# Enhance Combinatorial Testing With Metamorphic Relations

Xintao Niu, *Member, IEEE*, Yanjie Sun, Huayao Wu, *Member, IEEE*, Gang Li, Changhai Nie, *Member, IEEE*, Lei Yu, *Member, IEEE*, and Xiaoyin Wang, *Member, IEEE*

**Abstract**—Due to the effectiveness and efficiency in detecting defects caused by interactions of multiple factors, Combinatorial Testing (CT) has received considerable scholarly attention in the last decades. Despite numerous practical test case generation techniques being developed, there remains a paucity of studies addressing the automated oracle generation problem, which holds back the overall automation of CT. As a consequence, much human intervention is inevitable, which is time-consuming and error-prone. This costly manual task also restricts the application of higher testing strength, inhibiting the full exploitation of CT in industrial practice. To bridge the gap between test designs and fully automated test flows, and to extend the applicability of CT, this paper presents a novel CT methodology, named COMER, to enhance the traditional CT by accounting for Metamorphic Relations (MRs). COMER puts a high priority on generating pairs of test cases which match the input rules of MRs, i.e., the *Metamorphic Group (MG)*, such that the correctness can be automatically determined by verifying whether the outputs of these test cases violate their MRs. As a result, COMER can not only satisfy the t-way coverage as what CT does, but also automatically check as many test oracle violations as possible. Several empirical studies conducted on 31 real-world software projects have shown that COMER increased the number of *metamorphic groups* by an average factor of 75.9 and also increased the failure detection rate by an average factor of 11.3, when compared with CT, while the overall number of test cases generated by COMER barely increased.

**Index Terms**—Combinatorial testing, automated oracle, metamorphic relation, software testing

## 1 Introduction

Combinatorial Testing (CT), or Combinatorial Interaction Testing (CIT) [1], [2], has been extensively studied in the last decades, due to its effectiveness and efficiency in detecting failures caused by interactions of factors. The advantage of CT over other testing methods on testing space sampling is the utilization of a mathematical object, named Covering Array (CA), which is an elaborate data table that covers every possible $t$-size interaction of factors at least once. Enormous research efforts have been dedicated to generating covering arrays, yielding various practical methods and tools [3], [4], [5], [6], [7], [8], [9], [10] which greatly simplify the CA-based test case generation.

Despite many studies focusing on test case generation, the verification of the correctness of testing outputs, i.e., the oracle problem, is seldom discussed [11] in the CT community. The lack of support for test oracle automation leaves a gap between the test design and overall automated test flow [12], resulting in the intensive human effort required to check the system's behavior for all the generated test cases by CT. Therefore, the overall actual cost of CT is significantly more expensive than originally expected. To make this point more clear, we investigated several CT studies on industrial practice and summarized their numbers of generated test cases (Column "size") and additional compromises that were made to reduce the cost for giving oracles (Column "Additional compromises"), respectively, in Table 1. It is observed that all these studies suffered from the labor-intensive human oracle problem, and hence had to decrease the testing strength and reduce the number of modeling parameters (values), such that the total number of test cases to be generated could be reduced. However, these compromises apparently restrict the full exploitation of CT in practice. Particularly, considering the expensive human oracle cost, the work [13] adopted a 2-way covering array and merged some of the input parameters, which resulted in one fault being omitted (could be found by 3-way covering arrays).

Apart from the error-pone and labor-intensive manual-based oracle, other types of oracles [36], [37] were adopted in CT, including so-called implicit test oracles [12] which can be directly observed by the behaviors of the SUT, like crashes [2], exceptions [19], memory leaks [20], buffer overflow [22]. When the full specification of the SUT is available,

- *Xintao Niu, Yanjie Sun, Huayao Wu, Gang Li, and Changhai Nie are with the State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing 210023, China. E-mail: {niuxintao, hywu, changhainie}@nju.edu.cn, yanjiesun@smail.nju.edu.cn, lig@smaill.nju.edu.cn.*
- *Lei Yu is with the Department of Computer Science and Engineering, The University of Texas at Arlington, Arlington, TX 76019 USA. E-mail: ylei@cse.uta.edu.*
- *Xiaoyin Wang is with the Department of Computer Science, The University of Texas at San Antonio, San Antonio, TX 78249 USA. E-mail: xiaoyin.wang@utsa.edu.*

*Manuscript received 31 March 2021; revised 23 November 2021; accepted 25 November 2021. Date of publication 30 November 2021; date of current version 12 December 2022.*
*This work was supported in part by the National Key Research and Development Program of China under Grant 2018YFB1003800, in part by the National Natural Science Foundation of China under Grants 62102176, 61902174, and 62072226, in part by the Natural Science Foundation of Jiangsu Province under Grant BK20190291, and in part by Information Technology Laboratory, National Institute of Standards and Technology under Grant 70NANB18H207.*
*(Corresponding author: Changhai Nie.)*
*Recommended for acceptance by L. Mariani.*
*Digital Object Identifier no. 10.1109/TSE.2021.3131548*

TABLE 1
The Impact of Oracle Problem on Industrial CT Application: All These Subjects can Only be Tested With Covering Arrays of 2-way Strength

| Subjects | Size | Additional compromises |
|---|---|---|
| DSCS [13] | 122 | merging parameters |
| NeoKylin [13] | 172 | merging parameters |
| Dict [14] | 127 | - |
| IoT [14] | 96 | merging multiple SUTs into one SUT |
| ETL [14] | 160 | excluding some parameter values |
| EMBDD [14] | 296 | converting multiple values to an abstract value |
| DFM [14] | 19 | - |
| Browser [15] | 162 | - |
| Mobile [16] | 42 | removing couple of parameter values |

TABLE 2
Summary of Test Oracle in CT Applications

| Reference | Type of Oracles | Problems |
|---|---|---|
| [13], [14], [15], [16], [17], [18] | manual-based oracles | error-pone and labor-intensive |
| [2], [19], [20], [21], [22], [23] | implicit test oracles | limited faulty types and testing scenarios |
| [24], [25], [26], [27], [28], [29] | specified oracles | requiring of full specification and heavy reliance on the test expertise |
| [30], [31], [32], [33], [34], [35] | correct version as oracles | the existence of a correct version may not hold |

embedded assertions [24] or pre and post conditions [25] or a model-checker [26], [27] can also be utilized as the so-called specified oracles for the CT test cases. Another commonly used oracle in CT is the existence of a completely correct version [30], [31], [32], [35] of software, such that the correctness verification can be directly obtained through comparing the results between the currently analyzed version with the correct version. Although these studies have reduced the tedious manual work for the oracle problem, their limited faulty types, needing for correct software version or full specification, and heavy reliance on the expertise of test engineers may prevent these studies from being applied to more general testing scenarios. Table 2 gives a brief summary of the test oracles used in some representative CT studies and their corresponding problems. Obviously, the automated oracle still remains a pressing challenge and a necessity in the CT community.

To bridge the gap between the static CT-based test design and the overall automated CT flow, this paper presents a novel CT methodology, named COMER[1], focusing on the enhancement of the traditional CT procedure with automated oracle supports. COMER is inspired by Metamorphic Testing (MT) [38], which has been recognized as an effective method for alleviating the oracle problem [39], [40]. The intuition behind MT is that, instead of painstakingly building the accurate mapping between each generated test case and the expected output, it is easier to determine the relations between the outputs of several corresponding test cases. As a typical example, when testing the mathematical sine function, it is hard to get the accurate expected value of $\sin(23)$, but it is obvious that $\sin(23)$ must be equal to $\sin(\pi - 23)$. Any violation of this rule directly results in a detected failure of this sine function. These related test cases are called *Metamorphic Groups (MGs)* in MT, and their relations are called *Metamorphic Relations (MRs)*.

The key idea behind COMER is to generate as many related test cases (*Metamorphic Groups*) as possible, while maintaining the t-way coverage as CT does. For this, COMER works in two phases. In the first phase, COMER generates the abstract test cases (a set of input parameter values) to form a traditional t-way covering array. In addition to the aim of covering t-way combinations, when generating an abstract test case, COMER seeks to match the currently generated test case with one or more previously generated test cases in terms of the specific metamorphic relations. In the second phase, COMER converts the abstract test case into concrete, executable test cases. In this stage, COMER will replace some abstract integer values with specific practically meaningful values. The same as the first phase, COMER takes into consideration the metamorphic relations such that these selected values would assist in producing more pairs of related concrete test cases.

To evaluate COMER, we built a repository consisting of 31 real-world software projects. For each software in this repository, we created the corresponding abstract input parameter model, the program that can convert the abstract test case to a concrete, executable test case, and automated test execution script. Besides, the metamorphic relation of each software is also given.

We compared COMER with three baseline approaches, namely, CT, RT, and tri-MCT, in which CT and RT are two existing established approaches without using metamorphic relations, while tri-MCT is a simple version of COMER proposed by us in this paper which uses metamorphic relations. We applied all these approaches to the software in this repository and made the following main observations from the results:

- Compared to traditional CT, both two approaches, i.e., tri-MCT and COMER, obtained higher (about 113% and 149%, respectively) fault detection rates. Further, COMER is superior to tri-MCT at the number of needed test cases, but with almost the same fault detection rate.
- RT performed the worst among all the approaches in terms of fault detection rate. Yet, on the other hand, with respect to the time cost, RT is the most efficient approach, followed by CT, COMER, and tri-MCT.
- Compared to the approach using optimal oracles, the fault detection rate of COMER is about 42%.
- Among many investigated factors, the difficulty that the output rules of an MR can be satisfied has a higher correlation than other factors with the fault detection results.

To summarize, the contributions of this paper include:

- *Idea & Implementation*. We made the first attempt in the CT community to take into account the metamorphic

1. Created by joining (CO)mbinatorial testing and (ME)tamorphic (R)elation.

relations while generating covering arrays. Based on this idea, we designed two approaches, i.e., tri-MCT and COMER, to enhance the covering array generation, such that the oracles of the test cases in this covering array can be checked by the satisfaction of metamorphic relations.
- *Optimization*. Instead of simply applying MT to generate metamorphic groups, COMER adopts a tightly coupled strategy to integrate the processes of $\tau$-way combinations covering and metamorphic relations matching. As a result, COMER can generate covering arrays whose sizes are competitive to those of normal covering arrays but that contain significantly more metamorphic groups.
- *Repository*. We built a software repository containing 31 real-world subjects, along with the CT model buildings, abstract test case converting programs, test case execution scripts, and corresponding MRs. All these subjects can be easily obtained and executed with detailed instructions, which is a useful resource for both CT and MT empirical studies. They are now available at https://github.com/syjya/Comer.
- *Study*. We conducted a series of empirical studies to evaluate our approaches in terms of effectiveness and efficiency. We also investigated several features of the metamorphic relations that affect the performance of COMER.

The remainder of the paper is organized as follows. Section 2 introduces the background of combinatorial testing and metamorphic testing. Section 3 describes the details of the approach COMER. Section 4 presents the design of the empirical studies, including the research questions. Section 5 gives and analyses the evaluation results. Section 6 presents the related works. At last, Section 7 concludes this paper.

## 2 Background

This section formally introduces some basic definitions of CT and MT, paving the way for the description of the approach given in Section 3.

### 2.1 Combinatorial Testing

Combinatorial Testing (CT), or Combinatorial Interaction Testing (CIT), is a systematic testing method which focuses on testing various interactions between the factors that can affect the behavior of the System Under Test (SUT). Suppose the SUT is affected by $n$ parameters, and each parameter $p_i$ has a set of possible values $V_i$, where $1 \leq i \leq n$.

**Definition 2.1 (Test case).** *A* test case *in CT for a SUT is an assignment of a value $v_i$ from $V_i$ to $p_i$ for each $i$ with $1 \leq i \leq n$, which can be denoted as the set { $(p_1, v_1)$, $(p_2, v_2)$, ..., $(p_n, v_n)$ }.*

In practice, a test case can be a set of option values for configuration testing, or a set of selected events for GUI testing, or a set of input values for unit function testing, etc.

**Definition 2.2 (Combination).** *A $\tau$-way* combination *in CT is an assignment of a value $v_{c_i}$ from $V_{c_i}$ to $p_{c_i}$ for each $i$ with $1 \leq i \leq \tau$, $1 \leq c_i \leq n$, and $c_i \neq c_j$ if $i \neq j$, which can be denoted as the set {$(p_{c_1}, v_{c_1})$, $(p_{c_2}, v_{c_2})$, ..., $(p_{c_\tau}, v_{c_\tau})$}.*

TABLE 3
A 2-way Covering Array for MS-Word

| *ID* | *Highlight* | *Status Bar* | *Bookmarks* | *Smart tags* |
|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 1 |
| 2 | 0 | 0 | 1 | 1 |
| 3 | 0 | 1 | 0 | 0 |
| 4 | 1 | 0 | 0 | 1 |
| 5 | 1 | 0 | 1 | 0 |

Note that a $\tau$-way combination can be regarded as a $\tau$-subset of a corresponding test case, while a test case itself can be regarded as a $n$-way combination. The combination is a key object in CT since many failures are caused by a subset of elements instead of all the possible elements in a test input.

**Definition 2.3 (Covering array).** *A $\tau$-way Covering Array (CA) is a set of test cases in which each possible $\tau$-way combination in the SUT is contained by at least one test case. A $\tau$-way covering array can be denoted as $MCA(N; \tau, (|V_1|, |V_2|, \ldots, |V_n|))$, where $N$ is the number of the test cases. When $|V_1| = |V_2| = \ldots = |V_n| = v$, then the notation of the covering array can be simplified as $CA(N; \tau, n, v)$.*

Covering array is the foundation of the test set design for CT, which is effectively equivalent to the exhaustive test set in terms of detecting software failures caused by $\tau$-way combinations. As an example [41], consider that four options 'Highlight', 'Status Bar', 'Bookmarks' and 'Smart tags' may affect the behavior of the MS-Word application. Then Table 3 presents a 2-way covering array consisting of 5 test cases. Note that in each test case, the integers 0 and 1 are abstract values which represent the concrete settings *off* and *on*, respectively, of corresponding options. It can be easily checked that various 2-way combinations for these four options are covered by this covering array.

In practice when applying CT, the dependencies or constraints may result in many invalid $\tau$-way combinations [42]. In such a case, additional efforts are needed to handle these constraints such that all the test cases of CA are valid, i.e., do not violate these constraints.

**Definition 2.4 (Constraint).** *A* Constraint *in CT is a function $c$ which maps $n$ parameter values of a test case into true or false. Specifically, given a test case { $(p_1, v_1)$, $(p_2, v_2)$, ..., $(p_n, v_n)$ }, the function $c(v_1, v_2, \ldots, v_n)$ returns* true *or* false, *where the* true *value indicates that the test case satisfies this constraint, and* false *otherwise.*

When given a set of constraints $C$ in the SUT, a test case { $(p_1, v_1)$, $(p_2, v_2)$, ..., $(p_n, v_n)$ } is **valid** if and only if it satisfies each constraint in $C$, i.e., $\forall c \in C, c(v_1, v_2, \ldots, v_n) = true$.

### 2.2 Metamorphic Testing

Metamorphic Testing (MT) is one promising approach to alleviate the oracle problem in software testing. It utilizes necessary properties of the SUT in terms of multiple inputs and their expected outputs.

**Definition 2.5 (Metamorphic Relation (MR)).** *Given a SUT, let $x_1, x_2, ..., x_n$ ($n \geq 2$) be a set of test cases and $o_1, o_2, ...,$*

*$o_n$ be the corresponding outputs for these $n$ test cases. A* Metamorphic Relation (MR) *$(r_{in}, r_{out})$ of this SUT is an implication relation which declares the fact if these $n$ test cases satisfy the $r_{in}$ relation, i.e., $r_{in}(x_1, x_2, \ldots, x_n) = true$, then these $n$ outputs must satisfy the $r_{out}$ relation, i.e., $r_{out}(o_1, o_2, \ldots, o_n) = true$. Formally, $r_{in}(x_1, x_2, \ldots, x_n) \Rightarrow r_{out}(o_1, o_2, \ldots, o_n)$.*

As an example, to test the function $\sin(x)$, a metamorphic relation we can utilize is that if $x_1 + x_2 = \pi$ holds, then the outputs, i.e., $\sin(x_1)$ and $\sin(x_2)$, must be equal. Formally, $(x_1 + x_2 = \pi) \Rightarrow (\sin(x_1) = \sin(x_2))$. With MR, we do not need to specify the exact output, i.e., the oracle, for a test case; instead, we only need to figure out the relations between the outputs of multiple *related* inputs, which is normally easier than the former way.

**Definition 2.6 (Source and Follow-up).** *For a metamorphic relation of this SUT $(r_{in}, r_{out})$, suppose $k$ test cases $x_1$, $x_2$, ..., $x_k$ are specified. If it is possible to generate additional $n - k$ test cases $x_{k+1}$, $x_{k+2}$, ..., $x_n$ such that $r_{in}(x_1, x_2, \ldots, x_n) = true$, then we refer these $k$ specified test cases $x_1$, $x_2$, ..., $x_k$ to be* Source *test cases or* Sources *for short, and these $n - k$ generated test cases $x_{k+1}$, $x_{k+2}$, ..., $x_n$ to be* Follow-up *test cases or* Follow-ups *for short.*

As an example, for the previous *sine* function, let $x_1 = 23$ be the source test case. In order to satisfy the MR mentioned previously, it must construct the follow-up test case $x_2 = \pi - 23$ such that $x_1 + x_2 = \pi$. Having the source and follow-up test cases, if their outputs violate the MR, e.g., $\sin(23) \neq \sin(\pi - 23)$, then a failure is automatically observed. Note that, on the other hand, if the outputs of sources and follow-ups satisfy the MR, it does not necessarily mean that the SUT is correct. There is a chance that the outputs between test cases coincidently meet the MR but are not correct by themselves. Nevertheless, MT is still an important method that can alleviate the automated oracle problem for test cases.

**Definition 2.7 (Metamorphic Group (MG)).** *For a metamorphic relation of this SUT $(r_{in}, r_{out})$, if $x_1$, $x_2$, ..., $x_n$ are test cases that satisfy $r_{in}(x_1, x_2, \ldots, x_n) = true$, then we refer these $n$ test cases to be Metamorphic Group (MG). More specifically, an MG is actually a group of Sources ($x_1$, $x_2$, ..., $x_k$) and Follow-ups ($x_{k+1}$, $x_{k+2}$, ..., $x_n$) related to metamorphic relation $(r_{in}, r_{out})$.*

In the example of the *sine* function, source 23 and follow-up $\pi - 23$ together form a Metamorphic Group (MG).

