

Combinatorial Test Generation for Multiple Input Models with Shared Parameters

Chang Rao, Nan Li, *Member, IEEE*, Yu Lei, *Member, IEEE*, Jin Guo, YaDong Zhang, Raghu N. Kacker, D. Richard Kuhn, *Fellow, IEEE*

Abstract—Combinatorial testing typically considers a single input model and creates a single test set that achieves t -way coverage. This paper addresses the problem of combinatorial test generation for multiple input models with shared parameters. We formally define the problem and propose an efficient approach to generating multiple test sets, one for each input model, that together satisfy t -way coverage for all of these input models while minimizing the amount of redundancy between these test sets. We report an experimental evaluation that applies our approach to five real-world applications. The results show that our approach can significantly reduce the amount of redundancy between the test sets generated for multiple input models and perform better than a post-optimization approach.

Index Terms—Combinatorial Testing, T-way Test Generation, Multiple Input Models, Shared Parameters

1 INTRODUCTION

COMBINATORIAL testing (CT) has been shown to be a very effective approach to software testing [1] [2]. In particular, CT has been applied in situations where interactions of certain elements need to be tested, e.g., configuration testing [3], GUI testing [4], web application testing [5], security testing [6], product line testing [7] [8], and others [9] [10]. A t -way combinatorial test set, or simply a t -way test set, is designed to achieve t -way coverage that requires every value combination of any t parameters be covered by at least one test [11], where t is typically small and is referred to as the test strength.

CT is a black-box testing strategy in that it generates tests by modeling and sampling the input space of the subject application without access to the source code.

Many approaches have been proposed to build a t -way test set [2] [12] [13]. However, existing approaches have mainly focused on how to create a t -way test set for a single input model. In this paper, we consider the problem of t -way test generation for multiple input models with shared parameters. A shared parameter is a parameter that exists in more than one input model. Even though shared parameters exist in multiple models, their interactions need

to be tested only once. We refer to this problem as the CT-MM (Combinatorial Testing for Multiple Models) problem.

The CT-MM problem is found when we perform CT in practice, especially for large and/or complex software applications. For example, use case testing is one common approach to testing a software application. Each use case represents a scenario the user could use the application to achieve a goal. When we employ CT to perform use case testing, we could create one input model for each use case. Shared parameters may exist between the input models created for different use cases. Consider a loan management application. There could be different use cases for different types of loan applications. These use cases typically have some unique parameters that are specific to the loan types, and also some common parameters that are shared between the different loan types, e.g. parameters that represent the credit history of the applicant. Interactions between some shared parameters only need to be tested once when these parameters are processed by a common code module.

As a second example, based on the *write-a-little-test-a-little* strategy, smaller modules are often tested before they are integrated. When we employ CT to module testing, we could create one input model for each module. These modules are not completely independent because they could use some common components. Inputs that are processed by a common component may appear as shared parameters in the input models created for different modules. Since these parameters are processed by the same common component, their interactions only need to be tested once.

As a third example, Nguyen et al. present an approach that combines model-based testing (MBT) and combinatorial testing (CT) [14]. In their approach, a subject application is modeled as a Finite State Machine (FSM), where each transition is labeled with an event. An event represents a user action that may take one or more user inputs. To apply CT, a set of test paths is first generated from the FSM to achieve certain coverage, e.g., all-edge coverage. Second, an input model is created for each test path where each

- C. Rao is with the School of Information Science and Technology, and also with the Sichuan Key Laboratory of Transportation Information Engineering and Control, Southwest Jiaotong University, Chengdu, Sichuan, 611756, China. E-mail: changrao@my.swjtu.edu.cn.
- N. Li is with the Research and Development, Dassault Systems, New York, NY, 10014, USA. E-mail: nli@mdsol.com.
- Y. Lei is with the Department of Computer Science and Engineering, The University of Texas at Arlington, Arlington, TX, 76013, USA. E-mail: ylei@cse.uta.edu.
- J. Guo and Y. Zhang are with the School of Information Science and Technology, Southwest Jiaotong University, Chengdu, Sichuan, 611756, China. E-mail: {jguo_scce, ydzhang}@home.swjtu.edu.cn.
- R. Kacker and D. Kuhn are with the Information and Technology Laboratory, National Institute of Standards and Technology, Gaithersburg, MD, 20899, USA. E-mail: {raghu.kacker, kuhn}@nist.gov.

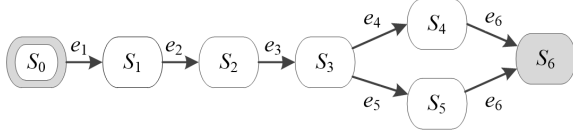


Fig. 1. An Example FSM

event is modeled as a parameter. Representative user inputs are selected for each event as the possible values of the corresponding parameter. Third, a t -way test set is created for each of the input models, using post-optimization to remove redundant tests. This is a CT-MM problem, since the same event could appear in different test paths, and thus the corresponding parameter(s) would be shared among the input models created for the test paths.

Figure 1 gives an example FSM [14]. This FSM models part of the payment process of a large open-source e-commerce and banking web application called Cyclos. In the payment process, the user first inputs the user name (e_1) and the payment amount (e_2). Next, the user schedules the payment process (e_3), i.e., *immediately pay* or *pay in the future*. If the user chooses *pay immediately*, the user may type a description about this payment (e_4). Otherwise, if the user chooses *pay in the future*, they need to choose the number of payments, which can be single or multiple (e_5). Last, the user presses a button to submit or cancel this payment (e_6). In the FSM, S_0 represents the initial state and S_6 represents the final state. Table 1 gives the action each event represents and the representative user inputs that each event takes.

TABLE 1
Event Information in the Example FSM

Event	Action	Representative User Inputs
e_1	input_user_ame	users_in_list; other_users;
e_2	input_payment_amount	too_small; valid; too_large;
e_3	select_payment_schedule	immediately; future;
e_4	type_payment_description	any_string;
e_5	select_payments_in_future	single; multiple;
e_6	press_button	submit; cancel;

To apply CT to this FSM, we first generate two test paths, including $path_1 = (e_1, e_2, e_3, e_4, e_6)$ and $path_2 = (e_1, e_2, e_3, e_5, e_6)$. Then, we create one input model for each test path. Event e_i is modeled as parameter p_i . The representative user inputs for e_i are modeled as the possible values of p_i . The model M_1 for $path_1$ consists of parameters p_1, p_2, p_3, p_4 and p_6 . The model M_2 for $path_2$ consists of parameters p_1, p_2, p_3, p_5 and p_6 . There are four shared parameters, i.e., p_1, p_2, p_3 and p_6 , in the two input models.

One straightforward approach to the CT-MM problem is to build a t -way test set for each input model using an existing t -way test generation algorithm. However, this approach would produce redundancy for shared parameters. That is, combinations of shared parameter values may be covered multiple times in multiple input models.

The technical challenge of the CT-MM problem is how to avoid redundant coverage of shared parameter value combinations in multiple input models.

One approach to addressing the above challenge is to apply post-optimization [14]. That is, we first generate a t -way test set for each input model, and remove the redundant tests afterwards. A test is redundant, and thus can be removed, if all the value combinations covered by this test are also covered by other tests.

In this paper, we propose a new approach that tries to avoid redundant tests in the first place. Our approach extends an existing t -way test generation approach, namely In-Parameter-Order-General (IPOG) [15], for a single input model to multiple input models. We refer to this approach as IPOG-MM, where MM again stands for Multiple Models. The main idea of IPOG-MM is the following. If a combination of some shared parameters appears in multiple input models, their value combinations, which we refer to as shared value combinations, could be covered in any of the test sets generated for these models. Our approach distinguishes shared value combinations from unique value combinations. We try to distribute these shared value combinations among multiple test sets in a way that minimizes the total number of tests. Specifically, our approach uses a concept called *capacity* to estimate the number of shared value combinations a given test set could potentially cover.

One might consider an alternative approach in which we create a super model that combines all the input models and then builds a single t -way test set for the super model. This approach would not work because each test in the resulting test set must be split into multiple test sets, one for each input model, so that they can be actually executed. This split of tests could make t -way coverage incomplete, because some combinations that are covered in the original test may no longer be covered in the split tests.

We report an experimental evaluation in which we apply our approach to five real-world applications. We compare our approach to the IPOG approach, i.e., using the IPOG approach to build a separate t -way test set for each input model. The results show that IPOG-MM removes a significant amount of redundant tests produced by the original IPOG approach. We also compare our approach to another approach, IPOG-PO, where we first use IPOG to build a separate t -way test set for each input model and then perform post-optimization to remove the redundant tests. The experimental results show that IPOG-MM can produce fewer tests in most cases than IPOG-PO while remaining competitive in terms of test generation time.

In summary, the major contributions of this paper are:

- 1) To the best of our knowledge, our work is the first to identify and formulate the CT-MM problem. The CT-MM problem is found in different domains, especially for large and/or complex applications.
- 2) We propose a test generation approach, i.e., IPOG-MM, for the CT-MM problem. IPOG-MM tries to minimize the number of tests by avoiding redundant coverage of shared value combinations.
- 3) We report an experimental evaluation on the effectiveness of the IPOG-MM approach. A prototype

TABLE 2
Input Models

Parameter = {values}	Input model = {parameters}
$p_1 = \{a, b\}$	$M_1 = \{p_1, p_2, p_3, p_4, p_5\}$
$p_2 = \{c, d\}$	$M_2 = \{p_1, p_2, p_3, p_4, p_6\}$
$p_3 = \{e, f\}$	
$p_4 = \{g, h, i\}$	
$p_5 = \{j, k\}$	
$p_6 = \{n, m\}$	

tool, *Stride*, is built that implements the IPOG-MM approach. Stride is made available to the public¹.

The paper is organized as follows. Section 2 shows a motivating example. Section 3 defines some basic concepts and the CT-MM problem. Section 4 introduces the existing IPOG approach. Section 5 presents the details of IPOG-MM approach. Section 6 summarizes the implementation of the tool. Section 7 reports some experimental results that demonstrate the effectiveness of our approach. Section 8 discusses the related work. Section 9 concludes the paper and discusses future work.

2 MOTIVATING EXAMPLE

In this section, we give an example to show that when directly applying an existing t -way test generation algorithm to the CT-MM problem, it could result in redundancy in the generated test sets. Furthermore, we show that though post-optimization can remove redundant tests, it can not remove redundant coverage of some shared value combinations. As a result, post-optimization can result in more tests than necessary.

Table 2 shows an SUT that has two input models, M_1 and M_2 . There are a total of six input parameters, p_1, p_2, p_3, p_4, p_5 , and p_6 . Parameters p_1, p_2, p_3 and p_4 are shared parameters, as they appear in both M_1 and M_2 . Parameter p_5 appears only in M_1 and p_6 only in M_2 . The values of each parameter are also shown in the Table 2.

Table 3 shows a test suite with two 2-way test sets, T_1 and T_2 , for M_1 and M_2 , respectively. Note that the two test sets are artificially bloated for the illustration purpose. Specifically, each test set has nine tests in Table 3. However, it requires only six tests in each test set to achieve 2-way coverage for M_1 and M_2 , e.g., if we use an existing tool called ACTS [16]. Also note that the motivating example, including the two test sets in Table 3, comes from [14], where the post-optimization approach was originally proposed.

We make two observations from Table 3. First, due to the existence of shared parameter combinations, there are shared value combinations in both test sets. For example, due to the fact that p_1 and p_2 form a shared parameter combination, all the value combinations of this parameter combination are covered in both T_1 and T_2 . Second, covering the shared value combinations in both test sets produces redundant tests. For example, τ_2 covers 10 value combinations, $(b, c), (b, f), (b, i), (b, k), (c, f), (c, i), (c, k), (f, i), (f, k)$, and (i, k) . As shown in Table 4, these value combinations are also covered by other tests. Removing τ_2

TABLE 3
Pairwise Tests

T_1	T_2
$\tau_0: (a, c, e, i, j)$	$\tau_9: (a, c, e, i, m)$
$\tau_1: (a, c, e, g, j)$	$\tau_{10}: (a, c, e, g, m)$
$\tau_2: (b, c, f, i, k)$	$\tau_{11}: (b, c, f, i, n)$
$\tau_3: (a, d, e, i, k)$	$\tau_{12}: (a, d, e, i, n)$
$\tau_4: (a, d, f, g, k)$	$\tau_{13}: (a, d, f, g, n)$
$\tau_5: (b, c, e, h, k)$	$\tau_{14}: (b, c, e, h, n)$
$\tau_6: (b, c, e, g, k)$	$\tau_{15}: (b, c, e, g, n)$
$\tau_7: (a, c, e, h, k)$	$\tau_{16}: (a, c, e, h, n)$
$\tau_8: (b, d, f, h, j)$	$\tau_{17}: (b, d, f, h, m)$

TABLE 4
Coverage of the VCs Between τ_2 And Other Tests

VCs	τ_0	τ_3	τ_4	τ_5	τ_6	τ_7	τ_8	τ_9	τ_{11}	τ_{14}	τ_{15}	τ_{17}
(b, c)				×	×				×	×	×	
(b, f)							×		×			×
(b, i)									×			
(b, k)				×	×							
(c, f)									×			
(c, i)	×							×	×			
(c, k)				×	×	×						
(f, i)									×			
(f, k)			×									
(i, k)		×										

would make T_1 by itself no longer a 2-way test set, as some pairs, e.g., $(b, i), (c, f), (f, i)$ are not covered by other tests in T_1 . However, if we remove τ_2 , T_1 and T_2 together still achieve 2-way coverage as all the pairs covered by τ_2 are also covered by some tests in T_2 . Thus, τ_2 is a redundant test. This suggests that when applying an existing t -way test generation algorithm to each input model separately, the redundant tests can be generated due to combinations of shared parameters.

Nguyen et al. [14] applied a post-optimization approach to reduce redundant tests. Similar to what we described above, they first generate a test set for each input model separately. Then, they apply the post-optimization approach as follows. For each test, they check whether the test covers any value combination that is not covered by any other tests. If so, the test is kept. Otherwise, the test is removed. This process is repeated until no more test can be removed. Note that the final test set produced by the post-optimization approach depends on the order in which the tests are checked.

