0, 0, 1 ,1)
	(0, 1, 0 ,0)
	(0, 1, 0 ,1)
	(0, 1, 1 ,0)
	(0, 1, 1 ,1)

— F_m : The mth failure in the SUT; for different failures, we can differentiate them based on their exception traces or other information.
— T_{F_m} : All the test cases that can trigger the failure F_m.
— $\mathcal{T}(c)$: All the test cases that contain the schema c. Based on the definition of MFS, we know that if schema c is an MFS for F_m, then $\mathcal{T}(c)$ must be a subset of T_{F_m}.
— $\mathcal{I}(t)$: All the schemas that are contained in the test case t, e.g., $\mathcal{I}(\ (111)\ ) = \{(1--)(-1-)(--1)(11-)(1-1)(-11)(111)\}$.
— $\mathcal{I}(T)$: All the schemas that are contained in a set of test cases T, i.e., $\mathcal{I}(T) = \bigcup_{t\in T}\mathcal{I}(t)$.
— $\mathcal{S}(T)$: All the schemas that are only contained in the set of test cases T. It is important to note that this set is different from $\mathcal{I}(T)$, as the schemas contained by the test cases in T can also be contained by other test cases that do not belong to this set. In fact, $\mathcal{S}(T)$ is computed by $\{c|c \in \mathcal{I}(T)\ and\ c\ \notin \mathcal{I}(T^*\backslash T\}$.
— $\mathcal{C}(T)$: The minimal schemas that are only contained by the set of test cases T. $\mathcal{C}(T)$ is a sub-set of $\mathcal{S}(T)$, which is defined as $\{c|c \in \mathcal{S}(T)\ and\ \ \nexists c' \prec c, s.t., c' \in \mathcal{S}(T)\}$.

PROPOSITION 3.6 (SUB SCHEMAS HAVE A LARGER SET OF TEST CASES). *For l-degree schema c_l and m-degree schema c_m, if $c_l \prec c_m$, then all the test cases that contain c_m must also contain c_l, i.e., $\mathcal{T}(c_m) \subset \mathcal{T}(c_l)$.*

PROOF. $\forall t \in \mathcal{T}(c_m)$, we have t contains c_m. Then as $c_l \prec c_m$, we must also have t contains c_l. This is because all the elements in c_l are also in c_m, which are contained in the test case t. Therefore, $t \in \mathcal{T}(c_l)$. Thus $t \in \mathcal{T}(c_m)$ implies $t \in \mathcal{T}(c_l)$, so it follows that $\mathcal{T}(c_m) \subset \mathcal{T}(c_l)$. □

Table IV illustrates an example of the SUT with four binary parameters (unless otherwise specified, the following examples also assume a SUT with binary parameters). The left column lists the schema (0,0,-,-) as well as all the test cases that contain this schema, while the right column lists the test cases for schema (0,-,-,-). We can observe that (0,-,-,-) $\prec$ (0,0,-,-), and the set of test cases which contain (0,-,-,-) includes the set of test cases that contain (0,0,-,-).

PROPOSITION 3.7 (ANY SET OF TEST CASES HAS A SET OF MINIMAL SCHEMAS). *For any set T of test cases of a SUT, there is a set of minimal schemas $\mathcal{C}(T)$, such that,*

$$T = \bigcup_{c\in\mathcal{C}(T)} \mathcal{T}(c)$$

PROOF. This is the key proposition of this paper, and we prove this by producing this set of schemas.

We have denoted the exhaustive set of test cases for SUT as T^* and let $T^*\backslash T$ be the test cases that are in T^* but not in T. Obviously $\forall t \in T$, we can always find at least

Table V. Example of the minimal schemas

T	$\mathcal{S}(T)$	$\mathcal{C}(T)$
(0, 0, 0 ,0)	(0, 0, 0, 0)	(0, 0, 0, -)
(0, 0, 0 ,1)	(0, 0, 0, 1)	(0, 0, -, 0)
(0, 0, 1 ,0)	(0, 0, 1, 0)	
	(0, 0, 0, -)	
	(0, 0, -, 0)	

one schema which is contained in t, i.e., we can find $c \in \mathcal{I}(t)$, such that $c \notin \mathcal{I}(T^*\backslash T)$. Specifically, at least the test case t itself as schema holds.

Then we collect all the schemas which are only contained by the test cases in T, which can be denoted as: $\mathcal{S}(T) = \{c|c \in \mathcal{I}(T)\ and\ c \notin \mathcal{I}(T^*\backslash T)\}$.

For the schemas in $\mathcal{S}(T)$, we have $\bigcup_{c\in\mathcal{S}(T)} \mathcal{T}(c) = T$. This is because first, $\forall t \in \mathcal{T}(c), c \in \mathcal{S}(T)$, it must have $t \in T$. This is because if not so, then $t \in T^*\backslash T$, which contradicts with the definition of $\mathcal{S}(T)$. So $t \in T$. Hence, $\bigcup_{c\in\mathcal{S}(T)} \mathcal{T}(c) \subset T$.

Then second, for any test case t in T, as at least one schema c' in $\mathcal{I}(t)$, such that c' in $\mathcal{S}(T)$ (The t itself as a schema holds). In other word, the test case t contains the schema c', which implies $t \in \mathcal{T}(c'), c' \in \mathcal{S}(T)$. And obviously $\mathcal{T}(c') \subset \bigcup_{c\in\mathcal{S}(T)} \mathcal{T}(c)$, so $t \in \bigcup_{c\in\mathcal{S}(T)} \mathcal{T}(c)$, therefore, $T \subset \bigcup_{c\in\mathcal{S}(T)} \mathcal{T}(c)$.

Since $\bigcup_{c\in\mathcal{S}(T)} \mathcal{T}(c) \subset T$ and $T \subset \bigcup_{c\in\mathcal{S}(T)} \mathcal{T}(c)$, so it follows $\bigcup_{c\in\mathcal{S}(T)} \mathcal{T}(c) = T$.

Then we denote the minimal schemas of $\mathcal{S}(T)$ as $\mathcal{C}(T) = \{c|c \in \mathcal{S}(T)\ and\ \not\exists c' \prec c, s.t., c' \in \mathcal{S}(T)\}$. For this set, we can have $\bigcup_{c\in\mathcal{C}(T)} \mathcal{T}(c) = T$. We also prove this by two steps, first and obviously, $\bigcup_{c\in\mathcal{C}(T)} \mathcal{T}(c) \subset \bigcup_{c\in\mathcal{S}(T)} \mathcal{T}(c)$. Then we just need to prove that $\bigcup_{c\in\mathcal{S}(T)} \mathcal{T}(c) \subset \bigcup_{c\in\mathcal{C}(T)} \mathcal{T}(c)$.

In fact by definition of $\mathcal{C}(T)$, $\forall c' \in \mathcal{S}(T)\backslash\mathcal{C}(T)$, we can have some $c \in \mathcal{C}(T)$, such that $c \prec c'$. According to Proposition 3.6, $\mathcal{T}(c') \subset \mathcal{T}(c)$. So for any test case $t \in \bigcup_{c\in\mathcal{S}(T)} \mathcal{T}(c)$, either $\exists c' \in \mathcal{S}(T)\backslash\mathcal{C}(T), s.t., t \in \mathcal{T}(c')$ or $\exists c \in \mathcal{C}(T), s.t., t \in \mathcal{T}(c)$. Both cases can deduce $t \in \bigcup_{c\in\mathcal{C}(T)} \mathcal{T}(c)$. So, $\bigcup_{c\in\mathcal{S}(T)} \mathcal{T}(c) \subset \bigcup_{c\in\mathcal{C}(T)} \mathcal{T}(c)$.

Hence, $\bigcup_{c\in\mathcal{S}(T)} \mathcal{T}(c) = \bigcup_{c\in\mathcal{C}(T)} \mathcal{T}(c)$, and $\mathcal{C}(T)$ is the set of schemas that holds this proposition. □

For example, Table V lists the $\mathcal{S}(T)$ and minimal schemas $\mathcal{C}(T)$ for the set of test cases T. We can see that for any other schema not in $\mathcal{C}(T)$, either we can find a test case not in T that contains the schema, e.g.,(0,0,-,-) with the test case (0,0,1,1) not in T, or that is the super schema of one of the two minimal schemas, e.g.,(0,0,0,0) is the super schema of both (0,0,0,-) and (0,0,-,0).

Let T_{F_m} denotes the set of all the test cases triggering failure F_m , then $\mathcal{C}(T_{F_m})$ actually is the set of MFS of F_m by definition of MFS.

From the construction process of $\mathcal{C}(T)$, one observation is that the minimal schema set $\mathcal{C}(T)$ is the subset of the schema set $\mathcal{S}(T)$, i.e., $\mathcal{C}(T) \subset \mathcal{S}(T)$, and for any schema in $\mathcal{S}(T)$, it either belongs to $\mathcal{C}(T)$, or is the super schema of one element of $\mathcal{C}(T)$. Then, we have the following proposition.

PROPOSITION 3.8 (SCHEMAS WILL BELONG TO $\mathcal{S}(T)$ IF TEST CASES ARE IN T). *For any test set T and schema c, if every test case contains c is in the set T, i.e., $\mathcal{T}(c) \subset T$, then it must be that $c \in \mathcal{S}(T)$.*

PROOF. First $c \in \mathcal{C}(\mathcal{T}(c))$ is obvious and in fact the minimal schemas for the set of test cases $\mathcal{T}(c)$ only contain one schema, which is exactly c itself. As discussed previously $\mathcal{C}(\mathcal{T}(c)) \subset \mathcal{S}(\mathcal{T}(c))$, so it must be $c \in \mathcal{S}(\mathcal{T}(c))$.

Then as $\mathcal{T}(c) \subset T$, it follows that $\mathcal{S}(\mathcal{T}(c)) \subset \mathcal{S}(T)$ by definition. In detail, $\mathcal{S}(\mathcal{T}(c)) = \{c | c \in \mathcal{I}(\mathcal{T}(c))\ and\ c \notin \mathcal{I}(T^* \backslash \mathcal{T}(c))\}$, so $\mathcal{S}(\mathcal{T}(c)) \subset \{c | c \in \mathcal{I}(T)\ and\ c \notin \mathcal{I}(T^* \backslash T)\}$, which is exactly $\mathcal{S}(T)$.

So as $c \in \mathcal{S}(\mathcal{T}(c))$, hence $c \in \mathcal{S}(T)$. □

Based on this proposition, we can find that if two schemas that have their test cases subsuming each other, then they are the relationships of super-schema and subschema.

For two different sets of test cases, there exist some relationships between the minimal schemas of them. The relationship varies with the associations of the two test sets. In fact, there are three possible associations between two different sets of test cases: *inclusion*, *disjointness*, and *intersection*, as shown in Figure 2. We did not show the case when the two sets are identical, because for this case the minimal schemas must also be identical. The relationship of the minimal schemas between two different sets of test cases is important as we will learn later the masking effects between multiple failures will make the MFS identifying process work incorrectly, i.e., the FCI approaches may isolate the minimal schemas for the set of test cases which are different from the expected failing set of test cases. And these properties can help to figure out the impact of masking effects on the FCI approaches. Next, we will discuss the relationship between minimal schemas of two sets of test cases with the three associations.

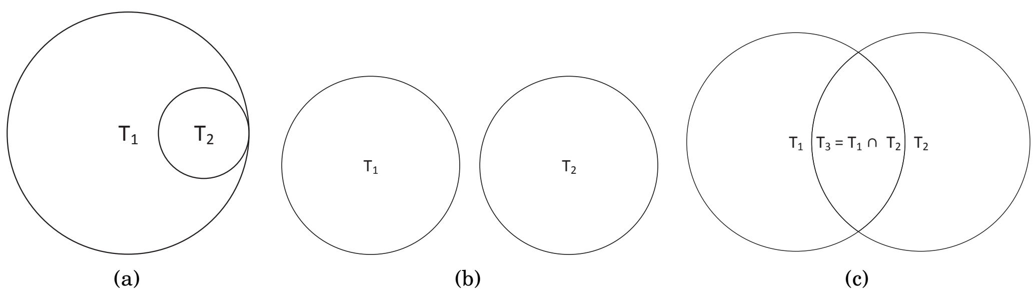


Fig. 2. Relationships between two test sets

3.2. Inclusion

It is the first relationship corresponding to Figure 2(a). We have the following proposition for two test sets that have an inclusion relationship.

PROPOSITION 3.9 (SCHEMAS FOR TEST SUBSET TEND TO BE SUPER SCHEMAS). *For two sets of test cases T_l and T_k, assume that $T_l \subset T_k$. Then $\forall c_l \in \mathcal{C}(T_l)$, $either\ c_l \in \mathcal{C}(T_k)\ or\ \exists c_k \in \mathcal{C}(T_k), s.t., c_k \prec c_l$.*

PROOF. Obviously $\forall c_l \in \mathcal{C}(T_l)$, $\mathcal{T}(c_l) \subset T_l \subset T_k$. According to Proposition 3.8, $c_l \in \mathcal{S}(T_k)$. So this proposition holds as the schema in $\mathcal{S}(T_k)$ either is in $\mathcal{C}(T_k)$, or must be the super-schemas of some schemas in $\mathcal{C}(T_k)$. □

Likewise, we can get the properties of the schemas identified for the larger set of test cases.

PROPOSITION 3.10 (SCHEMAS FOR TEST SUPERSET TEND TO BE SUBSCHEMAS). *For two sets of test cases T_l and T_k, assume that $T_l \subset T_k$. Then $\forall c_k \in \mathcal{C}(T_k)$, it could be* $(1)\ c_k \in \mathcal{C}(T_l)$ *,* $(2)\ \exists c_l \in \mathcal{C}(T_l), s.t., c_k \prec c_l$ *,or* $(3)\ \nexists c_l \in \mathcal{C}(T_l), s.t., c_k \prec c_l\ or\ c_k = c_l,\ or\ c_l \prec c_k$.

Table VI. Inclusion example

T_l	T_k
(0, 0, 0)	(0, 0, 0)
(0, 0, 1)	(0, 0, 1)
(0, 1, 0)	(0, 1, 0)
	(0, 1, 1)
$\mathcal{C}(T_l)$	$\mathcal{C}(T_k)$
(0, 0, -)	(0, -, -)
(0, -, 0)	

T_l	T_k
(0, 0, 0)	(0, 0, 0)
(0, 0, 1)	(0, 0, 1)
(0, 1, 0)	(0, 1, 0)
	(1, 1, 0)
	(1, 1, 1)
$\mathcal{C}(T_l)$	$\mathcal{C}(T_k)$
(0, 0, -)	(0, 0, -)
(0, -, 0)	(0, -, 0)
	(1, 1, -)

Table VII. Disjoint example

T_l	T_k
(0, 0, 0)	(1, 0, 0)
(0, 1, 0)	(1, 0, 1)
	(1, 1, 0)
$\mathcal{C}(T_l)$	$\mathcal{C}(T_k)$
(0, -, 0)	(1, 0, -)
	(1, -, 0)

This proposition is exactly the antithesis of Proposition 3.9. We need to note the third condition, i.e., $\nexists c_l \in \mathcal{C}(T_l), s.t., c_k \prec c_l \ or \ c_k = c_l, \ or \ c_l \prec c_k$. We refer to this condition as c_k is *irrelevant* to $\mathcal{C}(T_l)$. Furthermore, we can also say a schema is *irrelevant* to another schema if these two schemas are neither identical nor subsuming each other.

We illustrate these scenarios in Table VI. There are two parts in this table, with each part showing two sets of test cases: T_l and T_k, which have $T_l \subset T_k$. In the left part, we can see that in the schema in $\mathcal{C}(T_l)$: (0, 0, -) and (0, -, 0), both are the super-schemas of the one in $\mathcal{C}(T_k)$:(0,-,-). While in the right part, the schemas in $\mathcal{C}(T_l)$: (0, 0, -) and (0, -, 0) are both also in $\mathcal{C}(T_k)$. Furthermore, one schema in $\mathcal{C}(T_k)$: (1,1,-) is irrelevant to $\mathcal{C}(T_l)$.

3.3. Disjointness

This relationship corresponds to Figure 2(b). For two different test sets, one obvious property is listed as follows:

PROPOSITION 3.11 (NO RELATIONSHIPS BETWEEN THE SCHEMAS). *For two test sets T_1 and T_2, if $T_1 \bigcap T_2 = \emptyset$, we have, $\mathcal{S}(T_1) \bigcap \mathcal{S}(T_2) = \emptyset$.*

PROOF. Assume that $\mathcal{S}(T_1) \bigcap \mathcal{S}(T_2) \neq \emptyset$. Without loss of generality, let $c \in \mathcal{S}(T_1) \bigcap \mathcal{S}(T_2)$, we can learn that $\mathcal{T}(c)$ must both in T_1 and T_2, which is contradiction. $\square$

This property points out that the minimal schemas of two disjoint test cases should be irrelevant to each other. Table VII shows an example of this scenario. We can learn from this table that for two different test sets T_l and T_k, their minimal schemas, i.e., (0, - , 0) and (1, 0, -),(1, -, 0), respectively, are irrelevant to each other.

3.4. Intersection

This relationship corresponds to Figure 2(c), which is the most common scenario for two given sets of test cases, but is also the most complicated scenario for analysis. To conveniently illustrate the properties of the minimal schemas for this scenario, let $T_1 \bigcap T_2 = T_3$ as depicted in Figure 2(c). Then, we have the following properties:

Let us see that $C(T_1)$ is sub schema or identical or irrelevant to $C(T_3)$. C_{T_2} is also the same. Then we have the following conditions, if c_1 sub to c_3, c_2 sub to, c_1 identical to, c_2 identical. c_1 irrelevant, c_2 irrelevant.

Table VIII. Disjoint example

c_1c_3	c_2, c_3
sub	sub
sub	identical
identical	sub
identical	identical
-	irrelevant
irrelevant	-

PROPOSITION 3.12 (TWO SUB). *For $c_1 \in \mathcal{C}(T_1)$, and $c_2 \in \mathcal{C}(T_2)$, if there exist a schema $c_3 \in \mathcal{C}(T_3), s.t.c_1 \prec c_3, andc_2 \prec c_3$. Then it must have c_1 irrelevant to $C(T_2)$ and c_2 irrelevant to $C(T_1)$.*

PROOF. As we see that identical schemas must have the same set of test cases that contain them, then the only same set of test cases between T_1 and T_2 is $T_1 \bigcap T_2 = T_3$. So the only possible identical schema between $\mathcal{C}(T_1)$ and $\mathcal{C}(T_2)$ is in $\mathcal{C}(T_3)$. □

PROPOSITION 3.13 (ONE SUB, ONE IDENTICAL). *For $c_1 \in \mathcal{C}(T_1)$, and $c_2 \in \mathcal{C}(T_2)$, if there exist a schema $c_3 \in \mathcal{C}(T_3), s.t.c_1 \prec c_3, andc_2 = c_3$. Then it must have c_1 is sub of the schema of $C(T_2)$, which is c_2. And c_2 is the super schema of some schemas of $C(T_1)$, which is c_1.*

PROPOSITION 3.14 (ONE IDENTICAL, ONE SUB). *For $c_1 \in \mathcal{C}(T_1)$, and $c_2 \in \mathcal{C}(T_2)$, if there exist a schema $c_3 \in \mathcal{C}(T_3), s.t.c_1 \prec c_3, andc_2 = c_3$. Then it must have c_1 is sub of the schema of $C(T_2)$, which is c_2. And c_2 is the super schema of some schemas of $C(T_1)$, which is c_1.*

PROPOSITION 3.15 (TWO IDENTICAL). *For $c_1 \in \mathcal{C}(T_1)$, and $c_2 \in \mathcal{C}(T_2)$, if there exist a schema $c_3 \in \mathcal{C}(T_3), s.t.c_1 \prec c_3, andc_2 = c_3$. Then it must have c_1 is sub of the schema of $C(T_2)$, which is c_2. And c_2 is the super schema of some schemas of $C(T_1)$, which is c_1.*

PROPOSITION 3.16 (ONE IRRELEVANT). *For $c_1 \in \mathcal{C}(T_1)$, and $c_2 \in \mathcal{C}(T_2)$, if there exist a schema $c_3 \in \mathcal{C}(T_3), s.t.c_1 \prec c_3, andc_2 = c_3$. Then it must have c_1 is sub of the schema of $C(T_2)$, which is c_2. And c_2 is the super schema of some schemas of $C(T_1)$, which is c_1.*

PROPOSITION 3.17 (CAN BE IDENTICAL). *For two intersecting sets of test cases T_1 and T_2, and let $T_3 = T_1 \bigcap T_2$, if $\exists c_1 \in \mathcal{C}(T_1)$ and $c_2 \in \mathcal{C}(T_2)$ which are identical, then $c_1 = c_2 \in \mathcal{C}(T_3)$*

PROOF. As we see that identical schemas must have the same set of test cases that contain them, then the only same set of test cases between T_1 and T_2 is $T_1 \bigcap T_2 = T_3$. So the only possible identical schema between $\mathcal{C}(T_1)$ and $\mathcal{C}(T_2)$ is in $\mathcal{C}(T_3)$. □

Note that this proposition holds when some schemas in $\mathcal{C}(T_1 \bigcap T_2)$ are identical to some schemas in $\mathcal{C}(T_1)$ and $\mathcal{C}(T_2)$.