## 3 The Approach

In this section, we present an approach, named COMER, to enhance the traditional CT by taking into account the matchings of metamorphic relations. Note that in this section and the following sections, for the sake of simplicity, we only consider the metamorphic relations of which both the *Sources* and *Follow-ups* only contain one test case (which is one of the most common metamorphic relations in practice [39]).

### 3.1 Methodology Overview

Fig. 1 presents an overview of the framework of COMER, which consists of two main sub-processes. The first sub-process, which is referred to as *abstract test case generation*, focuses on generating a covering array. In this covering array, a test case consists of a set of integers, each of which represents a practical value or a set of practical values in the same equivalence class. Such a test case is called an *abstract* test case, which facilities the CA generation algorithm because it sets the algorithm free from the selection of a specific practical value when generating a test case. Overall, the sub-process of covering array generation follows the traditional one-test-at-a-time paradigm. That is, it continues generating test cases until all the $\tau$-way combinations are covered by these test cases. At each iteration of this sub-process, it first generates a set of candidate test cases and then selects the *best* one from that candidate set. Here, the *best* indicates the test case that covers the most uncovered $\tau$-way combinations in that candidate set. Note that, CT aims at covering all the $\tau$-way combinations by generating a set of test cases. Hence, in each iteration, selecting the "best" test case is a greedy approach to minimize the number of test cases (but usually not the optimal).

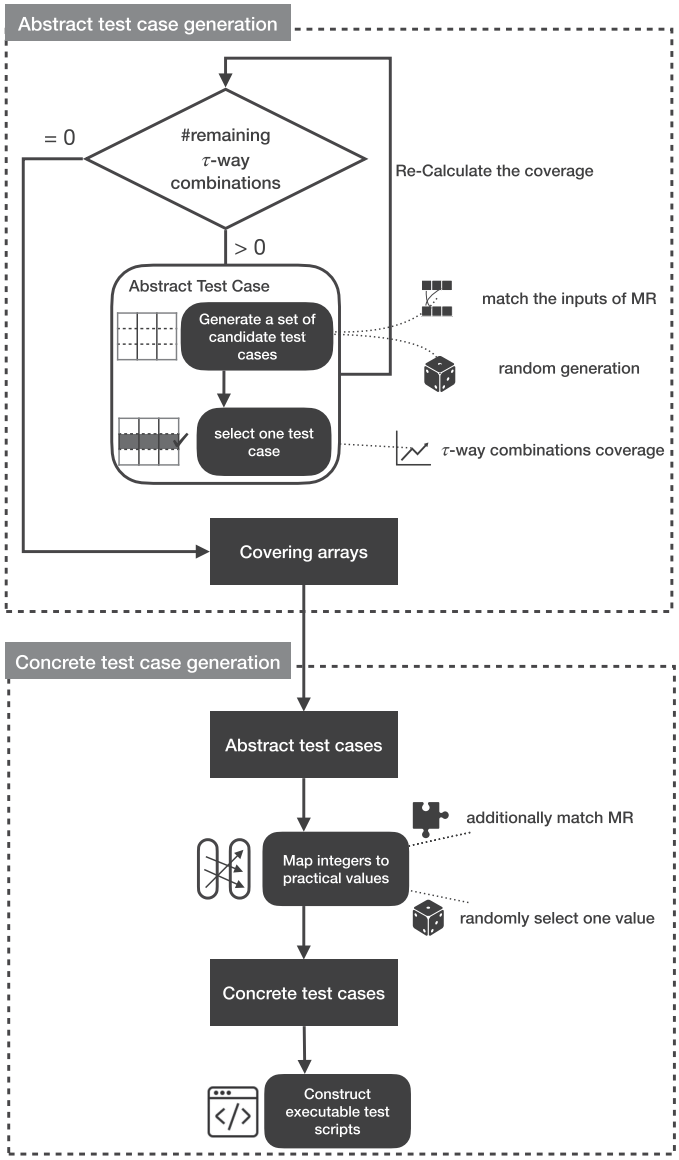



Fig. 1. The overview of COMER.

The second sub-process, which is referred to as *concrete test case generation*, aims at converting an abstract test case to a concrete and executable test case. For this, it first maps each integer in the abstract test case to a corresponding meaningful value (e.g., specific option values, event message, etc.). Then it constructs an executable script that can run the SUT with these specific, meaningful parameter values or options.

Note that the overall framework of COMER is similar to a traditional CT covering array generation process, except that it puts a high priority on the matchings of the metamorphic relations during the two sub-processes. By doing so, the generated test cases will form many *Metamorphic Groups (MGs)*, which will benefit the process of automatic correctness verification. The details of these two sub-processes are shown in the following two subsections.

### 3.2 Abstract Test Case Generation

This subprocess focuses on generating an abstract test case with two objectives: 1) the generated test case can match a previous test case to form an *MG*, and 2) can cover as many un-covered $\tau$-way combinations as possible. In order to achieve these two objectives, COMER adopts a two-stage mechanism. The first stage focuses on achieving the first objective by sampling a set of candidate test cases. The specific sampling strategy is shown in Algorithm 1.

**Algorithm 1.** Generate a set of Candidate Test Cases

**Input:** *MR*: metamorphic relation, *P*: parameters, *C*: a set of constraints, *PCA*: existing test cases, *REP*: the size of candidate test set, *PR*: a choice-controlling value
**Output:** *Candi*: a set of candidate abstract test cases

```
1  Candi ← { } ;
2  r ← rand() ;
3  if r ≤ PR then
4     t_pre ← rand_select(PCA) ;
5     repeat ← REP ;
6     for repeat > 0 do
7        t ← csp_solver(MR, t_pre, P, C);
8        Candi ← Candi ∪ {t};
9        repeat ← repeat - 1 ;
10    end
11 else
12    Candi ← random(P, C, REP);
13 end
14 return Candi ;
```

The inputs of this algorithm consist of one metamorphic relation *MR*, the abstract input parameters *P*, the constraints between these parameters *C*, previously generated abstract test cases *PCA*, the repeat times of abstract test case generation *REP*, and the value that controls the probability for choosing the specific strategy of test case generation *PR*.

The algorithm itself is a simple control flow which has two main branches. The specific entry is controlled by a randomly generated value *r* ranging from 0 to 1 (line 2). When this number *r* is not greater than a fixed value *PR* (line 3), the first branch is reached to generate a follow-up (line 7) of the source test case $t_{pre}$, which is randomly selected from the previously generated test cases (line 4). The process of generating a follow-up test case will repeat *REP* times (lines 5 and 6), which finally results in a candidate test set (line 14).

Specifically, COMER transforms the generation of a follow-up test case into a constraints satisfaction problem. Formally, let the selected abstract test case $t_{pre}$ = {($p_1, v_1$), ($p_2$, $v_2$), ..., ($p_n$, $v_n$)}, where $v_i \in V_i$, be the selected source test case. Let the metamorphic relation be $(r_{in}, r_{out})$, and the constraints $C$ = {$c_i$ | $c_i$ is a boolean constraint function of the SUT }. Then, the problem of obtaining a test case $t$ ={ ($p_1, v'_1$), ($p_2$, $v'_2$), ..., ($p_n$, $v'_n$) } such that the ($t_{pre}$, $t$) forms an *MG* can be formulated as the following formula.

$$\begin{aligned} Variables \quad & \{v'_1, v'_2, \ldots, v'_n\} \\ Domains \quad & D(v'_i) = V_i \quad \forall 1 \le i \le n \\ Constraints \quad & r_{in}(v_1, v_2, \ldots, v_n, v'_1, v'_2, \ldots v'_n) = true \\ & c_i(v'_1, v'_2, \ldots v'_n) = true \quad \forall c_i \in C \end{aligned} \tag{EQ1}$$

In Formula EQ1, *Domains* indicates that each abstract parameter value $v'_i$ can only be selected from $V_i$. The *Constraints* part has two rows, in which the first row indicates that the generated abstract test case should be the *follow-up* of the given source abstract test case. The second row indicates that the generated test case must be a valid test case (does not violate any constraint in *C*). After this, COMER will utilize a CSP solver to get one satisfying follow-up test case (line 7), and append it to the candidate test set (line 8). The solver we used is Choco [43], a conventional constraint programming library for Java that is built on an event-based propagation mechanism with backtrackable structures. In our experiments, all the constraints between parameter values and between test cases (source and follow-up test cases) are encoded into arithmetic forms, e.g., "$v_1 = v_2$", and "$v_1 > v_2$". Then, the solver will produce a solution if the constraints are satisfiable and if the computation resource is available. The correctness of generating a follow-up test case is guaranteed by Choco itself. We used the stable version of Choco (4.10.2) in the experiments. Furthermore, we manually checked some of the follow-ups (selected randomly), and all of them were correct (Note that it is not possible to manually check all of these follow-ups due to their large number).

The other branch (lines 11 to 13) randomly generates a candidate test set of size *REP*, of which the aim is to temporarily put aside the matchings of metamorphic relations. The reason COMER does not always generate *MGs* is that, in order to match the input rule of one metamorphic relation, normally, the source and follow-up test cases must be similar enough. However, such a pair of test cases is unfavorable to cover $\tau$-way combinations, and if COMER generates too many of them, it probably results in generating a large covering array with many redundant test cases (or even worse, the algorithm may not terminate).

The reason that such a pair of test cases is not always beneficial to CT is that the objectives of CT and MT may not always be consistent during the test generation. In the implementation of COMER, the metamorphic relations are encoded as the constraints between input parameter values. These constraints restrict the test input space of COMER.

Note that the original test space is the set of all the valid $\tau$-way combinations of parameter values. Metamorphic relations may reduce the test space to a smaller subspace. The extent to which the test space is reduced depends on how "strong" the metamorphic relations are, e.g., how many parameter values are involved in the metamorphic relations. In the worst case, a metamorphic relation may involve all the parameter values of a system. Consider a metamorphic relation which specifies that two tests are the same except for one parameter value. If we always try to match this metamorphic relation, only one parameter value would change during test generation. As a result, the combinations involving other parameter values may always be omitted, making it impossible to achieve combinatorial coverage.

For example, consider testing the TCAS, which is one program in the *Siemens* program set [44] and widely used as a subject in CT [29], [45], [46]. There are 12 input parameters of this program. One MR is that if the relation of the 4th parameter *Own_Tracked_Alt* (the altitude of the TCAS equipped airplane) and 6th parameter *Other_Tracked_Alt* (the altitude of the "threat") is not changed, meanwhile other parameter values are not changed too, then the outputs of the TCAS should be the same. Suppose that the source abstract test case is $\{(p_1, 0), \ldots, (p_4, 0), \ldots, (p_6, 0), \ldots, (p_{12}, 0)\}$, of which the 4th parameter value is equal to the 6th parameter value. Based on this metamorphic relation, the satisfying follow-up test case should be $\{(p_1, 0), \ldots, (p_4, 1), \ldots, (p_6, 1), \ldots, (p_{12}, 0)\}$, of which the 4th parameter value is also equal to the 6th parameter value while other parameter values are equal to those of the source test case. Note that these two test cases are the same except for 4th and 6th parameter values. As a result, only 21 2-way combinations are additionally covered by the follow-up test case, which is far smaller than the total 2-way combinations it contains (which is $\binom{12}{2} = 66$).

Therefore, to alleviate this issue, we additionally offer a chance to randomly generate candidate test cases which are likely to cover more uncovered $\tau$-combinations, such that COMER can quickly converge to the end and generate a covering array of smaller size.

Note that Algorithm 1 is the first step of the process for generating **one** abstract test case. The second step of this process is to select one test case from the candidate test set given by Algorithm 1. Particularly, COMER then focuses on selecting one test case from the candidate test set that can cover as many $\tau$-way combinations as possible (for the second objective). Note that this selection strategy follows a greedy manner which is widely adopted in the CA generation methods [3], [5], [6]. Although it does not necessarily guarantee to generate an optimal test set with the minimal size, its simplicity and efficiency benefit the overall framework of COMER [2]. Therefore, in this phase, we calculate the uncovered $\tau$-way combinations that each test case in the candidate test set can cover and select the one that has the most uncovered $\tau$-way combinations.

The stopping criteria for COMER is the same as other CT approaches in the stage of abstract test generation, i.e., to cover all the $\tau$-way combinations. Note that an abstract test case only covers a small set of $\tau$-way combinations. Therefore, we need to repeatedly generate abstract test cases, one at a time, until all the $\tau$-way combinations are covered.

### 3.3 Concrete Test Case Generation

In order to conduct a real test of the SUT under the given parameter values, the subprocess of converting an abstract test case to a concrete test case is needed. This subprocess of COMER consists of two phases. In the first phase, it maps the abstract value to a meaningful, practical value. Note that, in traditional CT, this value can be randomly chosen (as long as it belongs to the equivalence class which corresponds to the abstract integer), since it does not affect the coverage of $\tau$-way combinations. However, for COMER, the choice matters, as it may be relevant to the matchings of metamorphic relations. For example, when testing *sine* function, it is rational to classify the inputs into negative, zero, and positive numbers. With this input parameter model alone, it is not possible to generate two abstract test cases, $t_1$ and $t_2$, such that they match the input relation $t_1 + t_2 = \pi$. This is because the abstract input model only classifies the inputs into negative, zero, and positive numbers. Hence, with this model, we can only generate as inputs random numbers that are negative, or positive, or zero. For example, consider the domain of the double type in Java, which is $-1.79 * 10^{308}$ to $+1.79 * 10^{308}$. It is unlikely that the sum of a random negative number and a random positive number is equal to $\pi$. However, in the stage of *Concrete test case* generation, we can easily choose the two concrete values to satisfy the metamorphic relation. Suppose that we have selected a concrete value for one test case to be 23. The metamorphic relation can be satisfied by choosing the concrete value of another test case to be $\pi$ - 23.

In the implementation of COMER, we adopt a similar strategy as in the *abstract test case generation* stage to make the generated concrete test cases match the metamorphic relations. That is, we encode the metamorphic relations as constraints and transform the concrete test case generation problem into a constraints satisfaction solving problem.

More specifically, for each abstract value of each parameter, there exists a corresponding concrete value set, in which a concrete and meaningful value can be mapped to. Formally, let $K_i^j$ be the concrete value set which corresponds to the parameter $p_i$ with $j$th abstract value $v_{i_j}$, i.e., $(p_i, v_{i_j})$, for $1 \leq i \leq n, v_{i_j} \in V_i$. Suppose there are two abstract test cases $t_s^a$ and $t_f^a$. Let $t_s^a = \{(p_1, v_{1_{x_1}}), (p_2, v_{2_{x_2}}), \ldots, (p_n, v_{n_{x_n}})\}$ and $t_f^a = \{(p_1, v_{1_{x'_1}}), (p_2, v_{2_{x'_2}}), \ldots, (p_n, v_{n_{x'_n}})\}$, where $v_{i_{x_i}} \in V_i$ and $v_{i_{x'_i}} \in V_i$, for $1 \leq i \leq n$.

Then the problem of converting $t_s^a$ and $t_f^a$ into two concrete test cases $t_s^k = \{(p_1, k_1), (p_2, k_2), \ldots, (p_n, k_n)\}$, and $t_f^k = \{(p_1, k'_1), (p_2, k'_2), \ldots, (p_n, k'_n)\}$ such that the $(t_s^k, t_f^k)$ forms an *MG* can be formulated as the following formula.

$$\begin{aligned} Variables \quad & \{k_1, k_2, \ldots, k_n, k'_1, k'_2, \ldots, k'_n\} \\ Domains \quad & D(k_i) = K_i^{x_i}, D(k'_i) = K_i^{x'_i} \quad \forall 1 \leq i \leq n \\ Constraints \quad & r_{in}^k(k_1, k_2, \ldots, k_n, k'_1, k'_2, \ldots k'_n) = true \end{aligned} \tag{EQ2}$$

In Formula EQ2, *Domains* indicates that each concrete value $k_i$ and $k'_i$ can only be selected from the value set that corresponds to its corresponding abstract value. *Constraints*

2. Yet, COMER can also adopt other test case selection strategies, e.g., heuristic search-based strategy [4], [47], [48]. However, the chosen of test case selection strategy is beyond the scope of this paper.

TABLE 4
The Input Parameter Model for *ClosestPair*

| Params | Abstract values | Concrete corresponding values |
|---|---|---|
| $p_1$ | 0, 1 | 0: 0∼2 points in the first quadrant<br>1: 3∼5 points in the first quadrant |
| $p_2$ | 0, 1 | 0: 0∼2 points in the second quadrant<br>1: 3∼5 points in the second quadrant |
| $p_3$ | 0, 1 | 0: 0∼2 points in the third quadrant<br>1: 3∼5 points in the third quadrant |
| $p_4$ | 0, 1 | 0: 0∼2 points in the fourth quadrant<br>1: 3∼5 points in the fourth quadrant |

indicates that the generated concrete test cases should satisfy the input rule of the given metamorphic relation $r_{in}^k$ for concrete test cases. With this formula, once again, the utilization of the CSP solver can help to get a satisfactory solution.

Note that not every pair of abstract test cases can be converted to concrete test cases such that they can form an *MG*. In our implementation, the pair of abstract test cases should form an *MG* in the *abstract test case generation* stage at first (by EQ1). Then, the selected abstracted test cases will be converted to concrete test cases by considering the metamorphic relations for concrete values (by EQ2). For the remaining abstract test cases, we randomly select a concrete value (within its domain) for each abstract parameter value.

After the mapping procedure, the next phase is to write an executable script such that we can actually run the SUT with these test inputs. In this phase, COMER follows the traditional unit testing paradigm [49], e.g., JUnit, to write the test script. However, there is one difference between COMER and traditional unit testing frameworks. That is, we do not offer assertions (oracles) for each concrete test input. Instead, an automated script is used to find pairs of concrete tests such that their inputs match the input rule of a given metamorphic relation. After that, COMER will verify their outputs. A failure will be detected if they fail in matching the output rule of a given metamorphic relation.

### 3.4 Running Example

In this section, we will give an example to show how COMER works in detail. Consider the SUT *ClosestPair*, which is a program to give the distance between the closest pair of points among a set of points in the cartesian coordinate system. To test the SUT by COMER, We first build a simplified input parameter model for this program, of which the information is listed in Table 4. In this table, we give the abstract values and corresponding concrete values for each parameter of the SUT. The basic idea of this model is to check the program with different numbers of sampled points in four quadrants of the coordinate system.