After applying the post-optimization approach to the example, τ_3, τ_5 , and τ_7 are removed from T_1 , and τ_{13} and τ_{14} are removed from T_2 . Though post-optimization removes redundant tests, there is still redundant coverage of shared value combinations among the two test sets. For example, after removing the redundant tests $\tau_3, \tau_5, \tau_7, \tau_{13}$ and τ_{14} , (a, c) is still covered multiple times, i.e., by tests $\tau_0, \tau_1, \tau_9, \tau_{10}$, and τ_{16} . Such redundant coverage causes the number of tests to be more than necessary.

This example indicates that there are opportunities for further optimization for the CT-MM problem. In the rest of the paper, we present an approach that aims to avoid redundant tests from being generated in the first place during test generation.

1. <https://github.com/swjtu-railway/IPOG-MM>

3 PRELIMINARIES

This section introduces some basic concepts and defines the CT-MM problem. Assume the existence of a domain D of values.

Definition 1. (Parameter) A parameter p is a set of values. Formally, $p \subseteq D$.

A parameter combination P is a set of parameters. Let $\Pi(P) = p_1 \times p_2 \times \dots \times p_i \times \dots \times p_{|P|}$, where $p_i \in P$. We refer to $\pi \in \Pi(P)$ as a value combination of P . In the rest of the paper, we refer to parameter combination as PC and value combination as VC.

Definition 2. (Input Model) An input model $M = (P, C)$ consists of a (non-empty) set P of input parameters and a set C of constraints, where each constraint $c \in C$ is a function: $\Pi(P) \rightarrow \{\text{true}, \text{false}\}$.

Let $M = (P, C)$ be a model. A VC of P is also referred to as a test of M , or simply a test when M is implied. We will use τ to represent a test. A constraint c maps τ to a boolean value. A test τ is said to be valid if $c(\tau) = \text{true}$. Otherwise, τ is said to be invalid.

Definition 3. (SUT) A System Under Test (SUT) \mathcal{M} consists of a non-empty set of input models, M_1, M_2, \dots, M_m , where $M_i = (P_i, C_i)$, for $1 \leq i \leq m$. Formally, $\mathcal{M} = \{M_1, M_2, \dots, M_m\}$.

In the rest of the paper, we are concern with a single SUT and denote this SUT using $\mathcal{M} = \{M_1, M_2, \dots, M_m\}$. We use \mathcal{P} to denote the set of all the parameters in \mathcal{M} . That is, $\mathcal{P} = P_1 \cup P_2 \cup \dots \cup P_m$.

Let p be a parameter. Let $\mathcal{M}|_p = \{M_i | p \in P_i, 1 \leq i \leq m\}$. That is, $\mathcal{M}|_p$ includes all the models in which p appears. Let P be a PC. Let $\mathcal{M}|_P = \{M_i | P \subseteq P_i, 1 \leq i \leq m\}$. That is, $\mathcal{M}|_P$ includes all the models in which P appears.

Definition 4. (Shared Parameter) An input parameter $p \in \mathcal{P}$ is a *shared parameter*, if p appears in more than one input model of \mathcal{M} . Formally, p is shared if and only if $|\mathcal{M}|_p| > 1$.

A parameter $p \in \mathcal{P}$ is said to be a *unique parameter* if it is not a shared parameter.

Definition 5. (Shared PC) A PC P is a *shared PC*, if more than one input model contains P . Formally, P is shared if and only if $|\mathcal{M}|_P| > 1$.

A PC $P \subseteq \mathcal{P}$ is said to be a *unique PC* if only one input model contain P . It is possible that a PC is *unique* whereas every parameter in the PC is shared.

Further, a VC π of a shared PC is said to be a shared VC. Otherwise, π is a *unique VC*.

In the context of the CT-MM problem, we only consider interactions between parameters that appear altogether in at least one input model. In practice, parameters in different input models could interact due to poor modeling. Such interactions are not considered in our approach. If it is important to cover such interactions, an input model should be created that contain these parameters.

Definition 6. (Cover) A VC π is covered by a test τ , if $\pi \subseteq \tau$.

A VC π is covered by a test set T if there exists a test $\tau \in T$ such that π is covered by τ . A VC π is said to be valid if there exists a valid test τ such that π is covered by τ .

Definition 7. (T-way Test Suite) A t -way test suite \mathcal{T} for \mathcal{M} is a set of test sets T_i , where $1 \leq i \leq m$, one for each model M_i such that for every VC π of every t -way PC, there exists at least one test set T_i that covers π .

In the rest of the paper, we will use the term ‘test set’ to refer to a test set for a single input model and ‘test suite’ to refer to a test set for the entire SUT consisting of multiple input models. A test suite typically consists of a set of individual test sets, one for each input model in the SUT.

Definition 8. (The CT-MM Problem) The CT-MM problem is to find a t -way test suite \mathcal{T} for \mathcal{M} , such that the total number of tests in \mathcal{T} is minimum.

It is important to note that other metrics, e.g. the length of the test paths, the test execution cost at runtime, could be used to define the CT-MM problem.

We note that CT is a black-box approach. In CT, each test represents one external input to the system under test and is executed from an initial state to a final state. Furthermore, the system is typically reset to initial state between two test executions. Thus, these tests can be executed independently in terms that the execution of one test does not enable or disable the execution of another test, nor does it affect the outcome of another test.

4 THE IPOG ALGORITHM

As mentioned earlier, IPOG-MM is built on top of the IPOG approach. In this section, we give an overview of the IPOG approach [17]. We also discuss several issues with directly applying IPOG to multiple input models.

4.1 Overview

Assume that a system consists of a set P of parameters. The IPOG approach takes three major steps to generate a t -way test set T , including test set initialization, horizontal extension and vertical extension. First, IPOG initializes a test set T that satisfies t -way coverage for the first t parameters. This is simply an enumeration of all the t -way combinations for the first t parameters. Next IPOG extends T to build a t -way test set for the first $(t + 1)$ parameters by covering all the t -way combinations, i.e., all the t -way combinations that involve the new parameter and $(t - 1)$ parameters of the first t parameters. IPOG continues the extension for the first $(t + 2)$ parameters, the first $(t + 3)$ parameters, and so on until it builds a t -way test set for all the parameters. The extension to cover a new parameter is performed in the following two steps:

- *Horizontal extension:* IPOG extends each existing test in T by adding one value for the new parameter.
- *Vertical extension:* IPOG adds new tests to T , if needed.

Algorithm 1 shows the IPOG test generation approach.

4.2 Example

We use an example to explain the main idea of horizontal extension and vertical extension. Assume that an input model has three parameters, p_1, p_2, p_3 , where $p_1 = \{11, 12\}$, $p_2 = \{21, 22\}$ and $p_3 = \{31, 32, 33\}$. In order to build a 2-way test set for this model, IPOG first initializes a 2-way test set T for the first two parameters (Algorithm 1, line 2), as shown in Figure 2a.

Algorithm 1: The IPOG Approach

Input: The input model M , strength t
Output: A t -way test set T for M

```

1  $T \leftarrow \emptyset$ ;
2 initialize  $T$  with VCs of the first  $t$  parameters of  $M$ ;
3 for ( $\text{int } i = t + 1; i \leq n; i++$ ) do
4   let  $\Pi$  be the set of VCs involving  $p_i$  and  $(t - 1)$ 
   covered parameters;
5   // horizontal extension to cover  $p_i$ ;
6   for ( $\text{each test } \tau = (v_1, v_2, \dots, v_{i-1})$  in  $T$ ) do
7     choose a value  $v_i$  of  $p_i$ , and extend  $\tau$  into  $\tau' =$ 
        $(v_1, v_2, \dots, v_{i-1}, v_i)$ , such that  $\tau'$  covers
       the most VCs in  $\Pi$ ;
8     remove from  $\Pi$  the VCs covered by  $\tau'$ ;
9   // vertical extension to cover  $p_i$ ;
10  for ( $\text{each VC } \pi \in \Pi$ ) do
11    if ( $\pi$  has not been covered by an existing test) then
12      change an existing test, if possible, or other-
      wise add a new test to cover  $\pi$ ;
13    remove  $\pi$  from  $\Pi$ ;
14 return  $T$ ;
```

Second, IPOG extends T to cover p_3 . There are in total 12 VCs that involve p_3 and thus need to be covered: (11, 31), (11, 32), (11, 33), (12, 31), (12, 32), (12, 33), (21, 31), (21, 32), (21, 33), (22, 31), (22, 32) and (22, 33).

During horizontal extension (Algorithm 1, line 6 - 8), for each of the existing four tests, IPOG chooses a value of p_3 and adds it into the test to cover as many new VCs as possible. For example, assume that the tests $\tau_1 = (11, 21)$ and $\tau_2 = (11, 22)$ have been extended to $\tau_1 = (11, 21, 31)$ and $\tau_2 = (11, 22, 32)$, respectively. Now, IPOG extends the test $\tau_3 = (12, 21)$. If we extend τ_3 into (12, 21, 31), the new test would only cover one new VC, i.e., (12, 31), because (21, 31) has already been covered by τ_1 . Thus, IPOG extends τ_3 to $\tau_3 = (12, 21, 33)$ to cover two new VCs, i.e., (12, 33) and (21, 33). Figure 2b shows the test set after horizontal extension.

After horizontal extension, four VCs remaining uncovered, (11, 33), (12, 32), (21, 32) and (22, 33). IPOG performs vertical extension to cover them (Algorithm 1, line 10 - 13). To cover (11, 33), IPOG creates a new test $\tau_5 = (11, *, 33)$ in T . Note that $*$ here represents a *don't care* value of p_2 , which is a value that can be changed later to other values while preserving the coverage. For example, in order to cover (22, 33), IPOG can directly change the test $\tau_5 = (11, *, 33)$ into $\tau_5 = (11, 22, 33)$, i.e., without adding any new test. Figure 2c shows T after vertical extension. At this point, all the 12 VCs involving p_3 are covered, and T is a complete 2-way test set for all the three parameters, p_1, p_2 and p_3 .

4.3 Issues with Applying IPOG to the CT-MM Problem

The IPOG algorithm is designed to build a t -way test set for a single input model. There are several issues to be addressed if we directly apply IPOG to the CT-MM problem.

First, since shared VCs exist in multiple input models, we may face the following two problems.

	p_1	p_2		p_1	p_2	p_3		p_1	p_2	p_3	
τ_1	11	21		τ_1	11	21	31	τ_1	11	21	31
τ_2	11	22		τ_2	11	22	32	τ_2	11	22	32
τ_3	12	21		τ_3	12	21	33	τ_3	12	21	33
τ_4	12	22		τ_4	12	22	31	τ_4	12	22	31
								τ_5	11	22	33
								τ_6	12	21	32

(a) Initialization (b) Horizontal Extension (c) Vertical Extension

Fig. 2. An illustration of the IPOG Approach

- *Redundancy during initialization.* IPOG builds an exhaustive t -way test set for the first t parameters. If the first t parameters are shared across multiple input models, their combinations would be covered multiple times, one for each model.
- *Redundancy during horizontal and vertical extension.* IPOG tries to cover all the t -way VCs for each model during horizontal and vertical extension. This means that shared VCs, if exist, would be covered multiple times, one for each model.

One may consider making a simple improvement on the original IPOG approach to avoid redundant coverage on shared VCs, by tracking coverage of shared VCs across multiple models. When a shared VC is covered for one model, we mark it as covered and do not cover the VC again for other models. This approach, however, does not distinguish between unique and shared VCs, which may produce more tests than necessary as discussed below:

- During horizontal extension, IPOG extends each existing test to cover as many uncovered VCs as possible. No distinction is made on whether such VCs are unique or shared. This may cause more shared VCs to be covered than unique VCs. Thus, more unique VCs would have to be covered by vertical extension, which could produce more tests than necessary. Note that shared VCs could be covered in other test sets, while unique VCs must be covered in the current test set.
- During vertical extension, IPOG covers all the remaining VCs, i.e., VCs that have not been covered. The remaining VCs may include both unique and shared VCs. Again, while unique VCs must be covered in the current test set, shared VCs do not have to. In particular, shared VCs could be covered during horizontal extension of the later test sets, without adding any new test.

The IPOG-MM approach is developed to address above issues, as detailed in the following section.

5 THE IPOG-MM APPROACH

In this section, we present a new approach, IPOG-MM, for generating tests on multiple input models. Algorithm 2 shows the major steps of IPOG-MM. Similar to IPOG, IPOG-MM adopts the *one-parameter-at-a-time* framework. However, unlike IPOG, IPOG-MM makes a distinction between shared and unique PCs and VCs.

As shown in Algorithm 2, IPOG-MM consists of three major steps: 1) creating an initial test suite (line 3); 2)

Algorithm 2: The IPOG-MM Framework

Input: A set \mathcal{M} of input models, strength t
Output: A test suite \mathcal{T} that satisfies t -way coverage for \mathcal{M}

- 1 let \mathcal{P} be the union of all input parameters in \mathcal{M} ;
- 2 sort the parameters in \mathcal{P} with a non-ascending order by the domain size of each parameter;
- 3 build an initial test suite \mathcal{T} for \mathcal{M} ;
- 4 // test sets extension
- 5 **for** (each parameter $p \in \mathcal{P}$) **do**
- 6 **if** (p is a unique parameter) **then**
- 7 // cover a unique parameter
- 8 extend the only test set involving p to cover p ;
- 9 **else**
- 10 // cover a shared parameter
- 11 extend the multiple test sets involving p to cover p ;
- 12 **return** \mathcal{T} ;

extending a single test set to cover a unique parameter (line 8); 3) extending multiple test sets to cover a shared parameter (line 11).

To cover a unique parameter p , we use the same strategy as IPOG. That is, we perform horizontal and vertical extension to extend the only test set involving p to cover p (Algorithm 1, line 6-13).

The rest of this section is organized as follows. Section 5.1 discusses the initialization process. Section 5.2 discusses how to cover a shared parameter in IPOG-MM, Section 5.3 analyzes the complexity of IPOG-MM. Section 5.4 discusses constraints handling. Section 5.5 provides some miscellaneous considerations.

Note that in Section 5.1 and 5.2, we assume that the input models do not have constraints. We discuss how to deal with constraints in Section 5.4. Also, we assume that no input model is a sub-model of another model. A sub-model is an input model whose parameters are all contained in another input model (which is referred to as a super-model). We discuss how to deal with sub-models, and some other issues in Section 5.5.