For example, Table IX shows two test cases that interact with each other at test cases (1,1,0) and (1,1,1), and they have identical minimal schema, i.e., (1,1,-), which is also the minimal schema in $\mathcal{C}(T_3)$.

PROPOSITION 3.18 (CAN BE SUBSUMING EACH OTHER). *For two intersecting sets of test cases T_1 and T_2, let $T_3 = T_1 \bigcap T_2$, if $\exists c_1 \in \mathcal{C}(T_1)$ and $c_2 \in \mathcal{C}(T_2)$, and if c_1 is the super-schema of c_2, then $c_1 \in \mathcal{C}(T_3)$. (and vice versa).*

PROOF. We have proved previously if two schemas have a subsuming relationship, then their test cases must also have an inclusion relationship. And as the only inclusion relationship between T_1 and T_2 is that $T_3 \subset T_1$ and $T_3 \subset T_2$, the super schemas must be in $\mathcal{C}(T_3)$. □

Table IX. Example of Intersection by identical examples

T_1	T_2	$T_3 = T_1 \bigcap T_2$
(0, 1, 0)	(0, 0, 0)	(1, 1, 0)
(1, 1, 0)	(0, 0, 1)	(1, 1, 1)
(1, 1, 1)	(1, 1, 0)	
	(1, 1, 1)	
$\mathcal{C}(T_1)$	$\mathcal{C}(T_2)$	$\mathcal{C}(T_3)$
(-, 1, 0)	(0, 0, -)	(1, 1, -)
(1, 1, -)	(1, 1, -)	

Table X. Example of Intersect by subsuming examples

T_1	T_2	$T_3 = T_1 \bigcap T_2$
(0, 1, 0)	(0, 0, 0)	(1, 0, 0)
(1, 0, 0)	(1, 0, 0)	(1, 0, 1)
(1, 0, 1)	(1, 0, 1)	(1, 1, 0)
(1, 1, 0)	(1, 1, 0)	
	(1, 1, 1)	
$\mathcal{C}(T_1)$	$\mathcal{C}(T_2)$	$\mathcal{C}(T_3)$
(-, 1, 0)	(-, 0, 0)	(1, 0, -)
(1, 0, -)	(1, -, -)	(1, -, 0)
(1, -, 0)		

Table XI. Example of Intersection by irrelevant examples

T_1	T_2
(1, 0, 0)	(1, 0, 0)
(1, 0, 1)	(1, 1, 0)
$\mathcal{C}(T_1)$	$\mathcal{C}(T_2)$
(1, 0, -)	(1, -, 0)

It is noted that this proposition holds when some schemas in $\mathcal{C}(T_3)$ are also in $\mathcal{C}(T_1)$ (or $\mathcal{C}(T_2)$), and simultaneously the same schemas in $\mathcal{C}(T_3)$ must be the super-schema of the minimal schemas of another set of test cases, i.e., $\mathcal{C}(T_2)$ (or $\mathcal{C}(T_1)$).

Table X illustrates this scenario, in which, the minimal schemas of T_1: (1,0,-),(1,-,0) are also the schemas in $\mathcal{C}(T_3)$, and are the super schema of the minimal schema of T_2: (1,-,-).

Besides these two conditions, there can be that the schemas that are irrelevant to the other set. Formally.

PROPOSITION 3.19 (CAN BE IRRELEVANT TO EACH OTHER). *For two intersecting sets of test cases T_1 and T_2, let $T_3 = T_1 \bigcap T_2$, if $\exists c_1 \in \mathcal{C}(T_1)$ and $c_2 \in \mathcal{C}(T_2)$, and if c_1 is the super-schema of c_2, then $c_1 \in \mathcal{C}(T_3)$. (and vice versa).*

PROOF. We have proved previously if two schemas have a subsuming relationship, then their test cases must also have an inclusion relationship. And as the only inclusion relationship between T_1 and T_2 is that $T_3 \subset T_1$ and $T_3 \subset T_2$, the super schemas must be in $\mathcal{C}(T_3)$. □

It is noted that this proposition holds when some schemas in $\mathcal{C}(T_3)$ are also in $\mathcal{C}(T_1)$ (or $\mathcal{C}(T_2)$), and simultaneously the same schemas in $\mathcal{C}(T_3)$ must be the super-schema of the minimal schemas of another set of test cases, i.e., $\mathcal{C}(T_2)$ (or $\mathcal{C}(T_1)$).

For example, Table XI shows two test sets that interact with each other at test case (1,0,0), but their minimal schemas, (1,0,-) and (1,-,0), respectively, are irrelevant to each other.

It is noted that these three conditions can simultaneously appear when two sets of test cases intersect with each other.

3.5. Identify the MFS

According to Definition 3.5 and Proposition 3.7, we can determine that $\mathcal{C}(T_{F_m})$ actually is the set of failure-causing schemas of F_m. Then in theory, if we want to accurately figure out the MFS in the SUT, we need to exhaustively execute each possible test case, and collect the failing test cases T_{F_m}. This is impossible in practice, especially when the testing space is very large.

Traditional FCI approaches select a subset of the exhaustive test cases to execute. In this case, in order to identify the MFS, each of the remaining test cases must be predicted to be failing or not. Or alternatively, a set of schemas is given as the candidate of the MFS [Ghandehari et al. 2012]. For this case, these schemas should be ranked according to their likelihood to be the MFS. As giving a ranking of these candidate schemas can also be regard as a special case of making a prediction (with computing the probability), so we next only formally describe the mechanism of FCI approaches for the first case.

We refer to the observed failing test case as $T_{fail_{observed}}$, and the remaining failing test cases based on prediction as $T_{fail_{predicted}}$. We also denote the actual entire failing test cases as T_{fail}. Then the MFS identified by FCI approaches can be depicted as:

MFS = $\mathcal{C}(T_{fail_{observed}} \bigcup T_{fail_{predicted}})$.

Each FCI approach applies different way to predict $T_{fail_{predicted}}$ according to observed failing test cases; furthermore, as test cases generated are different, the failing test cases observed by different FCI approaches, i.e., $T_{fail_{observed}}$ also vary.

We offer an example using the OFOT approach [Nie and Leung 2011a] to identify the MFS. Suppose that the SUT has 3 parameters, each of which can take on 2 values, and assume the test case (1, 1, 1) failed. Then, the FCI process using OFOT approach can be illustrated in Table XII. In this table, test case *t* failed, and OFOT mutated one parameter value of test case *t* at a time to generate new test cases: $t_1; t_2; t_3$. The pass of t_1 indicates that this test case breaks the MFS in the original test case *t*. So, (1,-,-) should be one failure-causing factor, and as the other test cases (t_2, t_3) all failed , this means no other failure-inducing factors were broken; therefore, the MFS in *t* is (1,-,-).

Now let us explain this process with our formal model. Obviously $T_{fail_{observed}}$ is $\{$(1,1,1),(1,0,1),(1,1,0)$\}$. And as (0,-,-) broke the MFS, by theory [Nie and Leung 2011a], all the test cases that contain (0,-,-) should pass the testing (This conclusion is built on the assumption that the SUT just contains one failure-causing schema). As a result, (0,1,1),(0,0,1),(0,1,0),(0,0,0) should pass the testing. Further, as obviously the test case either passes or fails (the condition that a test case skips testing, i.e., does not produce an output, is labeled as a special case of failing), so the remaining test case (1,0,0), will be predicted to be failing, i.e., $T_{fail_{predicted}}$ is $\{$(1,0,0)$\}$. Finally, the MFS from the OFOT strategy can be described as: $\mathcal{C}(T_{fail_{observed}} \bigcup T_{fail_{predicted}})$ = $\mathcal{C}(\{(1,1,1),(1,0,1),(1,1,0),(1,0,0)\})$ = (1,-,-), which is identical to the one obtained previously.

Similarly, other FCI approaches can also be modeled using this formal description. It is noted that the test cases FCI predicts to be failing are not always identical to the actually failing test cases. In fact, a more general scenario for FCI approaches can be depicted as shown in Figure 3.

In Figure 3 area A denotes the test cases that should have passed testing but were predicted to be failing, area B depicts the test cases that the approach observed to be failing test cases, area C refers to the failing test cases that were not observed but were predicted to be failing test cases, and area D shows the failing test cases that

Table XII. OFOT with our strategy

original test case				Outcome
t	1	1	1	Fail
observed				
t_1	0	1	1	Pass
t_2	1	0	1	Fail
t_3	1	1	0	Fail
predicted				
t_4	0	0	1	Pass
t_5	0	1	0	Pass
t_6	1	0	0	Fail
t_7	0	0	0	Pass

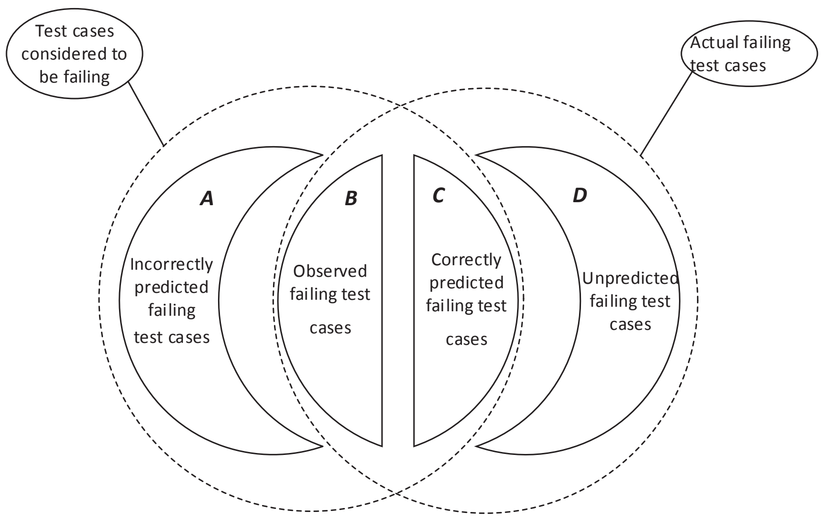


Fig. 3. A general model of FCI

are neither observed nor predicted. This figure actually represents the condition in which two sets of test cases intersect with each other; specifically, comparing to the Figure 2(c), we can learn that $A \bigcup B \bigcup C$= T_1, which indicates the test cases we think they are failing test cases. $D \bigcup B \bigcup C$= T_2 are the actual failing test cases, and $B \bigcup C$ = $T_1 \bigcap T_2 = T_3$ are their Intersections.

As we have known, in the case of Figure 2(c), there are various relationships between the schemas identified in T_1 and the schemas identified in T_2. According to Proposition **??**, some schemas in $\mathcal{C}(T_1)$ and in $\mathcal{C}(T_2)$ must be irrelevant to each other. Mapping to Figure 3, we can get that some schemas in $\mathcal{C}(A \bigcup B \bigcup C)$ and in $\mathcal{C}(D \bigcup B \bigcup C)$ must be irrelevant to each other, which means that the FCI approach will identify some minimal schemas that are irrelevant to the actual MFS, and must ignore some actual MFS. Moreover, under the appropriate conditions listed in Propositions 3.17 and 3.18, FCI may identify the identical schemas or super-schema or sub-schema of the actual MFS. In this case, the schemas (identical ones, super-schemas, or sub-schemas) are all depended on the area $B \bigcup C$, namely $T_1 \bigcap T_2$ in Figure 2(c). To identify the schemas as accurately as possible, the FCI approach needs to make $A \bigcup B \bigcup C$ as similar as possible to $D \bigcup B \bigcup C$.

However, even though each FCI approach tries to identify the MFS as accurately as possible, masking effects arising from different test cases will reduce its effectiveness. We next discuss the masking problem and how it affects the FCI approaches.

Table XIII. masking effects for exhaustive testing

T_1	$T_{mask(1)}$	T_*
(1, 1, 1, 1)	(1, 1, 0, 0)	(0, 1, 0, 0)
(1, 1, 1, 0)	(0, 1, 1, 1)	(0, 0, 0, 0)
(1, 1, 0, 1)		(1, 0, 0, 0)
		(1, 0, 1, 1)
		(0, 0, 1, 1)
actual MFS for 1	regarded as one failure	distinguishing failures
$\mathcal{C}(T_1 \bigcup T_{mask(1)})$	$\mathcal{C}(T_1 \bigcup T_{mask(1)} \bigcup T_*)$	$\mathcal{C}(T_1)$
(1, 1, -, -)	(-, -, 0, 0)	(1, 1, -, 1)
(-, 1, 1, 1)	(1, 1, -, -)	(1, 1, 1, -)
	(-, -, 1, 1)	

4. MASKING EFFECT

Definition 4.1. A *masking effect* occurs when a test case t contains an MFS for a particular failure, but it does not trigger the expected failure because another failure was triggered ahead of it that prevents t from being normally checked.

Taking the masking effects into account, when identifying the MFS for a specific failure, say, F_m, we should not ignore those test cases which did not trigger F_m but should have triggered it. We call these test cases $T_{mask(F_m)}$. Hence, the MFS for failure F_m should be $\mathcal{C}(T_{F_m} \bigcup T_{mask(F_m)})$.

As an example, in the motivation example in section 2, $F_{mask(F_{Ex\ 2})}$ is $\{$ (7,4,4,5),(11,4,4,5)$\}$, So the MFS for $Ex2$ is $\mathcal{C}(T_{F_{Ex\ 2}} \bigcup T_{mask(F_{Ex\ 2})})$, which is (-,-,4,5).

In practice with masking effects, however, it is not possible to correctly identifying the MFS, unless we fix some bugs in the SUT and re-execute the test cases to figure out $T_{mask(F_m)}$.

For traditional FCI approaches, without the knowledge of $T_{mask(F_m)}$, two common strategies can be adopted when facing the multiple failures problem, i.e., *regarded as one failure* and *distinguishing failures*. The former strategy treats all types of failures as one failure–*failure*, and others as *pass*, while the latter distinguishes the failures but with no special consideration of the masking effects, i.e., if a test case fails with a particular type of fault, this strategy presumes it does not contain other type of faults. We will separately discuss the two strategies under exhaustive testing condition and normal FCI testing condition.

4.1. Masking effects under exhaustive testing

4.1.1. Regarded as one failure strategy. This is the most common strategy. With this s-trategy, the minimal schemas are the set $\mathcal{C}(\bigcup_{i=1}^{L} T_{F_i})$, L is the number of all the failures in the SUT. Obviously, $T_{F_m} \bigcup T_{mask(F_m)} \subset \bigcup_{i=1}^{L} T_{F_i}$. So in this case, by Proposition 3.10, some schemas obtained may be the sub-schemas of some of the actual MFS, or be irrelevant to the actual MFS.

As an example, consider the test cases in Table XIII. Assume we need to characterize the MFS for error *1*. All the test cases that triggered error 1 are listed in column T_1; similarly, we list the test cases that triggered other failures in column $T_{mask(1)}$ and T_*, respectively, in which the former masked error 1, while the latter did not. Actually the MFS for error 1 should be (1,1,-,-) and (-,1,1,1) as we listed them in the column *actual MFS for 1*. However, when we use the *regarded as one failure* strategy, the minimal schemas obtained will be (-,-,0,0), (1,1,-,-), (-,-,1,1), in which (-,-,0,0) is irrelevant to the actual MFS for error 1, and (-,-,1,1) is a sub-schema of the actual MFS (-,1,1,1).

4.1.2. Distinguishing failures strategy. Distinguishing the failures by the exception traces or error code can help identify the MFS related to particular failure. Yilmaz [Yilmaz

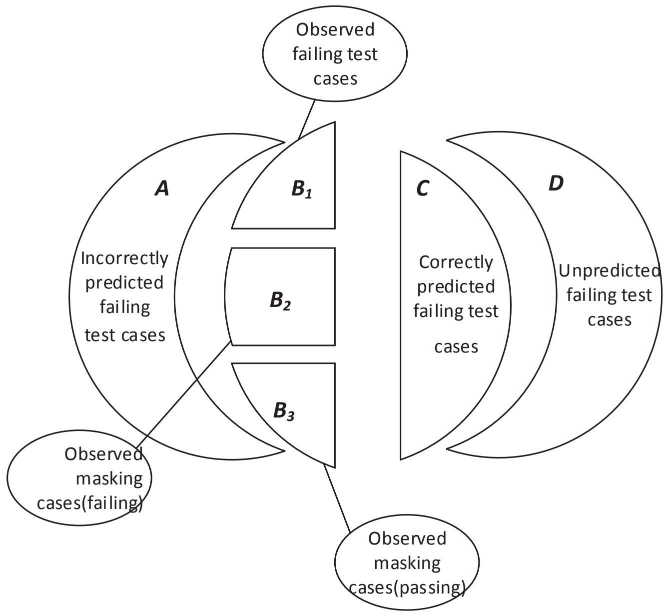


Fig. 4. FCI with masking effects

et al. 2014] proposed the *multiple-class* failure characterizing method instead of the *ternary-class* approach to make the characterizing process more accurate. Besides, other approaches can also be easily extended with this strategy for testing SUT with multiple failures.

This strategy focuses on identifying the set of $\mathcal{C}(T_{F_m})$. As $T_{F_m} \bigcup T_{mask(F_m)} \supset T_{F_m}$, some schemas obtained by this strategy may be the super-schema of some actual MFS. Moreover, some MFS may be irrelevant to the schemas obtained by this strategy, which means this strategy will ignore these actual MFS.

For the simple example shown in Table XIII, when using this strategy, we will get the minimal schemas (1, 1, -, 1) and (1, 1, 1, -), which are both super schemas of the actual MFS (1,1,-,-). Furthermore, no schemas obtained by this strategy have any relationship with the actual MFS (-,1,1,1), which means it was ignored.

It is noted that the motivation example in section 2 actually adopted this strategy. As a result, the schemas identified for Ex 2: (-,2,4,5), (-,3,4,5) are the super-schemas of the correct MFS(-,-,4,5).