Second, we give the metamorphic relation of this program: for two sets of points $A$ and $B$, if set $B$ is created by adding one more point to set $A$, then the distance of the closest pair of $B$ should be equal to or smaller than that of $A$, i.e., $ClosestPair(B) \leq ClosestPair(A)$. Note that this metamorphic relation cannot be directly applied to the input model in Table 4. This is because the model does not specify the number of points in each quadrant; instead, it only gives the range of the number of points. On the contrary, the metamorphic relation requires the specific number of points of two sets, with one set having exactly one more point than the other. Therefore, to apply COMER, we need to convert the metamorphic relation into other forms. For this example, we split the metamorphic relation into two parts (The following PART 1 and 2), of which the first part is for the abstract test cases, while the second part is for the concrete test cases.

$$
\begin{aligned}
Source \quad & \{v_1, v_2, v_3, v_4\} \\
Follow-up \quad & \{v_1', v_2', v_3', v_4'\} \\
Part\ MR \quad & v_1 \leq v_1', \quad v_2 = v_2', \quad v_3 = v_3', \quad v_4 = v_4' \\
& v_1 = v_1', \quad v_2 \leq v_2', \quad v_3 = v_3', \quad v_4 = v_4' \\
& v_1 = v_1', \quad v_2 = v_2', \quad v_3 \leq v_3', \quad v_4 = v_4' \\
& v_1 = v_1', \quad v_2 = v_2', \quad v_3 = v_3', \quad v_4 \leq v_4'
\end{aligned}
$$

**PART** 1: The metamorphic relation for the abstract values

In PART 1, the source and follow-up are two abstract test cases consisting of abstract values listed in Table 4. Specifically, $v_i$ and $v_i'$ are the values of $p_i$, where $1 \leq i \leq 4$. Since the original metamorphic relation requires that one set of points has one more point than the other, the corresponding abstract test cases should be generated by taking into account this property (Otherwise, there may do not exist any satisfying concrete test case). Hence, in this part, we let the range of points of one quadrant of the source test case be smaller than or equal to that of the follow-up test case. For example, $v_1 \leq v_1', v_2 = v_2', v_3 = v_3', v_4 = v_4'$ indicates that the ranges of the points of the source test case and follow-up test case are the same except the ones in the first quadrant, where the follow-up is larger than or equal to the source. By this metamorphic relation, when mapping to the concrete test cases, the additional point can be generated in the first quadrant.

$$
\begin{aligned}
Source \quad & \{A = \{(x,y) \,|\, x > 0, y > 0\}, \\
& \ B = \{(x,y) \,|\, x < 0, y > 0\}, \\
& \ C = \{(x,y) \,|\, x < 0, y < 0\}, \\
& \ D = \{(x,y) \,|\, x > 0, y < 0\}\} \\
Follow-up \quad & \{A' = \{(x,y) \,|\, x > 0, y > 0\}, \\
& \ B' = \{(x,y) \,|\, x < 0, y > 0\}, \\
& \ C' = \{(x,y) \,|\, x < 0, y < 0\}, \\
& \ D' = \{(x,y) \,|\, x > 0, y < 0\}\} \\
Remaining\ MR \quad & A' = A \cup \{(x,y)\}, x > 0, y > 0, \\
& B' = B, C' = C, D' = D \\
& B' = B \cup \{(x,y)\}, x < 0, y > 0, \\
& A' = A, C' = C, D' = D \\
& C' = C \cup \{(x,y)\}, x < 0, y < 0, \\
& A' = A, B' = B, D' = D \\
& D' = D \cup \{(x,y)\}, x > 0, y < 0, \\
& A' = A, B' = B, C' = C
\end{aligned}
$$

**PART** 2: The metamorphic relation for the concrete values

With the metamorphic relation in PART 1 alone, it is not enough to generate two sets of points, with one set exactly having one more point than the other. This is because PART 1 only describes the relation between the ranges of points instead of the specific number of points. Therefore, we give the supplemental information in PART 2. In the second part, the source and follow-up are two concrete test cases. Each concrete test case consists of four sets of points, i.e., $A, B, C, D$ or $A', B', C', D'$, of which each set represents one quadrant.

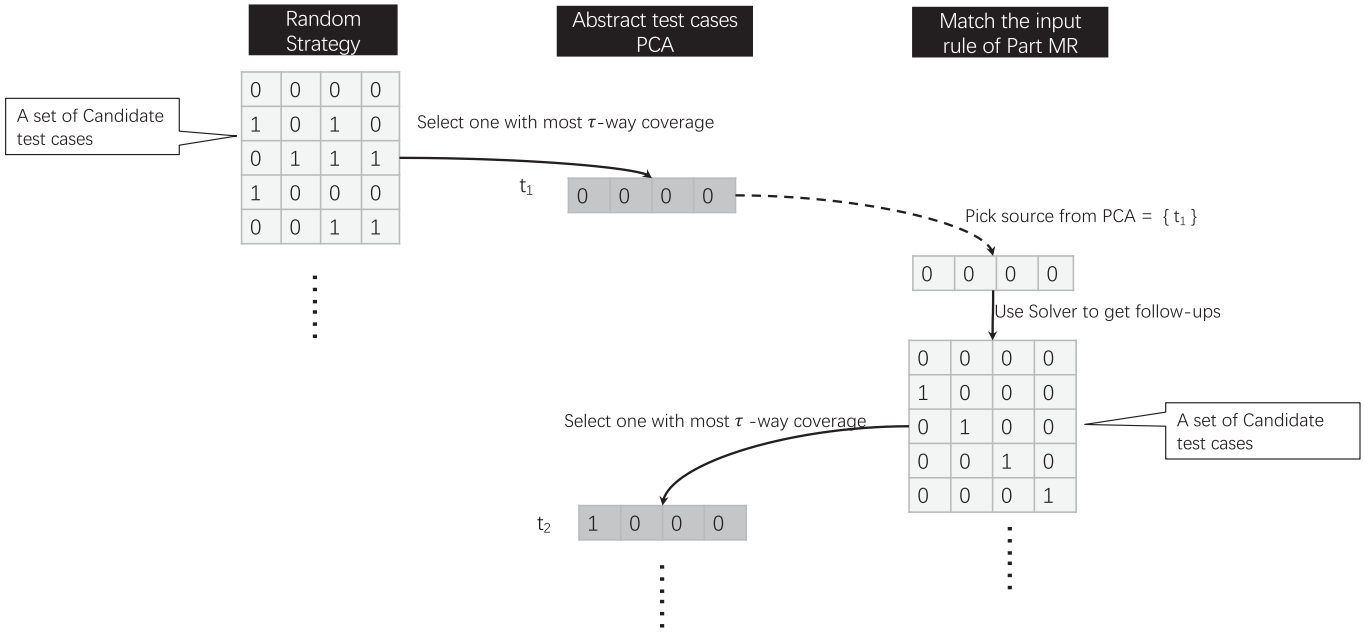



Fig. 2. Abstract test cases generation for ClosestPair.

When testing *ClosestPair*, we just need to merge all these points in four sets and send them to the program and execute it. In this part, the metamorphic relation involves the actual concrete value. For example, $A' = A \cup \{(x, y)\}, x > 0, y > 0, B' = B, C' = C, D' = D$ indicates that the follow-up has exactly one more point in the first quadrant when compared with the source test case; and apart from this additional point, other points of these two test cases are the same. There are four such relations, each of which determines one quadrant where the additional point is sampled.

With the input parameter model and metamorphic relations, we can easily apply the approach COMER to generate test cases for the SUT *ClosestPair*. The details are shown in Figs. 2 and 3, of which the former shows the generation of abstract test cases, while the latter shows the generation of concrete test cases.

Fig. 2 consists of three main parts(columns): the left part indicates the process of generating test case randomly, the right part indicates the process of generating test cases that match the metamorphic relation, and the middle part outputs the generated test cases by the previous two processes (Note: the two processes are adopted randomly during the test case generation according to Algorithm 1). In Fig. 2, it first adopts the random process and samples 5 test cases. Then, it selects the one that contains the most uncovered $\tau$-way combinations as $t_1$ (0, 0, 0, 0). Next, it turns to the metamorphic relation matching process. In this process, it first randomly picks one test case from previously generated test cases as the source test case. For this example, the selected test case is $t_1$ (it is the only test case that has been generated). Then it utilizes a CSP solver to obtain follow-up test cases that satisfy the constraints listed in PART 1. In this example, there are five satisfying test cases, and it also selects the one that contains the most uncovered $\tau$-way combinations as $t_2$ (1, 0, 0, 0). The abstract test case generation will continue until there is no uncovered $\tau$-way combination.

Fig. 3 shows the conversion from abstract test cases to concrete test cases. For abstract test cases $t_1$ and $t_2$, their concrete test cases must meet two rules: first, the concrete values should correspond to the abstract values. For example, the first parameter value of the abstract test case $t_1$ is 0, which indicates that the number of points in the first quadrant is between 0 and 2. Hence, two points (1, 2) and (2, 3) are satisfied as the concrete values. Second, the concrete test cases should satisfy the constraints listed in PART 2. Therefore, in this example, the concrete test cases of $t_2$ have exactly one more point (9, 11) in the first quadrant when compared with $t_1$. Note that these two rules may not always be satisfied. In that case, COMER will give up on the matching of the metamorphic relation and only keep on generating test cases that correspond to their abstract test cases.

COMER can not only be applied to programs with numerical inputs and outputs as in this example, but also to other types of programs that are supported by traditional CT. In fact, there are no particular constraints on the input and output types for COMER. It works for programs that take input values from a finite discrete set, e.g., the

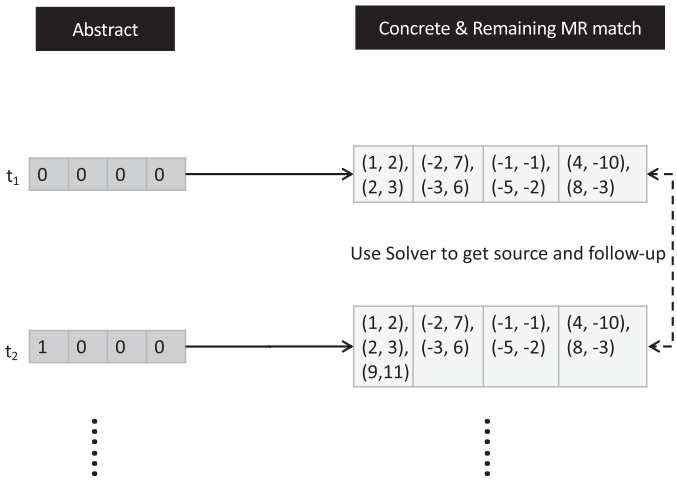



Fig. 3. Concrete test cases generation for closestpair.

configuration options, limited integers, GUI events, control signals. When the input values are infinite or non-discrete, we can utilize techniques such as equivalence partitioning to create a finite discrete set of inputs.

Additionally, the metamorphic relations do not restrict the types of programs that are supported by COMER. The metamorphic relations are very flexible: they can be arithmetic formulas among numerical variables, logic equations between sets, or even simple equivalence and in-equivalence relations between strings, etc., which is not limited to numerical programs only. Even in the worst case, i.e., there is no identified metamorphic relation, COMER can still work (under this condition, COMER will degrade into the traditional CT approach).

## 4 Experimental Setup

To evaluate the effectiveness and efficiency of COMER, we conducted a series of experiments, with the aim to answer the following research questions:

RQ1: Is COMER effective at handling the automated oracle problem when compared with other baseline approaches?

RQ2: What about the performance of these approaches in terms of time cost?

RQ3: Compared with using optimal oracles, how does COMER lose in fault detection by the mere use of MR?

RQ4: What features of the metamorphic relations affect the performance of COMER?

### 4.1 Benchmarks

We explored the benchmarks that are widely used in the studies of CT and MT [30], [42], [50], among which 158 unique software subjects are extracted as the initial candidate data set. We then inspected these subjects and filtered them according to the following three criteria: 1) the source code of the selected subjects must be available online, 2) the number of abstract input parameters of the subjects must be larger than 3, and 3) these subjects should have explicit metamorphic relations. Note that here by saying "explicit", we mean that after investigating the specification of the subject, the metamorphic relations can be easily identified by using the traditional metamorphic testing guidelines [39], [51], [52], [53], [54]. After the inspection process, 31 software subjects remained, as shown in Table 5.

For each of these subjects, we built or reused (given by previous studies) a corresponding CT abstract Input Parameter Model (IPM), which includes parameters, value set for each parameter, and constraints, respectively. We then identified or reused metamorphic relations for each subject based on this input model. After the modeling process, for each subject, we developed corresponding scripts that can map an abstract test input into a concrete and executable test case, along with the scripts that can search and count the matchings of the metamorphic relations between generated test cases. The accomplishment of the whole preparation task took us about three months, of which one month is used for the modeling process (the IPM modeling procedure accounted for about 90% of the time, while the MR obtaining procedure about 10%), and the remaining two months are used to write the corresponding scripts. This preparation task finally results in a new software repository for the evaluation of overlapping studies of combinatorial testing and metamorphic testing.

Table 5 lists the brief information of these subjects, which include the software name, description, lines of uncommented code, abstract input parameter model, constraints, number of metamorphic relations, type of metamorphic relations, and faults. In this table, the IPM information is given in an abbreviated form of the numbers of parameters with a given number of values $\#values^{\#parameters}$ (Column 'Abstract IPM'), and the constraint information is given in an abbreviated form of the numbers of constraints with a given number of involved parameters $\#parameters^{\#constraints}$ (Column 'Constraints'). With respect to metamorphic relations, we chose the common types of MRs, e.g., add, transpose, increment, etc. [51], [55], [56], that are suggested by MT practices (Column 'Types of MRs'). Table 6 gives a brief description of these types of metamorphic relations. Particularly, the type of "Special" indicates the MRs can not be directly classified into the traditional types of metamorphic relations. Nevertheless, these MRs in the experimental subjects of this paper are simple and can be easily identified.

To offer a detailed view of these metamorphic relations, for each subject, we summarized the inputs, outputs, detailed description of MRs, and where they are from in the Table of Summary in the Appendix, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TSE.2021.3131548. From that table, it can be observed that many MRs come from existing studies. For the remaining MRs, to guarantee the correctness of these newly identified MRs, we used the following strategy: the first two authors of this paper independently identified the MRs, and then we cross-checked their correctness. Note that, in general, the newly identified MRs are straightforward, and their correctness can be checked with reasonable effort.

It is desired to note that these metamorphic relations are different from the system constraints that appeared in traditional CT. Traditional system constraints describe the relationships between the input parameters. Furthermore, these constraints apply to every test case of the system, i.e., every test case should not violate the system constraints. However, metamorphic relationships describe the relations between test cases, and they involve both input and output parameter values. Furthermore, they do not apply to every test case. Instead, only these test cases which satisfy the input rules will be affected.

### 4.2 Approaches Under Comparison

Three baseline approaches are evaluated and compared to the proposed approach COMER. The first one is Random Testing (RT), which is a widely used baseline approach. In our comparison, RT does not use any abstract input parameter model. Instead, RT just randomly selects values from the valid domain of each parameter to generate test cases.

TABLE 5
The Software Subjects Under Evaluation

| *Software* | *Description* | *LOC* | *Abstract IPM* | *Constraints* | *#MRs* | *Types of MRs* | *Faults* |
|---|---|---|---|---|---|---|---|
| Schedule [57], [58] | Priority scheduler (Siemens suite) | 368 | $2^2 3^7 8^2$ | - | 2 | Special | real#1 |
| Determinant1 [59] | Matrix determinant computation | 98251 | $2^1 3^1 5^2$ | - | 2 | Transpose,Multiply | novel#1 |
| JAMA [59] | Matrix determinant computation | 2858 | $2^1 3^1 5^2$ | - | 2 | Transpose,Multiply | novel#1 |
| ClosestPair [60] | Finding the closest pair of points | 320 | $2^1 4^3$ | $3^1$ | 1 | Include | novel#1 |
| Printtoken [58], [61] | Lexical analyzer (Siemens suite) | 563 | $2^2 3^1 4^4 5^1 10^1 13^2$ | $4^2$ | 3 | Special | real#1 |
| Multi-MAXSUM [55] | Multi-Segment MAXSUM Algorithm | 22 | $2^1 3^4 4^1$ | $2^2 4^2$ | 2 | Include | seed#9 |
| SurroundedRegion [55] | Capture all regions of a board surrounded by a symbol | 59 | $2^1 3^2 5^2$ | $3^3$ | 2 | IncludeMatrix | seed#12 |
| MaxRectangle [55] | Find the largest rectangle in a 2D binary matrix | 62 | $2^1 3^2 5^2$ | $3^3$ | 2 | IncludeMatrix | seed#30 |
| InterleavingString [55] | Decide a string is the interleaving of other two strings | 12 | $2^1 3^2 5^4$ | $2^2$ | 1 | Include | seed#6 |
| QuickSort [55] | Quick sort algorithm | 40 | $3^6$ | $2^2 3^1 4^1$ | 3 | Add,Multiply,Permute | seed#5 |
| Bsearch1 [62] | Binary search within a sorted array | 37 | $3^4 5^1$ | $3^1 4^2 2^2$ | 2 | Add,Include | seed#5 |
| Spwiki [63] | Shortest path between two vertices in a graph | 52 | $3^4 4^1$ | $2^1 3^1$ | 2 | Swap,Multiply | seed#17 |
| DistinctSubsequence [55] | Count the distinct subsequences of an string | 14 | $2^1 4^2 5^2$ | $2^2$ | 1 | Include | seed#22 |
| Editingdistance [55] | Enhanced edit distance algorithm | 24 | $3^2 5^4$ | $2^2$ | 1 | Include | seed#14 |
| FirstMissingPositive [55] | Find the first missing positive integer | 13 | $2^1 3^5$ | $2^2 3^1 4^1$ | 2 | Permute,Include | seed#17 |
| HeapSort [55] | Heap sort algorithm | 42 | $3^6$ | $2^2 3^1 4^1$ | 3 | Add,Multiply,Permute | seed#23 |
| Schedule2 [57], [58] | Priority scheduler (Siemens suite) | 347 | $2^2 3^7 8^2$ | - | 2 | Special | seed#10 |
| Printtokens2 [58], [61] | Lexical analyzer (Siemens suite) | 355 | $2^2 3^1 4^4 5^1 10^1 13^2$ | $4^2$ | 3 | Special | seed#10 |
| TCAS [50] | Traffic collision avoidance system | 135 | $2^7 3^2 4^1 10^2$ | - | 4 | Special | seed#10 |
| Maxsub [55] | Kadane's MAXSUB algorithm | 13 | $3^5$ | $2^2 3^1 4^1$ | 1 | Reverse | seed#6 |
| Jodatime [64] | Date and time utilities | 31909 | $3^1 4^1 5^6 6^2$ | $2^1 3^2$ | 1 | Special | seed#28 |
| Klp [65] | Key-lock problem algorithm | 71 | $2^2 3^2$ | - | 2 | IncludeMatrix | seed#30 |
| Trisquarej [66] | Returns the type and square of a triangle | 70 | $3^3 4^1$ | $3^1 4^4$ | 4 | Swap,Multiply | seed#30 |
| Boyer [55] | Get the first occurrence of a pattern within a text | 248 | $2^1 3^6 4^1$ | $3^1$ | 2 | Exclude,Include | seed#14 |
| Lucene [67] | Text search engine library | 19205 | $3^6 4^2 5^1$ | $3^1$ | 3 | Special | seed#4 |
| Superstring [68] | Find the shortest common string | 61 | $2^1 3^1 5^4$ | $2^2$ | 1 | Include | seed#2 |
| Bsearch2 [69] | Binary search within a sorted array | 19 | $2^1 3^3 4^1$ | $2^3 3^2$ | 1 | Increment | seed#6 |
| RSA [70] | RSA encryption program | 11 | $3^1 4^2 12^1$ | - | 1 | Special | seed#4 |
| Shortest-path [71] | Get the shortest distance between nodes of the graph | 234 | $2^1 3^1 4^2 6^1$ | - | 1 | Special | real#1 |
| Rotate [71] | Rotate the matrix | 256 | $3^1 4^1 6^2$ | - | 1 | Increment | novel#1 |
| Foneway [71] | calculate the variance of a single factor | 1861 | $2^3 3^4$ | - | 1 | Permute | novel#1 |