Also note that in this section, we use a running example to explain the major steps of IPOG-MM. Assume $t = 2$. Table 5 shows the input models of the example.

TABLE 5
Input Models of the Running Example

Parameter = {values}	Input model = {parameters}
$p_1 = \{11, 12, 13, 14\}$	$M_1 = \{p_1, p_2, p_3, p_4\}$
$p_2 = \{21, 22, 23\}$	$M_2 = \{p_1, p_2, p_3, p_5\}$
$p_3 = \{31, 32, 33\}$	$M_3 = \{p_3, p_4, p_5\}$
$p_4 = \{41, 42\}$	$M_4 = \{p_4, p_5, p_6\}$
$p_5 = \{51\}$	
$p_6 = \{61\}$	

5.1 Initialization

For a given model M , IPOG-MM initializes the test set using unique VCs. That is, the initial test set for M does

not include shared VCs that also exist in other models. To minimize the number of tests, IPOG-MM selects a unique PC that has the largest number of VCs. This is similar to IPOG, where the parameters are sorted according to a non-ascending order of their domain sizes [18]. Doing so is likely to reduce the number of tests. Algorithm 3 shows the initialization process.

Algorithm 3: Initialize A Test Suite for \mathcal{M}

Input: A set \mathcal{M} of input models, strength t
Output: An initial test suite \mathcal{T} of test sets

- 1 let \mathcal{T} be an empty test suite;
- 2 **for** (each input model $M \in \mathcal{M}$) **do**
- 3 let T be an empty test set;
- 4 **if** (there exists a unique PC in M) **then**
- 5 let P be a unique PC that has the most VCs;
- 6 add into T each VC $\pi \in \Pi(P)$ as a test;
- 7 mark the parameters of P as covered;
- 8 $\mathcal{T} \leftarrow \mathcal{T} \cup \{T\}$;
- 9 **return** \mathcal{T} ;

Note that there may be no unique PCs in an input model M . In this case, the initial test set for M will be empty.

Consider the running example. For M_1 , the VCs of the PC (p_1, p_4) is chosen to create an initial a test set T_1 for M_1 . This is because the PC (p_1, p_4) is unique and has the largest number of VCs, i.e., $4 \times 2 = 8$. Based on this strategy, we initialize a test set for each input model. Table 6 shows the test suite after initialization.

TABLE 6
The Initial Test Suite for the Running Example

$T_1 : \{p_1, p_4\}$			
(11, 41)	(11, 42)	(13, 41)	(13, 42)
(12, 41)	(12, 42)	(14, 41)	(14, 42)
$T_2 : \{p_1, p_5\}$			
(11, 51)	(12, 51)	(13, 51)	(14, 51)
$T_3 : \{p_3, p_4, p_5\}$			
\emptyset			
$T_4 : \{p_4, p_6\}$			
(41, 61)		(42, 61)	

5.2 Covering a Shared Parameter

In this section, we present an algorithm, i.e., Algorithm 4 that is used to cover a shared parameter (Algorithm 2, line 11).

Algorithm 4 consists of two phases. In Phase 1, we extend the test sets to cover all the unique VCs and as many shared VCs as possible involving the shared parameter (Algorithm 4, line 3-9). We explain the details of Phase 1 in Section 5.2.1, including the notion of *capacity*. In Phase 2, we further extend the test sets to cover all the remaining shared VCs (Algorithm 4, line 11). We explain the details of Phase 2 in Section 5.2.2.

Algorithm 4: Cover a Shared Parameter

Input: A t -way test suite \mathcal{T} , a shared parameter p , strength t

Output: A t -way test suite that extends \mathcal{T} to cover p

- 1 //Phase 1: cover all unique VCs and most shared VCs
- 2 let $\mathcal{T}|_p$ be the set of non-empty test sets in \mathcal{T} that involves p , but does not cover p ;
- 3 **while** ($\mathcal{T}|_p$ is not empty) **do**
- 4 // Phase 1-1: select a test set
- 5 compute the $capacity(T, p)$ of each $T \in \mathcal{T}|_p$;
- 6 select a test set T that has the maximum $capacity$;
- 7 // Phase 1-2: extend the selected test set
- 8 extend T to cover p ;
- 9 remove T from $\mathcal{T}|_p$;
- 10 // Phase 2: cover the remaining shared VCs
- 11 extend multiple test sets in \mathcal{T} to cover the remaining shared VCs;
- 12 return \mathcal{T} ;

5.2.1 Phase 1: Cover all the Unique VCs

In Phase 1, we first identify test sets that are not empty, involve p , but do not cover p (Algorithm 4, line 2). A test set T is not extended in Phase 1 in the following two cases: 1) T is empty. This is possible if the model has no unique PCs. The horizontal extension of the IPOG framework requires an existing test when we try to cover a new parameter; 2) the shared parameter p to be covered has already been covered by T . This is possible because p could be covered by a test set during initialization.

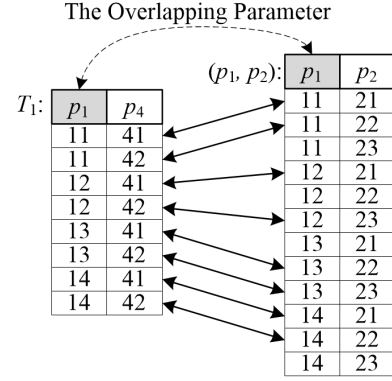
Phase 1-1: Selecting a Test Set. When we extend a test set to cover a new parameter p , we need to cover all the VCs of the PCs that involve p and $(t - 1)$ covered parameters. We aim to select a test set that is likely to cover the most uncovered VCs.

Assume that we have initialized a test set T for the first t parameters. When extending T to cover a new next parameter, it is difficult to calculate the number of uncovered VCs that T can cover. We introduce a notion, called *capacity*, to estimate this number.

To compute the *capacity* of T , we first introduce the concept of *matching pair*. Let P be a PC introduced by the new parameter p . Since IPOG-MM covers one parameter at a time, P consists of $(t - 1)$ parameters that have already been covered by T , in addition to p . Let π be a VC of P that has not been covered yet. Let τ be a test in T . π and τ are a matching pair if the $(t - 1)$ parameters have the same values in τ and π . $Match(T, P)$ denotes the total number of matching pairs between the tests in T and the VCs in P .

Note that a test (or VC) could form a matching pair with multiple VCs (or tests). However, when we count the number of matching pairs, each test (or VC) can only be used to form one matching pair.

The *capacity* with respect to a parameter p represents the number of VCs involving p that could potentially be covered by T . We compute $capacity(T, p)$ by adding the number of matching pairs of all the PCs that involve p . Specifically, the *capacity* is computed using Equation (1), where m represents the number of shared PCs that are involved in p and $(t - 1)$

Fig. 3. Possible matching pairs between T_1 and (p_1, p_2)

parameters covered by T .

$$capacity(T, p) = \sum_{i=1}^m (Match(T, P_i)) \quad (1)$$

We select a test set with the highest *capacity* as the test set to extend next. Note that the *capacity* of each test set is computed dynamically, since uncovered VCs may change after a test set is extended. Also note that *capacity* is an approximation of the number of VCs that T could actually cover, since we simply add all the number of matching pairs without considering possible *conflicts* between different PCs. The term *conflict* means that the shared parameter p that is being covered takes different values in the matched VCs.

We use the running example to illustrate how to compute matching pairs and capacity. Let \mathcal{P} be the set of all input parameters. After sorting, $\mathcal{P} = \{p_1, p_2, p_4, p_5, p_3, p_6, p_7\}$. As shown in Table 6, p_1 has been covered in both test sets T_1 and T_2 during initialization. We skip p_1 and cover the next parameter p_2 .

First, we identify two test sets, T_1 and T_2 , that satisfy two conditions (Algorithm 4, line 2): 1) these test sets involve p_2 ; 2) these test sets have not covered p_2 . Second, we compute the capacities of T_1 and T_2 . Adding p_2 into the test sets introduces one new shared PC, i.e., (p_1, p_2) . Table 7 shows the VCs of this PC. All of the 12 VCs of (p_1, p_2) have not been covered yet. Figure 3 shows the possible matching pairs between T_1 and (p_1, p_2) .

The only overlapping parameter between T_1 and (p_1, p_2) is p_1 . Note that p_1 has four values, i.e., 11, 12, 13, and 14. For the value 11, although there are three shared VCs with 11 in (p_1, p_2) , there are only two tests in T_1 with 11. Thus, there are only two matching pairs for value 11. Similarly, there are two matching pairs for each of the other values, i.e., 12, 13, and 14. Therefore, $Match(T_1, (p_1, p_2)) = 2 + 2 + 2 + 2 = 8$, $capacity(T_1, p_2) = 8$. Similarly, we get $Match(T_2, (p_1, p_2)) = 1 + 1 + 1 + 1 = 4$ and $capacity(T_2, p_2) = 4$. Since $capacity(T_1, p_2) > capacity(T_2, p_2)$, we select T_1 to extend first.

Phase 1-2: Extending the Selected Test Set. The general strategy is to cover all the unique VCs and as many shared VCs as possible. A new test could be added to cover a unique VC if the VC could not be otherwise covered. However, we never add a new test to cover a shared VC in this step. This strategy essentially gives higher priority to unique VCs than shared VCs. This is because unique VCs

can only be covered in the current test set, whereas shared VCs can be covered in other test sets. IPOG-MM does not construct new tests to cover the shared VCs in this phase, because other test sets may be able to cover them.

IPOG-MM makes two modifications on the original IPOG approach:

- During horizontal extension, when adding a new value of the new parameter p into existing test, IPOG-MM chooses a new value of p to cover as many unique VCs as possible. If two or more values of p can cover the same number of unique VCs, IPOG-MM chooses a value that can cover more shared VCs;
- During vertical extension, IPOG-MM only tries to cover all the remaining unique VCs. Shared VCs are only covered as side effects. That is, when we try to cover the unique VCs, some shared VCs would be covered without additional effort.

Consider the running example. We now extend T_1 to cover p_2 . There are two newly introduced PCs, (p_4, p_2) and (p_1, p_2) . Table 7 the VCs of these two PCs. Since (p_4, p_2) is a unique PC, all its VCs are unique VCs. Since (p_1, p_2) is a shared PC, all its VCs are shared VCs.

TABLE 7
Combinations Introduced by p_2 in the Running Example

(p_4, p_2)	(p_1, p_2)	
(41, 21)	(11, 21)	(13, 21)
(41, 22)	(11, 22)	(13, 22)
(41, 23)	(11, 23)	(13, 23)
(42, 21)	(12, 21)	(14, 21)
(42, 22)	(12, 22)	(14, 22)
(42, 23)	(12, 23)	(14, 23)

Next, we extend each test in T_1 . Recall that T_1 has eight tests as shown in Table 6. First, we extend $\tau_1 = (11, 41)$. If we add the value 21 to τ_1 , τ_1 becomes $(11, 41, 21)$. The new test covers one unique VC $(41, 21)$ and one shared VC $(11, 21)$. If we add 22 to τ_1 , the new test would also cover one unique VC $(41, 22)$, and one shared VC $(11, 22)$. We would cover one unique VC $(41, 23)$ and also one shared VC $(11, 23)$ if we add 23 to τ_1 . Thus, we can choose any of the three values, 21, 22 or 23 for τ_1 . Assume that we choose 21 for τ_1 . We remove the VCs covered by τ_1 .

Second, we extend $\tau_2 = (11, 42)$. Similarly, we add 22 into τ_2 . The new test covers one unique VC, $(41, 22)$, and one shared VC, $(11, 22)$. Then, we remove $(41, 22)$, and $(11, 22)$. We continue to do this for other tests.

After extending the test sets, all the unique VCs are covered. Table 8 shows T_1 and T_2 after covering p_2 .

TABLE 8
The Test Sets Involving p_2 after Phase 1 Extension

$T_1 : \{p_1, p_4, p_2\}$			
(11, 41, 21)	(11, 42, 22)	(13, 41, 22)	(13, 42, 23)
(12, 41, 23)	(12, 42, 21)	(14, 41, 21)	(14, 42, 22)

$T_2 : \{p_1, p_5, p_2\}$			
(11, 51, 23)	(12, 51, 22)	(13, 51, 21)	(14, 51, 23)

Note that in this example, we do not have to do the vertical extension after covering p_2 . But there will be remaining

shared VCs after covering the next parameter p_3 . We discuss how to cover the remaining shared VCs in Section 5.2.2.

5.2.2 Phase 2: Cover Remaining Shared VCs

In Phase 1, we have covered all the unique VCs, but not necessarily all the shared VCs. In Phase 2, we cover all the remaining shared VCs, if exist, either by updating existing tests (i.e., replacing don't care values), or by constructing new tests.

Algorithm 5 shows the details of Phase 2. The goal of the algorithm is to minimize the number of tests that are added to cover the remaining (shared) VCs. This is similar to the goal of the vertical extension in the original IPOG approach. The difference is that we need to minimize the number of tests that could be added to multiple test sets, instead of a single test set.

Algorithm 5: Cover Remaining Shared VCs

Input: A set Π of the remaining shared VCs, a shared parameter p , an existing test suite \mathcal{T} , strength t
Output: An extension of \mathcal{T} that covers all the VCs in Π

```

1 let  $\mathcal{T}|_p$  be a subset of test sets in  $\mathcal{T}$  that involve  $p$ ;
2 // reuse existing tests to cover shared VCs
3 for (each VC  $\pi \in \Pi$ ) do
4   change an existing test  $\tau \in \mathcal{T}|_p$  into  $\tau'$  to cover  $\pi$ ,
   if possible;
5   remove from  $\Pi$  the VCs that are covered by  $\tau'$ ;
6 // construct new tests to cover shared VCs
7 while ( $\Pi$  is not empty) do
8   let  $\tau$  be an empty test;
9   for (each test set  $T \in \mathcal{T}|_p$ ) do
10    create a test  $\tau'$  such that if added to  $T$ , it would
    cover the most VCs in  $\Pi$ ;
11    if ( $\tau'$  covers more VCs than  $\tau$ ) then
12       $\tau \leftarrow \tau'$ ;
13   add  $\tau$  to the corresponding test set as a new test;
14   remove from  $\Pi$  the VCs covered by  $\tau$ ;
15 return  $\mathcal{T}$ ;

```

In Algorithm 5, the test suite \mathcal{T} of test sets is obtained after Phase 1. When extending \mathcal{T} to cover remaining VCs, we first try to cover as many VCs as possible without adding a new test, i.e., by changing some don't care values in the existing tests. Then, we cover the remaining VCs by constructing new tests. For each test set that involves p , we create a test that would cover the most remaining VCs if added to the test set. Among all the tests created, one for each test set, we choose one that could cover the most remaining VCs. We add the chosen test to extend the corresponding test set and remove the VCs covered this chosen test. This process is repeated until all the remaining VCs are covered.