4.2. Masking effects under FCI approaches

With masking effects, the scenario of traditional FCI approaches is a bit more complicated than the previous two exhaustive testing scenarios, and is depicted in Figure 4. In this figure, areas A, C and D are the same as those in Figure 3, and area B is further divided into three sub-areas in which B_1 still represents the observed failing test cases for the current analysed failure, area B_2 represents the test cases that triggered other failures which masked the current failure, and area B_3 represents the test cases that triggered other failures which did not mask the current failure. It can be found that the actual MFS set for the SUT is $\mathcal{C}(B_1 \bigcup B_2 \bigcup C \bigcup D)$.

With this model, if we know which test cases mask the expected failure, i.e., if we have figured out B_2 and B_3, then the schemas that the FCI approach will identify can be described as $\mathcal{C}(A \bigcup B_1 \bigcup B_2 \bigcup C)$. We next denote the condition that we have known the masking effects in prior as *knowing masking effects*. However, as discussed before, to get this result is not possible without human involvement. Correspondingly, when using the *regarded as one failure* strategy, the set of MFS traditional FCI identify is $\mathcal{C}(A \bigcup B_1 \bigcup B_2 \bigcup B_3 \bigcup C)$. And for the *distinguishing failures* strategy, the MFS is $\mathcal{C}(A \bigcup B_1 \bigcup C)$. Next, we will discuss the influence of masking effects on the two strategies.

4.2.1. Using the regarded as one failure strategy. For the strategy *regarded as one failure*, the impacts of masking effects on FCI approaches can be analysed by comparing the MFS identified by this strategy to those by strategy *knowing masking effects*. Formally, it is to compare $\mathcal{C}(A \bigcup B_1 \bigcup B_2 \bigcup B_3 \bigcup C)$ (for *regarded as one failure*) with $\mathcal{C}(A \bigcup B_1 \bigcup B_2 \bigcup C)$ (for *knowing masking effects*). Intuitively, $\mathcal{C}(A \bigcup B_1 \bigcup B_2 \bigcup C)$ is more close to the actual MFS, i.e., $\mathcal{C}(B_1 \bigcup B_2 \bigcup C \bigcup D)$. In other word, when using *regarded as one failure* strategy, the deviation between the identified MFS and the actual MFS is larger than that of strategy *knowing masking effects*.

The gap between the two strategies can be further analysed by studying the relationships between these two MFS set. As $A \bigcup B_1 \bigcup B_2 \bigcup C \subset A \bigcup B_1 \bigcup B_2 \bigcup B_3 \bigcup C$, according to Proposition 3.10, schemas in $\mathcal{C}(A \bigcup B_1 \bigcup B_2 \bigcup B_3 \bigcup C)$ (for *regarded as one failure*) can be identical to some schemas in $\mathcal{C}(A \bigcup B_1 \bigcup B_2 \bigcup C)$ (for *knowing masking effects*), or be sub schemas of some of them, or irrelevant to all of them. Formally, $\forall c_{ro} \in \mathcal{C}(A \bigcup B_1 \bigcup B_2 \bigcup B_3 \bigcup C),\ \exists c_{km} \in \mathcal{C}(A \bigcup B_1 \bigcup B_2 \bigcup C),\ s.t.,\ c_{km} = c_{ro},\ or\ c_{ro} \prec c_{km},\ or\ c_{ro}\ irrelevant\ to\ \mathcal{C}(A \bigcup B_1 \bigcup B_2 \bigcup C)$, three possibilities in total.

Now let us get back to the relationships between the schemas obtained by *knowing masking effects* strategy and the actual MFS. It can be found that test cases that are used for identifying the MFS using this strategy, i.e., $A \bigcup B_1 \bigcup B_2 \bigcup C$, *intersect* the test cases that are used to compute the actual MFS($B_1 \bigcup B_2 \bigcup C \bigcup D$). It means that the minimal schemas obtained by this strategy can be identical, super-schema, sub-schema, and irrelevant to the actual MFS, four possibilities in total. Formally, $\forall c_{km} \in \mathcal{C}(A \bigcup B_1 \bigcup B_2 \bigcup C),\ \exists c_{as} \in \mathcal{C}(B_1 \bigcup B_2 \bigcup C \bigcup D),\ s.t.,\ c_{as} = c_{km},\ or\ c_{km} \prec c_{as},\ or\ c_{as} \prec c_{km}\ or\ c_{km}\ irrelevant\ to\ \mathcal{C}(B_1 \bigcup B_2 \bigcup C \bigcup D)$. Combining the previous three possibilities, there are 9 different conditions for the relationships between schemas obtained by strategy *regarded as one failure*, and the composite condition of *knowing masking effects* and the actual MFS. We list them in Table XIV.

In Table XIV, each condition is explained as following: A special case is the 9th condition in this table, as under this condition schema c_{ro} is irrelevant to all the schemas that are obtained by strategy *knowing masking effects*, so special case, the relationship makes nonsense. which are labeled as '—'.

Under each condition, what we cared is what the relationship bet c_{ro} with the actual MFS. As this can give us a clear sight of the c_{ro} when compared with corresponding c_{km}. We list the result in Table XV. Each row tells us a . All of these results can be proved by previous given propositions, which will be ignored in this paper. We just give an example of these conditions and their result.

To specifically discuss all these 9 conditions is redundant in this paper. Hence, we just select two particular conditions, and others can be analysed in a similar way. First, consider the condition that $\exists c_{ro} \in C() and () \prec c_{km}, \prec c_{km} \prec c_{as}$.

We can get the following propositions

In a simpler way, we can label this proposition as a rule $exists c'_{as} \in C(), s.t.,$.

Table gives an example of this condition and the result.

In this table.

Other 10 conditions and their results are list. We will not detail discuss them, as their proofs are in a similar way as proposition.

Table XIV. Possible conditions for Regard as one failure strategy

ID	c_{ro}[1]& c_{km} [2]	c_{km}[2]& c_{as}[3]
1	$\exists c_{km},\ s.t.,\ c_{km} = c_{ro}$	$\exists c_{as},\ s.t.,\ c_{as} = c_{km}$
2	$\exists c_{km},\ s.t.,\ c_{km} = c_{ro}$	$\exists c_{as},\ s.t.,\ c_{as} \prec c_{km}$
3	$\exists c_{km},\ s.t.,\ c_{km} = c_{ro}$	$\exists c_{as},\ s.t.,\ c_{km} \prec c_{as}$
4	$\exists c_{km},\ s.t.,\ c_{km} = c_{ro}$	$c_{km}\ irrelevant\ to\ \mathcal{C}(B_1 \bigcup B_2 \bigcup C \bigcup D)$
5	$\exists c_{km},\ s.t.,\ c_{ro} \prec c_{km}$	$\exists c_{as},\ s.t.,\ c_{as} = c_{km}$
6	$\exists c_{km},\ s.t.,\ c_{ro} \prec c_{km}$	$\exists c_{as},\ s.t.,\ c_{as} \prec c_{km}$
7	$\exists c_{km},\ s.t.,\ c_{ro} \prec c_{km}$	$\exists c_{as},\ s.t.,\ c_{km} \prec c_{as}$
8	$\exists c_{km},\ s.t.,\ c_{ro} \prec c_{km}$	$c_{km}\ irrelevant\ to\ \mathcal{C}(B_1 \bigcup B_2 \bigcup C \bigcup D)$
9	$c_{ro}\ irrelevant\ to\ \mathcal{C}(A \bigcup B_1 \bigcup B_2 \bigcup C)$	—
10		$\exists c_{as},\ s.t.,\ c_{as}\ irrelevant\ to\ \mathcal{C}(A \bigcup B_1 \bigcup B_2 \bigcup C)$

[1] $c_{ro} \in \mathcal{C}(A \bigcup B_1 \bigcup B_2 \bigcup B_3 \bigcup C)$, obtained by strategy *regarded as one failure*
[2] $c_{km} \in \mathcal{C}(A \bigcup B_1 \bigcup B_2 \bigcup C)$, obtained by strategy *knowing masking effects*
[3] $c_{as} \in \mathcal{C}(B_1 \bigcup B_2 \bigcup C \bigcup D)$, actual MFS

Table XV. Possible conditions for Regard as one failure strategy

ID	c_{ro}[1]& c_{km} [2]	c_{km}[2]& c_{as}[3]
1	$\exists c_{km},\ s.t.,\ c_{km} = c_{ro}$	$\exists c_{as},\ s.t.,\ c_{as} = c_{km}$
2	$\exists c_{km},\ s.t.,\ c_{km} = c_{ro}$	$\exists c_{as},\ s.t.,\ c_{as} \prec c_{km}$
3	$\exists c_{km},\ s.t.,\ c_{km} = c_{ro}$	$\exists c_{as},\ s.t.,\ c_{km} \prec c_{as}$
4	$\exists c_{km},\ s.t.,\ c_{km} = c_{ro}$	$c_{km}\ irrelevant\ to\ \mathcal{C}(B_1 \bigcup B_2 \bigcup C \bigcup D)$
5	$\exists c_{km},\ s.t.,\ c_{ro} \prec c_{ro}$	$\exists c_{as},\ s.t.,\ c_{as} = c_{km}$
6	$\exists c_{km},\ s.t.,\ c_{ro} \prec c_{ro}$	$\exists c_{as},\ s.t.,\ c_{as} \prec c_{km}$
7	$\exists c_{km},\ s.t.,\ c_{ro} \prec c_{ro}$	$\exists c_{as},\ s.t.,\ c_{km} \prec c_{as}$
8	$\exists c_{km},\ s.t.,\ c_{ro} \prec c_{ro}$	$c_{km}\ irrelevant\ to\ \mathcal{C}(B_1 \bigcup B_2 \bigcup C \bigcup D)$
9	$c_{ro}\ irrelevant\ to\ \mathcal{C}(A \bigcup B_1 \bigcup B_2 \bigcup C)$	—
10	—	$c_{as}\ irrelevant\ to\ \mathcal{C}(A \bigcup B_1 \bigcup B_2 \bigcup B_3 \bigcup C)$

[1] $c_{ro} \in \mathcal{C}(A \bigcup B_1 \bigcup B_2 \bigcup B_3 \bigcup C)$, obtained by strategy *regarded as one failure*
[2] $c_{km} \in \mathcal{C}(A \bigcup B_1 \bigcup B_2 \bigcup C)$, obtained by strategy *knowing masking effects*
[3] $c_{as} \in \mathcal{C}(B_1 \bigcup B_2 \bigcup C \bigcup D)$, actual MFS

From this 12 rules, we can learn that, when applying *regarded as one failure*, the results can be possible to sub schemas. parent. (). In these rules, sub schemas and irrelevant are more apparently than the super-schemas. One observation in this rules is that , the schemas are tend to be . (rules 1, 2, 5 tells that).

, And if we apply the *regarded as one failure* strategy, the minimal schemas are $\mathcal{C}(A \bigcup B_1 \bigcup B_2 \bigcup B_3 \bigcup C)$. Obviously, $A \bigcup B_1 \bigcup B_2 \bigcup B_3 \bigcup C \supset A \bigcup B_1 \bigcup B_2 \bigcup C$. So the minimal schema obtained by this strategy is either the sub-schema or identical to some schemas from the ones obtained by *knowing masking effects* strategy, or alternatively, there exist some schemas that are irrelevant to all of them. Taking these two properties together, we derive Table XVI

Table XVI. Masking effects influence on FCI with regarded as one failure strategy

1	If $c_m = c_{origin}$ and $c_{new} = c_{origin}$	Then, $\exists c'_m, s.t., c_{new} = c'_m$
2	If $c_m = c_{origin}$ and $c_{new} \prec c_{origin}$	Then, $\exists c'_m, s.t., c_{new} \prec c'_m$
3	If $c_m \prec c_{origin}$ and $c_{new} = c_{origin}$	Then, $\exists c'_m, s.t., c'_m \prec c_{new}$
4a	If $c_m \prec c_{origin}$ and $c_{new} \prec c_{origin}$	Then either,$c'_m, s.t., c_{new} \prec c'_m$
4b		Or,$c_{new}\ irrelevant\ to\ all\ c'_m$
5	If $c_{origin} \prec c_m$ and $c_{new} = c_{origin}$	Then, $\exists c'_m, s.t., c_{new} \prec c'_m$
6	If $c_{origin} \prec c_m$ and $c_{new} \prec c_{origin}$	Then, $\exists c'_m, s.t., c_{new} \prec c'_m$
7	If $c_{origin}\ irrelevant\ all\ c_m$ and $c_{new} = c_{origin}$	Then, $c_{new}\ irrelevant\ to\ all\ c'_m$
8a	If $c_{origin}\ irrelevant\ all\ c_m$ and $c_{new} \prec c_{origin}$	Then either,$\exists c'_m, s.t., c_{new} \prec c'_m$
8b		Or,$c_{new}\ irrelevant\ to\ all\ c'_m$
9a	If $c_{new}\ irrelevant\ to\ all\ c_{origin}$	Then either,$\exists c'_m, s.t., c_{new} \prec c'_m$
9b		Or,$c_{new}\ irrelevant\ to\ all\ c'_m$

There are totally 9 rules for this strategy, which are labeled from 1 to 9 respectively. In these rules, c_m and c'_m are the actual MFS, i.e., $c_m, c'_m \in C(B_1 \bigcup B_2 \bigcup C \bigcup D)$. c_{origin} is the minimal schema that are obtained by *knowing masking effects* strategy, i.e., $c_{origin} \in \mathcal{C}(A \bigcup B_1 \bigcup B_2 \bigcup C)$. c_{new} is the schema that are obtained by *regarded as one failure* strategy, i.e., $c_{new} \in \mathcal{C}(A \bigcup B_1 \bigcup B_2 \bigcup B_3 \bigcup C)$. c_{new} satisfies $c_{new} = c_{origin}$ or $c_{new} \prec c_{origin}$. Each rule in Table XVI describes one possible relationship between the schemas obtained by strategy *regarded as one failure* and the actual MFS. For example, rule 2, i.e., $If\ c_m = c_{origin}\ and\ c_{new} \prec c_{origin}, then\ \exists c'_m, s.t., c_{new} \prec c'_m$, indicates that the schema c_{new} by *regarded as one failure* strategy will be the sub schema of some actual MFS, if it is also the sub schema of the schema c_{origin} by strategy *knowing masking effects*, and c_{origin} is identical to some actual MFS. Some rules may offer more than one possible relationship between c_{new} and the actual MFS. For example, rule 4 states that c_{new} either be the sub schema of the actual MFS or be irrelevant to all the actual MFS. In this case, we divide the rule into several sub-rules. For rule 4, there are two sub-rules: 4a and 4b, each of which represents one possible relationship between c_{new} and the actual MFS c'_m.

Among these rules, only rule 1 and 3 state $\exists c'_m, s.t.\ c_{new} = c'_m$ or $c'_m \prec c_{new}$. This can be easily understood, as to make $c_{new} = c'_m$ or $c'_m \prec c_{new}$, according to Proposition 3.17 and 3.18, it must have $\mathcal{T}(c_{new}) \subset (B_1 \bigcup B_2)$. This condition does not hold if $c_{new} \prec c_{origin}$. As if so, then $\exists t \in B_3, s.t., t \in \mathcal{T}(c_{new})$, otherwise, there should be no schema c_{new} that is the sub-schema of c_{origin}. Consequently, $\mathcal{T}(c_{new}) \not\subset (B_1 \bigcup B_2)$ if $c_{new} \prec c_{origin}$. So to make $c_{new} = c'_m$ or $c'_m \prec c_{new}$, the schema c_{new} must be identical to c_{origin}, i.e., $c_{new} = c_{origin}$ and there must be $c_{origin} = c_m$ or $c_m \prec c_{origin}$, correspondingly. Apart from these two rules, the remaining rules indicate $c_{new} \prec c'_m$ or $c_{new}\ is\ irrelevant\ to\ all\ the\ actual\ MFS$, which implies the schemas that are obtained by strategy *regarded as one failure* are more likely to be subschemas or irrelevant schemas of the actual MFS when compared to that of the *knowing masking effects* strategy.

Next we will give examples to depict the rules except those with the condition '$c_{new} = c_{origin}$' (rule 1, 3, 5, 7). This is because these rules simply result in the relationship between c_{new} and actual MFS be the same as the relationship between c_{origin} and the actual MFS. Table **??** presents the examples of all the remaining rules. This table consists of three parts, with the upper part giving the test cases for each area in the abstract FCI model. Note that we only list the union of areas B_1, B_2 and C. This is because the union is the common element for computing the MFS of three strategies – *actual MFS, knowing masking effects, regarded as one failure*. The middle part of this table shows the minimal schemas using a particular method. The lower part depicts the sample of each possible rule in Table XVI. In this part, the left column indicates the specific rule id. Column 'c_m', 'c_{origin}', 'c_{new}', 'c'_m', respectively, indicates the schema which satisfies the rule in the corresponding row. The mark $*$ means the rule is irrelevant to this schema. For example, for rule 8b, i.e., if $c_{origin}\ irrelevant\ all\ c_m$ and $c_{new} \prec c_{origin}$ then $c_{new}\ irrelevant\ to\ all\ c'_m$, we marked '*' in the column 'c_m' and 'c'_m'.

4.2.2. Using distinguish strategy. For strategy *distinguishing failures*, the influence can be described as shown in Table XVII.

Similar to Table XVI, this table also lists the possible relationships among the schemas obtained by strategy *distinguishing failures*, schemas obtained by *knowing masking effects* strategy, and the actual MFS. As with the *distinguishing failures* strategy, the minimal schemas identified are actually $\mathcal{C}(A \bigcup B_1 \bigcup C)$. Obviously $A \bigcup B_1 \bigcup C \subset A \bigcup B_1 \bigcup B_2 \bigcup C$. So with this strategy, $c_{new} \in \mathcal{C}(A \bigcup B_1 \bigcup C)$ should be either the super-schema or identical to $c_{origin} \in \mathcal{C}(A \bigcup B_1 \bigcup B_2 \bigcup C)$. The main difference between the rules in Table XVII and those in Table XVI is that most rules with

Table XVII. Masking effects influence on FCI with distinguishing failures strategy

1	If $c_m = c_{origin}$ and $c_{new} = c_{origin}$	Then, $\exists c'_m, s.t., c_{new} = c'_m$
2	If $c_m = c_{origin}$ and $c_{origin} \prec c_{new}$	Then, $\exists c'_m, s.t., c'_m \prec c_{new}$
3	If $c_m \prec c_{origin}$ and $c_{new} = c_{origin}$	Then, $\exists c'_m, s.t., c'_m \prec c_{new}$
4	If $c_m \prec c_{origin}$ and $c_{origin} \prec c_{new}$	Then, $c'_m, s.t., c'_m \prec c_{new}$
5	If $c_{origin} \prec c_m$ and $c_{new} = c_{origin}$	Then, $\exists c'_m, s.t., c_{new} \prec c'_m$
6a	If $c_{origin} \prec c_m$ and $c_{origin} \prec c_{new}$	Then either $\exists c'_m, s.t., c_{new} = c'_m$
6b		Or $\exists c'_m, s.t., c'_m \prec c_{new}$
6c		Or $\exists c'_m, s.t., c_{new} \prec c'_m$
6d		Or c_{new} $irrelevant\ all\ c'_m$
7	If c_{origin} $irrelevant\ all\ c_m$ and $c_{new} = c_{origin}$	Then, c_{new} $irrelevant\ to\ all\ c'_m$
8a	If c_{origin} $irrelevant\ all\ c_m$ and $c_{origin} \prec c_{new}$	Then either,$\exists c'_m, s.t., c'_m \prec c_{new}$
8b		Or,c_{new} $irrelevant\ to\ all\ c'_m$
9	It may have c_{origin} $irrelevant\ to\ all\ c_{new}$	

strategy *distinguishing failures* result in $c'_m \prec c_{new}$. This is because with strategy *distinguishing failures*, the test cases that are used for identifying the MFS is less than that of *knowing masking effects*. So it is more likely to have $\mathcal{T}(c_{new}) \subset \mathcal{T}(c'_m)$, and hence $c'_m \prec c_{new}$.