The second one is traditional Combinatorial Testing (CT). To make a fair comparison, the CT framework used in the empirical studies is the same as COMER. Specifically, the algorithm of CT test generation is Adaptive Random Sampling (ARS) [72], [73], which is widely used as the baseline approach in CT approaches and other testing approaches. In a recent paper[74], empirical studies have compared ARS with other CT approaches. Based on that study, some observations can be made as follows: with respect to the number of generated test cases, ARS generates more test cases than other state-of-the-art approaches, e.g., [75], [76]. However, it is very simple and efficient such that we can generate a set of covering arrays as test suites in a very short time.Such characteristics of ARS benefit the overall process of our experiments. On the one hand, we can easily extend ARS with metamorphic relations matching strategy to form the basis of COMER. On the other hand, considering that we need to generate covering arrays for a large number of subjects (original ones and their mutants), using the ARS can accelerate the speed of our experiments. Additionally, more test cases also benefit the overall process of obtaining oracles of CT in our experiments. This is because it increases the chances of matching the *MGs*. Note that CT differs from COMER in that it does not consider the metamorphic relations when generating test cases (both in the abstract and concrete test case generation stages).

TABLE 6
The Types of the Metamorphic Relations Used in the Subjects

| *Relation* | *Change made to the input* |
|---|---|
| Add | Add a constant |
| Exclude | Remove an element |
| Include | Add a new constant |
| Increment | Add the value of each element regularly |
| IncludeMatrix | Add a row or column(Input is matrix) |
| Multiply | Multiply by a constant |
| Permute | Randomly permute the elements |
| Reverse | Reverse the input |
| Swap | Swap part content |
| Transpose | Transpose(Input is matrix) |
| Special | Other particular operations |

The third one is a simple version of COMER which is also newly provided by us in this paper. It is a trivial extension of CT by accounting for metamorphic relations, which is referred to as *tri-MCT*. *tri-MCT* first utilizes the traditional CT approach to generate a covering array. Then for each abstract test case in this covering array, *tri-MCT* tries to generate a corresponding test case such that these two test cases form an *MG* in terms of one metamorphic relation. Besides, in the concrete test case generation stage, it adopts the same strategy as COMER does, i.e., trying to make the follow-up concrete test case match the metamorphic relations.

The main differences between tri-MCT and COMER include:

- *Generating test cases.* During test generation, the strategy of COMER is the same all the way: it considers the satisfaction of metamorphic relations and t-way coverage together. However, the strategy of tri-MCT varies in two stages. In the first stage, it only considers t-way coverage until a covering array is generated. In the second stage, it only considers the matchings of metamorphic relations.

- *Generation of source and follow-ups.* For each source test case, the probability is 50 percent for COMER to generate a follow-up, while the probability is 100 percent for tri-MCT.
- *Termination criteria.* COMER terminates when a covering array is generated. tri-MCT terminates when a covering array that contains a set of follow-ups of each test case is generated.

### 4.3 Metrics and Other Settings

To evaluate the effectiveness of these four approaches (*COMER*, *RT*, *CT*, and *tri_MCT*) on the handling of the automated oracle problem, the most direct measurement is the number of *MG* matchings between generated test cases. More *MG* matchings provide more chances for the approach to verify the correctness of the SUT. Another related metric is the number of generated test cases, which assesses the cost of each approach to obtain these *MG* matchings.

Besides the *MGs* and the number of test cases, another important metric is the actual fault detection rate. With this regard, it is essential to figure out whether the generated test cases can detect the underlying faults in these subjects, i.e., can we find the violation of metamorphic relations of corresponding test cases. Note that among these 31 real-world subjects, there are some real faults as mentioned in the studies [57], [58], [77]. We also thoroughly explored the issue track of the *scipy*[3] to mine one additional fault of the program *Shortest-path* [78]. These existing faults are labeled as *'real'* in the column *Faults* of Table 5, followed by the number of faults. During the experiments, we discovered 5 previously unknown bugs. These faults are labeled as *'novel'* in the column *Faults* of Table 5.

These five previously unknown bugs belong to five different subjects, i.e., *Determinant1* , *JAMA* , *ClosetPair* , *Rotate* , and *Foneway* , in which *JAMA* and *Determinant1* are from the study [59], *Rotate* and *Foneway* are from the *scipy*[3], and *ClosetPair* is a program posted online [60]. For *Rotate* and *Foneway* , we submitted two issue reports [79], [80], and the developers have confirmed that they are real bugs (labeled as defect in the issue tracking system of Github). For *JAMA* and *Determinant1* , we sent emails on July 16, 2020, to the authors to describe the bugs we have found, but no reply has been received as of now. Hence, for the three programs, i.e., *JAMA* , *Determinant1* , and *ClosetPair* , the manual inspections were performed independently by three authors of this paper (the first, second, and fourth authors). At last, these programs are all determined to be buggy by three authors. In fact, all these five bugs are very clear and straightforward, and we have summarized their information online (https://github.com/syjya/Comer) such that everyone can check them. Based on the information we have posted, these five bugs are very easy to reproduce.

At last, as a supplement to the faulty software, we additionally use mutation testing [81] on the remaining correct software subjects, which results in 314 additional faulty software subjects (labeled as *'seed'* in the column *Faults* of Table 5, followed by the number of faults. For example, seed#23 indicates that there are 23 mutants of the program). The use of mutation testing to seed faults has been widely used in CT studies [82], [83], [84].

More specifically, for subjects *Printtokens2* , *Schedule2* , and *TCAS* , we reused the mutants from the website of Software Infrastructure Repository (SIR)[4]. For the remaining subjects, we created these mutants by utilizing existing mutation testing tools. Particularly, for the programs written in JAVA, we used $\mu$Java[5]. For the programs written in C and C++, we used Mull[6]. Both of the two tools support traditional mutation operators. For each program, we tried to generate 30 mutants at first. Then, we checked whether these mutants were valid. Specifically, we manually removed mutants that cannot be complied, mutants that can be complied but do not terminate, mutants that are equivalent mutants of the original correct program (the introduced change does not modify the meaning of the original program; hence, they do not contain any fault). As a result, not all the 30 mutants are included; only those valid ones are used in the experiments. We did not generate more mutants (larger than 30) to reduce the total time cost of experiments. In fact, for each mutant, we need to run various test cases on it, which requires a lot of time. Additionally, in our experiments, mutant testing tools just randomly picked lines of the original programs and used mutation operators to revise them. The process of mutant generation is fully automated, i.e., without any human intervention.

With the information of these faults, it is easy to measure these four approaches with respect to the fault detection rate. *Note*: for these four approaches, i.e., COMER, RT, CT and tri-MCT, the correctness of the outputs of generated test cases is determined by checking whether the output rule of a given metamorphic relation is violated. The detailed description of these faults (both real and seeded) is also given on the website https://github.com/syjya/Comer.

It is desired to note that in our studies, test generation and test evaluation (that uses an oracle) are two separate processes. For the test generation, all the approaches use their own algorithms to generate test cases. In the test evaluation step, for all the approaches, we use the same script to find pairs of test cases that can match the metamorphic relations. This script works on a set of generated test cases, and its process is very simple: for each test case in this set, this script searches through the other test cases and check whether they can satisfy the input rules of some metamorphic relations. Therefore, although some approaches, e.g., CT and RT, do not consider the metamorphic relations when generating test cases, we can use metamorphic relations as oracles for test evaluation. In fact, these metamorphic relations can be regarded as some types of constraints. Hence it is possible that the test cases generated by CT and RT coincidently satisfy these metamorphic relations. This point is also reflected in the results of the experiments (the number of matchings of metamorphic groups of CT or RT is not always 0). Further, in our experiments, *all the oracles used by these approaches are metamorphic relations; no other types of oracles (e.g., crashes or exceptions) are used (these simple faults are filtered out in the experiments).*

3. https://github.com/scipy

4. https://sir.csc.ncsu.edu/portal/index.php
5. https://github.com/jeffoffutt/muJava
6. https://github.com/mull-project/mull

In the implementation of COMER for this evaluation, the value of *REP* in Algorithm 1 is set to be 50. The larger *REP* , the better chance to select a test case with higher coverage in each iteration, the fewer test cases to generate, but the more time it takes. As suggested by study [72], we set REP to 50 to strike a balance between the number of test cases and test execution time. The value of *PR* in Algorithm 1 is set to be 0.5. We have tried other PR values from 0.1 to 0.9 and found that 0.5 is the best when considering the effectiveness and efficiency of COMER in the experiments. The strengths of covering arrays are set to be 2, 3, and 4, respectively, which are the common settings in the CT community [8], [74], [85]. Besides, to account for the randomness in the experiments, these approaches are repeated 30 times for each subject. The repeat number is also common in empirical studies [86], [87], [88]. All experiments are carried out on a machine with Intel Xeon Gold 5117 CPU@2.0GHz, 128GB RAM, and 64-bit Ubuntu 18.04.

## 5 Results

### 5.1 RQ1: The Effectiveness of COMER

To answer the first research question, we reported the number of the generated test cases, the matchings of *MGs*, and the detected faults, of three approaches, i.e., COMER, CT, and tri-MCT, respectively, in Figs. 4, 5, and 6, respectively. With respect to the approach RT, we set the number of generated test cases by RT to be the same as the other three approaches (CT, COMER, and tri-MCT) for all three test strengths, and then added their matchings of *MGs*, and the detected faults, to Figs. 5 and 6, respectively. In each of these figures, for three testing strengths, i.e., 2-way, 3-way, and 4-way, we reported the average results over 30 repeated runs obtained by each approach for each subject. To make these figures and the figures in the following experiments more readable, a base 10 logarithmic scale is used for *Y* axis.The raw data of these results (as well as the results of the following experiments) has been included in the Appendix, available in the online supplemental material.

When comparing COMER and CT, it can be first observed that COMER is superior to CT at the matchings of *MGs*, by average factors of 90.9, 36.8, and 100.1, respectively, for testing strengths 2, 3, and 4, as shown in Fig. 5 (The average factor is 75.9 for all three testing strengths). Regarding fault detection, COMER also outperforms CT by increasing the number of detected faults by factors of about 18.2, 5.0, and 10.8, on average, for testing strengths 2, 3, and 4, as shown in Fig. 6 (The average factor is 11.3 for all three testing strengths). These results provide clear evidence that COMER is more effective than CT at the handling of the automated oracle problem. On the other hand, with respect to the number of test cases, as shown in Fig. 4, COMER only increases by about 11.2%, 10.6%, and 1.2%, respectively, when compared with CT for testing strengths 2, 3, and 4, which are very trivial. These results show COMER is also very efficient at improving the matchings of *MGs* and fault detection.

When comparing tri-MCT and CT, a similar observation can be made with respect to *MG* matchings and fault detection. Particularly, tri-MCT outperforms CT by increasing the number of *MG* matchings by average factors 415.9, 273.5, and 465.5, respectively (The average factor is 385.0 for all three testing strengths), and the detected faults by average factors 27.3, 5.5, and 12.0, respectively (The average factor is 14.9 for all three testing strengths), for testing strengths 2, 3, and 4. Combining the comparisons between tri-MCT and CT, and between COMER and CT, it can be concluded that considering metamorphic relations is beneficial when generating test cases, since it can improve the automated fault detection rate by increasing the chances of detecting the violation of test oracles.

When comparing COMER and tri-MCT, not surprisingly, we can observe that tri-MCT is superior at the matchings of *MGs*, of which the numbers increased by factors about 6.3, 4.6, and 3.5, respectively, for testing strengths 2, 3, and 4. The reason why tri-MCT outperformed COMER at matching the *MGs* is that it tried to generate a follow-up test case for every test case in a covering array. Obviously, this strategy is not free. As shown in Fig. 4, the numbers of test cases generated by tri-MCT are about 93.4%, 94.9%, and 101.5%, respectively, larger than those of COMER, for testing strengths 2, 3, and 4 (On average, COMER reduces about 44.0% of generated test cases for all three testing strengths).

Furthermore, more matchings of *MGs* do not necessarily lead to a better fault detection result. As shown in Fig. 6, the fault detection results of COMER and tri-MCT follow a very similar distribution. In fact, the differences between the average number of detected faults of these two approaches are less than 1.14, 0.44, and 0.41, respectively, for 2-way, 3-way, and 4-way testing strengths, which are very trivial when compared to the total number of faults they can detect. Considering that COMER achieves nearly the same fault detection rate by using only about 56% test cases when compared to tri-MCT, we believe the optimization of COMER, i.e., integrating CT and MR into a tightly coupled optimized mode, is significant, especially when the SUT is of large scale.

With respect to the Random Testing, one observation is that it has a poor performance in terms of the *MG* matchings and the number of fault detection. Note that among the three sizes, RT performed best when using the same size as tri-MCT. This is because tri-MCT generated the largest number of test cases. Hence, with this size of test cases, RT obtained more chances to match the metamorphic relations, and consequently, detect more faults. However, even with the largest number of test cases, RT still performed poorly when compared to the other three approaches. In fact, there are 24, 22, and 17, out of 31 subjects that RT had 0 matchings, and 26, 25, and 24, out of 31 subjects, RT detected 0 faults, for testing strengths 2, 3, and 4, respectively, when generated the same size of test cases as tri-MCT. For the remaining subjects, the numbers of *MG* matchings and faults obtained by RT are still maintained at a low level. Therefore, compared to the other three approaches, the decrease is significant in terms of the matchings of *MG* and the detection of faults. The reason for the ineffectiveness of RT is that it does not consider the abstract input parameters of the system, resulting in poor searching through the input space of the programs under test. Therefore, although constructing such an abstract input space is time-consuming, it is worthwhile for that its high effectiveness in fault detection.

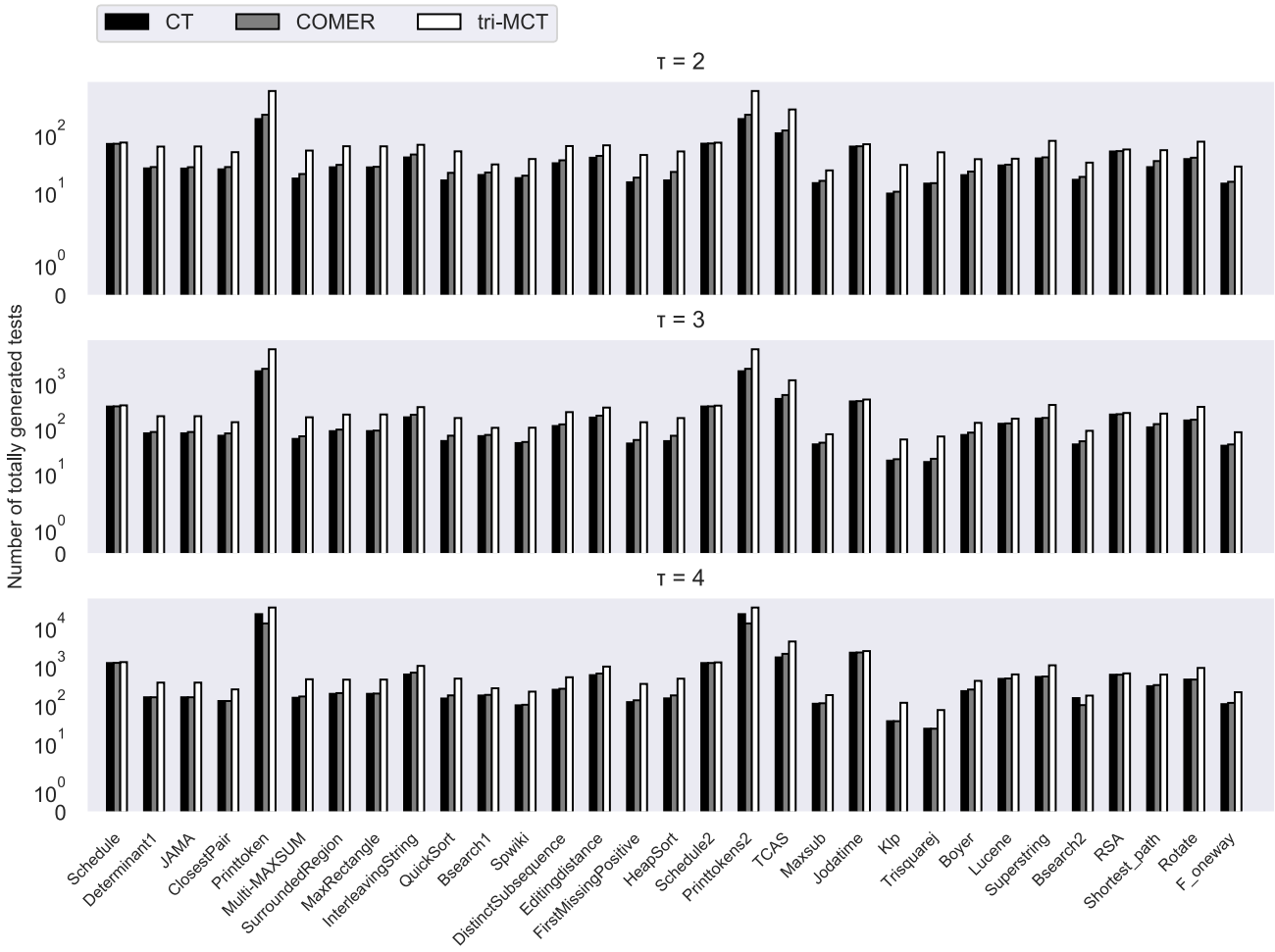



Fig. 4. The number of generated test cases by CT, COMER, tri-MCT.