In order to create a test τ' that would cover the most remaining VCs if added to a test set T , we create a set of candidate tests from which we choose one that covers the most remaining VCs. First, we check if there exists a PC P that have the most remaining VCs. If P does not exist, T

already covers all the VCs for the parameters involved in T . Thus no candidate test is built for T . Second, we create a candidate test τ from each remaining VC of P , such that in τ , all the parameters involved in the VC take the same values as in the VC and the other parameters take don't care values. Third, we extend each candidate test τ of T by adding a remaining shared VC $\pi \in \Pi$ into τ , such that 1) τ is compatible with π ; and 2) this addition covers the most remaining shared VCs. Note that c is compatible with a VC π if any parameter that exists in τ and π take the same value in them. We continue to select VCs to extend τ , until no further extension could be made, which happens when there exists no more don't care values or no compatible VCs can be selected from Π .

Let's continue with the running example. Table 9 shows T_1 and T_2 after covering p_3 in Phase 1.

TABLE 9
The Test Sets Involving p_3 after Phase 1 Extension

$T_1 : \{p_1, p_4, p_2, p_3\}$			
(11, 41, 21, 31)	(11, 42, 22, 32)	(13, 41, 22, 32)	(13, 42, 23, 33)
(12, 41, 23, 33)	(12, 42, 21, 31)	(14, 41, 21, 32)	(14, 42, 22, 33)
(13, *, 22, 31)	(11, *, *, 33)	(*, *, 23, 32)	

$T_2 : \{p_1, p_5, p_2, p_3\}$			
(11, 51, 23, 31)	(12, 51, 22, 32)	(13, 51, 21, 33)	(14, 51, 23, 31)

The following Table 10 shows the remaining shared VCs after Phase 1.

TABLE 10
Remaining Shared VCs Involving p_3

From: (p_1, p_3)	From: (p_2, p_3)
(11, 33)	(22, 31)
(13, 31)	(23, 32)

Since there are no *don't care* values in existing tests for reusing, we build new tests to cover the VCs in Π . Figure 4 shows the major steps for constructing a new test for T_1 (Algorithm 5, line 10 - 12).

At step 1, we initialize a set of candidate tests. The PCs (p_1, p_3) and (p_2, p_3) are both involved in T_1 . They have the same number of remaining VCs. Thus, for T_1 , we could use the VCs of either PC to initialize candidate tests. Without loss of generality, we use the VCs of (p_1, p_3) to initialize the candidate tests τ_{11} and τ_{12} .

At step 2, we extend the candidate tests. We use the VCs of (p_2, p_3) for the extension based on the greedy strategy. For τ_{11} , since there are no VCs could be used to extend, we assign don't care values to the parameters p_2 and p_4 , respectively. For τ_{12} , since it is compatible with the remaining VC (22, 31), we assign the value 22 to p_2 and don't care values to p_4 .

At step 3, we choose a candidate test that can cover the most remaining VCs. The candidate test τ_{12} is chosen because it covers more remaining VCs than τ_{11} , i.e., (13, 31), (22, 31).

A test could be similarly constructed for T_2 to cover the most remaining VCs. In this example, τ_{12} is selected and added into T_1 as a new test. After this extension, the remaining VCs (13, 31) and (22, 31) are covered and thus being removed from Π . We continue the above steps until

all the remaining VCs in Π are covered. Table 11 shows the final test suite that achieves 2-way coverage for the running example.

TABLE 11
The Final Test Suite for the Running Example

$T_1 : \{p_1, p_4, p_2, p_3\}$			
(11, 41, 21, 31)	(11, 42, 22, 32)	(13, 41, 22, 32)	(13, 42, 23, 33)
(12, 41, 23, 33)	(12, 42, 21, 31)	(14, 41, 21, 32)	(14, 42, 22, 33)
(13, *, 22, 31)	(11, *, *, 33)	(*, *, 23, 32)	

$T_2 : \{p_1, p_5, p_2, p_3\}$			
(11, 51, 23, 31)	(12, 51, 22, 32)	(13, 51, 21, 33)	(14, 51, 23, 31)

$T_3 : \{p_3, p_4, p_5\}$			
\emptyset			

$T_4 : \{p_4, p_6, p_5\}$	
(41, 61, 51)	(42, 61, 51)

5.3 Complexity Analysis

In this section, we assume that the strength is t . Also assume that there are n input models, at most k parameters in one input model, and at most d values for one parameter.

5.3.1 Space complexity

The space complexity is dominated by the number of VCs involving a new parameter when extending a test set to cover the new parameter p . There are at most $C(k-1, t-1)$ PCs when extending a test set. Each PC contains $O(d^t)$ VCs. Thus, the space complexity for the VCs is $O(d^t \times k^{t-1})$.

5.3.2 Time complexity for test set selection

To select the next test set for extension, the time complexity is dominated by computing the capacity for each test set when covering a new parameter. IPOG-MM computes the capacity of each test set by adding all the matching pairs between the tests and the shared VCs. Instead of directly computing matching pairs by matching the tests with shared VCs, we count the number of times that a distinct value combination occurs in the tests and the shared VCs. We compute the number of matching pairs by adding the minimum number of times that each distinct value combination occurs in the tests or the shared VCs.

According to [17], each time when covering a new parameter, there will be at most $O(d^t \times \log k)$ tests in a test set. And there will be at most $O(d^t)$ shared VCs for a given PC. Using the bitmap data structure [15], it takes us $O(1)$ time to check occurrence times that a distinct combination occur in a test and a VC. Thus, it takes $O(d^t \times \log k)$ to check matching pairs between the tests and the VCs of a shared PC. Since there are at most $O(k^{t-1})$ newly introduced shared PCs each time, to compute capacity for a test set, the time complexity is $O(d^t \times \log k \times k^{t-1})$.

5.3.3 Time complexity for test set extension

The time complexity is dominated by choosing the values of a new parameter p for the tests to cover the newly introduced VCs. Since we use a bitmap structure to store the VCs, which takes $O(1)$ time to check whether τ could

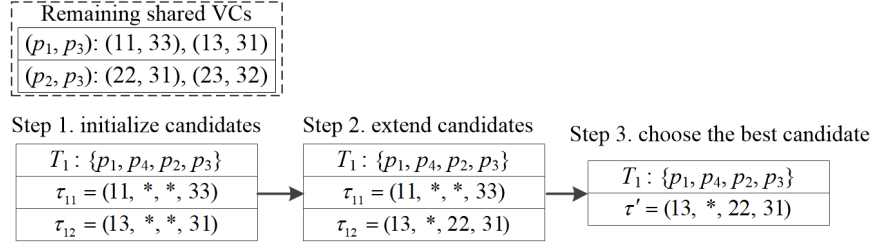


Fig. 4. Main steps of Phase 2

cover a newly introduced VC [15]. For a total of $O(k^{t-1})$ newly introduced PCs, it takes $O(k^{t-1})$ to choose a value of p and determine the number of VCs that a test can cover. To try each of the d values of p , it takes $O(d \times k^{t-1})$ to choose which value can cover the most number of VCs. Consider that in a test set, the number of tests is $O(\log k \times d^t)$, the time complexity for extending a test set to cover p is $O(d^{t+1} \times \log k \times k^{t-1})$.

In summary, for n test sets, the time complexity of IPOG-MM is $O(n \times d^{t+1} \times \log k \times k^{t-1})$.

5.4 Constraint Handling

Constraint handling can be added to IPOG-MM as follows. First, during initialization (Algorithm 3, line 5), we choose a unique PC that has the most valid VCs for initialization. Second, in Phase 1-1, when selecting a test set (Algorithm 4, line 5), we first remove the invalid VCs with respect to each input model. That is, we only try to cover VCs that are valid, i.e., that satisfy all the constraints. Since the constraints in different input models may be different, the same VC may be valid in one input model but invalid in a different model. A VC is considered shared between two input models only if it is valid in both input models. Third, when generating the tests to cover a unique parameter (Algorithm 2, line 8) or a shared parameter (modified horizontal and vertical extension), we check and ensure that the tests are valid. Fourth, in Phase 2, we check and ensure that the tests (Algorithm 5, lines 4 and 10) are valid.

5.5 Discussion

5.5.1 Single input model

When there is a single input model, IPOG-MM is naturally reduced to IPOG. This is because all the parameters of the single input model are treated as unique parameters.

5.5.2 Input model pre-processing

Input models can be pre-processed to improve efficiency of test generation. Let M_1 and M_2 be two input models. If both models have no constraints, M_1 is a sub-model of M_2 , or M_2 is a super-model of M_1 , if all the parameters of M_1 are also parameters of M_2 . In this case, M_1 could be removed. This removal does not affect test coverage.

If M_1 or M_2 or both have constraints, M_1 is a sub-model of M_2 , or M_2 is a super-model of M_1 , if whether the constraints in M_1 are stronger than the constraints in M_2 . In other words, every valid VC in M_1 is also a valid VC in M_2 . In this case, we could remove M_1 . Otherwise, we must keep M_1 . A constraint solver could be used to determine

whether the constraints of M_1 are stronger than those of M_2 . Alternatively, we could just keep both models as long as they contain constraints. In this case, our approach might produce more tests.

Also, some input models may have less than t parameters while the total number of parameters of the SUT is greater or equal to t . In this case, we could simply enumerate all possible combinations for these input models. And during the later test generation, we do not generate tests for such input models anymore.

5.5.3 Empty test sets

We suggest that a complete test to be added into a test set which remains empty after test generation. Recall that during initialization, if an input model M does not have any unique VCs, we create an empty test set for M . This test set may remain empty after test generation, as all the VCs for M are shared VCs and could be covered in other models. This does not affect t -way coverage. However, in some real-world applications, one may desire to have at least one test in each test set. For example, in [14], a test set is created for each path. No test in a test set means the corresponding path would not be executed, which may compromise the path coverage.

5.5.4 Model Evolution

As discussed in [19], input models could be changed due to model corrections and software changes such as bug fixes and enhancements. When this happens, the test sets must be updated in order to maintain t -way coverage. It is often desired to reuse as many existing tests as possible so that the time and effort spent on the existing tests could be saved [20].

We consider the following types of model changes and their impact on the existing test sets:

- *Adding a new input model:* The new model may or may not include shared parameters. In either case, we need to generate a test set for the new model to cover the VCs that are unique to this model. All the existing test sets could be reused. Note that when the new model includes shared PCs, the test set generated for the new model could cover some shared VCs. Thus, the existing test sets could potentially be reduced.
- *Removing an existing input model:* When we remove a model that does not include any shared PCs, we could simply remove the corresponding test set. The other test sets could be reused. However, when we remove a model that includes shared PCs, some

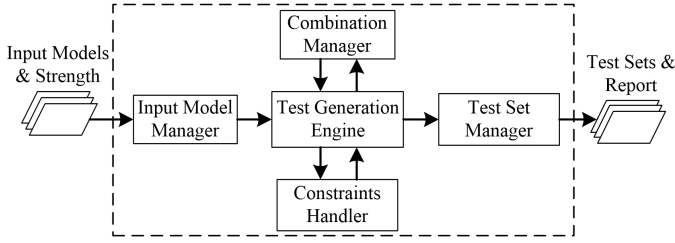


Fig. 5. Overview of the Stride tool

shared VCs (of these PCs) may be covered only in the test set for this model. In this case, the existing test sets must be updated to cover these shared VCs.

- *Modifying an existing input model:* An existing model could be modified in different ways, including adding and/or removing parameters, parameter values, and/or constraints. Let M be the model being modified. Let T be the existing test set for M . T must be updated to cover all the unique VCs in M , e.g., introduced by adding a new parameter or parameter value. In addition, if a test is removed due to some parameter value being removed from M , T must be updated to cover shared VCs that are covered only in this test. This could help minimize changes to, and thus maximize the reuse of, the other test sets. Note that it is possible that a shared VC π was originally covered in T , but could no longer be covered in T due to a constraint change. In this case, some other test sets must be updated to cover π

6 STRIDE: A PROTOTYPE TOOL

We have built a prototype tool, called Stride, that implements the IPOG-MM approach. Stride also implements two other approaches, IPOG and IPOG-PO, which are needed for our experimental evaluation, as explained in Section 7. The tool, with its source code, is made publicly available on GitHub². In this section, we discuss the major design decisions of the tool and our effort to verify the tool implementation.

6.1 Major Components

The Stride tool is written in Java. It takes as input a data file that contains a group of input models and a test strength. The output is a data file that contains a test suite that satisfies t -way coverage for the group of input models.

Figure 5 gives an overview of Stride.

There are five major components, including Input Model Manager, Test Generation Engine, Combination Manager, Test Set Manager, and Constraint Handler. The major responsibilities of these components are shown below:

- *Input Model Manager:* This component is responsible for managing input models. In particular, it is responsible for parsing the input model files, pre-processing the input models, and performing query operations, e.g., retrieving the models that contain a shared PC.

- *Test Generation Engine:* This component implements the core algorithms of the three test generation approaches, including IPOG, IPOG-PO, and IPOG-MM. The three approaches share common data structures and functions such as covering the unique VCs. IPOG-PO implements the post-optimization approach, which was originally proposed and implemented in the M[agi]C tool [21]. However, we found some faults in the implementation as confirmed by its developer.
- *Combination Manager:* This component is responsible for managing different types of combinations, including both shared/unique PCs and VCs. This component employs advanced data structures which are similar to [15], so that combinations can be generated, updated, and searched efficiently.
- *Test Set Manager:* This component is responsible for managing and outputting test sets, including adding new tests, updating existing tests, and performing query operations, e.g., checking whether a VC is covered by a test, retrieving the value of a given parameter, and others.
- *Constraint Handler:* This component is responsible for handling constraints. In particular, it performs two types of validity checks. One is to check the validity of a combination, and the other is to check the validity of a test. We used the constraint handler from our earlier work [22]. Currently, the tool only supports constraints that are written as forbidden tuples.