We give an example to illustrate these rules of strategy *distinguishing failures*, as depicted in Table **??**. Similar to Table **??**, we omit the samples that have the condition '$c_{origin} = c_{new}$'.

One rule to note is rule 9, which can make some c_{origin} *removed* from the newly minimal schemas, i.e., $\not\exists c_{new}, s.t., c_{new} = c_{origin}\ or\ c_{origin} \prec c_{new}$. For the Table **??** example, in the last row for rule 9, we find that for the schema $c_{origin} - -$(1,1,0,0,1,-), which is identical to the one in actual MFS, there exists no c_{new} which is identical to or the super-schema of this schema. Consequently, in this case, this strategy may ignore some actual MFS compared with *knowing masking effects*.

In fact, besides rule 9, there is another condition under which the FCI approach may ignore some actual MFS, namely when one c_{new} is the only schema that is *related* to c_{origin}, (*related* means not irrelevant, and in this case it is either identical to or the super-schema), and the corresponding c_{origin} is the only schema which is related to the actual MFS c_m. Then, if the c_{new} is irrelevant to all the actual MFS, we will ignore the actual MFS c_m. Note that this *ignoring* event is based on the condition that the c_{new} is irrelevant to all the actual MFS, and this condition is common for both strategies. But rule 9, which can also lead to MFS ignored, only can be realized when applying *distinguishing failures* strategy on the FCI approaches. This indicates that this strategy has a larger chance to ignore some actual MFS than the *regarded as one failure* strategy.

4.3. Summary of the masking effects on the FCI approach

From the analysis of the formal model, we can learn that masking effect does influence the FCI approaches, and even worse, both the *regarded as one failure* and *distinguishing failures* strategies have their own problems in handling this effect. Specifically when compared with the *knowing masking effects* condition, the strategy *regarded as one failure* has a larger possibility of getting more sub-schemas of the actual MFS and getting more schemas which are irrelevant to the MFS, while strategy *distinguishing failures* may get more super schemas of the MFS and can also get more irrelevant MFS. Further, both strategies may ignore the actual MFS with the *distinguishing failures* strategy more likely to ignore the MFS than the *regarded as one failure* strategy.

5. TEST CASE REPLACING STRATEGY

The main reason that the FCI approach fails to work properly is that it cannot determine B_2 and B_3, i.e., if the test case triggers other failures which are different from the currently analysed one, it cannot figure out whether this test case will trigger the current expected failure because of the masking effects. So to limit the impact of this effect on the FCI approach, it is important to reduce the number of test cases that trigger other different failures, as it can reduce the probability that expected failure may be masked by other failures.

In the exhaustive testing, as all the test cases will be used to identify the MFS, there is no room left to improve the performance unless we fix other failures and re-execute all the test cases. However, if only a subset of all test cases is used to identify the MFS (which is how the traditional FCI approach works), it is important to make the right selection to limit the size of $\mathcal{T}(mask_{F_m})$ to be as small as possible.

5.1. Replacing test cases that trigger unexpected failures

The basic idea is to pick the test cases that trigger other failures and generate new test cases to replace them. The regenerated test cases should either pass in the execution or trigger F_m. The replacement must satisfy the condition that the newly generated ones will not negatively influence the original identifying process.

Normally, when we replace the test case that triggers an unexpected failure with a new test case, we should keep some part of the original test case. We call this part the *fixed part*, and mutate the other part with different values from the original one. For example, if a test case (1,1,1,1) triggered an unexpected failure, and the fixed part is (-,-,1,1). Then, we can replace it with a test case (0,0,1,1) which may either pass or trigger the expected failure.

The *fixed part* can vary for different FCI approaches, e.g, for the OFOT [Nie and Leung 2011a] algorithm, the parameter values are the fixed part except for the one that needs to be validated, while for the FIC_BS [Zhang and Zhang 2011] approach, the fixed parts are dynamically changed, depending on the outcome of the execution of last generated test case.

This replacement may need to be executed multiple times for one fixed part as it may not always possible to find a test case that coincidentally satisfied our requirement. One replacement method is randomly choosing test cases until the satisfied test case is found. While this method may be simple and straightforward, however, it also may require trying many times. So to handle this problem and reduce the cost, we propose a replacement approach by computing the *strength* of the test case with the other failures, and then we select the test case from a group of candidate test cases that has the least *strength* related to the other failures.

To explain the *strength* notion, we first introduce the *strength* that a parameter value is related to a particular failure. We use $all(o)$ to represent the number of executed test cases that contain this parameter value, and $m(o)$ to indicate the number of test cases that trigger the failure F_m and contain this parameter value. Then, the *strength* that a parameter value is related to a particular failure, i.e., $S(o, F_m)$, is $\frac{m(o)}{all(o)+1}$. This heuristic formula is based on the idea that if a parameter value frequently appears in the test cases that trigger a particular failure, then it is more likely to be the inducing factor that triggers this failure. We add 1 in the denominator for two reasons: (1) avoid division by zero when the parameter value has never appeared before, (2) reduce the bias when a parameter value rarely appears in the test set but coincidentally appears in a failing test case with a particular failure.

With the *strength* associated with a parameter value, we then define the *strength* of a test case f is related to a particular failure F_m as:

$$S(f, F_m) = \frac{1}{k} \sum_{o \in f} S(o, F_m) \qquad \textbf{(EQ1)}$$

where k is the number of parameters in f, and o is the specific parameter value in f. The *strength* that a test case is related to a failure is defined as the average *strength* of the relevance between each parameter value in the test case and this failure. For a selected test case, we want its ability to trigger another failure to be as small as possible, such that the masking effects can be alleviated. In practice, the relevance *strength* varies between test case with different failures. As a result, we cannot always find a test case that, for any failure, the *strength* that this test case is related to that failure is the least. With this in mind, we have to settle for a test case, such that the maximal possible failure (except for the one that is currently analysed) it can trigger should be the least likely to be triggered when compared with that of other test cases. In other word, we need to find a test case, so that the maximal *strength* that it is related to another failure is minimal. Formally, we should choose a test case f, s.t.,

$$\min_{f \in R} \max_{m \leq L \& m \neq n} S(f, F_m) \tag{EQ2}$$

where L is the number of failures, and n is the current analysed failure. R is the set of all possible test cases that contain the $fixed$ part except those that have been tested. The test case is from the set R because the FCI approach needs to keep the $fixed$ part when generating additional test cases and the test case should not have been executed. Obviously $|R| = \prod_{i \notin fixed}(v_i) \ - |\{t \mid t \; contains \; the \; fixed \; part \; \& \; t \; is \; executed\}|$.

We can further resolve this problem. Consider the test case f satisfies EQ2. Without loss of generality, we assume that the failure $F_k, k \neq n$ is the failure with which the test case f has the maximal related *strength* compared to the other failures. Then, a natural property for f is that any other test case f' which satisfies that failure F_k is the maximal related failure for this test case and must have $S(f, F_k) <= S(f', F_k)$. Formally, to obtain such a test case is to solve the following formula:

$$\begin{aligned} \min \quad & S(f, F_k) \\ \text{s.t.} \quad & f \in R \\ & S(f, F_k) > S(f, F_i), \quad 1 \leq i \leq L \;\&\; i \neq k, n \end{aligned} \tag{EQ3}$$

To solve EQ2, we just need to find a particular failure F_k, and the corresponding test case f_k that satisfies EQ3, such that the related *strength* between f_k and F_k is smaller than the related *strength* between any other failures and their corresponding test cases that satisfy EQ3. Formally, we need to find:

$$\begin{aligned} \min \quad & S(f, F_k) \\ \text{s.t.} \quad & 1 \leq k \leq L \;\&\; k \neq n \\ & f, F_k \; satisfies \; EQ3 \end{aligned} \tag{EQ4}$$

According to EQ4, to determine such a test case lies in solving EQ3 because if it is solved we just need to rank the one that has the minimal value from the solutions to EQ3. As to EQ3, it can be formulated as an 0-1 integer linear programming (ILP) problem. Assume the SUT has K parameters in which the ith parameter has V_i values. And the SUT has L failures. We then define the variable x_{ij} as:

$$x_{ij} = \begin{cases} 1 & the \; ith \; parameter \; of \; the \; test \; case \; take \; the \; jth \; value \; for \; that \; parameter \\ 0 & otherwise \end{cases}$$

We then take $o_{m_{ij}}$ to be the related *strength* between the *j*th value of the *i*th parameter of the SUT and the failure F_m. And we use a set R of parameter values to define the fixed part in the test case we should not change, i.e., R =

$\{(i,j)|i\ is\ the\ fixed\ parameter\ in\ the\ test\ case, j\ is\ the\ corresponding\ value\}$. As we can generate redundant test cases, so we keep a set of test cases $T_{executed}$ to guide the generation of different test cases. Then EQ3 can be transformed into following ILP formula:

$$
\begin{aligned}
\min \quad & \frac{1}{|K|}\sum_{i=0}^{K}\sum_{j=0}^{V_i} o_{m_{ij}} \times x_{ij} & & \text{(EQ5)} \\
\text{s.t.} \quad & 0 \leq x_{ij} \leq 1 & i = 0..K-1, j = 0,..V_i - 1 & \quad (1) \\
& x_{ij} \in \mathbb{Z} & i = 0..K-1, j = 0,..V_i - 1 & \quad (2) \\
& \sum_{j=0}^{V_j} x_{ij} = 1 & i = 0...K-1 & \quad (3) \\
& x_{ij} = 1 & (i,j) \in R & \quad (4) \\
& \sum_{i=0}^{K}\sum_{j=0}^{V_i}(o_{m_{ij}} - o_{m'_{ij}}) \times x_{ij} \geq 0 & 1 \leq m' \neq m \leq L & \quad (5) \\
& \sum_{(i,j)\in t} x_{ij} < K & t \in T_{existed} & \quad (6)
\end{aligned}
$$

In EQ5, constraints (1) and (2) indicate that the variable x_{ij} is a 0-1 integer. Constraint (3) indicates that a parameter in one test case can only take on one value. Constraint (4) indicates that the test case should not change values of the fixed part. Constraint (5) indicates that the related strength between the test case and failure F_m is higher than that of the other failures. Constraint (6) indicates that the test cases generated should not be the same as the test cases in $T_{existed}$.

As we have formulated the problem into a 0-1 integer programming problem, we just need to utilize an ILP solver to solve this formula. In this paper, we use the solver introduced in [Berkelaar et al. 2004], which is a mixed Integer Linear Programming (MILP) solver that can handle satisfaction and optimization problems.

The complete process of replacing a test case with a new one while keeping some fixed part is depicted in Algorithm 1.

The inputs to this algorithm consist of the failure F_m, the fixed part of which we want to keep from the original test case s_{fixed} , the set of values that each parameter can take $Param$ and the set of matrix $o_1, o_2, ...o_m, ..., o_L$, where o_m ($1 \leq m \leq L$) is recorded the related strength between each specific parameter with each value and the failure F_m, i.e., $o_m = \{o_{m_{ij}}|0 \leq i \leq K-1, 0 \leq j \leq V_i\}$. The output of this algorithm is a test case t_{new} which either triggers the expected F_m or passes.

The outer loop of this algorithm (lines 1 - 19) contains two parts:

The first part (lines 2 - 9) generates a new test case which is supposed to be least likely to trigger failures different from F_m. The basic idea for this part is to search each failure different from F_m (line 3) and find the best test case that has the least related strength with other failures. In detail, for each failure we set up an ILP solver (line 4) and use it to get an optimal test case for that failure according to EQ5 (line 5). We compare the optimal value for each failure, and choose the one with less strength related to other failures (lines 6 - 9).

The second part is to check whether the newly generated test case is as expected (lines 10 - 16). We first execute the SUT under the newly generated test case (line 10) and update the related strength matrix ($o_1...o_L$) for each parameter value that is

ALGORITHM 1: Replacing test cases triggering unexpected failures

Input: failure F_m,fixed part s_{fixed}, set of values that each option can take $Param$, the related strength matrix $o_1...o_L$

Output: t_{new} the regenerate test case,The frequency number

```
1  while not MeetEndCriteria() do
2      optimal ← MAX ; t_new ← null ;
3      forall the F_k ∈ F_1, ...F_m, F_{m+1}...F_L do
4          solver ← setup(s_fixed, Param, F_m, o_1...o_L);
5          (optimal', t'_new) ← solver.getOptimalTest() ;
6          if optimal' < optimal then
7              t_new ← t'_new;
8          end
9      end
10     result ← execute(t_new);
11     updatetRelatedStrengthMatrix(t_new) ;
12     if result == PASS or result == F_m then
13         return t_new;
14     else
15         continue;
16     end
17 end
18 return null
```

involved in this newly generated test case (line 11). We then check the execution result. If the test case passes or triggers the same failure – F_m, a satisfied test case is obtained (line 12) and returned (line 13). Otherwise, we will repeat the process, i.e., generate a new test case and check again (lines 14 - 15).

Note that this algorithm has another exit, besides finding an expected test case (line 12), which is when the function *MeetEndCriteria()* returns *true* (line 1). We did not explicitly show function *MeetEndCriteria()*, because this is dependent on the computing resource and the desired accuracy. In detail, if we want to get a high quality result and have enough computing resource, it is desirable to try many times to get the expected test case; otherwise, a relatively small number of attempts is recommended.

In this paper, we just set 3 as the greatest number of iterations for this function. When it ends with *MeetEndCriteria()* returning true, it will return null (line 18), which means we cannot find an expected test case.

5.2. A case study using the replacement strategy

Suppose we have to test a system with eight parameters, each of which has three options. And when we execute the test case T_0 – (0, 0, 0, 0, 0, 0, 0, 0), a failure–$e1$ is triggered. Furthermore, there are two more potential failures, $e2$ and $e3$, that may be triggered during the testing; and they will mask the desired failure $e1$. Next, we will use FIC_BS [Zhang and Zhang 2011] with replacement strategy to identify the MFS for $e1$. The process is shown in Figure 5. In this figure, there are two main columns. The left main column indicates the executed test cases during testing as well as the executed results, and each executed test case corresponds to a specific label, T_1 – T_8, at the left. The right main column lists the related strength matrix when a test case triggers $e2$ or $e3$. In detail, the matrix records the related strength between each parameter (Columns $P1$ – $P8$) for each value it can take (Column v) with the unexpected failure (Column F). The executed test case, shown in bold, indicates the one that triggers the other failure and should be replaced in the next iteration.

The completed MFS identifying process listed in Figure 5 works as follows: firstly the original FCI approach determines which $fixed$ part needed to be test in each iteration. Then the extra test case will be generated to fill in the remaining part. After executing the extra test case, if the result of the execution is normal, i.e., did not trigger unexpected failure (e_2, e_3), then the original FCI process will continue until the MFS is identified. Otherwise, the replacement strategy starts when an unexpected failure is triggered. The replacement process will mutate the parameter values that is not in the $fixed$ part according to Algorithm 1. After the replacement process, the control for the MFS identifying process will be passed back to the original FCI approach. Next we will specifically explain how the replacement works with an example in this figure.

From Figure 5, for the test case that triggered $e2$ – (2, 1, 1, 1, 0, 0, 0, 0) (in this case, the fixed part of the test case is (-, -, -, -, 0, 0, 0, 0), in which the last four parameter values are the same as the original test case T_0), we generate the related matrix at left. Each element in this matrix is computed as $\frac{m(o)}{all(o)+1}$; for example, for the $P7$ parameter with value 0, we can find two test cases that contain this element, i.e., T_0 and T_1, so $all(o)$ is 2. And only one test case triggers the failure $e2$, which means m(o) = 1. So the final related strength between this parameter value with $e2$ is $\frac{1}{2+1} = 0.33$. All the related strength with $e3$ is labeled with a short slash as there is no test case triggering this failure in this iteration. After this matrix has been determined, we can obtain the optimal test case with the ILP solver, which is T_1'–(1, 2, 2, 2, 0, 0, 0, 0), with its related strength 0.167, which is smaller than that all.

This replacement process is started each time a new test case that triggered another failure until we finally get the MFS. Sometimes we could not find a satisfied replacing test case in just one trial like T_1 to T_1'. When this happened, we needed to repeat searching the proper test case. For example, for T_4 which triggered $e3$, we tried three times– T_4', T_4'', T_4''' to finally get a satisfied T_4''' which passes the testing. Note that the *related strength* matrix continues to change as the test case is generated and executed so that we can adaptively find an optimal one.

5.3. Complexity analysis

The complexity of our approach relies on two variables: the number of test cases that triggered other failures which need to be replaced, and the number of test cases that need to be tried to generate a non-masking-effect test case. The complexity is the product of these two variables.

The first variable is dependent on the extra test cases that are needed to identify the MFS, and this number varies in different FCI approaches. Table XVIII lists the number of test cases that each algorithm needed to get the MFS, where d indicates the number of MFS in the SUT. k means the number of the parameters of the SUT. t is the degree of MFS in the SUT. c is an upper bound, and satisfies $d \leq \frac{c}{2} loglogk$. v is the number of values that a parameter can take.

Note that each algorithm may have some restrictions, details of which are shown in [Zhang and Zhang 2011].