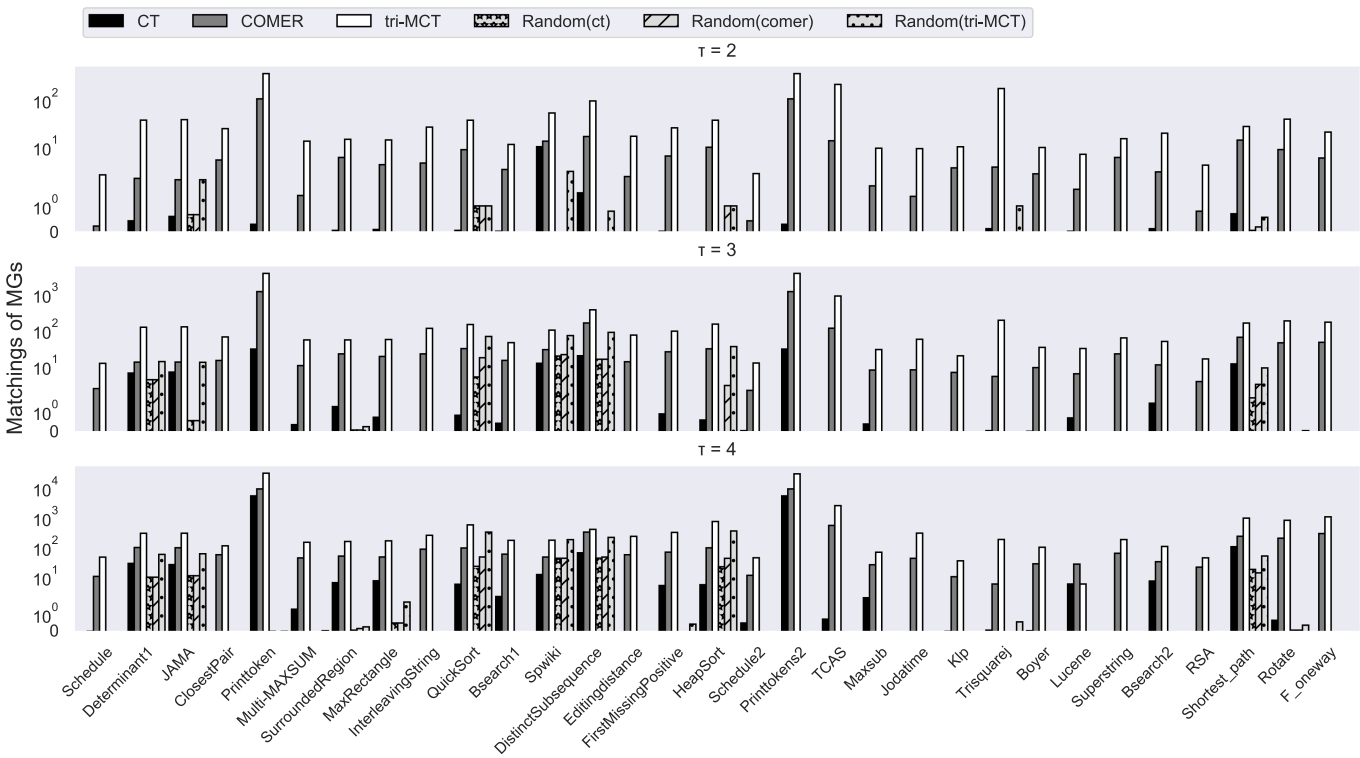



Fig. 5. The matchings of MGs by CT, COMER, tri-MCT, and RT.

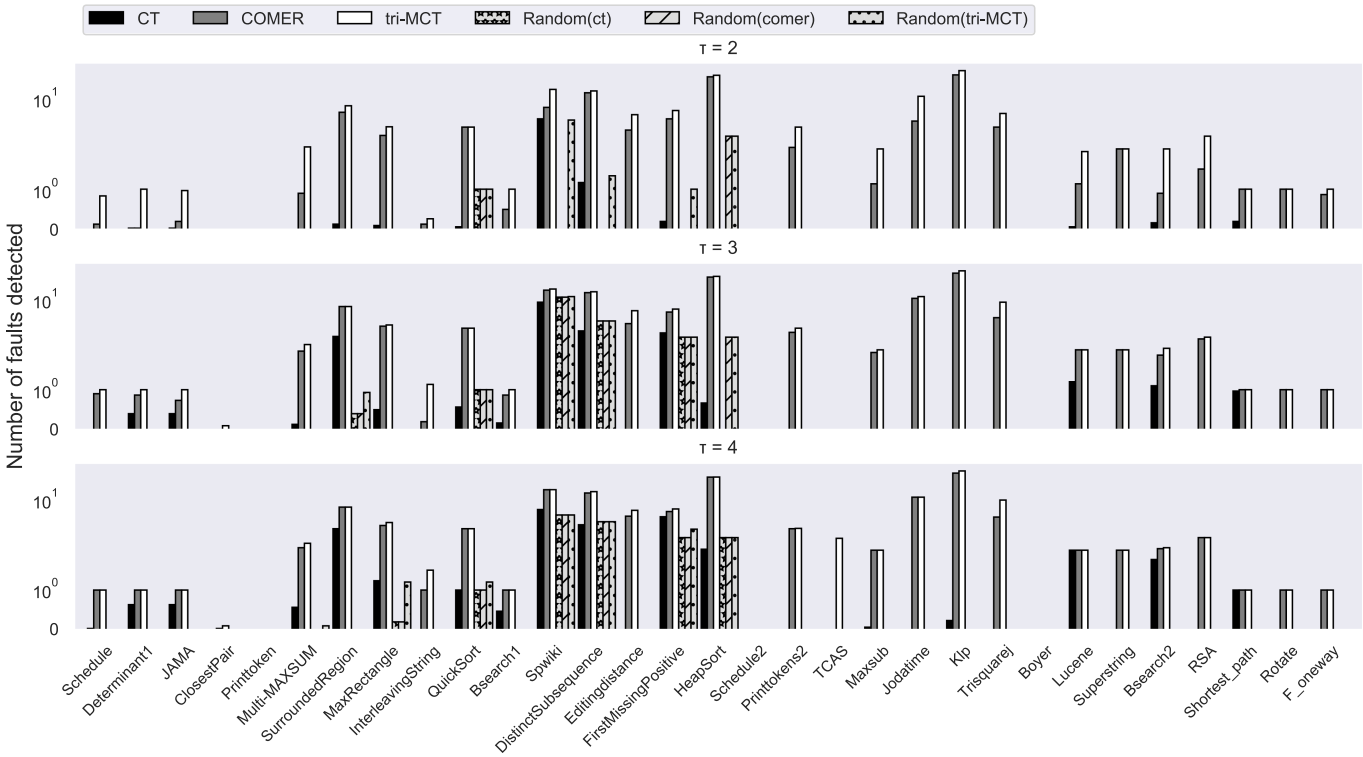



Fig. 6. The number of detected faults by CT, COMER, tri-MCT, and RT.

**Finding 1.** Using metamorphic relations and constructing abstract input space are beneficial to automatically detect faults. Further, COMER is effective at improving the number of matchings of *MGs* and the fault detection rate, while remaining a relatively small size of testing cases.

### 5.2 RQ2: The Time Cost of the Approaches

The second research question aims at investigating the time cost of these approaches. We reported the average time consumed to generate abstract test cases and concrete test cases in Figs. 7 and 8, respectively, for approaches COMER, CT, tri-MCT, and RT. Note that RT does not need to generate abstract test cases; hence, we only reported its concrete test generation time. In this experiment, RT generated the same number of concrete test cases as tri-MCT, which is larger than COMER and CT.

With respect to the time cost of abstract test generation, we have: tri-MCT $>$ COMER $>$ CT, for most subjects. One exception is subject *Lucene* , on which COMER takes less time than CT. On average, COMER spends about 486.9%, 410.2%, and 363.9% more time than CT at abstract test generation, for testing strengths 2, 3, and 4, respectively, while tri-MCT consumes about 47.6%, 57.6%, and 78.9% more time than COMER. This is as expected, because both tri-MCT and COMER need additional computation time for dealing with metamorphic relations, while CT does not. Additionally, COMER is faster than tri-MCT because COMER does not need to generate as many test cases as tri-MCT. The gap between the time cost of COMER and CT is much larger than that between tri-MCT and COMER. This suggests that using the constraint solver to generate follow-ups is time-consuming when compared to generating covering arrays.

With respect to the time costs of concrete test generation, it can be first observed that, for all the three approaches, i.e., COMER, CT, and tri-MCT, the time costs are much smaller than that of abstract test generation for all the subjects. In fact, the time cost of concrete test generation is no more than 8s, while the maximal time cost of abstract generation is about 662.03s. The reason why the concrete test generation is much more efficient is that at this stage, all the approaches do not need to take care of the $\tau$-way coverage, which is the most time-consuming part for covering array generation. Regarding COMER, CT, and tri-MCT, a second observation is that CT is the most efficient at concrete test generation, followed by COMER, and tri-MCT is the last one, for most subjects. On average, COMER consumes about 113.6%, 131.4%, and 160.8% more time than CT at concrete test generation, for testing strengths 2, 3, and 4, respectively, while tri-MCT consumes about 74.4%, 60.2%, and 91.6% more time than COMER. This result is similar to that of abstract test generation.

With respect to RT, it shows a very efficient result. When compared to CT, RT only increases by 125.18%, 154.97%, and 160.46%, respectively, for test strengths 2, 3, and 4. Considering that RT needs larger test sets (116.81%, 117.94%, and 103.46% larger than CT, for 2-way, 3-way and 4-way), RT is competitive to CT in terms of the efficiency of concrete test generation. Combining the fact that RT does not need to generate abstract test cases, it can conclude that RT is the most efficient approach among all the approaches. This is as expected, since RT does not need to take care of $\tau$-way coverage and metamorphic relation satisfaction.

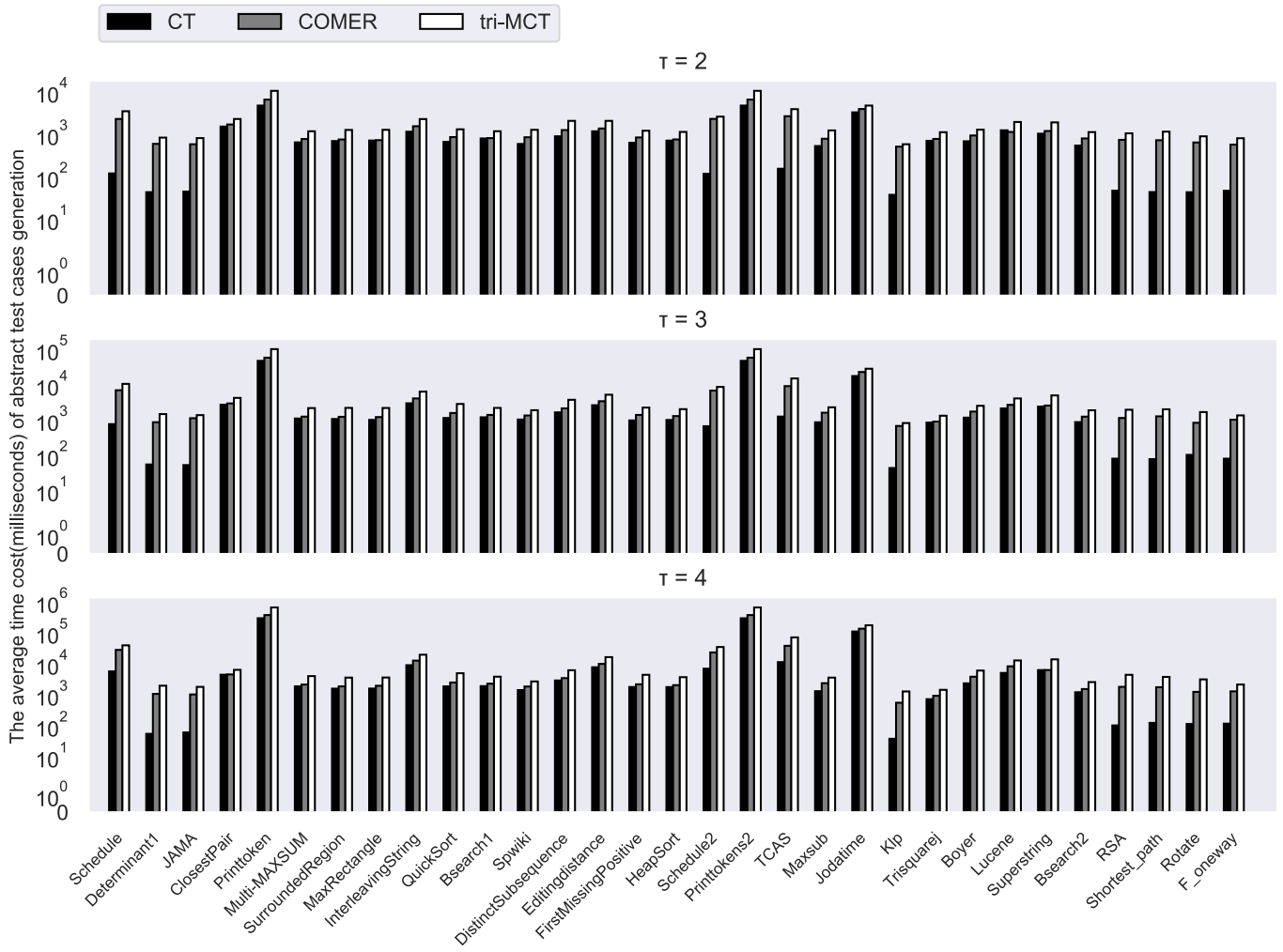



Fig. 7. The average time cost (milliseconds) of abstract test cases generation for COMER, CT, and tri-MCT.

Since the difference between CT and COMER in terms of the time cost is that COMER additionally utilized a solver to get follow-ups, it is desired to know how the constraint solver contributed to the total cost of COMER. Therefore, in Fig. 9, we reported the average number of times the solver succeeded and failed to get follow-ups for each subject.

One observation from Fig. 9 is that with the increase of the number of test cases, the number of successfully and unsuccessfully calling solvers both increase. Particularly, for the same subject, the higher the testing strength, the larger the number of generated test cases, and the more successful and unsuccessful solver-calls. Also, for the subjects with larger input space, e.g., *Printtoken*, the number of solver-calls is also larger than others. Second, we can observe that there are 22 out of 31 subjects on which the number of successful solver-calls is larger than unsuccessful solver-calls, and 9 subjects on which the number of unsuccessful solver-calls is larger. The specific gaps between successful and unsuccessful solver-calls have no clear trend, which vary among subjects. Considering that utilizing solver to get follow-ups is actually to solve constraint satisfaction problems, we believe the rate of success with respect to getting a follow-up depends on the nature of the subject itself, e.g., how difficult the metamorphic relations can be satisfied of these subjects.

**Finding 2.** With respect to the time cost, RT is the most efficient approach, followed by CT, COMER, and tri-MCT. It suggests that the $\tau$-way combination covering and metamorphic relation satisfaction is useful but not free.

### 5.3 RQ3: The Loss in Fault Detection by the Mere use of MR

The third research question seeks to figure out the loss in fault detection of COMER by mere use of MR when compared with the same strategy but with using practically *optimal* oracles. Here, the practically optimal oracles tell the *pass* or *fail* for **each** of the generated test cases. In order to give such an *optimal* oracle, we need to utilize a completely correct version of the subject under testing. After that, we can tell the *pass* or *fail* for a test case of a faulty version by checking whether the outcome of this test case is equal to that of the correct version.