In addition, Stride includes a component that generates random constraints. This component is only used for the purpose of evaluation, and is thus not needed during production use.

6.2 Tool Verification

Several steps have been taken to ensure the correctness of the tool. At the unit level, we have created JUnit³ test cases to help test key functions, such as finding shared combinations, computing capacity, and test set extension, including horizontal and vertical extension. At the system level, we have created small input models, run the tool and manually checked that the output tests are as expected.

We have also implemented an assistive analysis tool to automatically check the validity and coverage of the resulted tests. The inputs of the tool include the set of input models, strength t , and the generated test suite. The output of the tool includes the validity and coverage of the generated test sets. In particular, if the test suite fails to achieve the expected coverage, the tool will produce a warning message and report the remaining uncovered VCs. This tool has been integrated inside the Stride after the verification and is also publicly available.

7 EXPERIMENTS

Our experiments are designed to evaluate the effectiveness and efficiency of IPOG-MM in comparison with the following two approaches:

2. <https://github.com/swjtu-railway/IPOG-MM>

3. <https://junit.org/junit4/>

- 1) **IPOG**: This is the baseline approach upon which IPOG-MM is built upon. IPOG does not made any effort to avoid redundant tests that may be generated in multiple test sets.
- 2) **IPOG-PO**: This is a competition approach to IPOG-MM. IPOG-PO also builds upon IPOG, but it uses post-optimization to avoid redundant coverage of shared VCs.

We note that the Nguyen et al [14] used post-optimization to avoid redundant tests in multiple test sets, and provided a 2-way test generation tool called M[agi]C [21]. However, the M[agi]C tool does not produce correct results due to some implementation issues⁴. This is the reason why we had to re-implement the post-optimization approach with IPOG.

7.1 Research Questions

Our experiments are designed to ask the following three research questions:

RQ1: How does IPOG-MM compare to IPOG?

IPOG-MM builds on top of IPOG by considering how to avoid redundant coverage of shared VCs. This question helps to evaluate the impact of this consideration on the test generation process.

RQ2: How does IPOG-MM compare to IPOG-PO?

IPOG-MM tries to avoid redundant coverage of shared VCs *during* test generation, whereas IPOG-PO tries to do so in a post-mortem manner, i.e. *after* test generation. This question is designed to compare the effectiveness and efficiency of these two approaches.

RQ3: How does the amount of shared VCs affect the effectiveness of IPOG-MM?

IPOG-MM is centered on the idea of how to avoid redundant coverage of shared VCs. This question is designed to evaluate the impact of shared VCs on the effectiveness of IPOG-MM.

7.2 Subjects

In the experiments, we used the following applications as our experimental subjects:

- 1) **NotePad**: an Android application for editing text and taking notes.
- 2) **OpenManager**: a free and open source app for file management.
- 3) **iosched**: a scheduling application for Google I/O developer conference.
- 4) **k9mail**: an Android application for email management on smartphones.
- 5) **cyclos**, a Java web application for banking. Because it is a large system, only the payment part is modeled.

We selected these applications because they were used to evaluate the post-optimization approach in [14]. Use of the

⁴ The implementation of the post-optimization of the public available version of M[agi]C is not correct, and it cannot satisfy pair-wise coverage after the post-optimization. This has been confirmed by the original developers.

same applications allows us to make a direct comparison between IPOG-MM and the post-optimization approach. Also, the FSM models of these subjects were given in [14]. This helps to remove one major variable that could potentially impact the comparison.

Table 12 shows the size of each FSM model in terms of number of states (States #) and transitions (Transitions #), as given in [14].

TABLE 12
Sizes of FSM Subjects

Subjects	NotePad	iosched	cyclos	k9mail	Open-Manager
States #	10	17	27	19	19
Transitions #	20	58	52	53	60

7.3 Input Models

We create both unconstrained and constrained input models. The purpose of generating constrained/unconstrained input models is to evaluate the impact of constraints on the effectiveness of our approach. Many practical applications have constraints in their input models [13]. Constraint handling is thus considered to be an important feature for practical applications of CT.

7.3.1 Input models for unconstrained experiments

A total of ten groups of unconstrained input models are created for the experiments. Each group consists of multiple input models. Table 13 gives some statistics of these input model groups. Note that we have removed sub-models in each model group.

TABLE 13
Statistics of Input Model Groups

Model Group	Total # of Parameters	Total # of Input Models	# of Parameters in Each Model		
			max	min	average
G_1	20	14	4	2	2.6
G_2	55	43	5	2	2.6
G_3	42	23	9	1	6.2
G_4	47	33	4	1	2.7
G_5	59	42	3	1	2.3
G_6	42	103	14	1	10.1
G_7	28	52	5	2	4.4
G_8	42	167	12	1	10.7
G_9	42	128	14	1	9.6
G_{10}	52	48	14	2	8.3

Note that the ten groups of input models are from the test paths of the subject FSM models. For each subject, we first use the M[agi]C tool to generate a set of test paths from its FSM model, and then construct a group of input models, one for each test path. The M[agi]C tool provides a number of algorithms that could be used to generate test paths from an FSM, among which four algorithms are applicable to the FSMs in our experiments. The test paths generated by each of the four algorithms achieve *all-edge* coverage. In our experiments, we tried every algorithm that could be applied. If more than one algorithm can be applied to an FSM, then multiple groups of input models are created for the FSM, one for each algorithm.

Table 14 shows the ten input model groups created for our subjects. If a path generation algorithm can be applied to a FSM, the name of the input model group is shown in Table 14; otherwise, the corresponding cell is empty. Note that since the test paths are different, for $i, j = 1, 2, \dots, 10, i \neq j$, the input model group $G_i \neq G_j$, even if they are generated based on the same FSM.

TABLE 14
Input Model Groups

Algorithm	NotePad	iosched	cyclos	k9mail	Open-Manager
<i>breadth-first</i>	G_1	G_2	G_3	G_4	G_5
<i>breadth-first with loop (global)</i>			G_6		
<i>breadth-first with loop (local)</i>		G_7	G_8		
<i>uniform coverage</i>			G_9	G_{10}	

7.3.2 Input models for constrained experiments

For each unconstrained input model group (or simply each unconstrained group), we randomly generate five sets of forbidden tuples. The forbidden tuples are used to simulate constraints that may exist in real-world applications. We create a constrained input model group (or simply a constrained group) by combining an unconstrained group and each of the forbidden tuple sets created for the group. Thus, there are a total of $10 \times 5 = 50$ groups of constrained input models.

To distinguish the five constrained groups for an unconstrained group, in the rest of the paper, we name the constrained groups as $CG1$, $CG2$, $CG3$, $CG4$, and $CG5$, respectively.

Constraints are often expressed as forbidden tuples, allowed tuples, or logical expressions [12]. Constraints could be identified from requirements [23], derived from UML diagrams [24], or learned using machine learning techniques [25]. Our experiments adopt random constraints for evaluation for two reasons. First, in the literature, we could not find subject applications that provide multiple input models with constraints. For example, in [14], the subject applications do not have constrained input models. Second, random constraints have been commonly used in several studies that evaluate the impact of constraints on CT, such as [26].

A random constraint generator is implemented to generate random constraints. The constraint generator allows the user to control the following options:

- 1) The input models involved in forbidden tuples. We randomly choose from one to ten input models to be involved in forbidden tuples. If the total number of input models is less than ten, then we randomly specify from one to the total input models to be involved in forbidden tuples.
- 2) Number of forbidden tuples in an input model. In our experiments, we randomly create from one to ten forbidden tuples in an input model.

- 3) Size of a forbidden tuple. In our experiments, we choose a random size between 2 and k to construct the forbidden tuples, where k is the total number of parameters in the input model.
- 4) Parameters involved in a forbidden tuple. We randomly select the parameters with two or more values from the input model.
- 5) Parameter values in a forbidden tuple. For each selected parameter, we randomly choose a value from its value domain.

Table 15 gives the information of the randomly generated forbidden tuples, in terms of number of forbidden tuples and size of forbidden tuple for each model group.

TABLE 15
Forbidden Tuples in Constrained Groups

Model Group	# of Tuples in a Constrained Group			Size of Tuples in a Constrained Group		
	max	min	average	max	min	average
G_1	5	3	3.0	2	2	2.0
G_2	11	1	4.4	2	2	2.0
G_3	21	4	9.6	5	2	2.4
G_4	7	1	4.0	2	2	2.0
G_5	3	1	2.0	2	2	2.0
G_6	38	15	28.2	5	2	2.8
G_7	9	3	5.2	2	2	2.0
G_8	21	1	9.2	5	2	2.6
G_9	19	1	11.2	4	2	2.2
G_{10}	10	3	6.8	2	2	2.0

7.4 Metrics

We measure the effectiveness and efficiency of the three approaches, i.e., IPOG, IPOG-PO, IPOG-MM, in terms of number of generated tests and the amount of time taken for test generation. In addition, for **RQ1**, in order to make explicit the comparison between IPOG-MM and IPOG, we compute the following test reduction ratio:

$$reduction_ratio = \frac{(IPOG \text{ tests } \# - IPOG-MM \text{ tests } \#)}{IPOG \text{ tests } \#} \times 100\% \quad (2)$$

For **RQ2**, in order to make explicit the comparison between IPOG-MM and IPOG-PO, we also compute the following test reduction ratio:

$$reduction_ratio = \frac{(IPOG-PO \text{ tests } \# - IPOG-MM \text{ tests } \#)}{IPOG-PO \text{ tests } \#} \times 100\% \quad (3)$$

For all **RQs**, the shared VC ratio for each unconstrained or constrained group is computed as follows:

$$Shared_VC_Ratio = \frac{\# \text{ of valid shared VCs}}{\# \text{ of valid VCs}} \times 100\% \quad (4)$$

In constrained experiments, a shared VC may be valid in some input models, and invalid in other input models. In Equation (4), we count a shared VC only if it is valid in more than one input model. To evaluate the impact of shared VC ratio, we perform Spearman Rank Correlations to measure

the degree of correlation between the shared VC ratio and the test reduction ratio by IPOG-MM.

We note that the shared VC ratio is a variable that depends on the input models.

7.5 Procedure

There are a total of 10 groups of unconstrained input models and 50 groups of constrained input models. For each input model group, we use the three approaches, i.e., IPOG, IPOG-PO, and IPOG-MM to create test suites for test strength $t = 2$ to 6. Thus, we have conducted a total of $10 \times 5 = 50$ unconstrained experiments and a total of $50 \times 5 = 250$ constrained experiments. For each experiment, we record the number of tests and also measure the amount of time used to run the experiments.

All the experimental subjects, input models and results are available on GitHub⁵. All the experiments are carried out on the platform with Intel Core i5 CPU (3.20 GHz×2) and 8GB Memory.

7.6 Results for RQ1

In this section, we present both unconstrained and constrained experimental results for RQ1. We also provide some additional discussion on the results.

7.6.1 Unconstrained Experiments

Table 16 shows the detailed results of unconstrained experiments.

Among the 50 unconstrained experiments, there are 25 experiments where IPOG-MM generates fewer tests than IPOG. IPOG-MM achieves the highest test reduction rate of 94.82% for G_8 with $t = 2$.

For the other 25 experiments, IPOG-MM generated the same number of tests as IPOG. We have investigated the possible reasons. One reason is that in some experiments, there are no shared VCs between input models. This means that all the VCs are unique VCs. There is no opportunity for optimization as there is no redundant coverage of shared VCs. Thus, IPOG-MM generates the same number of tests as IPOG. A second reason is that in some experiments, there are only a small number of shared VCs. These shared VCs are all covered as side effect when we try to cover unique VCs. That is, no additional tests are generated to cover shared VCs. Thus, avoiding redundant coverage of shared VCs does not help generate fewer tests.

We observe that in some experiments, as strength t increases, the number of tests generated by IPOG remains the same, whereas the number of tests generated by IPOG-MM increases. For example, this happens when t increases from 5 to 6 for G_6 . The reason is that in each input model of G_6 , at most five parameters have multiple values, while other parameters are single-value parameters. When t increases from 5 to 6, the total number of VCs to be covered in each input model remains the same. Thus, IPOG generates the same number of tests. However, IPOG-MM considers shared VCs across multiple input models. As t increases from 5 to 6, the number of shared VCs decreases. This means

that there are fewer opportunities for optimization. Thus, IPOG-MM generates more tests.

Table 17 shows the average number of tests and reduction ratio for each strength for the unconstrained experiments where one or more shared VCs exist between input models.

TABLE 17
IPOG vs IPOG-MM (Unconstrained): Test Generation Results by Strength

Strength	IPOG	IPOG-MM	Reduction Ratio
2	479.90	72.40	84.91%
3	1181.25	200.50	83.23%
4	2893.67	598.50	79.32%
5	5296.00	1673.20	68.41%
6	5296.00	3141.00	40.69%

We observe that the reduction ratio decreases as the test strength increases. This can be explained by the fact that the shared VC ratio decreases as the test strength increases, and thus there are fewer chances to do optimization. In general, the bigger a VC, the more parameters it involves, the less chance it is shared between different models.

In terms of generation time, we focus on the experiments where at least one approach spent more than one second. There are 12 such unconstrained experiments.

Table 18 shows the detailed results for these 12 unconstrained experiments. IPOG-MM is slower than IPOG because IPOG-MM has to perform additional computations to avoid redundant coverage of shared VCs.

TABLE 18
IPOG vs IPOG-MM (Unconstrained): Test Generation Time

Group	Strength	IPOG	IPOG-MM
G_6	3	0.45s	2.17s
	4	1.13s	10.42s
	5	3.92s	20.53s
	6	4.65s	29.46s
G_8	3	0.59s	3.18s
	4	0.92s	16.80s
	5	3.40s	43.27s
	6	4.89s	73.99s
G_9	3	0.26s	1.15s
	4	0.40s	4.40s
	5	0.63s	8.95s
	6	0.87s	9.17s
Average		1.84s	18.62s

7.6.2 Constrained Experiments

Among the 250 constrained experiments, we focus on 170 such experiments where the shared VC ratio is greater than 0, which provides opportunity for optimization.

Table 19 shows the average results of the 170 experiments, including average shared VC ratios, average number of tests, and average reduction ratios. For each input model group, the results are averaged over five random forbidden tuple groups. The detailed results for individual random forbidden tuple groups are available on our Github.