To get the magnitude of the second variable, we need to figure out the probability of a test case that could trigger other failure. The first consideration is the *fixed* part, as the additional generated test case should somehow contain this part. As mentioned before, we can generate $(v-1)^{k-p}$ (p is the number of parameter values in the *fixed* part) possible test cases that contain the *fixed* part. Apart from the one that needs to be replaced, there remain $(v-1)^{k-p} - 1$ candidate test cases, which indicates the complexity is $O((v-1)^{k-p} - 1)$. However, to avoid the exponential computation, we use the $MeetEndCriteria()$ function to end Algorithm 1 when the attempts to find a proper

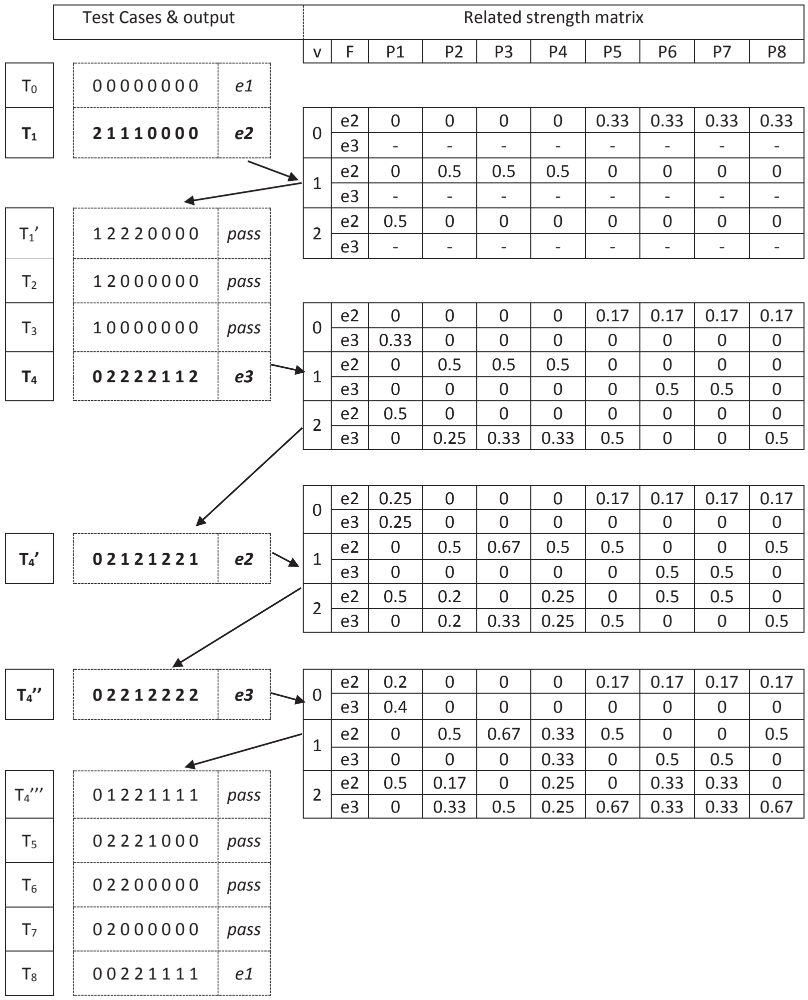


Fig. 5. A case study using our approach

Table XVIII. The number of test cases each FCI approach needed to identify MFS

Method	**number of test cases to identify MFS**
Charles ELA	depends on the covering array
Martinez with safe value [Martínez et al. 2008; 2009]	$O(\hat{d}logk + \hat{d}^2)$
Martinez without safe values [Martínez et al. 2008; 2009]	$O(d^2 + dlogk + log^c k)$
Martinez' ELA [Martínez et al. 2008; 2009]	$O(ds^v logk)$
Shi SOFOT [Shi et al. 2005]	$O(k)$
Nie OFOT [Nie and Leung 2011a]	$O(k \times d)$
Ylimaz classification tree	depends on the covering array
FIC [Zhang and Zhang 2011]	$O(k)$
FIC_BS [Zhang and Zhang 2011]	$O(t(logk + 1) + 1)$
Ghandehari's suspicious based [Ghandehari et al. 2012]	depends on the number and size of MFS
TRT [Niu et al. 2013]	$O(d \times t \times logk + t^d)$

Table XIX. The complexity of the second part

Method	**fixed part**
Charles ELA	$O(min(N,(v-1)^{k-t}-1))$
Martinez with safe value [Martínez et al. 2008; 2009]	$O(min(N,(v-1)-1)) \sim O(min(N,(v-1)^{k-1}-1))$
Martinez without safe values [Martínez et al. 2008; 2009]	−
Martinez' ELA [Martínez et al. 2008; 2009]	$O(min(N,(v-1)^{k-t}-1))$
Shi SOFOT [Shi et al. 2005]	$O(min(N,(v-1)-1)))$
Nie OFOT [Nie and Leung 2011a]	$O(min(N,(v-1)-1)))$
Ylimaz classification tree	$O(min(N,(v-1)^{k-t}-1)$
FIC [Zhang and Zhang 2011]	$O(min(N,(v-1)-1)) \sim O(min(N,(v-1)^{k-1}-1))$
FIC_BS [Zhang and Zhang 2011]	$O(min(N,(v-1)-1)) \sim O(min(N,(v-1)^{k-1}-1))$
Ghandehari's suspicious based [Ghandehari et al. 2012]	$O(min(N,(v-1)^{k-t}-1))$
TRT [Niu et al. 2013]	$O(min(N,(v-1)-1)) \sim O(min(N,(v-1)^{k-1}-1))$

test case is over a prior given constant, say N, so the complexity for the second part is $O(min(N,(v-1)^{k-p}-1))$.

We note that exponent $k-p$ has a significant effect on the complexity of the second value. The greater p is, the less test cases that can be generated. Different approach has different p. For example, for the OFOT approach, p is equal to $k-1$. And for the FIC_BS approach, p varies ranging from $k-1$ to 1. While for the non-adaptive approaches, as the $fixed$ part is commonly the schemas that are needed to be covered, so the p for these approaches is at least equal to t. We have listed all of them in Table XIX. It is noted that the approach $Martinez\ without\ safevalues$ has no such complexity, because it works when $v=2$, and this results in not having other test cases to be replaced if we test a fixed part when triggering other failures.

Above all, the cost of replacement strategy varies in different FCI approaches. Note that this cost is computed for the worst case, and in practice much less test cases are needed to identify the MFS. This is because first, not every test case generated by the FCI approach needs to be replaced; and second, a satisfied test case can usually be found before the whole searching space is explored. From complexity analysis, in fact, we cannot determine which approach is better than others, as the cost of each approach is dependent on different factors. For example in Table XVIII, the extra test cases needed for OFOT[Nie and Leung 2011a] and FIC_BS[Zhang and Zhang 2011] are $O(k \times d)$ and $O(t(logk+1)+1)$, respectively. Values t, k and d determine which one is better in practice. So for different SUT with different MFS, which FCI approach is better can be completely changed. Besides, the less cost in practice is not always good for identifying MFS. A potential problem is, with less candidate test cases, the replacement strategy may not find a satisfied test case, which will result in a low quality of the identified schemas.

6. EMPIRICAL STUDIES

To investigate the impact of masking effects on FCI approaches in real software testing scenarios and to evaluate the performance of our approach in handling this effect, we conducted several empirical studies. Each of the studies focuses on addressing one particular issue, as follows:

Q1: Do masking effects exist in real software that contains multiple failures?

Q2: How well does our approach perform compared to traditional approaches?

Q3: Is the ILP-based test case searching technique efficient compared to the random selection?

Q4: Compared to another masking effects handling approach FDA-CIT [Yilmaz et al. 2014], does our approach have any advantages ?

Table XX. Software under survey

software	versions	LOC	classes	bug pairs
HSQLDB	2.0rc8	139425	495	#981 & #1005
	2.2.5	156066	508	#1173 & #1179
	2.2.9	162784	525	#1286 & #1280
JFlex	1.4.1	10040	58	#87 & #80
	1.4.2	10745	61	#98 & #93

6.1. The existence and characteristics of masking effects

In the first study, we surveyed two kinds of open-source software systems to gain an insight into the existence of multiple failures and their effects. The software under study were HSQLDB and JFlex. The first is a database management software written in pure Java and the second is a lexical analyser generator. The reason that we chose these two systems is because they both contain different versions and are all highly configurable so that the options and their interactions can affect their behaviour. Additionally, they all have a developer community so that we can easily obtain the real bugs reported in the bug tracker forum. Table XX lists the program, the versions surveyed, number of lines of uncommented code, number of classes in the project, and the bug's id [3]for each of the software.

6.1.1. Study setup. We first looked through the bug tracker forum and focused on the bugs which are caused by the options interactions. 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 example, through analysing the source code of the test file of bug#981 for HSQLDB, we found the failure-inducing interaction for this bug is (*preparestatement*, *placeHolder*, *Long string*). These three parameter values together form the condition that triggers the bug. The analysed result was later regarded as the "prior MFS".

We further built the testing scenario for each version of the software listed in Table XX. The testing scenario is constructed so that we can reproduce different failures by controlling the inputs to the test file. For each version, the source code of the testing file as well as other detailed information is available at https://code.google.com/p/merging-bug-file.

Next, we built the input model which consists of the options related to the failure-inducing interactions and additional options that are commonly used. The detailed model information is shown in Tables XXI and XXII for HSQLDB and JFLex, respectively. Each table is organised into three groups: (1) *common options*, which lists the options as well as their values under which every version of this software can be tested; (2) *specific options*, under which only the specific version can be tested; and (3) *configure space*, which depicts the input model for each version of the software, presented in the abbreviated form $\#values^{\#number\ of\ parameters} \times$ ..., e.g., $2^9 \times 3^2 \times 4^1$ indicates the software has 9 parameters that can take 2 values, 2 parameters 3 values, and only one parameter 4 values.

We then generated the exhaustive set of test cases consisting of all possible interactions of these options. For each of them, we executed the prepared testing file. We recorded the output of each test case to observe whether there were test cases containing prior MFS that did not produce the corresponding bug. Later we refer to those test cases that contain the MFS but did not trigger the expected failure as the *masked* test cases.

[3]http://sourceforge.net/p/hsqldb/bugs
http://sourceforge.net/p/jflex/bugs

Table XXI. Input model of HSQLDB

versions	common options	values
	Server Type	server, webserver, inprocess
	existed form	mem, file
	resultSetTypes	forwad, insensitive, sensitive
	resultSetConcurrencys	read_only, updatable
	resultSetHoldabilitys	hold, close
	StatementType	statement, prepared
	sql.enforce_strict_size	true, false
	sql.enforce_names	true, false
	sql.enforce_refs	true, false
versions	**specific options**	**values**
2.0rc8	more	true, false
	placeHolder	true, false
	cursorAction	next,previous,first,last
2.2.5	multiple	one, multi, defailure
	placeHolder	true, false
2.2.9	duplicate	dup, single, defailure
	defailure_commit	true, false
versions	**Config space**	
2.0rc8	$2^9 \times 3^2 \times 4^1$	
2.2.5	$2^8 \times 3^3$	
2.2.9	$2^8 \times 3^3$	

Table XXII. Input model of JFlex

versions	common options	values
	generation	switch, table, pack
	charset	default, 7bit, 8bit, 16bit
	public	true, false
	apiprivate	true, false
	cup	true, false
	caseless	true, false
	char	true, false
	line	true, false
	column	true, false
	notunix	true, false
	yyeof	true, false
versions	**specific options**	**values**
1.4.1	hasReturn	has, non, default
	normal	true, false
1.4.2	lookAhead	one, multi, default
	type	true, false
	standalone	true, false
versions	**Config space**	
1.4.1	$2^{10} \times 3^2 \times 4^1$	
1.4.2	$2^{11} \times 3^2 \times 4^1$	

6.1.2. Results and discussion. Table XXIII lists the results of our survey. Column “al-l tests” gives the total number of test cases executed. Column “failure” indicates the number of test cases that failed during testing, and column “masking” indicates the number of masked test cases. The percentage in the parentheses indicates the proportion of masked test cases and the failing test cases.

We observed that for each version of the software under analysis listed in Table XXIII, test cases with masking effects do exist, i.e., test cases containing MFS did not trigger the corresponding bug. In fact, there are about 768 out of 4608 test cases (16.7%) in hsqldb with 2rc8 version. This rate is about 16.7%, 50%, 25%, and 16.7%, respectively, for the remaining software versions.

So the answer to **Q1** is that in practice, when SUT have multiple failures, masking effects do exist widely in the test cases.

Table XXIII. Number of failures and their masking effects

software	versions	all tests	failure	masking
HSQLDB	2cr8	18432	4608	768 (16.7%)
-	2.2.5	6912	3456	576 (16.7%)
-	2.2.9	6912	3456	1728 (50%)
JFlex	1.4.1	36864	24576	6144 (25%)
-	1.4.2	73728	36864	6144 (16.7%)

It is notable that in Yilmaz's [Yilmaz et al. 2014] paper, a similar study about the existence of the masking effects has been conducted. The main difference between that work and ours is that Yilmaz's work quantify impact of the masking effects as the number of t-degree schemas that only appear in the test cases that triggered other failures. Here, the t-degree schemas can be either MFS or not. Our work, however, quantify the masking effects as the number of test cases that are masked by unexpected failures. These test cases should contain some MFS, i.e., they should have triggered the expected failure if they did not trigger any other failure. The reason that we quantify the masking effects in such way is because our work seeks to handle the masking effects in the MFS identifying process. As the test cases which contain the MFS but do not produce the corresponding failure will significantly affect the MFS identifying results, their number can better reflect the impact of the masking effects on the FCI approach.

6.2. Comparing our approach with traditional algorithms

The second study aims to compare the performance of our approach with traditional approaches in identifying MFS under the impact of masking effects. To conduct this study, we need to apply our approach and traditional algorithms to identify MFS in a variety of software and evaluate their results. The five versions of software in Table XX used as test objects are far from the requirement for a general evaluation. However, to construct many real testing objects is time-consuming as we must carefully study the detail of that software as well as the bug tracker report. To compromise, we synthesized 10 more testing objects. These synthesized objects are ten small programs which can directly return outputs when executed with given inputs. To make the synthetic objects as similar as possible to the real software, we firstly analysed the characterizations, such as the number of parameters, the number of failures, and the possible masking effects, of the real software. We observed that the number of parameters of the SUT ranged from 8 to 30, the number of different failures in the SUT ranged from 2 to 4, and the number of MFS for a failure ranged from 1 to 2, in which the degree of the MFS ranged from 1 to 6. Then for each characterization, we randomly selected one value in the corresponding range and assigned it to the input model by adjusting the relationships between the inputs and outputs of these programs.

Table XXIV lists the testing model for both the real and synthesizing testing objects. In this table, column 'Object' indicates the SUT under test. For the real SUT listed in Table XX, we label the five software as $H2cr8$, $H2.2.5$, $H2.2.9$, $J1.4.1$, $J1.4.2$, respectively. While for the synthesized ones, we label them in the form of '*syn+ id*'. Column 'Model' presents the input space for each testing object. Column 'Failures' shows the different failures in the software and their masking relationships. In this column, '$>$' means the left failure will mask the right failure, i.e., if the left failure is triggered, then the right failure will not be triggered. Furthermore, '$>$' is transitive so that the left failure can mask all the failures in the right. For example, for the $H2cr8$ object, we can find three failures : e_1, e_2, and e_3. By using the formula $e_1 > e_2 > e_3$, we indicate that the failure e_2 will mask e_3 and e_1 will mask both e_2 and e_3. Here for the simplicity of the experiment, we did not build more complex testing scenarios such as the mask-

Table XXIV. The testing models used in the case study

Object	Model	Failures	MFS for each failure
H2cr8	$2^9 \times 3^2 \times 4^1$	$e_1 > e_2 > e_3$	$(5_1, 6_0, 7_0)_{e_1}, (5_1, 8_2, 9_2)_{e_2}, (5_1, 8_2, 9_1)_{e_2}, (5_1, 8_3, 9_2)_{e_3}, (5_1, 8_3, 9_1)_{e_3}$
H2.2.5	$2^8 \times 3^3$	$e_1 > e_2$	$(6_1, 7_0)_{e_1}, (5_2)_{e_2}$
H2.2.9	$2^8 \times 3^3$	$e_1 > e_2 > e_3$	$(6_0)_{e_1}, (0_1, 5_1, 7_0)_{e_2}, (0_0, 5_1, 7_0)_{e_2}, (5_1, 7_0)_{e_3}$
J1.4.1	$2^{10} \times 3^2 \times 4^1$	$e_1 > e_2$	$(0_0)_{e_1}, (1_0)_{e_2}$
J1.4.2	$2^{11} \times 3^2 \times 4^1$	$e_1 > e_2$	$(1_0, 2_1)_{e_1}, (0_1)_{e_2}$
syn1	$2^5 \times 3^3 \times 4^1$	$e_1 > e_2$	$(2_1, 3_0)_{e_1}, (1_1, 2_1)_{e_2}, (1_0, 3_0)_{e_2}$
syn2	$2^6 \times 3^2 \times 4^1$	$e_1 > e_2 > e_3$	$(4_1, 6_0, 7_1, 8_0)_{e_1}, (1_1, 3_1, 5_1)_{e_2}, (2_0, 3_1, 6_0)_{e_3}$
syn3	$2^5 \times 3^3$	$e_1 > e_2 > e_3$	$(2_1, 3_0)_{e_1}, (1_0)_{e_2}, (4_1)_{e_2}, (6_0, 7_0)_{e_3}$
syn4	$2^7 \times 3^2 \times 4^1$	$e_1 > e_2 > e_3$	$(0_1, 2_1, 5_0, 6_1)_{e_1}, (2_1, 4_0)_{e_2}, (6_1, 7_0)_{e_2}, (3_0, 4_0, 5_0)_{e_3}$
syn5	$2^4 \times 3^3 \times 4^2$	$e_1 > e_2$	$(0_0, 1_1, 3_0, 6_1, 8_0)_{e_1}, (2_0, 3_0, 4_1)_{e_2}$
syn6	$2^9 \times 3^2$	$e_1 > e_2 > e_3 > e_4$	$(2_0, 7_1, 8_1)_{e_1}, (3_1, 5_1)_{e_2}, (4_0)_{e_2}, (3_1, 6_0, 7_1)_{e_3}, (3_1, 7_1, 8_0)_{e_4}$
syn7	$2^{10} \times 3^1 \times 4^1$	$e_1 > e_2 > e_3$	$(3_1, 4_0, 5_0)_{e_1}, (2_0, 4_0, 7_1, 9_0)_{e_2}, (6_1, 10_0, 11_1)_{e_3}$
syn8	$2^{11} \times 3^1 \times 4^1$	$e_1 > e_2$	$(1_0, 3_1, 4_0, 7_1, 9_0, 12_1)_{e_1}, (0_0, 2_1, 3_1, 7_1, 10_0, 11_1)_{e_2}$
syn9	$2^4 \times 4^3$	$e_1 > e_2$	$(3_1, 5_0)_{e_1}, (5_0, 6_1)_{e_2}$
syn10	$2^7 \times 3^3 \times 4^1$	$e_1 > e_2$	$(0_1, 3_0, 4_1, 7_0)_{e_1}, (2_0, 3_0, 5_1)_{e_2}, (2_0, 3_0, 5_0)_{e_2}$

ing effects can happened in the form $e_1 > e_2,\ e_2 > e_3,\ e_3 > e_1$ or even $e_1 > e_2,\ e_2 > e_1$. The last column shows the MFS for each failure. The MFS is presented in an abbreviated form $\{\#index_{\#value}\}_{failure}$, e.g., for the object $H2cr8$, $(5_1, 6_0, 7_0)_{e_1}$ actually means (-, -, -, -, -, 1, 0, 0, -, -, -, -) is the MFS for the failure e_1.

6.2.1. Study setup. After preparing the objects under testing, we then applied our approach (FIC_BS with replacement strategy) to identify the MFS. Specifically, for each SUT we selected each test case that failed during testing and fed it into our FCI approach as the input. Then, after the identifying process was completed, we recorded the identified MFS and the extra test cases needed. For the traditional FIC_BS approach, we designed the same experiment. But as the objects being tested have multiple failures for which the traditional FIC_BS can not be applied directly, we adopted two traditional strategies on the FIC_BS algorithm, i.e., *regarded as one failure* and *distinguishing failures* as described in Section 3.2. The purpose of recording the generated additional test cases is to quantify the additive cost of our approach.

We next compared the identified MFS of each approach with the prior MFS to quantify the degree that each suffers from masking effects. There are five metrics used in this study, listed as follows:

(1) *Accurate number* : the number of identified MFS which are actual prior MFS.
(2) *Super number*: the number of identified MFS that are the super schemas of some prior MFS.
(3) *Sub number* : the number of identified MFS that are the sub schemas of some prior MFS.
(4) *Ignored number* : the number of schemas that are in the prior MFS, but irrelevant to the identified MFS.
(5) *Irrelevant number* : the number of schemas in the identified MFS that are irrelevant to the prior MFS.

Among these five metrics, the *accurate number* directly indicates how effectively the FCI approaches performed, since to identify as many actual MFS as possible is the target for every FCI approach. Metrics *ignored number* and *irrelevant number* indicate the extent of deviation for the FCI approaches, specifically, the former indicates how much information about the MFS will miss, while the latter indicates how serious the distraction would be due to the useless schemas identified by the FCI approach. *Super number* and *sub number* are the metrics in between, i.e., to identify some schemas that is *super* or *sub* schemas of the actual MFS is better than identifying *irrelevant* ones or ignoring some MFS, but it is worse than identifying the schema that is identical to some actual MFS. This is intuitive, as given the *super / sub* schemas, we just need to *remove / add* some elements of the original schemas to get the actual MFS. While for the *irrelevant* or *ignore* ones, however, more efforts will be needed (e.g., both *adding* and *removing* operations will be needed to revise the irrelevant schemas to the actual MFS).