To conduct a fair evaluation and answer the research question, we compared the fault detection results of two approaches with the *same* test cases (both are generated by COMER) but with different correctness determining strategies, i.e., one is checking whether one metamorphic relation is violated (the original strategy of COMER), while the other is

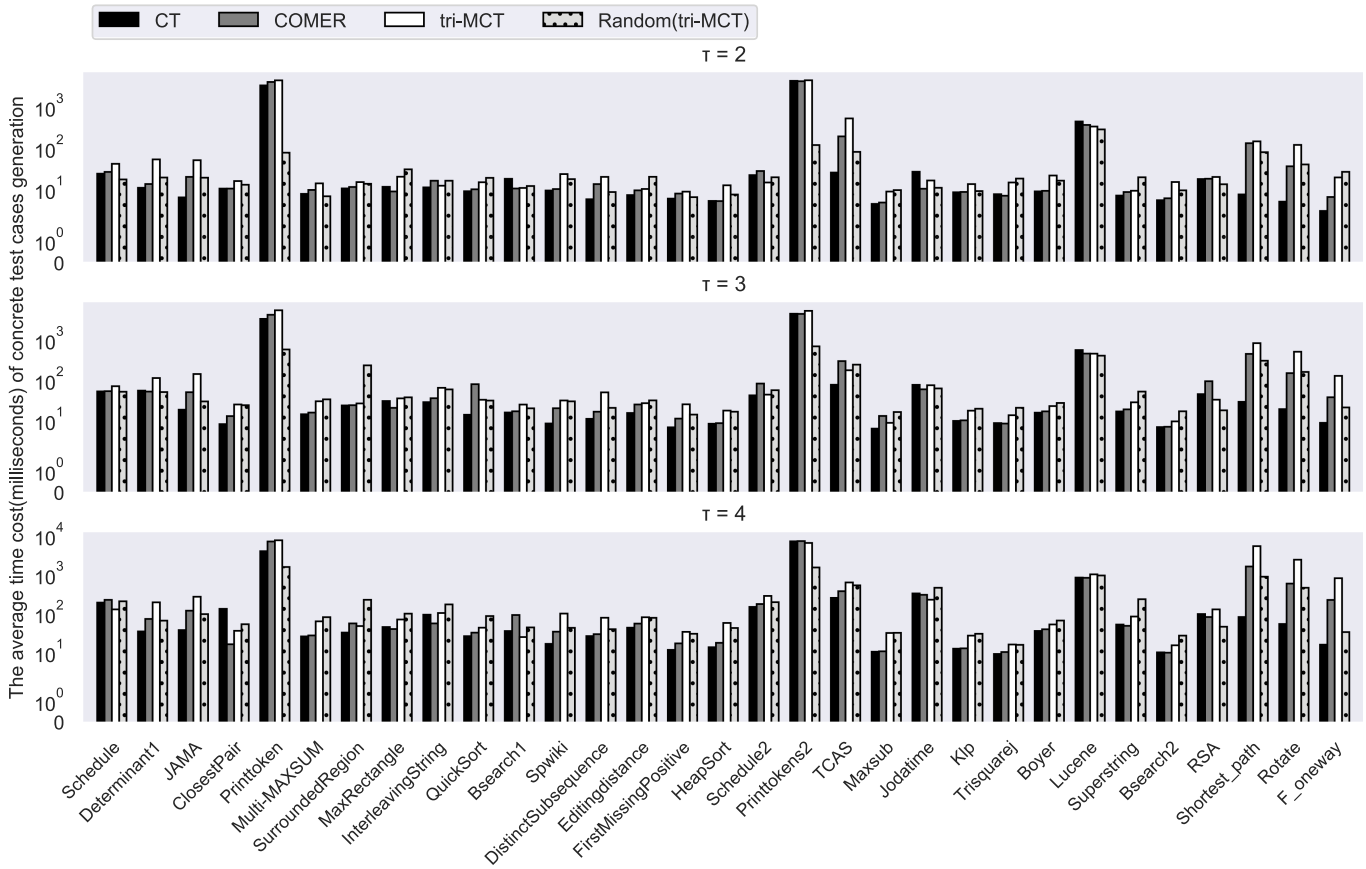



Fig. 8. The average time cost (milliseconds) of concrete test cases generation for COMER, CT, tri-MCT, and RT.

querying the *optimal* oracles. Since one correct version of the SUT is needed in this comparison, the subjects used in this evaluation only include the ones with seeded faults in Table 5.

Results are shown in Fig. 10. In this figure, we reported the average numbers of faults that are detected by the strategy of utilizing metamorphic relation and using optimal oracles, respectively, for three testing strengths.

One obvious observation from this figure is that by merely utilizing metamorphic relation, COMER cannot detect all the faults in these subjects. In fact, for subjects *Schedule2*, *TCAS*, and *Boyer*, COMER did not detect any fault. This result is as expected because a metamorphic relation is only a partial property of the SUT, which is not the full specification of how the SUT works. As a result, the satisfaction of an MR doesn't mean the correctness of the SUT(false negative may exist). Yet, the average fault detection of COMER with using MR reached about 34.2%, 44.7%, and 47.5%, for testing strengths 2, 3, and 4, respectively, when compared with using optimal oracles. Considering that the *optimal* oracle is very expensive and usually not available (and may not exist at all, e.g., the untestable programs [89]), this fault detection result of COMER with using MR is acceptable.

Another observation is that the number of detected faults varies among subjects. For example, COMER omitted all the faults for subjects *Schedule2*, *TCAS*, and *Boyer*, but detected all the faults for subject *Superstring*. For the remaining subjects at testing strength 2, there are 6 subjects at which COMER has detected over or equal to 50% of the faults, while 13 subjects less than 50%. For testing strengths 3 and 4, there are both 9 subjects at which COMER has detected over or equal to 50% of the faults, while 10 subjects less than 50%.

The third observation is that with the increase of testing strength, the number of detected faults by merely utilizing metamorphic relation tends to be relatively stable. In fact, when the test strength increases from 2-way to 3-way, COMER detected about 30% more faults, while when the test strength increases from 3-way to 4-way, it only detected about 6% more faults.

This result shows that the capability of the fault detection of COMER mainly lies in itself (e.g., the types of metamorphic relations that are utilized) when the testing strength is more than 2. Specifically, after further investigation, compared to the input rules of an MR, the output rules have a larger influence on the fault detection results. Further, the more specific the output rules of a metamorphic relation, the easier COMER will detect faults. For example, if a metamorphic relation shows that the outputs between source and follow-up test case should satisfy an equation (e.g., a simple equal relation), it tends to detect more faults. On the other hand, if the output rule of an MR is less stringent, e.g., inequality relation, it is less helpful for fault detection. More investigation about the relation between the types of MR and the fault detection rate can be found in the results in RQ4.

It is worth noting again that, on average, COMER achieved a fault detection rate of about 42% when compared with using optimal oracles, but it does not mean that COMER is extremely disadvantageous and makes no strong contribution in the CT community. This is because:

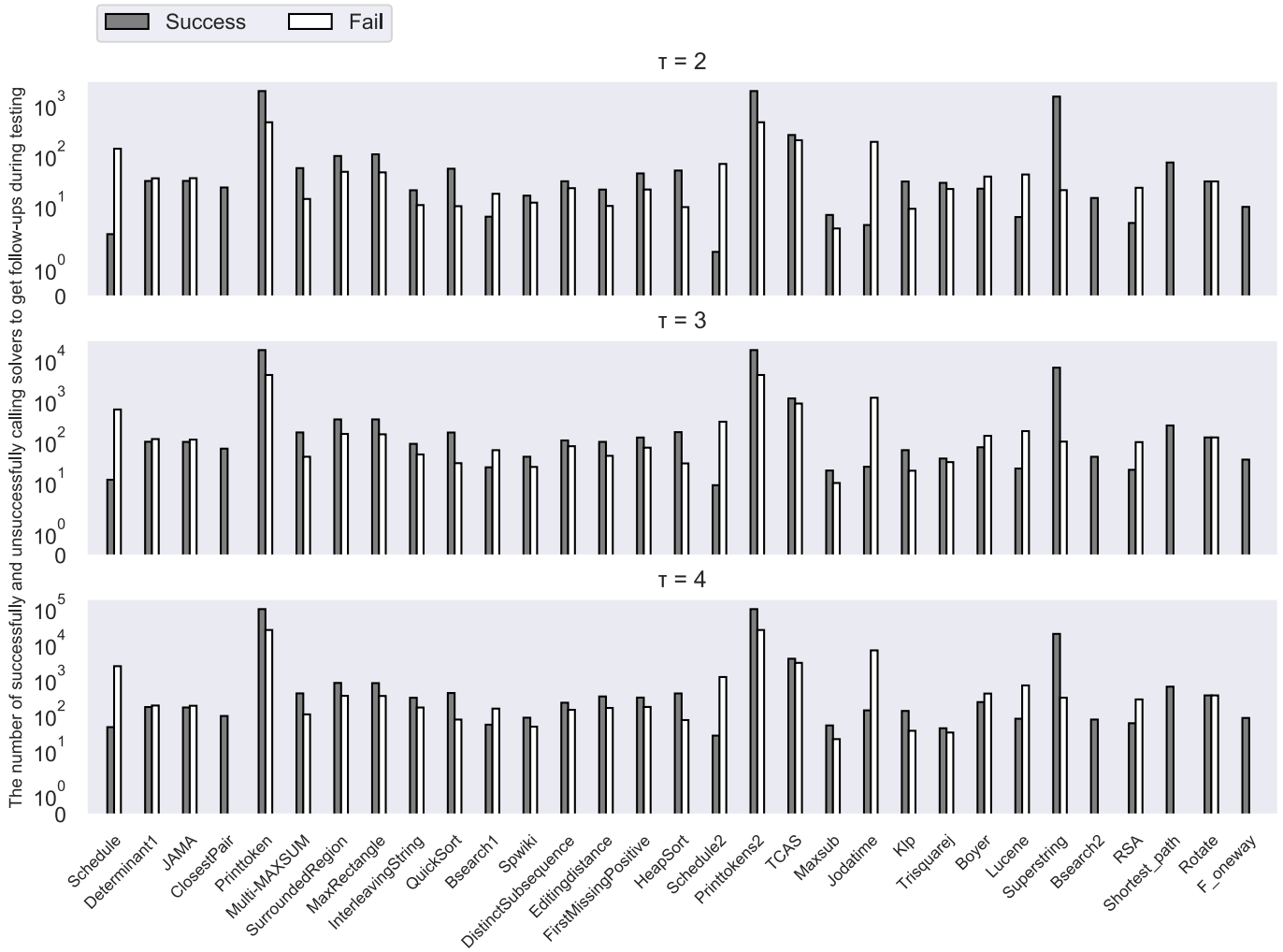



Fig. 9. The number of successfully and unsuccessfully calling solvers to get follow-ups during testing.

- First, as has been clarified in this paper, the automated oracle still remains an open problem and a pressing challenge in the CT community. Our research is the first attempt in this community to systematically solve this problem by taking into account the metamorphic relations during test generation.
- Second, our error detection effectiveness is not extremely low. In fact, traditional CT and random testing only reached about 8.24% and 5.76%, respectively, when compared to the approach using optimal oracles. In comparison with them, our approach COMER improved by about 409.7% and 629.2%, respectively.
- Last, but not least, the optimal approach is a very ideal approach, which assumes the existence of a completely correct version of a SUT. Such an assumption is very strong and is not practical. In fact, in our experiments, subjects *Schedule*,*Determinant1*, *JAMA*, *ClosestPair*, *Printtoken*, *Shortest_path*, *Rotate*, and *F_oneway*, do not have such correct versions. Therefore, the average 42% fault detection rate does not mean our approach is extremely disadvantageous.

**Finding 3.** By merely utilizing metamorphic relation, COMER achieved about a 42% fault detection rate when compared with using optimal oracles. The number of detected faults varies among subjects but remains stable when the testing strength is larger than 2.

### 5.4 RQ4: The Impact of the Features of Metamorphic Relations

In the previous study, we observed that the fault detection result varies among subjects. Since the fault detection is directly related to the metamorphic relation, it is desired to investigate the correlations between the features of the metamorphic relations and fault detection, which can provide practical guidelines when applying COMER. To answer the question, we first identified three main features of MRs that may have influences on the performance of COMER, which are listed in Table 7, along with their brief introductions. Among these features, the first two features indicate the degree of difficulty that a source and a follow-up test case can be generated according to the input rules of

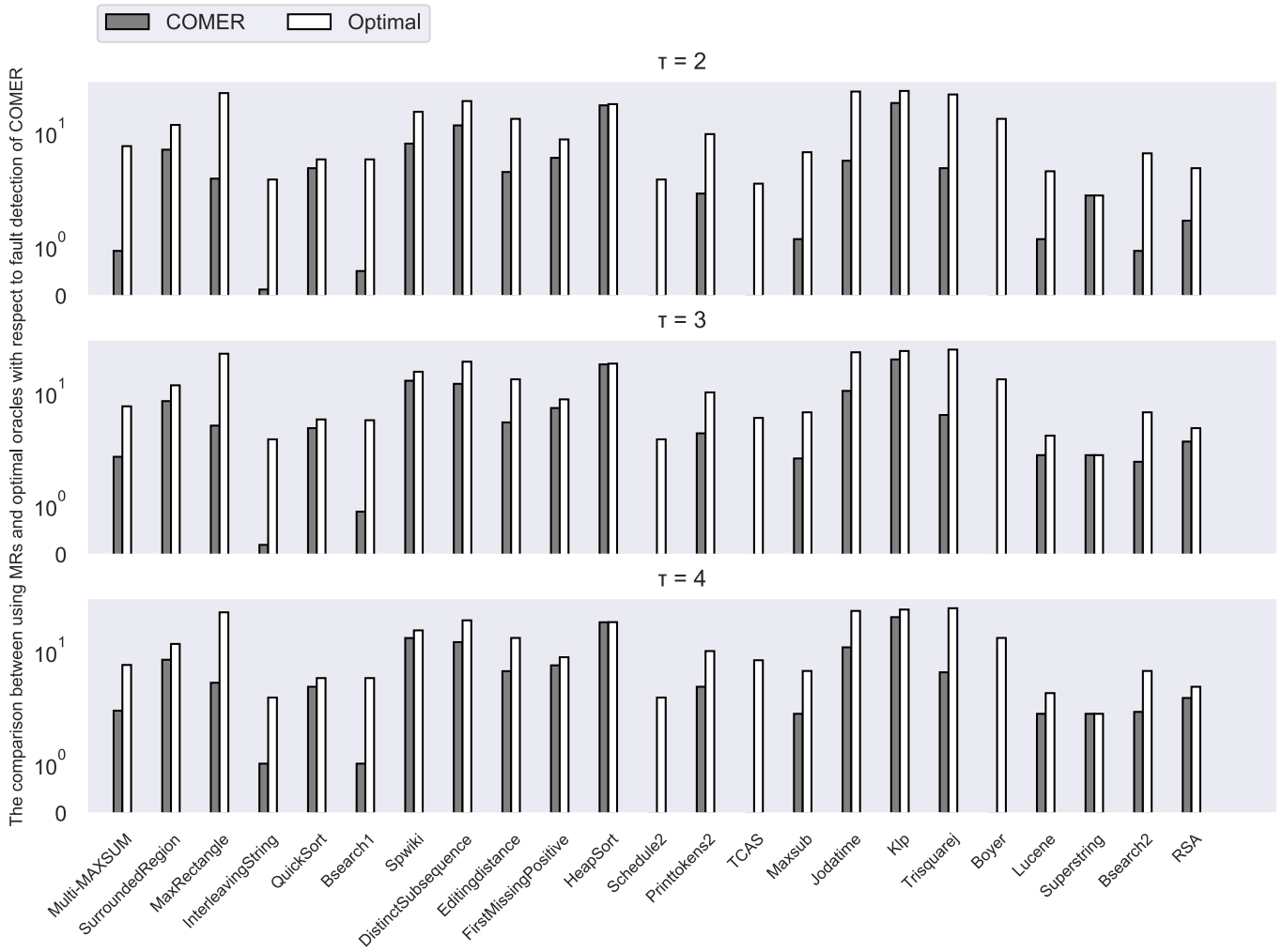



Fig. 10. The comparison between using MRs and optimal oracles with respect to fault detection of COMER.

an MR, while the last feature indicates the degree of difficulty that the output rules of an MR can be satisfied by the outputs of a source and a follow-up test case. Then, we measured the correlations between each feature and two metrics, i.e., the number of *MG* matchings, and the number of detected faults, respectively. We used the Pearson correlation coefficient as effect size and its corresponding p-value for the significance test.

Regarding the correlation between different features and the number of *MG* matchings, we reported the results in Fig. 11, with three subfigures for strengths 2, 3, and 4. Each box in the box plot represents the distribution of correlation coefficients between a feature and the number of *MG* matchings obtained by COMER, for each run at which that feature's correlation was statistically significant ($p < 0.05$). The closer the correlation coefficient is to 0, the weaker the relationship between the feature and the *MG* matching. We also reported the percentage of cases (over 30 runs) in which a feature's correlation was statistically significant in the parentheses of its label in the horizontal axis.

From Fig. 11, it can be easily observed that the first two features are moderately correlated to the *MG* matching, with the medium correlation coefficients ranging from about 0.32 to 0.39 for 2-way to 4-way testing strengths, respectively, while the last feature is negligibly correlated to the *MG* matching (the correlation coefficient is 0 for all three testing strengths). The reason for the negligible relation of the last feature is that the last feature does not affect the input form of the test cases. As shown in Algorithm 1, COMER only takes into account the input rules of MRs during the test generation, while the output rule is used to check the outputs of these test cases, which is not related to the matchings of the *MGs*.

With respect to the first two features, although they indicate the difficulties of the generation of sources and follow-ups, the results do not show a very high correlation (larger than 0.7) between them and the matchings of *MGs* as expected. We believe the reason for this result is that there exist some other factors during the test generation that may affect the results, e.g., the IPMs of the subjects.

Regarding the correlation between different features and the number of detected faults, we reported the results in Fig. 12. From this figure, it can be first observed that the correlation between the first two features and the number of detected faults is very weak. Particularly, when $\tau = 3$, our experimental data shows no correlation between the feature 'Follow Generate' and fault detection, i.e., the correlation coefficient is 0. Hence, there is no output in this figure for 'Follow Generate' when $\tau = 3$. In fact, when $\tau = 2$ and 4, the coefficients between 'Follow Generate' and fault detection are also very small (about -0.07 and 0.12, very close to 0).

TABLE 7
The Investigated Features of Metamorphic Relations

| Feature | Short Description |
|---|---|
| Source Generate | Given one MR, the percentage of the test cases that can be treated as source test cases among all the possible test cases. |
| Follow Generate | Given one MR, the percentage of the test cases that can be treated as follow-up test cases among all the possible test cases. |
| Output Match | Given one MR, the degree of the difficulty that its output rule can be satisfied. In this study, the degrees of the difficulty are classified into 5 main levels (from easy to difficult):<br>1) The "unequal" relation between two outputs with single values<br>2) The "equal" relation between two outputs with single limited values, e.g., enumerated type<br>3) The "equal" relation between two outputs with a set of limited values<br>4) The "equal" relation between two outputs with single unlimited values (e.g., float number)<br>5) The "equal" relation between two outputs with a set of unlimited values |

These results are consistent with each other. Combining the fact that the first two features are moderately correlated to the number of *MG* matchings, one implication is that a high number of *MG* matchings does not necessarily lead to a high number of detected faults. This implication is consistent with the results shown in the RQ1, where tri-MCT generated more *MGs*, but didn't significantly detect more faults than COMER.