The results in Table 19 show that IPOG-MM performs better than IPOG in most experiments. Specifically, the detailed results (available on our Github) show that IPOG-MM can generate fewer tests than IPOG in 125 out of the

5. <https://github.com/swjtu-railway/IPOG-MM>

TABLE 16
IPOG vs IPOG-MM (Unconstrained): Detailed Tests Generation Results

Group	Strength	Shared VC Ratio	IPOG	IPOG -MM	Reduction Ratio	Group	Strength	Shared VC Ratio	IPOG	IPOG -MM	Reduction Ratio
G_1	2	10.64%	31	23	25.81%	G_6	2	93.83%	1213	105	91.34%
	3	0.00%	35	35	0.00%		3	90.01%	2905	324	88.85%
	4	0.00%	35	35	0.00%		4	85.65%	6199	949	84.69%
	5	0.00%	35	35	0.00%		5	79.62%	11383	2403	78.89%
	6	0.00%	35	35	0.00%		6	71.27%	11383	4693	58.77%
G_2	2	14.75%	193	85	55.96%	G_7	2	48.00%	98	59	40.82%
	3	5.86%	193	193	0.00%		3	43.23%	98	74	24.49%
	4	0.00%	193	193	0.00%		4	28.95%	98	98	0.00%
	5	0.00%	193	193	0.00%		5	0.00%	98	98	0.00%
	6	0.00%	193	193	0.00%		6	0.00%	98	98	0.00%
G_3	2	42.86%	182	85	53.30%	G_8	2	96.22%	1836	95	94.83%
	3	41.31%	370	206	44.32%		3	89.37%	4072	313	92.31%
	4	38.58%	736	460	37.50%		4	79.89%	7918	1033	86.95%
	5	33.61%	1312	880	32.93%		5	69.03%	10798	2846	73.64%
	6	26.64%	1312	1312	0.00%		6	57.54%	10798	6732	37.66%
G_4	2	13.68%	55	55	0.00%	G_9	2	87.07%	1010	95	90.59%
	3	6.94%	55	55	0.00%		3	78.03%	1630	315	80.67%
	4	0.00%	55	55	0.00%		4	66.24%	2284	924	59.54%
	5	0.00%	55	55	0.00%		5	52.80%	2860	2110	26.22%
	6	0.00%	55	55	0.00%		6	38.95%	2860	2841	0.66%
G_5	2	0.59%	54	54	0.00%	G_{10}	2	56.58%	127	68	46.45%
	3	0.00%	54	54	0.00%		3	31.56%	127	124	2.36%
	4	0.00%	54	54	0.00%		4	14.09%	127	127	0.00%
	5	0.00%	54	54	0.00%		5	5.02%	127	127	0.00%
	6	0.00%	54	54	0.00%		6	1.46%	127	127	0.00%

170 experiments. In particular, IPOG-MM can achieve the highest reduction ratio of 94.83% for model group G_8 with test strength equal to 2 for forbidden tuple group $CG3$.

Similar to the unconstrained results in Table 17, in some experiments, when t increases, e.g., for G_6 from $t = 5$ to $t = 6$, IPOG generates the same number of tests whereas IPOG-MM generates more tests.

Table 20 shows the average number of tests and reduction ratio for each strength for the 170 constrained experiments. Similar to the unconstrained results in Table 17, the reduction ratio decreases as the test strength increases. This is also due to the same reason, i.e., the number of shared VCs decreases as the test strength increases.

Note that for constrained experiments on strength $t = 5$ and $t = 6$, IPOG still generates the same number of tests on average. Again, this is because at most 5 parameters have more than one value in an input model. Thus, the total number of VCs to be covered remains the same when t is increased from 5 to 6. Note that the same forbidden tuples are used to perform the experiments with different strengths for the same model group.

In terms of generation time, we focus on the experiments where IPOG and/or IPOG-MM spent more than one second. There are 60 such experiments. To save space, we do not present the detailed results for all of the 60 experiments, which are available on our Github. As an example, Table 21 shows the results for the model group G_6 with test strength equal to 5.

We observe that the average times, in Table 21, i.e., 3.82s for IPOG and 20.73s for IPOG-MM, are similar to those in Table 18 for the same experiment, i.e., 3.92s for IPOG and 20.53s for IPOG-MM. Thus, the overhead ratio between unconstrained and constrained experiments is close. However, there are significant variations for individual experiments. In particular, the overhead ratio of individual (constrained)

experiments varies from 4.45 to 8.90.

7.6.3 Discussion

Table 22 shows an overall comparison between IPOG and IPOG-MM. Recall that there are a total of 300 experiments, including both unconstrained and constrained experiments. We focus on the results of 204 (out of 300) experiments where there is one or more shared VCs between input models. Note that **IPOG-MM = IPOG** indicates that IPOG-MM and IPOG generated the same number of tests, and **IPOG-MM < IPOG** indicates that IPOG-MM generated fewer tests than IPOG.

For the 204 experiments, There are a total of 54 experiments where IPOG-MM generated the same number of tests as IPOG (26.47%). There are a total of 150 experiments (73.53%) where IPOG-MM generated fewer tests.

For **RQ1**, we make the following conclusion:

In most cases, IPOG-MM can generate fewer tests than IPOG, with a test reduction ratio up to 94.83%, when there exist shared VCs between multiple input models. IPOG-MM takes more time than IPOG, but all the experiments take no more than 90 seconds.

7.7 Results for RQ2

In this section, we present both unconstrained and constrained experimental results for RQ2. We also provide some additional discussion on the results. We only focus on the 204 out of the total 300 experiments where the shared VC ratio is greater than 0, which means both IPOG-MM and IPOG-PO have opportunities for optimization.

TABLE 19
IPOG vs IPOG-MM (Constrained): Tests Generation Results
(Note: The results are average results among five random forbidden tuples for each model group)

Model Group	Strength	Shared VC Ratio	IPOG	IPOG-MM	Reduction Ratio
G_1	2	10.92%	29.2	22.4	23.25%
G_2	2	14.85%	188.6	84.8	55.01%
	3	5.95%	188.6	188.6	0.00%
G_3	2	43.19%	176	81.8	53.52%
	3	41.64%	363.6	199.2	45.22%
	4	39.12%	723.6	451.2	37.62%
	5	33.84%	1279.4	870.6	31.87%
	6	26.86%	1279.4	1279.4	0.00%
G_4	2	14.37%	51	51	0.00%
	3	7.69%	51	51	0.00%
G_5	2	6.26%	52	52	0.00%
G_6	2	93.83%	1207.6	105	91.31%
	3	90.03%	2890	319.6	88.94%
	4	85.68%	6177.8	938.8	84.80%
	5	79.65%	11241.6	2410.8	78.55%
	6	71.31%	11241.6	4803.6	57.26%
G_7	2	48.90%	92.8	56.2	39.43%
	3	43.99%	92.8	72.2	22.17%
	4	29.69%	92.8	92.8	0.00%
G_8	2	96.00%	1832	95	94.81%
	3	89.37%	4065.6	312.8	92.31%
	4	79.89%	7906	1034.8	86.91%
	5	69.04%	10768.6	2852.8	73.51%
	6	57.56%	10768.6	6705.4	37.73%
G_9	2	87.10%	1002.4	94.8	90.54%
	3	78.05%	1619.4	314.6	80.57%
	4	66.26%	2263.8	922.8	59.24%
	5	52.84%	2839.8	2096.6	26.17%
	6	39.01%	2839.8	2821.2	0.65%
G_{10}	2	56.76%	120.4	66.6	44.67%
	3	31.96%	120.4	117.8	2.16%
	4	14.40%	120.4	120.4	0.00%
	5	5.15%	120.4	120.4	0.00%
	6	1.49%	120.4	120.4	0.00%

TABLE 20
IPOG vs IPOG-MM (Constrained): Test Generation Results by Strength

Strength	IPOG	IPOG-MM	Reduction Ratio
2	475.20	70.96	85.07%
3	1173.93	196.98	83.22%
4	2475.33	593.47	76.02%
5	5249.96	1670.24	68.19%
6	5249.96	3146.00	40.08%

TABLE 21
IPOG vs IPOG-MM (Constrained): Test Generation Time for G_6 with $t = 5$

Approach	Constrained Group.					
	CG_1	CG_2	CG_3	CG_4	CG_5	average
IPOG	4.54s	4.11s	4.11s	2.49s	3.85s	3.82s
IPOG-MM	23.00s	29.35s	18.30s	22.15s	20.84s	20.73s
overhead ratio ¹	5.06	4.71	4.45	8.90	5.41	5.43

1. overhead ratio = (time cost of IPOG-MM) / (time cost of IPOG)

7.7.1 Unconstrained Experiments

There are 34 (out of 50) unconstrained experiments where the shared VC ratio is greater than 0. Table 23 shows the detailed results for these experiments.

There are 20 experiments where IPOG-MM generates fewer tests than IPOG-PO. IPOG-MM achieves the highest

TABLE 22
Overall Comparison between IPOG and IPOG-MM

Strength	Unconstrained		Constrained	
	IPOG-MM = IPOG	IPOG-MM < IPOG	IPOG-MM = IPOG	IPOG-MM < IPOG
2	2	8	10	40
3	2	6	10	30
4	2	4	10	20
5	1	4	5	20
6	2	3	10	15
Sum.	9	25	45	125
Total # of Experiments: 204				

test reduction ratio of 42.87% for G_8 with $t = 4$.

There are 13 experiments where IPOG-MM generates the same number of tests as IPOG-PO. There is only one unconstrained experiment, G_7 with $t = 2$, where IPOG-MM generates more tests than IPOG-PO. In this experiment, IPOG-PO generates 57 tests in the experiment while IPOG-MM generated 59 tests.

Table 24 shows the average number of tests and the reduction ratio for each strength. IPOG-MM generates fewer tests than IPOG-PO for each strength, but the reduction ratio does not show a consistent relationship with test strength. This is because both IPOG-PO and IPOG-MM are heuristic approaches. The reduction ratio depends on the number of shared VCs and also other factors, e.g., some heuristic decisions made by the two approaches.

TABLE 24
IPOG-PO vs IPOG-MM (Unconstrained): Test Generation Results by Strength

Strength	IPOG-PO	IPOG-MM	Reduction Ratio
2	82.70	72.40	12.45%
3	269.50	200.50	25.60%
4	855.17	598.50	30.01%
5	2236.80	1673.20	25.20%
6	3424.60	3141.00	8.28%

In terms of generation time, we focus on the experiments where IPOG-PO and/or IPOG-MM spent more than one second. There are 12 such experiments. Table 25 shows the detailed results of these experiments. As Table 25 shows, IPOG-MM is slower than IPOG-PO in most of these experiments, but on average, the difference is small.

TABLE 25
IPOG-PO vs IPOG-MM (Unconstrained): Test Generation Time

Group	Strength	IPOG-PO	IPOG-MM
G_6	3	1.23s	2.17s
	4	7.96s	10.42s
	5	36.60s	20.53s
	6	46.73s	29.46s
G_8	3	1.53s	3.18s
	4	8.03s	16.80s
	5	22.55s	43.27s
	6	37.04s	73.99s
G_9	3	0.54s	1.15s
	4	1.16s	4.40s
	5	2.16s	8.95s
	6	2.68s	9.17s
Average		14.02s	18.62s

TABLE 23
IPOG-PO vs IPOG-MM (Unconstrained): Detailed Tests Generation Results

Group	Strength	Shared VC Ratio	IPOG -PO	IPOG -MM	Reduction Ratio	Group	Strength	Shared VC Ratio	IPOG -PO	IPOG -MM	Reduction Ratio
G_1	2	10.64%	23	23	0.00%	G_7	3	43.23%	75	74	1.33%
G_2	2	14.75%	107	85	20.56%	G_7	4	28.95%	98	98	0.00%
	3	5.86%	193	193	0.00%		2	96.22%	118	95	19.49%
G_3	2	42.86%	93	85	8.60%	G_8	3	89.37%	544	313	42.46%
	3	41.31%	222	206	7.20%		4	79.89%	1808	1033	42.87%
	4	38.58%	460	460	0.00%		5	69.03%	4125	2846	31.01%
	5	33.61%	880	880	0.00%		6	57.54%	7022	6732	4.13%
	6	26.64%	1312	1312	0.00%	G_9	2	87.07%	118	95	19.49%
G_4	2	13.68%	55	55	0.00%		3	78.03%	422	315	25.35%
G_4	3	6.94%	55	55	0.00%		4	66.24%	1055	924	12.41%
	2	0.59%	54	54	0.00%		5	52.80%	2165	2110	2.54%
G_5	2	93.83%	132	105	20.45%		6	38.95%	2842	2841	0.04%
G_6	3	90.01%	521	324	37.81%	G_{10}	2	56.58%	70	68	2.86%
	4	85.65%	1583	949	40.05%		3	31.56%	124	124	0.00%
	5	79.62%	3887	2403	38.18%		4	14.09%	127	127	0.00%
	6	71.27%	5820	4693	19.36%		5	5.02%	127	127	0.00%
G_7	2	48.00%	57	59	-3.51%		6	1.46%	127	127	0.00%

7.7.2 Constrained Experiments

Table 26 gives the average results of the 170 constrained experiments, where the shared VC ratio is greater than 0.