Besides these specific metrics, we also give a composite metric to measure the overall performance of each approach. The composite metric *aggregate* is defined as follows:

$$Aggregate = \frac{accurate + related(super) + related(sub)}{accurate + super + sub + irrelevant + ignored}$$

In this formula, *accurate, super, sub, irrelevant,* and *ignored* represent the value of each specific metric. To refine the evaluation of different *super / sub* schemas, we design a *related* function which gives the similarity between the schemas (either super or sub) and the real MFS, so that we can quantify the specific effort for changing a *super / sub* schema to the real MFS. The similarity between two schemas S_A and S_B is computed as:

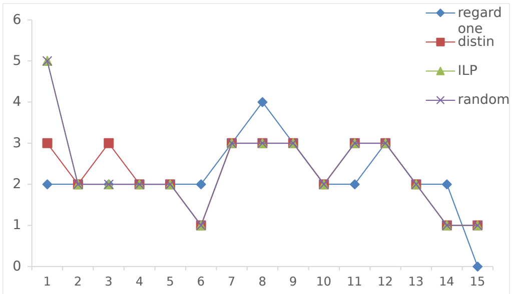
$$Similarity(S_A, S_B) = \frac{number\ of\ same\ elements\ in\ S_A\ and\ S_B}{\max(Degree(S_A), Degree(S_B))}$$

.

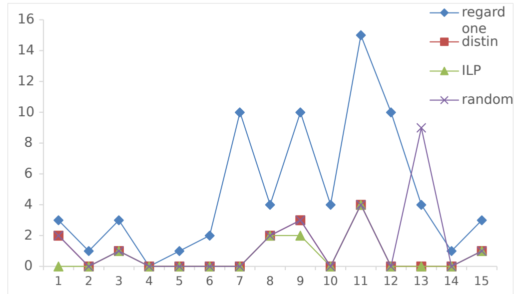
For example, the similarity of (- 1 2 - 3) and (- 2 2 - 3) is $\frac{2}{3}$. This is because S_A and S_B have the same third and last elements, and both of them are three-degree.

So the *related* function is the summation of similarity of all the super or sub schemas with their corresponding MFS.

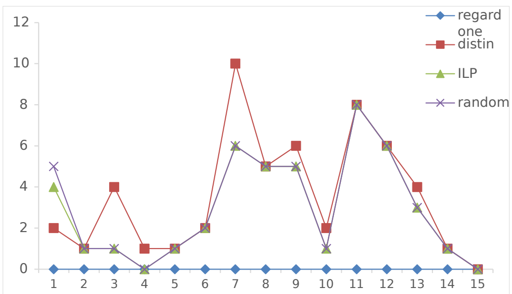
6.2.2. Results and discussion. Figure 6 depicts the results of the second case study. There are seven sub-figures in this figure, i.e., Figure 6(a) to Figure 6(g). They indicate the results of the number of accurate MFS each approach identified, the number of identified schemas which are the sub-schema / super-schema of some prior MFS, the number of ignored prior MFS, the number of identified schemas which are irrelevant to all the prior MFS, the aggregate value, and the extra test cases each algorithm needed, respectively. For each sub-figure, there are four polygonal lines, each of which shows the results for one of the four strategies: *regarded as one failure*, *distinguishing failures*, *replacement strategy based on ILP searching*, *replacement strategy based on random searching* (The last one will be discussed in the next case study). Specifically, each point in the polygonal line indicates the specific result of a particular strategy for the corresponding testing object. For example in Figure 6(a), the point marked with '♦' at (1,2) indicates that the approach using *regarded as one failure* strategy identified 2 accurate MFS in the testing object–HSQLDB 2cr8. The raw data for this experiment can be found in Table XXV of the Appendix. Note that all the data except for the metric *ignored number* are based on all failing test cases for each testing object, i.e., we got the data by comparing the union of the schemas identified in each the failing test cases to the prior actual MFS. As for metric *ignored number*, however, we found that if we merged all the schemas identified in each failing test case, there is no MFS ignored. We therefore use the average score of *ignored number* for each failing test case, which can be seen in the parentheses in Column *ignore* of Table XXV. Next we will discuss the results for each metric.



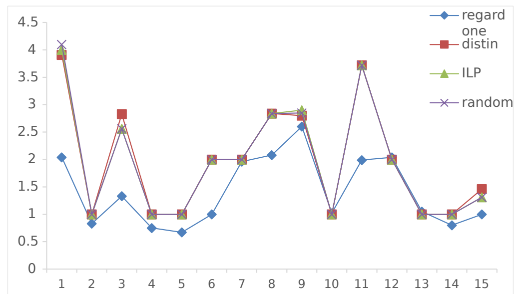
(a) Result of the number of accurately identified MFS



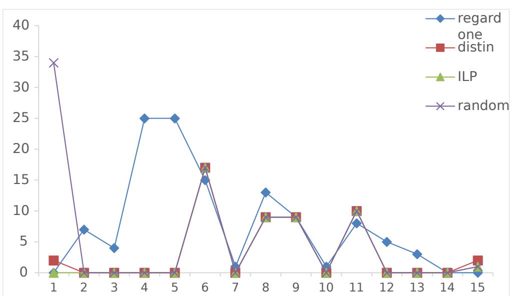
(b) Result of the number of identified sub-schemas of the MFS



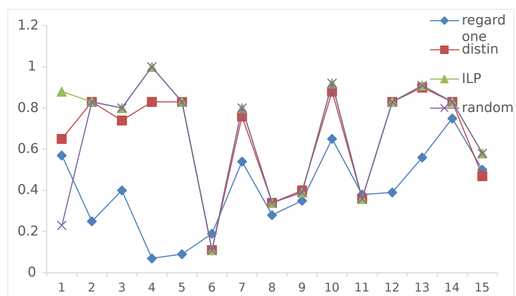
(c) Result of the number of identified super-schemas of the MFS



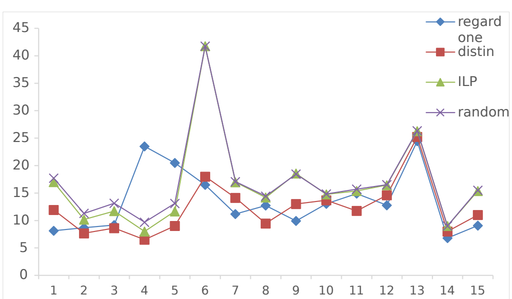
(d) Result of the number of ignored MFS



(e) Result of the number of identified irrelevant schemas of the MFS



(f) The aggregative result for the five metrics



(g) Needed test cases

Fig. 6. Result of the evaluation of each approach

Accurate number: Figure 6(a) shows the number of accurate schemas that each approach achieved. It appears that there is no outstanding strategy, i.e., there does not exist a strategy that can always perform better or worse than others. For example, for the testing object 1, *ILP* performed the best in obtaining the accurate MFS, while for the testing object 2, *distinguishing failures* identified the most accurate MFS and for testing object 3, *regarded as one failure* did the best. However, upon closer inspection, we can find that strategy *distinguishing failures* performed a little better than strategy *regarded as one failure*. This can be reflected in that there are four testing objects (1, 3, 11, and 15) on which strategy *distinguishing failures* performed better, while for strategy *regarded as one failure* there are only three superior testing objects (6, 8, and 14). This subtle difference can be explained by our formal analysis in Section 4. Specifically, for the strategy *regarded as one failure*, only rule 1 in Table XVI can result in the FCI approaches identifying the schemas that are identical to some actual MFS, while for *distinguishing failures*, there are two rules (rule 1 and 6a in Table XVII). As a result, *distinguishing failures* strategy has a slighter larger chance than *regard as one failure* to identify the schemas that are identical to the actual MFS.

We can further find that strategies *replacement strategy based on ILP searching* (short for *ILP* later) and *distinguishing failures* have similar results. This can be easily understood, as strategy *ILP* is actually a refinement version of the strategy *distinguishing failures*, which also make the failures distinguished with each other. The main difference between *ILP* and *distinguishing failures* is that the former has to replace the test cases that triggered any failure other than the currently analysed one while the latter will not change the generated test cases. As a result, the comparison of other metrics (sub, super, ignore, irrelevant numbers) also showed the similarity between strategy *ILP* and *distinguishing failures*.

Sub number & super number: Figure 6(b) and 6(c) depicts the results for *sub number* and *super number*, respectively. These two figures firstly showed a clear trend for strategies *regarded as one failure* and *distinguishing failures*, i.e., the former identified more sub schemas of actual MFS than the latter, while the latter identified more super schemas of actual MFS than the former. This is consistent with our formal analysis. Specifically, there are 6 rules (rules 2, 4a, 5, 6, 8a, 9a listed in Table XVI) for strategy *regarded as one fault* that can lead to the identified schemas being sub schemas of actual MFS, while *distinguishing failures* strategy has 2 such rules (rules 5, 6c in Table XVII). But for the rules that can result in the schemas being super schemas of actual MFS, strategy *regarded as one failure* only has one (rule 3 in Table XVI), while *distinguishing failure* has 5 such rules (rules 2, 3, 4, 6b, 8a in Table XVII).

Although offering similar result as *distinguishing failures* strategy, our strategy *ILP* tend to identify fewer sub schemas and super schemas of actual MFS than strategy *distinguishing failures* (testing objects 1, 9 in Figure 6(b) and testing objects 3, 4,7,9,10 13 in Figure 6(c)). We believe this is an improvement, as too many sub schemas and super schemas will make it harder to identify the actual MFS. One issue is the redundancy problem, as many sub or super schemas in fact point to the same actual MFS.

Ignore number & irrelevant number: The results of the two negative performance metrics are given in Figure 6(d) and 6(e), respectively.

There are two main observations: first, for strategies *regarded as one failure* and *distinguishing failures*, we can find that the former identified more irrelevant schemas of the actual MFS, while the latter ignored more actual MFS. This observation is as expected, because in our formal analysis the strategy *regarded as one failure* has more rules than *distinguish failures* that can lead to the schemas being irrelevant to the actual MFS, in detail, the former has 4 such rules (rules 4b, 7, 8b, 9b in Table XVI) while the latter has three (rules 6d, 7,8b in Table XVII). And for strategy *distinguish* *failures*, rule 9 in Table XVII increases the chance to ignore the actual MFS than the strategy *regarded as one failure*.

The second observation is that *ILP* did a good job at reducing the scores for these two negative metrics. Specifically, for *ignored number*, our approach performed better than strategy *distinguishing failures* at testing object 3 and 15 in Figure 6(d), but is not as good as strategy *regarded as one failure*. In fact, strategy *regarded as one failure* has a significant advantage at reducing the number of ignored MFS as it tends to associate the failures with all the failing test cases. However, when we consider the *irrelevant number*, we can find that our approach is the best among all three strategies (better than *distinguishing failures* at testing object 1 in Figure 6(e), and better than strategy *regarded as one failure* for most testing objects) . We believe this improvement is caused by our test cases replacing strategy, as it can increase the test cases that are useful for identifying the MFS and decrease those useless test cases.

Aggregative for the five metrics: The composite results are given in Figure 6(f). This metric gives an overall evaluation of the quality of the identified schemas. From this figure, we can find that *ILP* performed the best, next the *distinguishing failures*, the last is the *regarded as one failure* (See the testing object 1, 3 and 4 in Figure 6(f)).

It is as expected that *ILP* performed better than *distinguishing failures* as it is actually the refinement version of latter. It is a bit of surprise to find, however, that strategy *distinguishing failures* performed better than *regarded as one failure* at almost all the testing objects. This result cannot be derived from the formal analysis. A possible explanation is that in these testing objects constructed, the possibility of triggering a masking effect is relatively small. Consequently if we take the strategy *regarded as one failure*, we are more likely to misjudge a test case which triggered other failures to be the failing test case for the failure we currently focus on.

Test cases: The number of test cases generated for identifying the MFS indicates the cost of FCI approach. The result is listed in Figure 6(g). We can obviously find that strategy *ILP* generated more test cases than the other strategies. In specific, the gap between the *ILP* and other two strategies ranged from about 2 to 5 (except for the 6th testing object, which exceeds 20), this is acceptable when comparing to all the test cases that each approach needed. The increase in test cases for our approach is necessary, as additional test cases must be generated when some test cases are not satisfied for identifying the MFS of the currently analysed failure. As for strategies *distinguishing failures* and *regarded as one failure*, there is no significant difference between the number of test cases generated.

Above all, we draw three conclusions, which help to answer **Q2**:

1) The results of strategy *distinguishing failures* and *regard one failure* are consistent with the previous formal analysis.

2) Considering the quality of the MFS each approach identified, we can find that our *ILP* approach achieves the best performance, followed by the strategy *distinguishing failures*.

3) Although our approach need more test cases than the other two strategies, it is an acceptable number.

6.3. Evaluating the ILP-based test case searching method

The third empirical study aims to evaluate the efficiency of the ILP-based test case searching component of our approach. To conduct this study, we implemented an FCI approach which is also augmented by the *replacing test cases* strategy, but the test case is randomly replaced.

6.3.1. Study setup. The setup of this case study is based on the second case study, and uses the same SUT model as shown in Table XXIV. We apply the new random searching based FCI approach to identify the MFS in the prepared SUTs. To avoid the bias coming from the randomness, we repeat the new approach 30 times to identify the MFS in each failing test case. We then compute the average additional test cases as well as other metrics listed in section 6.2.1 for the random-based approach.

6.3.2. Results and discussion. The evaluation of this random-based approach is also shown in Figure 6, in which the polygonal line marked with '$\times$' in each sub-figure indicates the results. The raw data can also be found in the column 'R' of Table XXV in the appendix.

Compared to the ILP-based approach, we can firstly observe that there is little distinction between them in terms of the metrics: accurate schemas, super-schemas, sub-schemas, ignored schemas, irrelevant schemas (for some particular cases the ILP-based approach performs slightly better, e.g., in Figure 6(b) for the first testing object, the ILP-based approach identified less sub schemas than that of the Random-based approach and in Figure 6(c) still for the first object the ILP -based approach identified less super schemas than that of the random-based approach). The similar quality of the identified MFS between these two approaches is conceivable as they both use the *test case replacement* strategy, although the test cases generated may be different.

Secondly, when considering the cost, we find that the ILP-based approach performs better, which can reduce on average 1 to 2 test cases compared to the random-based procedure. It shows that our integer programming based searching technique can find a satisfied test case more rapidly than the random approach.

In summary, the answer for **Q3** is that searching for a satisfied test case affects the performance of our approach, especially regarding the number of extra test cases, and the ILP-based test cases can handle the masking effects at a relatively smaller cost than the random-based approach.

6.4. Comparison with Feedback driven combinatorial testing

The *FDA-CIT* [Yilmaz et al. 2014] approach can handle masking effects so that the generated covering array can cover all the t-degree schemas without being masked by the MFS. There is an integrated FCI approach in the FDA-CIT, of which this FCI approach has two versions, i.e., *ternary-class* and *multiple-class*. In this paper, we use the multiple-class version for our comparative approach, as Yilmaz claims that it performs better than the former [Yilmaz et al. 2014].

The FDA-CIT process starts with generating a t-way covering array (In [Yilmaz et al. 2014], this is a test case-aware covering array [Yilmaz 2013]). After executing the test cases in this covering array, it records the outcome of each test case and then applies the classification tree method (Wekas implementation of C4.5 algorithm(J48) [Hall et al. 2009]) on the test cases to characterize the MFS for each failure. It then labels these MFS as the schemas that can trigger masking effects. And later if the interaction coverage is not satisfied (here the interaction coverage criteria is different from the traditional covering array, details see [Yilmaz et al. 2014]), it will re-generate a covering array that aims to cover these schemas that were masked by these MFS labeled as masking effects and then repeat the previous steps.

The main target of FDA-CIT is to make the generated test cases to cover all the t-degree schemas. In order to achieve this goal, FDA-CIT needs to repeatedly identify the schemas that can trigger the masking effects. So to make the two approaches (FDA-CIT and ILP) comparable, we need to collect all the MFS that FDA-CIT characterized in each iteration and then compare them with the MFS identified by our approach.

6.4.1. Study setup. As FDA-CIT used a post-analysis (classification tree) technique on covering arrays, we first generated 2 to 4 ways covering arrays. The covering array generating method is based on augmented simulated annealing [Cohen et al. 2003], as it can be easily extended with constraint dealing and seed injecting [Cohen et al. 2007b], which is needed by the FDA-CIT process. As different test cases will influence the results of the characterization process, we generated 30 different 2 to 4 way covering arrays and fed them to the FDA-CIT. Then after running the FDA-CIT, we recorded the MFS identified, and by comparing them with prior actual MFS, we can evaluate the quality of the identified schemas according to the metrics mentioned in the previous case study.

Besides the FDA-CIT, we also applied our ILP-based approach to the generated covering array. Specifically, for each failing test case in the covering array, we separately applied our approach to identify the MFS for that case. In fact, we can reduce the number of extra test cases if we utilize the other test cases in the covering array [Li et al. 2012]), but we did not utilize the information to simplify the experiment. Similarly, we then recorded the MFS that are identified by our approach, and evaluate them according to the corresponding metrics. In addition, we recorded the overall test cases (including the initially generated covering array) that this approach needed and compared the magnitude of these test cases with that of FDA-CIT.

As mentioned before, the FCI approach in FDA-CIT, i.e., classification tree algorithm, is a post-analysis technique. Given different set of test cases, the results identified by the classification tree algorithm are also different. Then a nature question is, what the schemas identified by FDA-CIT will be if the classification tree method is applied on the test cases generated by our approach ILP? To figure this question out is of importance as first, we can learn that whether the test cases generated by ILP can help FDA-CIT approach to improve its quality of the identified schemas; second, the comparison between ILP and FDA-CIT will be more fair as they share the same test cases. For this, a new approach that based on FDA-CIT is introduced, which is augmented by replacing the original test cases in FDA-CIT with those generated by ILP approach. Then the schemas identified by the classification tree algorithm in FDA-CIT are recorded and evaluated. This new approach is referred to as *FDA-CITs* later.

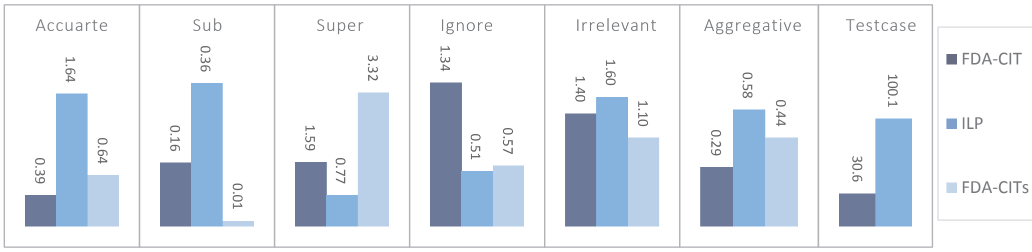
6.4.2. Result and discussion. The result is listed in Figure 7. We conducted three groups of experiments. The first one generated 30 different 2-way cover arrays for each testing object, and then for each covering array we applied the three approaches to identify the MFS. The average evaluation results for the experiments based on 30 covering arrays are listed in the Sub-figure 7(a). The other two groups of experiments starts with 3-way covering arrays and 4-way covering arrays, of which their results are depicted in Sub-figure 7(b) and 7(c) respectively.

In each sub-figure, there are 7 columns, showing the outcomes for the previous mentioned 6 metrics and one more metric (Column *Testcase*), which indicates the overall test cases that each approach needed. Each column has three bars (Except for the Column *Testcase*, as the overall test cases for ILP and FDA-CITs are the same), which indicate the results for approach FDA-CIT, ILP and FDA-CITs, respectively.