With respect to the last feature, there exists a modest correlation between this feature and the number of detected faults, with the medium coefficients ranging from about 0.25 to 0.27. This result implies a tendency, although not very strong, that the more difficult the output rule of an MR

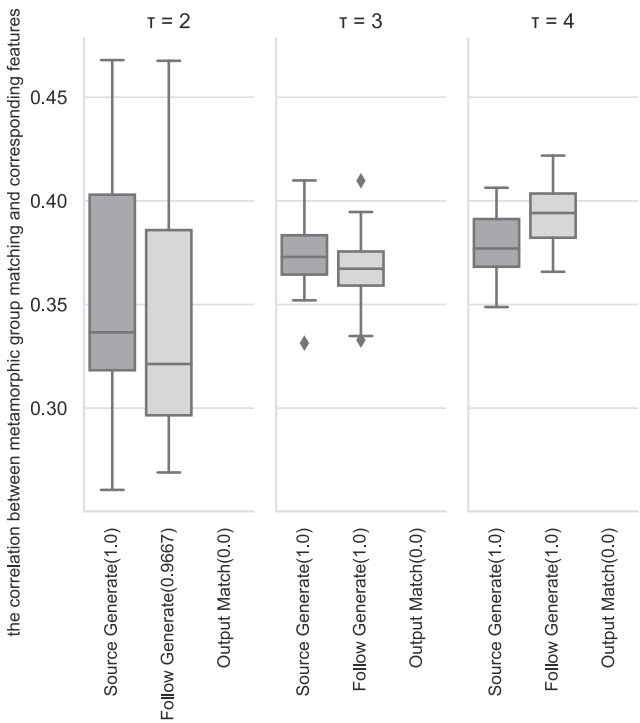



Fig. 11. The correlation between different features and the matchings of metamorphic groups.

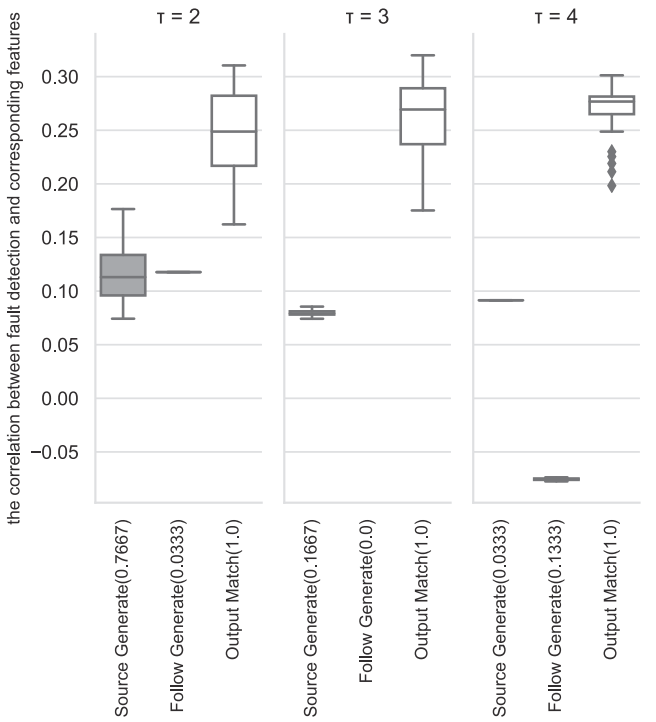



Fig. 12. The correlation between different features and the fault detection.

can be satisfied, the easier a fault can be detected. A supporting example for this implication is the application of COMER on subject *MaxRectangle* [55], of which the MR indicates that the output (a single value) of one matrix must be larger than the output of another matrix if the former matrix has one more row than the latter matrix (other cells of these two matrixes must be equal). Note that this rule is very easy to be satisfied (Level 1 in Table 7). This is because it does not require an exact value for the output of the follow-up test case (matrix). Instead, any number which is smaller than the output of the source test case is accepted. Hence, even though we generated two test cases with wrong outputs, we missed this fault because these outputs coincidently satisfied the output rules of the MR.

**Finding 4.** The degree of difficulty that the input rules of an MR can be satisfied is moderately correlated to the performance of COMER in terms of the number of *MG* matchings, while the degree of the difficulty that the output rules of an MR can be satisfied is modestly correlated to the number of detected faults.

### 5.5 Threats to Validity

The main threat to internal validity is the abstract input parameter models that are built for each subject. Different models, e.g., chosen parameters, value sets, and different metamorphic relations, may affect the results of the experiments. To reduce this threat, in terms of the IPM, for some of the subjects, we used the existing IPMs [30], [90]. For the remaining subjects without existing IPMs, we built IPMs by following the common CT modeling guidelines, i.e., to select possible parameter values that may affect the behavior of the SUT and identify the constraints between them [30]. With

respect to metamorphic relations, we also reused the MRs for some subjects of which the MRs have been found by previous studies [51], [55]. For the remaining subjects, we chose the common types of MRs, e.g., shift, reverse, increment, etc. [51], [54], [91] that are suggested by MT practices.

Threats to external validity mainly lie in the benchmark suites used in our experiments. To reduce this threat, we chose several benchmarks widely used in both CT and MT, among which 31 unique software subjects are finally extracted. These subjects vary in sizes, types, and input spaces. Besides, in addition to the seeded bugs, we explored the issue tracks for several subjects to identify the real bugs of these subjects. Our approach further identified previously unknown bugs. We believe these benchmarks and real bugs are representative and diversified enough for our evaluation.

## 6 Related Works

*CT Test Case Generation* . Test case generation is a major topic in the CT community. Various practical methods and tools have been proposed, which can be classified into three categories [1], [92]: mathematical methods [93], [94], [95], [96], [97], greedy methods [3], [5], [6], [72], and heuristic search methods [4], [7], [8], [47], [48]. Mathematical methods can guarantee the minimality of the number of test cases needed to be generated, but impose certain restrictions on input models, e.g., the number of values should be the same for all the parameters [98]. Greedy methods are very efficient but may generate more test cases than the other two methods [99]. Heuristic search methods are very time-consuming, but they can generate very small test sets. In fact, for many complex input models, the lower bound is obtained by heuristic search methods, including the recently proposed methods [75], [76], [100]. In this paper, since we do not focus on minimizing the covering array, we choose the greedy method, ARS [72], [73], [74], to generate test cases for its simplicity and efficiency. Based on this approach, we can easily extend the metamorphic group matching strategies during the test case generation. Note that using ARS even benefits the overall CT process in our experiments. This is because ARS is likely to generate a covering array of larger size, which also offers more chances to match the metamorphic rules than the covering arrays of smaller size.

*The Studies in CT Which Focus on Oracle Problem.* In spite of the fact that some studies of CT have mentioned how to obtain oracles in their specific testing scenarios, very few focus on tackling the automated oracle problem in CT in a systematical way [11]. Inspired by the oracle classification criteria proposed by Barr *et al.* [12], Kruse [36] also classified the oracles used in CT into several common categories. Kuhn *et al.* [101] utilized the equivalence class as the oracle. Specifically, the generated test cases must have the same output as long as their input data are in the same equivalence class. Ukai *et al.* [102] generated a new test case by joining previously generated test sets, such that the oracle can be reused and manual efforts are reduced. Wotawa *et al.* [103] utilized metamorphic testing to test the logic-based non-monotonic reasoning system. In their study, they used a meta-relation which tests with same parameter values but different sequences must have the same results. Further, to enhance the test generation, they utilized CT to generate covering arrays for different sequences. Note that in that work, they combined the MT with CT in a totally different way from our work. Specifically, it uses CT to enhance the test set generation for MT, while our approach is to use the MT to enhance the automated oracle supporting for CT.

*Applications of MT to Other Techniques.* As a well-established and successful testing method, MT has been integrated into other software engineering techniques. Chen *et al.* [104] first combined metamorphic testing with fault-based testing. Their approach alleviated the oracle problem both for real and symbolic inputs of fault-based testing. Dong *et al.* [105] improved the efficiency of evolutionary testing by integrating the metamorphic relation in the fitness function such that it can help to accelerate evolutionary convergence. Xie *et al.* [57], [58] extended the Spectrum-based Fault Localization (SBFL) with metamorphic testing, such that SBFL does not need an oracle for each test case. Jiang *et al.* [106] developed a new Automated Program Repair (APR) paradigm by integrating MT with APR. The repair effectiveness of their technique was empirically demonstrated to be comparable to the original APR with given oracles. Xu *et al.* [107] augmented an image classification algorithm using two metamorphic relations. Empirical studies showed their approach is superior to the original classifier in terms of accuracy. For more details on the studies which focus on integrating MT into other techniques, readers are referred to two recent surveys [39], [40].

## 7 Conclusion

This paper focused on the less well-investigated, but yet very important, oracle problem in combinatorial testing. To get rid of tedious manual efforts and to extend CT to more widely testing scenarios, we proposed a novel CT methodology, i.e., COMER, to enhance the CT with automated oracle supports. The basic idea of COMER is to generate as many *MGs* as possible, while meeting the t-way coverage. The correctness of the SUT can be automatically determined by verifying whether the outputs of these *MGs* violate their MRs, and hence greatly alleviate the automated oracle problem. To evaluate the effectiveness and efficiency of COMER, we built a software repository with 31 real-world software subjects, along with the corresponding abstract input parameter models, programs for converting abstract test cases to concrete test cases, specific metamorphic relations, and automated test execution scripts. The empirical studies based on this software repository show traditional CT suffers from the automated oracle problem, while COMER can improve the *MG* matching by an average factor of 75.9 and also improve the failure detection rate by an average factor of 11.3. These results evidently show that COMER is a potential solution for the automated problem in CT.

## References

[1] C. Nie and H. Leung, "A survey of combinatorial testing," *ACM Comput. Surv.*, vol. 43, no. 2, pp. 11:1–11:29, Feb 2011.

[2] D. R. Kuhn, R. Kacker, and Y. Lei, "Practical combinatorial testing," *NIST Special Pub.*, vol. 800, pp. 800–142, 2010.

[3] D. M. Cohen, S. R. Dalal, M. L. Fredman, and G. C. Patton, "The AETG system: An approach to testing based on combinatorial design," *IEEE Trans. Softw. Eng.*, vol. 23, no. 7, pp. 437–444, Jul. 1997.

[4] M. B. Cohen, C. J. Colbourn, and A. C. Ling, "Augmenting simulated annealing to build interaction test suites," in *Proc. 14th Int. Symp. Softw. Reliability Eng.*, 2003, pp. 394–405.

[5] Y. Lei, R. Kacker, D. R. Kuhn, V. Okun, and J. Lawrence, "IPOG/IPOG-D: Efficient test generation for multi-way combinatorial testing," *Softw. Testing, Verification Rel.*, vol. 18, no. 3, pp. 125–148, 2008.

[6] R. C. Bryce and C. J. Colbourn, "The density algorithm for pairwise interaction testing," *Softw. Testing, Verification Rel.*, vol. 17, no. 3, pp. 159–182, 2007.

[7] Y. Jia, M. B. Cohen, M. Harman, and J. Petke, "Learning combinatorial interaction test generation strategies using hyperheuristic search," in *Proc. 37th Int. Conf. Softw. Eng.*, 2015, pp. 540–550.

[8] J. Lin, C. Luo, S. Cai, K. Su, D. Hao, and L. Zhang, "Tca: An efficient two-mode meta-heuristic algorithm for combinatorial test generation (t)," in *Proc. 30th IEEE/ACM Int. Conf. Automated Softw. Eng.*, 2015, pp. 494–505.

[9] B. J. Garvin, M. B. Cohen, and M. B. Dwyer, "Evaluating improvements to a meta-heuristic search for constrained interaction testing," *Empirical Softw. Eng.*, vol. 16, no. 1, pp. 61–102, 2011.

[10] D. R. Kuhn, R. Kacker, and Y. Lei, "Automated combinatorial test methods: Beyond pairwise testing," *Crosstalk J. Defense Softw. Eng.*, vol. 21, no. 6, pp. 22–26, 2008.

[11] R. Tzoref-Brill, "Chapter two - Advances in combinatorial testing," in *Advances in Computers*, A. M. Memon, Ed. New York, NY, USA: Elsevier, 2019, pp. 79–134.

[12] E. T. Barr, M. Harman, P. McMinn, M. Shahbaz, and S. Yoo, "The oracle problem in software testing: A survey," *IEEE Trans. Softw. Eng.*, vol. 41, no. 5, pp. 507–525, May 2015.

[13] X. Li, R. Gao, W. E. Wong, C. Yang, and D. Li, "Applying combinatorial testing in industrial settings," in *Proc. IEEE Int. Conf. Softw. Qual., Rel. Secur.*, 2016, pp. 53–60.

[14] L. Hu, W. E. Wong, D. R. Kuhn, and R. N. Kacker, "How does combinatorial testing perform in the real world: An empirical study," *Empirical Software Eng.*, vol. 25, pp. 2661–2693, Apr. 2020.

[15] X. Deng, T. Wu, J. Yan, and J. Zhang, "Combinatorial testing on implementations of html5 support," in *Proc. IEEE Int. Conf. Softw. Testing, Verification Validation Workshops*, 2017, pp. 262–271.

[16] R. Krishnan, S. M. Krishna, and P. S. Nandhan, "Combinatorial testing: Learnings from our experience," *SIGSOFT Softw. Eng. Notes*, vol. 32, no. 3, pp. 1–8, May 2007.

[17] Z. Zhang, X. Liu, and J. Zhang, "Combinatorial testing on id3v2 tags of mp3 files," in *Proc. IEEE 5th Int. Conf. Softw. Testing, Verification Validation*, 2012, pp. 587–590.

[18] B. R. Sagi and R. Silvestrini, "Application of combinatorial tests in video game testing," *Qual. Eng.*, vol. 29, no. 4, pp. 745–759, 2017.

[19] X. Niu, C. Nie, J. Y. Lei, H. Leung, and X. Wang, "Identifying failure-causing schemas in the presence of multiple faults," *IEEE Trans. Softw. Eng.*, vol. 46, no. 2, pp. 141–162, Feb. 2020.

[20] I. Abal, C. Brabrand, and A. Wasowski, "42 variability bugs in the linux kernel: A qualitative analysis," in *Proc. 29th ACM/IEEE Int. Conf. Autom. Softw. Eng.*, New York, NY, USA, 2014, pp. 421–432.

[21] D. Jarman *et al.*, "Applying combinatorial testing to large-scale data processing at adobe," in *Proc. IEEE Int. Conf. Softw. Testing, Verification Validation Workshops*, 2019, pp. 190–193.

[22] W. Wang *et al.*, "A combinatorial approach to detecting buffer overflow vulnerabilities," in *Proc. IEEE/IFIP 41st Int. Conf. Dependable Syst. Netw.*, 2011, pp. 269–278.

[23] M. Albonico, S. D. Alesio, J. Mottu, S. Sen, and G. Sunyé, "Generating test sequences to assess the performance of elastic cloud-based systems," in *Proc. IEEE 10th Int. Conf. Cloud Comput.*, 2017, pp. 383–390.

[24] Y. Ledru, L. du Bousquet, O. Maury, and P. Bontron, "Filtering tobias combinatorial test suites," in *Fundamental Approaches to Software Engineering*, M. Wermelinger and T. Margaria-Steffen, Eds. Berlin, Germany: Springer, 2004, pp. 281–294.

[25] T. Triki, "Filtering and reduction techniques of combinatorial tests," Ph.D. dissertation, Dept. Comput. Sci., Université de Grenoble English Alpes, Grenoble, France, 2013.

[26] A. Calvagna, A. Fornaia, and E. Tramontana, "Random versus combinatorial effectiveness in software conformance testing: a case study," in *Proc. 30th Annu. ACM Symp. Appl. Comput., Salamanca, Spain*, R. L. Wainwright, J. M. Corchado, A. Bechini, and J. Hong, Eds. 2015, pp. 1797–1802.

[27] M. Al-Hajjaji, T. Thüm, M. Lochau, J. Meinicke, and G. Saake, "Effective product-line testing using similarity-based product prioritization," *Softw. Syst. Model.*, vol. 18, no. 1, pp. 499–521, 2019.

[28] R. Bartholomew, "An industry proof-of-concept demonstration of automated combinatorial test," in *Proc. 8th Int. Workshop Autom. Softw. Test*, 2013, pp. 118–124.

[29] D. R. Kuhn and V. Okun, "Pseudo-exhaustive testing for software," in *Proc. 30th Annu. IEEE/NASA Softw. Eng. Workshop*, 2006, pp. 153–158.

[30] L. S. G. Ghandehari, M. N. Bourazjany, Y. Lei, R. N. Kacker, and D. R. Kuhn, "Applying combinatorial testing to the siemens suite," in *Proc. 6th Int. Conf. Softw. Testing, Verification Validation Workshops* , 2013, pp. 362–371.

[31] L. Sh. Ghandehari, Y. Lei, R. Kacker, R. Kuhn, T. Xie, and D. Kung, "A combinatorial testing-based approach to fault localization," *IEEE Trans. Softw. Eng.*, vol. 46, no. 6, pp. 616–645, Jun. 2020.

[32] Z. Zhang and J. Zhang, "Characterizing failure-causing parameter interactions by adaptive testing," in *Proc. Int. Symp. Softw. Testing Anal.*, 2011, pp. 331–341.

[33] L. d. Bousquet, M. Delahaye, and C. Oriat, "Applying a pairwise coverage criterion to scenario-based testing," in *Proc. IEEE 9th Int. Conf. Softw. Testing, Verification Validation Workshops*, 2016, pp. 83–91.

[34] H. Felbinger, F. Wotawa, and M. Nica, "Mutation score, coverage, model inference: Quality assessment for t-way combinatorial test-suites," in *Proc. IEEE Int. Conf. Softw. Testing, Verification Validation Workshops*, 2017, pp. 171–180.

[35] N. Xintao, N. Changhai, Y. Lei, and A. T. S. Chan, "Identifying failure-inducing combinations using tuple relationship," in *Proc. IEEE 6th Int. Conf. Softw. Testing, Verification Validation Workshops*, 2013, pp. 271–280.

[36] P. M. Kruse, "Test oracles and test script generation in combinatorial testing," in *Proc. IEEE 9th Int. Conf. Softw. Testing, Verification Validation Workshops*, 2016, pp. 75–82.

[37] F. Medeiros, C. Kästner, M. Ribeiro, R. Gheyi, and S. Apel, "A comparison of 10 sampling algorithms for configurable systems," in *Proc. 38th Int. Conf. Softw. Eng.*, New York, NY, USA, 2016, pp. 643–654.