TABLE 26
IPOG-PO vs IPOG-MM (Constrained): Test Generation Results
(Note: The results are average results among five random forbidden tuples for each model group)

Model Group	Strength	Shared VC Ratio	IPOG -PO	IPOG -MM	Reduction Ratio
G_1	2	10.92%	22.4	22.4	0.00%
G_2	2	14.85%	106.4	84.8	20.30%
	3	5.95%	188.6	188.6	0.00%
G_3	2	43.19%	90.2	81.8	9.30%
	3	41.64%	216.4	199.2	7.95%
	4	39.12%	454.2	451.2	0.61%
	5	33.84%	865.2	870.6	-0.62%
	6	26.86%	1279.4	1279.4	0.00%
G_4	2	14.37%	51	51	0.00%
	3	7.69%	51	51	0.00%
G_5	2	6.26%	52	52	0.00%
G_6	2	93.83%	130	105	19.22%
	3	90.03%	498	319.6	35.79%
	4	85.68%	1495.8	938.8	37.14%
	5	79.65%	3802.8	2410.8	36.58%
	6	71.31%	5705.8	4803.6	15.77%
G_7	2	48.90%	55.6	56.2	-1.08%
	3	43.99%	72.4	72.2	0.27%
	4	29.69%	92.8	92.8	0.00%
G_8	2	96.00%	118	95	19.49%
	3	89.37%	542	312.8	42.29%
	4	79.89%	1805.8	1034.8	42.70%
	5	69.04%	4115.6	2852.8	30.68%
	6	57.56%	6998.6	6705.4	4.19%
G_9	2	87.10%	117.4	94.8	19.25%
	3	78.05%	420.4	314.6	25.17%
	4	66.26%	1045.8	922.8	11.76%
	5	52.84%	2149	2096.6	2.44%
	6	39.01%	2821.8	2821.2	0.02%
G_{10}	2	56.76%	67.2	66.6	0.84%
	3	31.96%	117.6	117.8	-0.17%
	4	14.40%	120.4	120.4	0.00%
	5	5.15%	120.4	120.4	0.00%
	6	1.49%	120.4	120.4	0.00%

On average, IPOG-MM can generate fewer tests than IPOG-PO. The detailed results (available on our Github) shows IPOG-MM generates fewer tests than IPOG-PO in 93

out of the 170 experiments. Specifically, IPOG-MM achieves the highest reduction ratio of 42.86% for constrained group CG_1 of model group G_8 with $t = 4$.

IPOG-MM generates more tests than IPOG-PO in 9 out of the 170 constrained experiments. Both IPOG-MM and IPOG-PO are heuristic approaches. Thus, it is possible for IPOG-MM to generate more tests than IPOG-PO. For example, in the experiment of G_3 (constrained group CG_4 , $t = 5$), IPOG-MM generates 27 more tests than IPOG-PO. In general, we find that in the experiments where IPOG-MM generates more tests than IPOG-PO, the number of shared VCs for some PCs is often significantly larger than the number of unique VCs for any PC. In IPOG-MM, unique VCs are covered in Phase 1, whereas shared VCs are covered in Phase 2. Thus, more VCs need to be covered in Phase 2 in these experiments. Unlike Phase 1, which includes both horizontal and vertical extension, Phase 2 only includes vertical extension. Recall that horizontal extension could cover VCs without adding any new test. As a result, compared to Phase 1, Phase 2 typically has to generate more tests to cover the same set of VCs. This explains why IPOG-MM generates more tests in these experiments. In contrast, IPOG-PO does not distinguish between unique and shared VCs. Both types of VCs are covered in the same way during test generation.

Table 27 shows the average number of tests and reduction ratio for each strength for the 170 constrained experiments. Similar to unconstrained experiments, the average reduction ratio does not show a consistent relationship with test strength. This is again due to the fact that both IPOG-PO and IPOG-MM are heuristic approaches.

TABLE 27
IPOG-PO vs IPOG-MM (Constrained): Test Generation Results by Strength

Strength	IPOG-PO	IPOG-MM	Reduction Ratio
2	81.02	70.96	12.42%
3	263.30	196.98	25.19%
4	835.80	593.47	28.99%
5	2210.60	1670.24	24.44%
6	3385.20	3146.00	7.07%

In terms of generation time, we focus on the experiments

where IPOG-PO and/or IPOG-MM spent more than one second. There are 60 such experiments. To save space, we do not present the detailed results for all of the 60 experiments, which are available on our Github.

We note that for these 60 experiments, the average times taken by IPOG-PO is 12.91s, and the average times taken by IPOG-MM is 18.12s.

7.7.3 Discussion

Table 28 shows an overall comparison between IPOG-PO and IPOG-MM. We focus on the 204 (out of 300) experiments where there is one or more shared VC during test generation. Note that $MM < PO$ indicates IPOG-MM generates fewer test than IPOG-PO. $MM = PO$ indicates IPOG-MM generates fewer test than IPOG-PO. And $MM > PO$ indicates IPOG-MM generates more tests than IPOG-PO.

TABLE 28
Overall Comparison between IPOG-PO and IPOG-MM

Strength	Unconstrained			Constrained		
	$MM < PO$	$MM = PO$	$MM > PO$	$MM < PO$	$MM = PO$	$MM > PO$
2	6	3	1	28	17	5
3	5	3	0	21	18	1
4	3	3	0	16	12	2
5	3	2	0	15	9	1
6	3	2	0	13	12	0
Sum.	20	13	1	93	68	9
Total # of Experiments: 204						

For the total of 204 experiments, there are a total of 113 experiments where IPOG-MM generates fewer tests than IPOG-PO (55.39%). There are a total of ten experiments where IPOG-MM generates more tests than IPOG-PO (4.90%). There are a total of 81 experiments where IPOG-MM generates the same number of tests as IPOG-PO (39.71%).

For **RQ2**, we make the following conclusion:

In most cases, IPOG-MM can generate fewer tests than IPOG-PO with a test reduction ratio up to 42.87%, when there exist shared VCs between multiple input models. IPOG-MM takes more time than IPOG-PO, but all the experiments also take no more than 90 seconds.

7.8 Results for RQ3

To systematically discuss the impact of shared VCs, we propose two hypotheses for the correlation between shared VC ratio and the test reduction ratio of IPOG-MM over IPOG.

- Null hypothesis (H_0): There is no correlation between the shared VC ratio and the test reduction ratio. That is, the shared VC ratio does not affect the test reduction ratio.
- Alternative hypothesis (H_1): As the shared VC ratio increases, IPOG-MM achieves a higher test reduction ratio.

To determine the rejection and acceptance of the hypotheses, we use the Spearman's rank correlation coefficient [27] [28] to calculate the correlation between the shared

VC ratio and the reduction ratio. Note that Spearman's rank correlation coefficient is a robust and non-parametric correlation test method for discovering the strength of a link between two sets of data [29].

Table 29 shows the calculation results by strength. Note that the p -value is computed in MATLAB [30] using the permutation testing method [31] instead of t -testing, since there is no evidence that shows our data satisfies Student's t -distribution.

TABLE 29
Spearman's Rank Correlation for the Experiments

Strength	Unconstrained		Constrained	
	ρ	p -value	ρ	p -value
2	0.87	< 0.05	0.87	< 0.05
3	0.95	< 0.05	0.94	< 0.05
4	0.93	< 0.05	0.91	< 0.05
5	0.90	0.08	0.88	< 0.05
6	0.98	< 0.05	0.93	< 0.05

As Table 29 shows, the correlation coefficient ρ is always larger than 0.85 and reaches at most 0.98. This indicates there is a strong and positive correlation between the shared VC Ratio and the reduction ratio. The p -value is less than 0.05 in most cases, which provides a strong evidence that the correlation holds with a high possibility instead of by chance [32].

Thus, for **RQ3**, we reject the null hypothesis (H_0), and accept the alternative hypothesis (H_1). We make the following conclusion:

There exists a high positive correlation between shared VC ratio tends and the test reduction ratio achieved by IPOG-MM. This suggests that IPOG-MM can be more effective when the shared VC ratio is high.

7.9 Threats to Validity

One threat to external validity is that the subjects used in our experiments may not be representative. The subjects we used are of different sizes and represent different application domains. Further, the subjects are also used in other studies e.g., [14] and [33]. In the constrained test generation part, we used a tool to randomly generate constraints for the input models. It is important to note that the results do not depend on the actual implementation of the SUTs, only the input models that were constructed based on the test paths and constraints.

There is also a threat that the implementation of IPOG, IPOG-PO and IPOG-MM may not be correct. We carefully examined the results to ensure the correctness. For small subjects, such as *Notepad*, we have manually checked the generated test suite covers all valid VCs and satisfy constraints. For large subjects, such as *cyclos*, we implemented a tool to automatically check the coverage and validity of the tests.

8 RELATED WORK

In this section we review existing work on CT, including both CT for single input model (Section 8.1) and CT for

multiple input models (Section 8.2). We also briefly review existing work on test set reduction (Section 8.3), which is related to our work as it also tries to avoid redundancy in a test set.

8.1 CT for Single Input Model

A large body of work has been reported on t -way test generation for a single input model [2] [12] [13]. Mathematical approaches, e.g., [34] [35] [36], could construct optimal t -way test sets, but they often impose restrictions on an input model and typically do not support constraints, limiting their applications in practice.

Computational approaches can be classified into greedy approaches and search-based approaches. Many greedy approaches e.g., AETG [3] [26] [37] [38], TCG [39], DDA [40] [41], PICT [42], FoCuS [43], the MDD (Multivalued Decision Diagram) based approach [44], adopt a *one-test-at-a-time* framework, where each test is constructed to cover as many uncovered VCs as possible. Other greedy approaches, including IPOG [17] and its variations [18] [45] [46] [47], adopt a *one-parameter-at-a-time* framework, where t -way test set is built to cover the first t parameters, and then is repeatedly extended to cover the remaining parameters, one at a time.

Search-based approaches, e.g. [48] [49] [50] [51] [52] [53], typically start with a randomly construct test set and then use some search strategies to find a t -way test set that is as small as possible.

More recently approaches are reported that convert the problem of t -way test set generation to a constraint satisfaction problem and then use a constraint solver to generate tests [54] [55] [56]. These approaches could leverage the power of existing constraint solvers and handle constraints naturally.

Our work differs from the above work in that we focus on t -way test generation for multiple input models. In principle, the general idea of our approach, i.e., trying to avoid redundant coverage of shared VCs across multiple input models, could be applied to these single-model test generation approaches to extend them for multi-model test generation.

IPOG-MM is built on the top of IPOG. However, there are major differences between the two. IPOG-MM addresses multiple input models with shared parameters. During initialization, IPOG-MM creates initial test sets with the VCs of unique PCs instead of the VCs of first t parameters. During test sets extension, IPOG-MM distinguishes shared parameters and unique parameters. As a result, it extends each test set differently.

IPOG-MM uses the notion of *capacity* to estimate the number of VCs a test set could potentially cover, which is used to choose which test set to extend. Bryce et. al [40] [41] used in their DDA algorithms the concepts of *factor density* and *level density* to estimate the number of VCs a parameter value could potentially cover in a test, which are used to determine which parameter value to use.

While the notion of *capacity* is similar to the two density notions, they are computed very differently and are used for different purposes.

8.2 CT for Multiple Input Models

Anna et al. [57] described a method for combinatorial test generation for multiple input models. Their input models are required to have a sequential relationship, i.e., the output of the previous model is used as the input of the next model. Kampel et al. [58] presented a test generation approach for hierarchical models. There are two levels of input models. Each lower-level model is considered to be an abstract parameter in the upper-level model. During test generation, they first generate a t -way test set for each input model at the lower level. Next, they generate an abstract t -way test set at the upper level, where each test of a lower level model is considered as an abstract value of the corresponding abstract parameter at the upper level. A structure called *nested covering array* is used to represent the final test set. In contrast, our work does not assume a sequential or hierarchical relationship between input models. Instead, input models in our work are in a peer-to-peer relation. Also, no nesting relationship exists between test sets.

Two approaches have been reported on applying CT to systems that are modeled as FSMs. Nguyen et al. [14] first select test paths from an FSM, and then construct an input model from each of the selected test paths. A pairwise test set is constructed for each input model. A post-optimization approach is used to remove redundant tests that may exist in multiple test sets. In contrast, IPOG-MM tries to avoid generating redundant tests in the first place.

Chang et al. [10] reported an empirical study where they constructed multiple input models from the FSMs of a track circuit receiver used in high-speed rail systems, and then generated pairwise test sets for these input models. This study did not remove redundant tests, but it provided the original motivation for the CT-MM problem.

8.3 Test Set Reduction

The problem of test set reduction has been studied extensively in the literature, especially in the context of regression testing [59].

Test set reduction is typically performed with respect to a set of test requirements. A test requirement is an entity that must be covered, e.g. statement, branch, or interaction. The reduced test set is a subset of the original test set that covers the same set of test requirements. Most approaches, e.g. [60] [61] [62], adopt a greedy framework to construct the reduced test set. Each time when they add a test into the reduced test set, they try to select a test that covers the most uncovered test requirements. Some heuristics are developed to optimize the framework, e.g., by covering test requirements that are more difficult to cover first.

Post-optimization [14] can be considered as a test set reduction approach. Instead of trying to add tests into the reduced test set, it tries to remove redundant tests one at a time. A test is redundant if it could be removed while still preserving the same coverage, i.e., covering the same set of test requirements as the original test set.

Test set reduction is in principal a post-mortem approach. That is, redundant tests are removed after they come into existence. In contrast, IPOG-MM tries to prevent redundant tests from being generated in the first place.

9 CONCLUSIONS AND FUTURE WORK

Existing research on CT has mainly considered a single input model. In this paper, we formulated the CT-MM problem, i.e. CT for multiple input models, for the first time. We developed a test generation approach, i.e., IPOG-MM, to address the CT-MM problem. IPOG-MM distinguishes unique VCs from shared VCs, and tries to minimize the number of tests by avoiding redundant coverage of shared VCs. Our experimental results show that IPOG-MM can significantly reduce the number of tests in comparison with two other approaches, i.e., IPOG and IPOG-PO.

The key insight behind existing work on CT for single input model is that many faults in practical applications are caused by interactions between a small number of factors [2]. We believe that this insight also applies when there exist multiple input models. Similar to existing approaches to CT for single input model, IPOG-MM constructs a t -way test suite for multiple input models to cover every t -way interaction at least once. Thus, it is reasonable to believe that tests generated by IPOG-MM would be effective for fault detection. This has been demonstrated by the experimental results in [14]. As part of our future work, we plan to conduct a thorough evaluation of the fault detection effectiveness of the tests generated by IPOG-MM. In particular, we plan to evaluate the impact of avoiding redundant tests on the numbers and types of fault that could be detected in different types of application.

There are several additional directions for future work. First, IPOG-MM currently assumes the same test strength for all the input models. We plan to support different test strengths for different input models. Second, our algorithm currently minimizes the total number of tests. We plan to investigate other optimization goals, e.g., the total length of tests, or the total cost of test execution. The length of a test could be defined as the length of a test path in an FSM or in the source code. The cost of a test could be defined as the time or other resources taken to execute a test. Third, we currently use randomly generated constraints to evaluate the impact of constraints on the performance of IPOG-MM. We plan to investigate the characteristics of constraints that occur in real-world applications and use these characteristics to improve our random constraint generator. We also plan to use real-world constraints, in addition to random constraints, to evaluate the impact of constraints. Finally, we plan to develop algorithms to support model evolution. These algorithms aim to reuse existing test sets as many as possible, i.e., without building them from scratch, when one or more input models are changed due to model correction and/or updates.