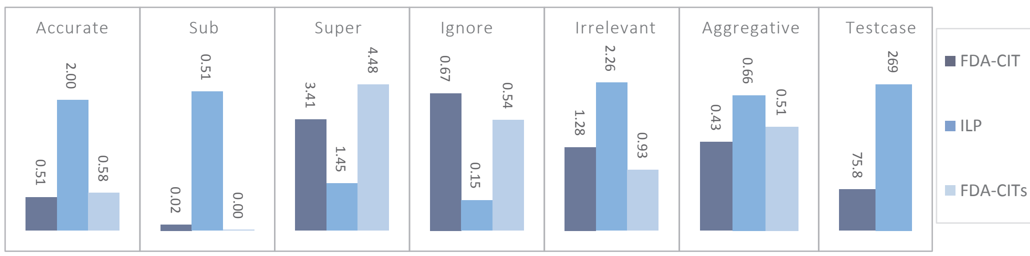
Note that in Figure 7, the results for each metric is the average evaluation for all the results of the experiments on the 15 testing objects in Table XXIV. The raw results for each testing object are listed in Table XXVI in the appendix . The raw data is organised the same way as Table XXV, except that we added a column t which indicates the strength of the covering array generated for this experiment.

With respect to the relationships between the results and the degree t of the covering arrays, we have the following observations:

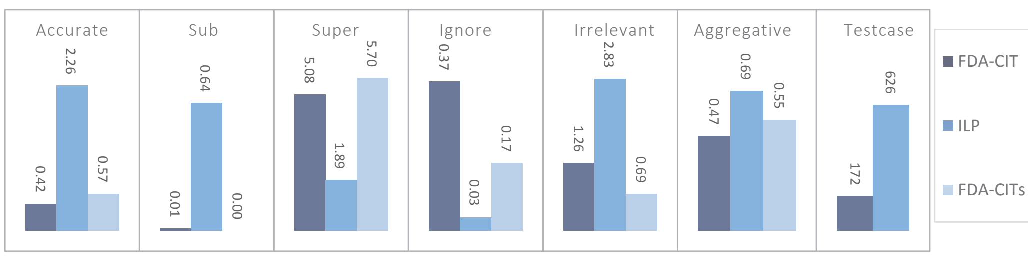
First, for every metric in our study, the order of the performance of each approach is stable against the change of degree t. Take for example the metric *accurate number*. No matter what t is (2 , 3 or 4), *ILP* always obtained the most schemas that are identical to the actual MFS, and then is *FDA-CITs*, and the last is *FDA-CIT*. This observation



(a) Result for the 2-way covering array



(b) Result for the 3-way covering array



(c) Result of the 4-way covering array

Fig. 7. Three approaches augmented with the replacing strategy

indicates that the difference between the performance of these approaches is not dependent on the characteristics of the covering array, but instead on the approaches themselves.

Second, with increasing t, the overall performance of each approach is improved. For example, the score of the aggregative metric of *ILP* is 0.55, 0.66 and 0.69, respectively, for t equals to 2, 3 and 4. The improvement is mainly because with increasing t, the number of test cases also increased. Based on this, the approach will observe more failing test cases (See area B in Figure 3), so that we can get the schemas more close to the actual MFS.

Third, for different approaches in our study, the effect of the change of t on the scores of other metrics varies. Specifically, for *ILP*, with increasing t, metrics *accurate number, sub number, super number, irrelevant number* also increase, while metric *ignore number* decreases. This is mainly because *ILP* is based on *FIC_BS* [Zhang and Zhang 2011], which works on single failing test case. As we all know, when t increases, the number of test cases also increases. Then when applying our approach, more different schemas may be identified from those additional failing test cases, so the number of accurate MFS, sub-schemas of the actual MFS, super-schemas of the actual MFS, and schemas that are irrelevant to the actual MFS will increase. Furthermore, some actual MFS that had been ignored before may be obtained. For *FDA-CIT* and *FDA-CITs*, however, we find that *sub number, irrelevant number* decrease with increasing t. We believe this result is due to the use of the classification tree method. A typical classification tree works by partitioning the test cases according to some aspects. Here, the aspect is the parameter value of the SUT. And one path (conjunction of nodes from the root to one leaf in the tree) in this tree is deemed as an MFS. So when test cases increase, the classification may need more nodes to classify the test cases. This induces the so-called 'over fitting' problem. As a result, the schemas identified by *FDA-CIT* and *FDA-CITs* tend to be the super schemas of the actual MFS, leading to a decrease of *sub number* and *irrelevant number*.

Other observations include:

First, when compared with the original approach *FDA-CIT*, *FDA-CITs* has obvious advantages at almost all the metrics except for *super number*. In detail, *FDA-CITs* obtained more schemas that are identical to the actual MFS (*accurate number*), less schemas that are the sub schemas of actual MFS (*sub number*), and lower scores for the two negative metrics (*ignored number* and *irrelevant number*). At last, the schemas identified by *FDA-CITs* showed an overall higher quality than that of the original *FDA-CIT* (*aggregative metric*). We have discussed previously that *FDA-CIT* tends to identify super-schemas of actual MFS when the test cases increase. So for metric *super number*, it is no surprise that *FDA-CITs* identified more super schemas of actual MFS than *FDA-CIT*, because it used the test cases generated by *ILP*, which were more than that of the original FDA-CIT. The difference between the overall performance of *FDA-CIT* and *FDA-CITs* is also expected. In fact, this result is consistent with our previous observation that when t increases, the overall performance for each approach also increases.

Second, in terms of the quality of the MFS identified, we can clearly find that our approach performed better than the other approaches. This is mainly manifested in that our approach obtained more accurate schemas and identified less irrelevant ones. We believe this gap is mainly caused by the FCI approach. Because for *ILP* and *FDA-CITs*, the test cases used to identify the MFS are the same. The only difference is how they utilize them to identify MFS. However, this result does not mean that FIC_BS is better than the classification tree method under all conditions. The classification tree method has its own advantage, i.e., it does not need to generate additional test cases, and as a result, *FDA-CIT* generated less test cases than that of *ILP*.

In fact, another reason why our approach generated more test cases is that the F-CI approach, i.e., FIC_BS , works on single test case, so when there are many failing test cases in the covering array, we need to repeatedly use our approach to identify the MFS for each failing test case. This process may produce many redundant test cases, because many failing test cases contain the same MFS, and when we have already identified the MFS in one test case, there is no need to identify it again in other failing test cases. Jieli [Li et al. 2012] introduced a method that utilizes the previous generated test cases to reduce such redundance. Here we did not use this technique to simplify our experiment. We believe if we utilize the MFS that are already identified in previous iteration, the overall number of test cases will decrease.

Above all, we can conclude three points in this experiment, which provide answer to **Q4**:

1)The degree t of the covering array does not determine the order of the performance of different approaches, but for each approach, the bigger the t is, the better its performance will be.

2)When taking the test cases generated by our approach *ILP*, *FDA-CITs* performed better than the original *FDA-CIT* approach.

3)Considering the quality of the MFS each approach identified, *ILP* performed better than the other two approaches, although it needed more test cases.

Based on these observations, a recommendation for selecting masking handling techniques in practice is that to get a more precise identification of the MFS in the SUT, *ILP* is preferred, and for a small size of test cases, *FDA-CIT* may be a better choice.

6.5. Threats to validity

6.5.1. internal threats. There are two threats to internal validity. First, the characteristics of the actual MFS in the SUT can affect the FCI results. This is because the magnitude and location of the MFS can make the FCI approaches generate different test cases. And as a result, it can make the observed failing test cases and predicted failing test cases different (See Figure 3). In the worst case, the FCI approach happens to identify the exact actual MFS, and in that condition our test case replacing strategy is of no use. In this paper, we used 15 testing objects, in which 5 are real software systems with real faults and 10 synthetic ones with injected faults. To reduce the influence caused by different characteristics of the MFS, we need to build more testing objects and injected more different types of faults for a more comprehensive study of our approach.

The second threat is that we just applied our test case replacing strategy on one FCI approach – FIC_BS [Zhang and Zhang 2011]. Although we believe the test case replacing strategy can also improve the quality of the identified MFS for other FCI approaches when the testing object is suffering from masking effects, the extent to which their results can be refined may vary for different FCI approaches. For example, for FIC_BS [Zhang and Zhang 2011] used in this paper, there are about $(v-1)$ to $(v-1)^{k-1}$ candidate test cases that can be replaced (See Table XIX) when one test case triggered other failures, while for OFOT [Nie and Leung 2011a], we only have $(v-1)$ candidates. As a result, FIC_BS can have a higher chance than OFOT to find a satisfied test case. So to learn the difference between the improvement of different FCI approaches when applying our test case replacing strategy, we need to try more FCI approaches in the future.

6.5.2. external threats. One threat to external validity comes from the real software we used. In this paper we have only surveyed two types of open-source software with five different versions, of which the program scale is medium-sized. This may impact the generality of our results.

Another important threat is that our approach is based on the assumption that different errors in the software can be easily distinguished by information such as exception traces, state conditions, or the like. If we cannot directly distinguish them, our approach does not work. In such case, one potential solution is to use the clustering techniques to classify the failures according to available information [Zheng et al. 2006; Jones et al. 2007; Podgurski et al. 2003]. If we cannot classify them because we do not have enough information (e.g., the black box testing) or it is too costly, we believe the only approach is to take the *regarded as one failure* strategy. With this strategy, we must aware that the MFS identified are likely to be sub-schemas or irrelevant schemas of the actual MFS.

The third threat comes from the possible masking relationships between multiple failures in the real software. In this paper, we just focus on the condition that the masking effects are transitive, i.e., if failure A masks B, failure B masks C, then failure A must mask the failure C. In practice, the relationships between multiple failures may be more complicated. One possible scenario is that two failures are in a loop, for which the two failures can even mask each other in a particular condition. Such a case will make our formal analysis invalid and will significantly complicate the relationships between schemas and their corresponding test cases. A new formal model should be proposed to handle that type of masking effects.

7. RELATED WORKS

Shi and Nie presented an approach for failure revealing and failure diagnosis in CT [Shi et al. 2005], which first tests the SUT with a covering array, then reduces the value schemas contained in the failing test case by eliminating those appearing in the passing test cases. If the failure-causing schema is found in the reduced schema set, failure diagnosis is completed with the identification of the specific input values which caused the failure; otherwise, a further test suite based on SOFOT is developed for each failing test cases, and the schema set is then further reduced, until no more faults are found or the fault is located. Based on this work, Wang proposed an AIFL approach which extended the SOFOT process by adaptively mutating factors in the original failing test cases in each iteration to characterize failure-inducing interactions [Wang et al. 2010].

Nie et al. introduced the notion of Minimal Failure-causing Schema(MFS) and proposed the OFOT approach which is an extension of SOFOT that can isolate the MFS in SUT [Nie and Leung 2011a]. This approach mutates one value with different values for that parameter, hence generating a group of additional test cases each time to be executed. Compared with SOFOT, this approach strengthens the validation of the factor under analysis and can also detect the newly imported faulty interactions.

Delta debugging [Zeller and Hildebrandt 2002] is an adaptive divide-and-conquer approach to locate interaction failure. It is very efficient and has been applied to real software environment. Zhang et al. also proposed a similar approach that can efficiently identify the failure-inducing interactions that has no overlapped part [Zhang and Zhang 2011]. Later, Li improved the delta-debugging based approach by exploiting useful information in the executed covering array [Li et al. 2012].

Colbourn and McClary proposed a non-adaptive method [Colbourn and McClary 2008]. Their approach extends a covering array to the locating array to detect and locate interaction failures. C. Martinez proposed two adaptive algorithms. The first one requires safe value as the assumption and the second one removes this assumption when the number of values of each parameter is equal to 2 [Martínez et al. 2008; 2009]. Their algorithms focus on identifying faulty tuples that have no more than 2 parameters.

Ghandehari et al. defined the suspiciousness of tuple and suspiciousness of the environment of a tuple [Ghandehari et al. 2012]. Based on this, they ranked the possible tuples and generated the test configurations. They further utilized the test cases generated from the inducing interaction to locate the fault [Ghandehari et al. 2013].

Yilmaz proposed a machine learning method to identify inducing interactions from a combinatorial testing set [Yilmaz et al. 2006]. They constructed a classification tree to analyze the covering arrays and detect potential faulty interactions. Beside this, Fouché [Fouché et al. 2009] and Shakya [Shakya et al. 2012] made some improvements in identifying failure-inducing interactions based on Yilmaz's work.

Our previous work [Niu et al. 2013] proposed an approach that utilizes the tuple relationship tree to isolate the failure-inducing interactions in a failing test case. One novelty of this approach is that it can identify the overlapped faulty interaction. This work also alleviates the problem of introducing new failure-inducing interactions in additional test cases.

In addition to the studies that aim at identifying the failure-inducing interactions in test cases, there are others that focus on working around the masking effects.

Constraints handling become more and more popular in CT these years. A constraint is an invalid interaction that should not appear in the test case. It can be deemed as the masking effect which are known in prior [Yilmaz et al. 2014]. Cohen [Cohen et al. 2007a; 2007b; 2008] studied the impact of the constraints that render some generated test cases invalid in CT. They also proposed an approach that integrates the incremental SAT solver with the covering arrays generating algorithm to avoid those invalid interactions. Further study was conducted [Petke et al. 2013] to show that with consideration of constrains, higher-strength covering arrays with early failure detection are practical.

Besides, there are additional works that aim to study the impacts of constraints for CT [Garvin et al. 2011; Bryce and Colbourn 2006; Calvagna and Gargantini 2008; Grindal et al. 2006; Yilmaz 2013]. Among them, [Bryce and Colbourn 2006] distinguished the constraints into two types: *hard* and *soft*, which the former cannot be included in the test case, while the latter can be permitted, but not desirable. [Grindal et al. 2006] comprehensively compared the performance of four strategies at handling the constraints in the covering array. [Calvagna and Gargantini 2008] proposed an heuristic strategy to handle the constraints. It can support an ad-hoc inclusion or exclusion of interactions such that the user can customize output of the covering array. [Garvin et al. 2011] refined the simulated annealing algorithm to efficiently construct the covering array with considering the constraints. [Yilmaz 2013] introduced the test case-specific constraints; differing from the system-wide constraints, this constraint can only be triggered in some specific test cases.

Chen et al. addressed the issues of shielding parameters in combinatorial testing and proposed the Mixed Covering Array with Shielding Parameters (MCAS) to solve the problem caused by shielding parameters [Chen et al. 2010]. The shielding parameters can disable some parameter values to expose additional interaction errors, which can be regarded as a special case of masking effects.

Dumlu and Yilmaz proposed a feedback-driven approach to work around the masking effects [Dumlu et al. 2011]. Specifically, they first used classification tree to classify the possible failure-inducing interactions and eliminate them. Then they generate new test cases to detect possible masked interaction in the next iteration. They further extended their work [Yilmaz et al. 2014] by proposing a multiple-class CTA approach to distinguish failures in SUT. In addition, they empirically studied the impacts of masking effects on both ternary-class and multiple-class CTA approaches.

These works can be categorized into 3 groups according to their relationships with our work. First, the works that aim to identifying the MFS in the SUT. Our work also focuses on identifying the MFS, but instead of single failure, our work considers the impacts of multiple failures on the FCI approaches, and based on this, a test case replacement strategy is proposed that can assist these FCI approaches in reducing the negative effects. Second, the works that aim to handling the constraints. As discussed before the constraints can be deemed as a special masking effects. Our work differs from them in that the masking effects handled in this paper are those that can be dynamically triggered; that is, we did not know them in prior. Another difference between our work with these constraints handling works is that their target is to avoid the constraints when generating covering array. However, our work aims to removing the masking effects of the FCI approaches. Last, the work that is most similar to our work [Yilmaz et al. 2014], which also considered the masking effects that are dynamically appeared in test cases. But different from our work, it mainly focused on reducing the masking effects in the covering array, so that the covering array can support a comprehensive validation of all the t-degree schemas. The approach used to reduce this negative effect is to use the FCI approach to identify the schemas that can trigger this effect in each iteration. Our approach, however, aims to handling the masking effects that happened in these FCI approaches themselves, and our approach alleviates the masking effects by augmenting the FCI approaches with a test case replacement strategy.

8. CONCLUSIONS

Masking effects of multiple failures in SUT can bias the results of traditional failure-inducing interactions identifying approaches. In this paper, we formally analysed the impact of masking effects on FCI approaches and showed that the two traditional strategies, i.e., *regarded as one fault* and *distinguishing failures*, are both inefficient in handling such impact. We further presented a test case replacement strategy for FCI approaches to alleviate such impact.

In our empirical studies, we extended FIC_BS [Zhang and Zhang 2011] with our strategy. The comparison between our approach and traditional approaches was performed on several open-source software. The results indicated our strategy assists the traditional FCI approach in achieving better performance when facing masking effects in SUT. We also empirically evaluated the efficiency of the test case searching component by comparing it with the random searching based FCI approach. The results showed that the ILP-based test case searching method can perform more efficiently. Last, we compared our approach with existing technique for handling masking effects – FDA-CIT [Yilmaz et al. 2014], and observed that our approach achieved a more precise result which can better support debugging, though our approach required more test cases than FDA-CIT.

As for the future work, we need to do more empirical studies to make our conclusions more general. Our current experiments focus on medium-sized software. We would like to extend our approach to more complicated, large-scaled testing scenarios. Another promising work in the future is to integrate the white-box testing technique into the FCI approaches. We believe gaining insight into source code can help figure out the relationships between multiple failures, and hence facilitate the FCI approaches obtaining more accurate results. And last, because the extent to which the FCI suffers from masking effects varies with different algorithms, combining these different FCI approaches would be desired in the future to further improve identifying MFS for multiple failures.

REFERENCES

James Bach and Patrick Schroeder. 2004. Pairwise testing: A best practice that isnt. In *Proceedings of 22nd Pacific Northwest Software Quality Conference*. Citeseer, 180–196.

Michel Berkelaar, Kjell Eikland, and Peter Notebaert. 2004. lp_solve 5.5, Open source (Mixed-Integer) Linear Programming system. Software. (May 1 2004). http://lpsolve.sourceforge.net/5.5/ Last accessed Dec, 18 2009.

Renée C Bryce and Charles J Colbourn. 2006. Prioritized interaction testing for pair-wise coverage with seeding and constraints. *Information and Software Technology* 48, 10 (2006), 960–970.

Renée C Bryce, Charles J Colbourn, and Myra B Cohen. 2005. A framework of greedy methods for constructing interaction test suites. In *Proceedings of the 27th international conference on Software engineering*. ACM, 146–155.

Andrea Calvagna and Angelo Gargantini. 2008. A logic-based approach to combinatorial testing with constraints. In *Tests and proofs*. Springer, 66–83.

Baiqiang Chen, Jun Yan, and Jian Zhang. 2010. Combinatorial testing with shielding parameters. In *Software Engineering Conference (APSEC), 2010 17th Asia Pacific*. IEEE, 280–289.

David M. Cohen, Siddhartha R. Dalal, Michael L Fredman, and Gardner C. Patton. 1997. The AETG system: An approach to testing based on combinatorial design. *Software Engineering, IEEE Transactions on* 23, 7 (1997), 437–444.

Myra B Cohen, Charles J Colbourn, and Alan CH Ling. 2003. Augmenting simulated annealing to build interaction test suites. In *Software Reliability Engineering, 2003. ISSRE 2003. 14th International Symposium on*. IEEE, 394–405.

Myra B Cohen, Matthew B Dwyer, and Jiangfan Shi. 2007a. Exploiting constraint solving history to construct interaction test suites. In *Testing: Academic and Industrial Conference Practice and Research Techniques-MUTATION, 2007.* IEEE, 121–132.

Myra B Cohen, Matthew B Dwyer, and Jiangfan Shi. 2007b. Interaction testing of highly-configurable systems in the presence of constraints. In *Proceedings of the 2007 international symposium on Software testing and analysis*. ACM, 129–139.

Myra B Cohen, Matthew B Dwyer, and Jiangfan Shi. 2008. Constructing interaction test suites for highly-configurable systems in the presence of constraints: A greedy approach. *Software Engineering, IEEE Transactions on* 34, 5 (2008), 633–650.