[38] T. Chen, S. Cheung, and S. Yiu, "Metamorphic testing: A new approach for generating next test cases," Dept. Comput. Sci., Hong Kong Univ. Sci. Technol., Hong Kong., *HKUST-CS98–01, Tech. Rep.* 1998.

[39] S. Segura, G. Fraser, A. B. Sanchez, and A. Ruiz-Cortés, "A survey on metamorphic testing," *IEEE Trans. Softw. Eng.*, vol. 42, no. 9, pp. 805–824, Sep. 2016.

[40] T. Y. Chen *et al.*, "Metamorphic testing: A review of challenges and opportunities," *ACM Comput. Surv.*, vol. 51, no. 1, pp. 1–27, Jan. 2018.

[41] J. Bach and P. Schroeder, "Pairwise testing: A best practice that isn't," in *Proc. 22nd Pacific Northwest Softw. Qual. Conf.*, 2004, pp. 180–196.

[42] M. B. Cohen, M. B. Dwyer, and J. Shi, "Constructing interaction test suites for highly-configurable systems in the presence of constraints: A greedy approach," *IEEE Trans. Softw. Eng.*, vol. 34, no. 5, pp. 633–650, Sep./Oct. 2008.

[43] C. Prud'homme, J.-G. Fages, and X. Lorca, *Choco Solver Documentation*, TASC, INRIA Rennes, LINA CNRS UMR 6241, COSLING S.A.S., 2016. [Online]. Available: http://www.choco-solver.org

[44] M. Hutchins, H. Foster, T. Goradia, and T. Ostrand, "Experiments on the effectiveness of dataflow- and control-flow-based test adequacy criteria," in *Proc. 16th Int. Conf. Softw. Eng.*, 1994, pp. 191–200.

[45] K. Shakya, T. Xie, N. Li, Y. Lei, R. Kacker, and R. Kuhn, "Isolating failure-inducing combinations in combinatorial testing using test augmentation and classification," in *Proc. IEEE 5th Int. Conf. Softw. Testing, Verification Validation*, 2012, pp. 620–623.

[46] L. Sh. Ghandehari, Y. Lei, R. Kacker, R. Kuhn, T. Xie, and D. Kung, "A combinatorial testing-based approach to fault localization," *IEEE Trans. Softw. Eng.*, vol. 46, no. 6, pp. 616–645, Jun. 2020.

[47] K. J. Nurmela, "Upper bounds for covering arrays by tabu search," *Discrete Appl. Math.*, vol. 138, no. 1, pp. 143–152, 2004.

[48] H. Wu, C. Nie, F. Kuo, H. Leung, and C. J. Colbourn, "A discrete particle swarm optimization for covering array generation," *IEEE Trans. Evol. Comput.*, vol. 19, no. 4, pp. 575–591, Aug. 2015.

[49] P. Hamill, *Unit Test Frameworks: Tools for High-Quality Software Development*. Newton, MA, USA: O'Reilly, 2009.

[50] H. Do, S. G. Elbaum, and G. Rothermel, "Supporting controlled experimentation with testing techniques: An infrastructure and its potential impact," *Empirical Softw. Eng., Int. J.*, vol. 10, no. 4, pp. 405–435, 2005.

[51] C. Murphy, G. E. Kaiser, L. Hu, and L. Wu, "Properties of machine learning applications for use in metamorphic testing," in *Proc. 20th Int. Conf. Softw. Eng. Knowl. Eng.*,vol. 8, 2008, pp. 867–872.

[52] J. Mayer and R. Guderlei, "An empirical study on the selection of good metamorphic relations," in *Proc. 30th Annu. Int. Comput. Softw. Appl. Conf.*, 2006, pp. 475–484.

[53] H. Zhang, L. Liu, and P. Zhang, "Predicting metamorphic relations based on path features," *J. Phys., Conf. Series*, vol. 1650, no. 3, 2020, Art. no. 032008.

[54] U. Kanewala, J. M. Bieman, and A. Ben-Hur , "Predicting metamorphic relations for testing scientific software: A machine learning approach using graph kernels," *Softw. Testing, Verification Rel.*, vol. 26, no. 3, pp. 245–269, 2016.

[55] H. Jin, Y. Jiang, N. Liu, C. Xu, X. Ma, and J. Lu, "Concolic metamorphic debugging," in *Proc. IEEE 39th Annu. Comput. Softw. Appl. Conf.*, 2015, pp. 232–241.

[56] X. Xie, J. W. K. Ho, C. Murphy, G. Kaiser, B. Xu, and T. Y. Chen, "Testing and validating machine learning classifiers by metamorphic testing," *J. Syst. Softw.*, vol. 84, no. 4, pp. 544–558, Apr. 2011.

[57] X. Xie, W. E. Wong, T. Y. Chen, and B. Xu, "Spectrum-based fault localization: Testing oracles are no longer mandatory," in *Proc. 11th Int. Conf. Qual. Softw.*, 2011, pp. 1–10.

[58] X. Xie, W. E. Wong, T. Y. Chen, and B. Xu, "Metamorphic slice: An application in spectrum-based fault localization," *Inf. Softw. Technol.*, vol. 55, no. 5, pp. 866–879, 2013.

[59] J. Mayer and R. Guderlei, "An empirical study on the selection of good metamorphic relations," in *Proc. 30th Annu. Int. Comput. Softw. Appl. Conf.*, USA, 2006, pp. 475–484.

[60] "Closest pair of points O(nlogn) algorithm - problem with some data in c++ implementation," Accessed Jul. 06, 2020. [Online]. Available: https://stackoverflow.com/questions/54000950/closest-pair-of-points-onlogn-algorithm-problem-with-some-data-in-c-implem

[61] Y. Lei, X. Mao, and T. Y. Chen, "Backward-slice-based statistical fault localization without test oracles," in *Proc. 13th Int. Conf. Qual. Softw.*, 2013, pp. 212–221.

[62] T. Y. Chen, F. -C. Kuo, L. Ying, and A. Tang, "Metamorphic testing and testing with special values," in *Proc. ACIS Int. Conf. Softw. Eng.*, vol. 8, 2004, pp. 128–134.

[63] Y. Cao, Z. Q. Zhou, and T. Y. Chen, "On the correlation between the effectiveness of metamorphic relations and dissimilarities of test case executions," in *Proc. 13th Int. Conf. Qual. Softw.*, 2013, pp. 153–162.

[64] A. Carzaniga, A. Goffi, A. Gorla, A. Mattavelli, and M. Pezzè, "Cross-checking oracles from intrinsic software redundancy," in *Proc. 36th Int. Conf. Softw. Eng.*, 2014, pp. 931–942.

[65] A. Barus, T. Y. Chen, D. Grant, F.-C. Kuo, and M. F. Lau, "Testing of heuristic methods: A case study of greedy algorithm," in *Proc. IFIP Central East Eur. Conf. Softw. Eng. Techn.*, 2008, pp. 246–260.

[66] G. Dong, T. Guo, and P. Zhang, "Security assurance with program path analysis and metamorphic testing," in *Proc. IEEE 4th Int. Conf. Softw. Eng. Serv. Scie.*, 2013, pp. 193–197.

[67] C. Murphy and G. Kaiser, "Empirical evaluation of approaches to testing applications without test oracles," Columbia Univ. Comput. Sci., New York, NY, Tech. Rep. CUCS-039-10, 2010. [Online]. Available: https://academiccommons.columbia.edu/doi/10.7916/D8BV7QGC/download

[68] F.-H. Su, J. Bell, C. Murphy, and G. Kaiser, "Dynamic inference of likely metamorphic properties to support differential testing," in *Proc. IEEE/ACM 10th Int. Workshop Autom. Softw. Test*, 2015, pp. 55–59.

[69] A. Gotlieb and B. Botella, "Automated metamorphic testing," in *Proc. 27th Annu. Int. Comput. Softw. Appl. Conf.*, 2003, pp. 34–40.

[70] C.-a. Sun, Z. Wang, and G. Wang, "A property-based testing framework for encryption programs," *Front. Comput. Sci.*, vol. 8, no. 3, pp. 478–489, 2014.

[71] P. *et al.*, "SciPy 1.0: Fundamental algorithms for scientific computing in Python," *Nat. Methods*, vol. 17, pp. 261–272, 2020.

[72] R. Huang, X. Xie, T. Y. Chen, and Y. Lu, "Adaptive random test case generation for combinatorial testing," in *Proc. IEEE 36th Annu. Comput. Softw. Appl. Conf.*, 2012, pp. 52–61.

[73] C. Nie, H. Leung, and K.-Y. Cai, "Adaptive combinatorial testing," in *Proc. 13th Int. Conf. Qual. Softw.*, 2013, pp. 284–287.

[74] H. Wu, C. Nie, J. Petke, Y. Jia, and M. Harman, "An empirical comparison of combinatorial testing, random testing and adaptive random testing," *IEEE Tran. Softw. Eng.*, vol. 46, no. 3, pp. 302–320, Mar. 2020.

[75] C. Luo *et al.*, "AutoCCAG: An automated approach to constrained covering array generation," in *Proc. IEEE/ACM 43rd Int. Conf. Softw. Eng.*, 2021, pp. 201–212.

[76] J. Lin, S. Cai, C. Luo, Q. Lin, and H. Zhang, "Towards more efficient meta-heuristic algorithms for combinatorial test generation," in *Proc. 27th ACM Joint Meeting Eur. Softw. Eng. Conf. Symp. Found. Softw. Eng.*, 2019, pp. 212–222.

[77] H. Liu, F. Kuo, D. Towey, and T. Y. Chen, "How effectively does metamorphic testing alleviate the oracle problem?," *IEEE Trans. Softw. Eng.*, vol. 40, no. 1, pp. 4–22, Jan. 2014.

[78] "Different algorithms giving different results in shortest_path," June 2020. [Online]. Available: https://github.com/scipy/scipy/issues/12424

[79] "BUG: Picture rotated 180 degrees and rotated -180 degrees should be consistent," July 2020. [Online]. Available: https://github.com/scipy/scipy/issues/12543

[80] "p-value varies with the order of the data," March 2020. [Online]. Available: https://github.com/scipy/scipy/issues/11669

[81] Y. Jia and M. Harman, "An analysis and survey of the development of mutation testing," *IEEE Trans. Softw. Eng.*, vol. 37, no. 5, pp. 649–678, Sep./Oct. 2011.

[82] M. Papadakis, C. Henard, and Y. L. Traon, "Sampling program inputs with mutation analysis: Going beyond combinatorial interaction testing," in *Proc. IEEE 7th Int. Conf. Softw. Testing, Verification Validation*, 2014, pp. 1–10.

[83] C. Henard, M. Papadakis, M. Harman, Y. Jia, and Y. Le Traon , "Comparing white-box and black-box test prioritization," in *Proc. 38th Int. Conf. Softw. Eng.*, 2016, pp. 523–534.

[84] M. Aggarwal and S. Sabharwal, "Combinatorial test set prioritization using data flow techniques," *Arabian J. Sci. Eng.*, vol. 43, no. 2, pp. 483–497, Feb. 2018.

[85] H. Mercan, C. Yilmaz, and K. Kaya, "Chip: A configurable hybrid parallel covering array constructor," *IEEE Trans. Softw. Eng.*, vol. 45, no. 12, pp. 1270–1291, Dec. 2019.

[86] R. E. Lopez-Herrejon, J. Javier Ferrer, F. Chicano, E. N. Haslinger, A. Egyed, and E. Alba, "A parallel evolutionary algorithm for prioritized pairwise testing of software product lines," in *Proc. Annu. Conf. Genetic Evol. Comput.*, 2014, pp. 1255–1262.

[87] A. Arcuri and L. Briand, "Adaptive random testing: An illusion of effectiveness?" in *Proc. Int. Symp. Softw. Testing Anal.*, 2011, pp. 265–275.

[88] J. Campos, Y. Ge, N. Albunian, G. Fraser, M. Eler, and A. Arcuri, "An empirical evaluation of evolutionary algorithms for unit test suite generation," *Info. Softw. Technol.*, vol. 104, pp. 207–235, 2018.

[89] E. J. Weyuker, "On testing non-testable programs," *Comput. J.*, vol. 25, no. 4, pp. 465–470, Nov. 1982.

[90] L. S. Ghandehari, Y. Lei, D. Kung, R. Kacker, and R. Kuhn, "Fault localization based on failure-inducing combinations," in *Proc. IEEE 24th Int. Symp. Softw. Rel. Eng.*, 2013, pp. 168–177.

[91] U. Kanewala, "Techniques for automatic detection of metamorphic relations," in *Proc. IEEE 7th Int. Conf. Softw. Testing, Verification Validation Workshops*, 2014, pp. 237–238.

[92] J. Torres-Jimenez and I. Izquierdo-Marquez , "Survey of covering arrays," in *Proc. 15th Int. Symp. Symbolic Numeric Algorithms Sci. Comput.*, 2013, pp. 20–27.

[93] C. J. Colbourn, "Combinatorial aspects of covering arrays," *Le Matematiche*, vol. 59, no. 1, 2, pp. 125–172, 2004.

[94] A. Hartman and L. Raskin, "Problems and algorithms for covering arrays," *Discrete Math.*, vol. 284, no. 1, pp. 149–156, 2004.

[95] C. J. Colbourn and J. H. Dinitz, *The CRC Handbook of Combinatorial Designs*. Boca Raton, FL, USA: CRC Press, 2007.

[96] S. Raaphorst, L. Moura, and B. Stevens, "Variable strength covering arrays," *J. Combinatorial Des.*, vol. 26, no. 9, pp. 417–438, 2018.

[97] L. Kampel and D. E. Simos, "A survey on the state of the art of complexity problems for covering arrays," *Theor. Comput. Sci.*, vol. 800, pp. 107–124, 2019.

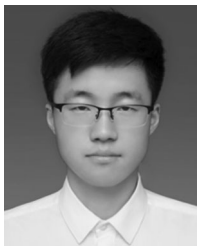
[98] J. Zhang, Z. Zhang, and F. Ma, *Mathematical Construction Methods*. Berlin, Germany: Springer, 2014.

[99] S. K. Khalsa and Y. Labiche, "An orchestrated survey of available algorithms and tools for combinatorial testing," in *Proc. IEEE 25th Int. Symp. Softw. Rel. Eng.*, 2014, pp. 323–334.
[100] C. Luo *et al.*, "LS-sampling: An effective local search based sampling approach for achieving high t-wise coverage," in *Proc. 29th ACM Joint Meeting Eur Softw. Eng. Conf. Symp. Found. Softw. Eng.*, 2021, pp. 1081–1092.
[101] D. R. Kuhn, R. N. Kacker, Y. Lei, and J. Torres-Jimenez , "Equivalence class verification and oracle-free testing using two-layer covering arrays," in *Proc. IEEE 8th Int. Conf. Softw. Testing, Verification Validation Workshops*, 2015, pp. 1–4.
[102] H. Ukai, X. Qu, H. Washizaki, and Y. Fukazawa, "Reduce test cost by reusing test oracles through combinatorial join," in *Proc. IEEE Int. Conf. Softw. Testing, Verification Validation Workshops*, 2019, pp. 260–263.
[103] F. Wotawa, "Combining combinatorial testing and metamorphic testing for testing a logic-based non-monotonic reasoning system," in *Proc. IEEE Int. Conf. Softw. Testing, Verification Validation Workshops*, 2018, pp. 348–351.
[104] T. Y. Chen, T. H. Tse, and Zhiquan Zhou, "Fault-based testing in the absence of an oracle," in *Proc. 25th Annu. Int. Comput. Softw. Appl. Conf.*, 2001, pp. 172–178.
[105] G. Dong, S. Wu, G. Wang, T. Guo, and Y. Huang, "Security assurance with metamorphic testing and genetic algorithm," in *Proc. IEEE/WIC/ACM Int. Conf. Web Intell. Intell. Agent Technol.*, 2010, pp. 397–401.
[106] M. Jiang, T. Y. Chen, F.-C. Kuo, D. Towey, and Z. Ding, "A metamorphic testing approach for supporting program repair without the need for a test oracle," *J. Syst. Softw.*, vol. 126, pp. 127–140, 2017.
[107] L. Xu, D. Towey, A. P. French, S. Benford, Z. Q. Zhou, and T. Y. Chen, "Enhancing supervised classifications with metamorphic relations," in *Proc. IEEE/ACM 3rd Int. Workshop Metamorphic Testing*, 2018, pp. 46–53.

**Xintao Niu** (Member, IEEE) received the PhD degree in computer science and technology from Nanjing University in 2019. He is currently an assistant researcher with the Department of Computer Science and Technology, Nanjing University. His research interests include the area of fault localization, combinatorial testing, software Testing, and software repair.

**Yanjie Sun** received the BS degree from the Nanjing University of Science and Technology. She is currently working toward the master's degree with the Department of Computer Science and Technology, Nanjing University. Her research interests include software testing, combinatorial testing, and metamorphic testing.

**Huayao Wu** (Member, IEEE) received the PhD degree in computer science and technology from Nanjing University in 2018. He is currently an assistant researcher with the Department of Computer Science and Technology, Nanjing University. His research interests include combinatorial testing, software testing, and search based software engineering.



**Gang Li** received the BS degree from the Qingdao University of Technology. He is currently working toward the master's degree with the Department of Computer Science and Technology, Nanjing University. His research interests include random testing and combinatorial testing.

**Changhai Nie** (Member, IEEE) is currently a professor of software engineering with the National Key Laboratory for Novel Software Technology, Department of Computer Science and Technology, Nanjing University. His research interests include software testing and search base software engineering, combinatorial testing, search based software testing, and software testing methods comparison and combination.

**Lei Yu** (Member, IEEE) received the PhD degree in computer science from North Carolina State University. He is currently a professor with the Department of Computer Science and Engineering, University of Texas, Arlington. His research interests include the area of software analysis, testing and verification, and special focus on combinatorial testing. He was a member of Technical Staff with Fujitsu Network Communications Inc. for about three years.

**Xiaoyin Wang** (Member, IEEE) received the PhD degree from Software Engineering Institute, Peking University. His advisor is Prof. Hong Mei and he also did research under the supervision of Prof. Lu Zhang and Prof. Tao Xie. From October 2008 to September 2009, he visited Singapore Management University as a research fellow, where he worked with Prof. David Lo. In January 2012, he began to work with Prof. Dawn Song, as a postdoc with UC Berkeley. In August 2013, he joined Computer Science Department, University of Texas at San Antonio.