Myra B Cohen, Peter B Gibbons, Warwick B Mugridge, and Charles J Colbourn. 2003. Constructing test suites for interaction testing. In *Software Engineering, 2003. Proceedings. 25th International Conference on*. IEEE, 38–48.

Charles J Colbourn and Daniel W McClary. 2008. Locating and detecting arrays for interaction faults. *Journal of combinatorial optimization* 15, 1 (2008), 17–48.

Emine Dumlu, Cemal Yilmaz, Myra B Cohen, and Adam Porter. 2011. Feedback driven adaptive combinatorial testing. In *Proceedings of the 2011 International Symposium on Software Testing and Analysis*. ACM, 243–253.

Sandro Fouché, Myra B Cohen, and Adam Porter. 2009. Incremental covering array failure characterization in large configuration spaces. In *Proceedings of the eighteenth international symposium on Software testing and analysis*. ACM, 177–188.

Brady J Garvin, Myra B Cohen, and Matthew B Dwyer. 2011. Evaluating improvements to a meta-heuristic search for constrained interaction testing. *Empirical Software Engineering* 16, 1 (2011), 61–102.

Laleh Sh Ghandehari, Yu Lei, David Kung, Raghu Kacker, and Richard Kuhn. 2013. Fault localization based on failure-inducing combinations. In *Software Reliability Engineering (ISSRE), 2013 IEEE 24th International Symposium on*. IEEE, 168–177.

Laleh Shikh Gholamhossein Ghandehari, Yu Lei, Tao Xie, Richard Kuhn, and Raghu Kacker. 2012. Identifying failure-inducing combinations in a combinatorial test set. In *Software Testing, Verification and Validation (ICST), 2012 IEEE Fifth International Conference on*. IEEE, 370–379.

Mats Grindal, Jeff Offutt, and Jonas Mellin. 2006. Handling constraints in the input space when using combination strategies for software testing. (2006).

Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H Witten. 2009. The WEKA Data Mining Software: An Update; SIGKDD Explorations, Volume 11, Issue 1. (2009).

James A Jones, James F Bowring, and Mary Jean Harrold. 2007. Debugging in parallel. In *Proceedings of the 2007 international symposium on Software testing and analysis*. ACM, 16–26.

Yu Lei, Raghu Kacker, D Richard Kuhn, Vadim Okun, and James Lawrence. 2008. IPOG/IPOG-D: efficient test generation for multi-way combinatorial testing. *Software Testing, Verification and Reliability* 18, 3 (2008), 125–148.

Jie Li, Changhai Nie, and Yu Lei. 2012. Improved Delta Debugging Based on Combinatorial Testing. In *Quality Software (QSIC), 2012 12th International Conference on*. IEEE, 102–105.

Conrado Martínez, Lucia Moura, Daniel Panario, and Brett Stevens. 2008. Algorithms to locate errors using covering arrays. In *LATIN 2008: Theoretical Informatics*. Springer, 504–519.

Conrado Martínez, Lucia Moura, Daniel Panario, and Brett Stevens. 2009. Locating errors using ELAs, covering arrays, and adaptive testing algorithms. *SIAM Journal on Discrete Mathematics* 23, 4 (2009), 1776–1799.

Changhai Nie and Hareton Leung. 2011a. The minimal failure-causing schema of combinatorial testing. *ACM Transactions on Software Engineering and Methodology (TOSEM)* 20, 4 (2011), 15.

Changhai Nie and Hareton Leung. 2011b. A survey of combinatorial testing. *ACM Computing Surveys (CSUR)* 43, 2 (2011), 11.

Xintao Niu, Changhai Nie, Yu Lei, and Alvin TS Chan. 2013. Identifying Failure-Inducing Combinations Using Tuple Relationship. In *Software Testing, Verification and Validation Workshops (ICSTW), 2013 IEEE Sixth International Conference on*. IEEE, 271–280.

Justyna Petke, Shin Yoo, Myra B Cohen, and Mark Harman. 2013. Efficiency and early fault detection with lower and higher strength combinatorial interaction testing. In *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering*. ACM, 26–36.

Andy Podgurski, David Leon, Patrick Francis, Wes Masri, Melinda Minch, Jiayang Sun, and Bin Wang. 2003. Automated support for classifying software failure reports. In *Software Engineering, 2003. Proceedings. 25th International Conference on*. IEEE, 465–475.

Kiran Shakya, Tao Xie, Nuo Li, Yu Lei, Raghu Kacker, and Richard Kuhn. 2012. Isolating Failure-Inducing Combinations in Combinatorial Testing using Test Augmentation and Classification. In *Software Testing, Verification and Validation (ICST), 2012 IEEE Fifth International Conference on*. IEEE, 620–623.

Liang Shi, Changhai Nie, and Baowen Xu. 2005. A software debugging method based on pairwise testing. In *Computational Science–ICCS 2005*. Springer, 1088–1091.

Charles Song, Adam Porter, and Jeffrey S Foster. 2012. iTree: Efficiently discovering high-coverage configurations using interaction trees. In *Proceedings of the 2012 International Conference on Software Engineering*. IEEE Press, 903–913.

Ziyuan Wang, Baowen Xu, Lin Chen, and Lei Xu. 2010. Adaptive interaction fault location based on combinatorial testing. In *Quality Software (QSIC), 2010 10th International Conference on*. IEEE, 495–502.

Cemal Yilmaz. 2013. Test case-aware combinatorial interaction testing. *Software Engineering, IEEE Transactions on* 39, 5 (2013), 684–706.

Cemal Yilmaz, Myra B Cohen, and Adam A Porter. 2006. Covering arrays for efficient fault characterization in complex configuration spaces. *Software Engineering, IEEE Transactions on* 32, 1 (2006), 20–34.

C. Yilmaz, E. Dumlu, M.B. Cohen, and A. Porter. 2014. Reducing Masking Effects in CombinatorialInteraction Testing: A Feedback DrivenAdaptive Approach. *Software Engineering, IEEE Transactions on* 40, 1 (Jan 2014), 43–66.

Andreas Zeller and Ralf Hildebrandt. 2002. Simplifying and isolating failure-inducing input. *Software Engineering, IEEE Transactions on* 28, 2 (2002), 183–200.

Zhiqiang Zhang and Jian Zhang. 2011. Characterizing failure-causing parameter interactions by adaptive testing. In *Proceedings of the 2011 International Symposium on Software Testing and Analysis*. ACM, 331–341.

Alice X Zheng, Michael I Jordan, Ben Liblit, Mayur Naik, and Alex Aiken. 2006. Statistical debugging: simultaneous identification of multiple bugs. In *Proceedings of the 23rd international conference on Machine learning*. ACM, 1105–1112.

Online Appendix to: Identifying minimal failure-causing schemas in the presence of multiple failures

XINTAO NIU and CHANGHAI NIE, State Key Laboratory for Novel Software Technology, Nanjing University
JEFF Y. LEI, Department of Computer Science and Engineering, The University of Texas at Arlington
Hareton Leung and ALVIN CHAN, Department of computing, Hong Kong Polytechnic University

A. THE DETAIL OF THE EXPERIMENTS

Table XXV. Result of the evaluation of each appraoch

Subject	accurate				sub				super				ignore				irrelevant				overall				test cases			
	O [1]	D [2]	I [3]	R [4]	O	D	I	R	O	D	I	R	O	D	I	R	O	D	I	R	O	D	I	R	O	D	I	R
HSQLDB 2cr8	2	3	5	5	3	2	0	2	0	2	4	5	0(2.04)	0(3.91)	0(4.0)	0(4.1)	0	2	0	34	0.57	0.65	0.88	0.23	8.125	11.92	17	17.72
HSQLDB 2.2.5	2	2	2	2	1	0	0	0	0	1	1	1	0(0.83)	0(1.0)	0(1.0)	0(1.0)	7	0	0	0	0.25	0.83	0.83	0.83	8.67	7.67	10.17	11.3
HSQLDB 2.2.9	2	3	2	2	3	1	1	1	0	4	1	1	0(1.33)	0(2.83)	0(2.56)	0(2.56)	4	0	0	0	0.4	0.74	0.8	0.8	9.167	8.61	11.72	13.14
JFlex 1.4.1	2	2	2	2	0	0	0	0	0	1	0	0	0(0.75)	0(1.0)	0(1.0)	0(1.0)	25	0	0	0	0.07	0.83	1	1	23.5	6.5	8	9.68
JFlex 1.4.2	2	2	2	2	1	0	0	0	0	1	1	1	0(0.67)	0(1.0)	0(1.0)	0(1.0)	25	0	0	0	0.09	0.83	0.83	0.83	20.5	9	11.67	13.12
synthez 1	2	1	1	1	2	0	0	0	0	2	2	2	0(1.0)	0(2.0)	0(2.0)	0(2.0)	15	17	17	17	0.19	0.11	0.11	0.11	16.5	18	41.75	41.75
synthez 2	3	3	3	3	10	0	0	0	0	10	6	6	0(1.96)	0(2.0)	0(2.0)	0(2.0)	1	0	0	0	0.54	0.76	0.8	0.8	11.19	14.12	16.96	17.08
synthez 3	4	3	3	3	4	2	2	2	0	5	5	5	0(2.08)	0(2.84)	0(2.84)	0(2.84)	13	9	9	9	0.28	0.34	0.34	0.34	12.73	9.46	14.18	14.44
synthez 4	3	3	3	3	10	3	2	3	0	6	5	5	0(2.6)	0(2.8)	0(2.9)	0(2.85)	9	9	9	9	0.35	0.4	0.39	0.39	9.91	13.02	18.55	18.45
synthez 5	2	2	2	2	4	0	0	0	0	2	1	1	0(1.02)	0(1.0)	0(1.0)	0(1.0)	1	0	0	0	0.65	0.88	0.92	0.92	13.04	13.7	14.77	14.84
synthez 6	2	3	3	3	15	4	4	4	0	8	8	8	0(1.99)	0(3.72)	0(3.72)	0(3.72)	8	10	10	10	0.38	0.36	0.36	0.36	14.91	11.75	15.37	15.71
synthez 7	3	3	3	3	10	0	0	0	0	6	6	6	0(2.04)	0(2.0)	0(2.0)	0(2.0)	5	0	0	0	0.39	0.83	0.83	0.83	12.77	14.59	16.44	16.53
synthez 8	2	2	2	2	4	0	0	9	0	4	3	3	0(1.05)	0(1.0)	0(1.0)	0(1.0)	3	0	0	0	0.56	0.9	0.91	0.91	24.45	25.25	26.27	26.37
synthez 9	2	1	1	1	1	0	0	0	0	1	1	1	0(0.8)	0(1.0)	0(1.0)	0(1.0)	0	0	0	0	0.75	0.83	0.83	0.83	6.8	8	9	9
synthez 10	0	1	1	1	3	1	1	1	0	0	0	0	0(1.0)	0(1.46)	0(1.31)	0(1.31)	0	2	1	1	0.5	0.47	0.58	0.58	9.08	11	15.38	15.53

[1] O denotes the strategy *regarded as one failure*.
[2] D denotes the strategy *distinguishing failures*.
[3] I denotes the replacement strategy based on ILP searching.
[4] R denotes the replacement strategy based on randomly searching.

DOI:http://dx.doi.org/10.1145/0000000.0000000

Table XXVI. Comparison with FDA-CIT

Subject		**accurate**			**sub**			**super**			**ignore**			**irrelevant**			**overall**			**test cases**		
	t	F [1]	I [2]	Fs [3]	F	I	Fs	F	I	F	F	I	Fs	F	I	Fs	F	I	Fs	F	I	Fs
HSQL2cr8	2	0.17	2.27	1.57	0.57	0	0	0.17	0.4	2.17	3.87	2.3	2	2.53	0	1.97	0.12	0.51	0.39	23.6	70.1	70.1
	3	1.47	3.67	1	0	0	0	4.67	2	6.07	0.63	0.3	0.17	3	0	1.47	0.51	0.87	0.6	76.6	241.8	241.8
	4	0.83	4.8	1	0	0	0	9.03	3.37	8	0	0	0	0.97	0	0	0.65	0.9	0.71	183.5	606.6	606.6
HSQL2.2.5	2	1	1.97	0.37	0	0	0	2.4	0.73	3.8	0.4	0	0	1.4	0	0.1	0.38	0.87	0.56	26.7	68.8	68.8
	3	0	2	0.4	0	0	0	5	1	3.8	0	0	0	0	0	0	0.52	0.83	0.56	67	202.4	202.4
	4	0	2	0.33	0	0	0	5	1	4	0	0	0	0	0	0	0.53	0.83	0.56	130.1	503.3	503.3
HSQL2.2.9	2	0.9	1.77	0.9	0	0.77	0	1.47	0.47	6.8	1.93	0.53	0	2.37	0	0.2	0.28	0.72	0.58	29.2	78.3	78.3
	3	1	2	0.83	0	1	0	5.13	0.93	7.1	0.2	0	0	0.1	0	0	0.61	0.8	0.61	72.8	221.7	221.7
	4	1	2	1	0	1	0	5.87	1	6.7	0	0	0	0	0	0	0.64	0.8	0.62	129.8	560.3	560.3
JFlex 1.4.1	2	0	2	0	0	0	0	4.03	0	4	0	0	0	0	0	0	0.49	1	0.5	30.5	87.3	87.3
	3	0	2	0	0	0	0	4	0	0	0	0	4	0	0	0	0.5	1	0.5	73.4	269.2	269.2
	4	0	2	0	0	0	0	4	0	0	0	0	0	0	0	0	0.5	1	0.5	190.6	724.7	724.7
JFlex 1.4.2	2	0.3	1.97	0.93	0	0	0	3.6	1	2.16	0.03	0	0	0.63	0	0	0.5	0.83	0.62	34.3	106.9	106.9
	3	0	2	0.97	0	0	0	5	1	2.1	0	0	0	0.03	0	0	0.52	0.83	0.61	72.3	305.7	305.7
	4	0	2	1	0	0	0	5	1	2	0	0	0	0	0	0	0.53	0.83	0.61	186.8	836.9	836.9
synthez 1	2	0.97	1	1	0	0	0	1.7	1.93	2	0	0.07	0	0.33	14.3	0	0.66	0.13	0.78	40.3	342.87	342.87
	3	1	1	1	0	0	0	2	2	2	0	0	0	0	16.73	0	0.78	0.12	0.78	93.4	809.1	809.1
	4	1	1	1	0	0	0	2	2	2	0	0	0	0	17	0	0.78	0.12	0.78	218.8	1532.8	1532.8
synthez 2	2	0.17	1.3	0.73	0.37	0	0	0	0.4	2.37	2.27	1.2	1.03	1.37	0	1.2	0.11	0.52	0.4	19.77	54.4	54.4
	3	0.73	2.23	0.5	0	0	0	1.9	1.3	7.1	1.2	0.43	0.53	2.2	0	1.33	0.36	0.82	0.52	59.5	171.5	171.5
	4	0.63	2.97	0.1	0	0	0	5.3	2.33	16.1	0.53	0	0	2.6	0	1	0.44	0.89	0.54	152.7	415.1	415.1
synthez 3	2	0.43	2.97	0.73	0	0.93	0	4.3	1.73	5.3	0.47	0.17	0.5	1.03	3.77	1.13	0.37	0.46	0.37	48.6	138.7	138.7
	3	0.2	3	0.87	0	1.57	0	7.2	3.67	6.57	0.07	0	0	0.83	6.77	0.07	0.38	0.38	0.44	106.3	315.3	315.3
	4	0.03	3	1	0	1.97	0	10.4	3	6	0	0	0	0.43	8.56	0	0.38	0.34	0.45	147.9	565.7	565.7
synthez 4	2	0.3	2.3	0.33	0.07	0.63	0	2.63	1.97	7.7	1.93	0.63	0.4	3.4	1.4	1.97	0.24	0.6	0.44	42.7	142.2	142.2
	3	0.37	2.97	0.07	0	1.26	0	6.5	3.53	10.97	0.83	0.07	0	2.5	3.43	1.03	0.39	0.54	0.51	86.5	373.2	373.2
	4	0.07	3	0	0	1.77	0	11.7	4.67	11.4	0	0	0	1.33	6.73	0.03	0.48	0.44	0.55	202.2	899.7	899.7
synthez 5	2	0.2	1.2	0.8	0.3	0	0	0.1	0.03	0.83	1.4	0.77	0.97	0.7	0	1	0.2	0.59	0.4	21.9	46.9	46.9
	3	0.87	1.4	0.53	0	0	0	0.5	0.23	3.03	1	0.6	0.77	0.37	0	1.63	0.46	0.71	0.43	76.9	150.3	150.3
	4	0.7	1.9	0.37	0	0	0	1.77	0.33	6.5	0.9	0.1	0.03	1.87	0	2.03	0.34	0.92	0.54	232.9	433.2	433.2
synthez 6	2	0.23	2.63	0.17	0.2	2	0	2.93	1.63	9.63	2.6	0.5	0.4	3.03	3.7	2	0.19	0.42	0.37	45.7	132.6	132.6
	3	0.1	3	0.1	0	2.83	0	7.4	3.83	12.5	1.2	0.17	0.03	2.3	6.5	0.67	0.31	0.38	0.43	99.5	338.9	338.9
	4	0	3	0	0	3.8	0	10.2	6.03	14.5	0.47	0	0	1.8	9.1	0.03	0.37	0.36	0.44	152.6	781.9	781.9
synthez 7	2	0.13	1.43	0.83	0.23	0	0	0.1	0.63	1.4	2.53	1.03	0.93	1.93	0	1.97	0.09	0.61	0.38	20.3	58.8	58.8
	3	0.87	2.17	0.93	0	0	0	0.43	1.23	2.97	1.77	0.17	0.13	3.2	0	2.87	0.2	0.88	0.44	52.6	164.7	164.7
	4	1	2.87	1	0	0	0	3.23	2.53	4.6	0.27	0	0	4.5	0	2.27	0.35	0.9	0.51	145.3	413.1	413.1
synthez 8	2	0	0.2	0.17	0.03	0	0	0	0	0.03	0.3	0.13	0.13	0.17	0	0.3	0.01	0.1	0.05	16.1	45.2	45.2
	3	0	0.6	0.5	0.1	0	0	0	0	0.03	0.97	0.47	0.53	0.63	0	0.87	0.02	0.3	0.17	43.1	64.3	64.3
	4	0	1.33	0.8	0.1	0	0	0	0.07	0.67	1.53	0.4	0.5	1.4	0	0.93	0.04	0.67	0.41	109.3	145.6	145.6
synthez 9	2	1	1	1	0	0	0	0.46	0.6	0.77	0.53	0	0.23	0.63	0.6	0.67	0.54	0.7	0.6	36.2	43.4	43.4
	3	1	1	1	0	0	0	1	1	1	0	0	0	0	0	0	0.83	0.83	0.83	84.3	145	145
	4	1	1	1	0	0	0	1	1	1	0	0	0	0	0	0	0.83	0.83	0.83	188	291.6	291.6
synthez10	2	0	0.63	0	0.6	1	0.2	0	0	0.83	1.8	0.37	1.9	1.5	0.3	3.97	0.23	0.61	0.17	23.4	84.9	84.9
	3	0.07	0.97	0	0.23	1	0.03	0.36	0	1.9	2.23	0.03	1.97	4.03	0.53	3.97	0.13	0.66	0.2	73.4	263.2	263.2
	4	0	1	0	0.07	1	0	1.7	0	2	1.87	0	2	4.03	1	4	0.21	0.58	0.2	202.2	685.9	685.9

[1] F is for the FDA-CIT approach.
[2] I is for the our approach with replacement strategy based on ILP searching.
[3] Fs is for the FDA-CITs approach.