ACKNOWLEDGMENT

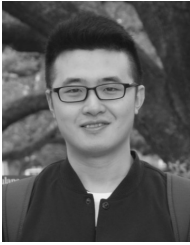
This research is partly supported by China Scholarship Council (201707000108), National Natural Science Foundation of China (Grant No. 61703349), China State Railway Corporation (Grant No. N2018G062 and K2018G011) and Fundamental Research Funds for the Central Universities (Grant No. 2682017CX101). In addition, Lei's work is partly supported by a research grant (70NANB15H199) from National Institute of Standards and Technology.

Disclaimer: We identify certain software products in this document, but such identification does not imply recommendation by the US National Institute of Standards and Technology, nor does it imply that the products identified are necessarily the best available for the purpose.

REFERENCES

- [1] D. R. Kuhn and D. R. Wallace, "Software fault interactions and implications for software testing," *IEEE Transactions on Software Engineering*, vol. 30, no. 6, pp. 418–421, 2004.
- [2] C. Nie and H. Leung, "A survey of combinatorial testing," *ACM Computing Surveys (CSUR)*, vol. 43, no. 2, p. 11, 2011.
- [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 Transactions on Software Engineering*, vol. 23, no. 7, pp. 437–444, 1997.
- [4] X. Yuan, M. B. Cohen, and A. M. Memon, "Gui interaction testing: Incorporating event context," *IEEE Transactions on Software Engineering*, vol. 37, no. 4, pp. 559–574, 2010.
- [5] W. Wang, S. Sampath, Y. Lei, R. N. Kacker, D. R. Kuhn, and J. Lawrence, "Using combinatorial testing to build navigation graphs for dynamic web applications," *Software Testing, Verification and Reliability*, vol. 26, no. 4, pp. 318–346, 2016.
- [6] D. E. Simos, D. R. Kuhn, A. G. Voyiatzis, and R. N. Kacker, "Combinatorial methods in security testing," *IEEE Computer*, vol. 49, no. 10, pp. 80–83, 2016.
- [7] M. F. Johansen, O. Haugen, and F. Fleurey, "An algorithm for generating t -wise covering arrays from large feature models," in *Proceedings of the International Conference on Software Product Lines*. Springer, 2012, pp. 46–55.
- [8] C. Henard, M. Papadakis, G. Perrouin, J. Klein, P. Heymans, and Y. L. Traon, "Bypassing the combinatorial explosion: Using similarity to generate and prioritize t -wise test configurations for software product lines," *IEEE Transactions on Software Engineering*, vol. 40, no. 7, pp. 650–670, 2014.
- [9] N. Li, Y. Lei, H. R. Khan, J. Liu, and Y. Guo, "Applying combinatorial test data generation to big data applications," in *Proceedings of the International Conference on Automated Software Engineering*. ACM, 2016, pp. 637–647.
- [10] C. Rao, J. Guo, N. Li, Y. Lei, Y. Zhang, Y. Li, and Y. Cao, "Applying combinatorial testing to high-speed railway track circuit receiver," in *Proceedings of the International Conference on Software Testing, Verification and Validation Workshops*. IEEE, 2017, pp. 199–207.
- [11] P. Ammann and J. Offutt, *Introduction to software testing*. Cambridge University Press, 2016.
- [12] S. K. Khalsa and Y. Labiche, "An orchestrated survey of available algorithms and tools for combinatorial testing," in *Proceedings of the International Symposium on Software Reliability Engineering*. IEEE, 2014, pp. 323–334.
- [13] H. Wu, C. Nie, J. Petke, Y. Jia, and M. Harman, "A survey of constrained combinatorial testing," *arXiv: Software Engineering*, 2019.
- [14] C. D. Nguyen, A. Marchetto, and P. Tonella, "Combining model-based and combinatorial testing for effective test case generation," in *Proceedings of the International Symposium on Software Testing and Analysis*. ACM, 2012, pp. 100–110.
- [15] Y. Lei, R. Kacker, D. R. Kuhn, V. Okun, and J. Lawrence, "Ipog: A general strategy for t -way software testing," in *Proceedings of the International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)*. IEEE, 2007, pp. 549–556.
- [16] L. Yu, Y. Lei, R. N. Kacker, and D. R. Kuhn, "Acts: A combinatorial test generation tool," in *Proceedings of the International Conference on Software Testing, Verification and Validation*. IEEE, 2013, pp. 370–375.
- [17] Y. Lei, R. Kacker, D. R. Kuhn, V. Okun, and J. Lawrence, "Ipog/ipog-d: efficient test generation for multi-way combinatorial testing," *Software Testing, Verification and Reliability*, vol. 18, no. 3, pp. 125–148, 2008.
- [18] M. A. Forbes, J. Lawrence, Y. Lei, R. N. Kacker, and D. R. Kuhn, "Refining the in-parameter-order strategy for constructing covering arrays," *Journal of Research of the National Institute of Standards and Technology*, vol. 113, no. 5, pp. 287–297, 2008.

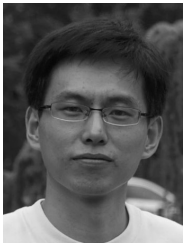
- [19] R. Tzoref-Brill and M. Shahar, "Modify, enhance, select: Co-evolution of combinatorial models and test plans," in *Proceedings of the Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. ACM, 2018, pp. 235–245.
- [20] A. Vahabzadeh, S. Andrea, and A. Mesbah, "Fine-grained test minimization," in *Proceedings of the International Conference on Software Engineering*. IEEE, 2018, pp. 210–221.
- [21] C. D. Nguyen, "M[jagi]c tool and fsm subjects," <http://selab.fbk.eu/magic/>, accessed: 09-01-2019.
- [22] L. Yu, F. Duan, Y. Lei, R. N. Kacker, and D. R. Kuhn, "Constraint handling in combinatorial test generation using forbidden tuples," in *Proceedings of the International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*. IEEE, 2015, pp. 1–9.
- [23] M. Grindal and J. Offutt, "Input parameter modeling for combination strategies," in *Proceedings of the International Conference on Software Engineering*. IEEE, 2007, pp. 255–260.
- [24] P. Satish, K. Sheeba, and K. Rangarajan, "Deriving combinatorial test design model from uml activity diagram," in *Proceedings of the International Conference on Software Testing, Verification and Validation Workshops*. IEEE, 2013, pp. 331–337.
- [25] C. D. Nguyen and P. Tonella, "Automated inference of classifications and dependencies for combinatorial testing," in *Proceedings of the International Conference on Automated Software Engineering*. ACM, 2013, pp. 622–627.
- [26] 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 Transactions on Software Engineering*, vol. 34, no. 5, pp. 633–650, 2008.
- [27] K. H. Zou, K. Tuncali, and S. G. Silverman, "Correlation and simple linear regression," *Radiology*, vol. 227, no. 3, pp. 617–628, 2003.
- [28] J. L. Myers, A. D. Well, and R. F. Lorch Jr, *Research design and statistical analysis*. Routledge, 2013.
- [29] V. H. Durelli, J. Offutt, N. Li, M. E. Delamaro, J. Guo, Z. Shi, and X. Ai, "What to expect of predicates: An empirical analysis of predicates in real world programs," *Journal of Systems and Software*, vol. 113, pp. 324–336, 2016.
- [30] H. Moore, *MATLAB for Engineers*. Pearson, 2017.
- [31] F. Pesarin, *Multivariate permutation tests: with applications in biostatistics*. Wiley Chichester, 2001, vol. 240.
- [32] J. Cohen, *Statistical power analysis for the behavioral sciences*. Routledge, 2013.
- [33] P. Tonella, R. Tiella, and C. D. Nguyen, "Interpolated n-grams for model based testing," in *Proceedings of the 36th International Conference on Software Engineering*. IEEE, 2014, pp. 562–572.
- [34] A. Hartman and L. Raskin, "Problems and algorithms for covering arrays," *Discrete Mathematics*, vol. 284, no. 1–3, pp. 149–156, 2004.
- [35] C. J. Colbourn, "Covering arrays from cyclotomy," *Designs, Codes and Cryptography*, vol. 55, no. 2–3, pp. 201–219, 2010.
- [36] L. Moura, S. Raaphorst, and B. Stevens, "Upper bounds on the sizes of variable strength covering arrays using the lovsz local lemma," *Theoretical Computer Science*, vol. 800, pp. 146–154, 2019.
- [37] M. B. Cohen, M. B. Dwyer, and J. Shi, "Exploiting constraint solving history to construct interaction test suites," in *Proceedings of the Testing: Academic and Industrial Conference Practice and Research Techniques*. IEEE, 2007, pp. 121–132.
- [38] L. Kampel, M. Leithner, and D. E. Simos, "Sliced aetg: a memory-efficient variant of the aetg covering array generation algorithm," *Optimization Letters*, pp. 1–14, 2019.
- [39] Y.-W. Tung and W. S. Aldiwan, "Automating test case generation for the new generation mission software system," in *Proceedings of the Aerospace Conference*, vol. 1. IEEE, 2000, pp. 431–437.
- [40] R. C. Bryce and C. J. Colbourn, "The density algorithm for pairwise interaction testing," *Software Testing, Verification and Reliability*, vol. 17, no. 3, pp. 159–182, 2007.
- [41] —, "A density-based greedy algorithm for higher strength covering arrays," *Software Testing, Verification and Reliability*, vol. 19, no. 1, pp. 37–53, 2009.
- [42] J. Czerwinka, "Pairwise testing in real world," in *Proceedings of the 24th Pacific Northwest Software Quality Conference*, vol. 200. Citeseer, 2006.
- [43] I. Segall, R. Tzorefbrill, and E. Farchi, "Using binary decision diagrams for combinatorial test design," in *Proceedings of the International Symposium on Software Testing and Analysis*. ACM, 2011, pp. 254–264.
- [44] A. Gargantini and P. Vavassori, "Efficient combinatorial test generation based on multivalued decision diagrams," in *Proceedings of the International Symposium on Software Reliability Engineering*, vol. 8855. IEEE, 2014, pp. 220–235.
- [45] L. Yu, Y. Lei, M. Nourozborazjany, R. N. Kacker, and D. R. Kuhn, "An efficient algorithm for constraint handling in combinatorial test generation," in *Proceedings of the International Conference on Software Testing, Verification and Validation*. IEEE, 2013, pp. 242–251.
- [46] K. Kleine and D. E. Simos, "An efficient design and implementation of the in-parameter-order algorithm," *Mathematics in Computer Science*, vol. 12, no. 1, pp. 51–67, 2018.
- [47] H. Zhong, L. Zhang, and S. Khurshid, "Combinatorial generation of structurally complex test inputs for commercial software applications," in *Proceedings of the International Symposium on the Foundation of Software Engineering*. ACM, 2016, pp. 981–986.
- [48] J. Torres-Jimenez and E. Rodriguez-Tello, "New bounds for binary covering arrays using simulated annealing," *Information Sciences*, vol. 185, no. 1, pp. 137–152, 2012.
- [49] H. Wu, C. Nie, F. Kuo, H. Leung, and C. J. Colbourn, "A discrete particle swarm optimization for covering array generation," *IEEE Transactions on Evolutionary Computation*, vol. 19, no. 4, pp. 575–591, 2015.
- [50] J. Torresjimenez, H. Avilageorge, and I. Izquierdomarquez, "A two-stage algorithm for combinatorial testing," *Optimization Letters*, vol. 11, no. 3, pp. 457–469, 2017.
- [51] Y. Jia, M. B. Cohen, M. Harman, and J. Petke, "Learning combinatorial interaction test generation strategies using hyperheuristic search," in *Proceedings of the International Conference on Software Engineering*, vol. 1. IEEE, 2015, pp. 540–550.
- [52] 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," in *Proceedings of the International Conference on Automated Software Engineering*. ACM, 2015, pp. 1–12.
- [53] S. Esfandyari and V. Rafe, "A tuned version of genetic algorithm for efficient test suite generation in interactive t-way testing strategy," *Information and Software Technology*, vol. 94, pp. 165–185, 2018.
- [54] A. Calvagna and A. Gargantini, "A formal logic approach to constrained combinatorial testing," *Journal of Automated Reasoning*, vol. 45, no. 4, pp. 331–358, 2010.
- [55] A. Yamada, T. Kitamura, C. Artho, E. Choi, Y. Oiwa, and A. Biere, "Optimization of combinatorial testing by incremental sat solving," in *Proceedings of the International Conference on Software Testing, Verification and Validation*. IEEE, 2015, pp. 1–10.
- [56] A. Yamada, A. Biere, C. Artho, T. Kitamura, and E.-H. Choi, "Greedy combinatorial test case generation using unsatisfiable cores," in *Proceedings of the International Conference on Automated Software Engineering*. ACM, 2016, pp. 614–624.
- [57] A. Zamansky, A. Shwartz, S. Khoury, and E. Farchi, "A composition-based method for combinatorial test design," in *Proceedings of the International Conference on Software Testing, Verification and Validation Workshops*. IEEE, 2017, pp. 249–252.
- [58] L. Kampel, B. Garn, and D. E. Simos, "Combinatorial methods for modelling composed software systems," in *Proceedings of the International Conference on Software Testing, Verification and Validation Workshops*. IEEE, 2017, pp. 229–238.
- [59] S. Yoo and M. Harman, "Regression testing minimization, selection and prioritization: a survey," *Software Testing, Verification and Reliability*, vol. 22, no. 2, pp. 67–120, 2012.
- [60] M. J. Harrold, R. Gupta, and M. L. Soffa, "A methodology for controlling the size of a test suite," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 2, no. 3, pp. 270–285, 1993.
- [61] T. Y. Chen and M. F. Lau, "A new heuristic for test suite reduction," *Information and Software Technology*, vol. 40, no. 5–6, pp. 347–354, 1998.
- [62] D. E. Blue, I. Segall, R. Tzorefbrill, and A. Zlotnick, "Interaction-based test-suite minimization," in *Proceedings of the International Conference on Software Engineering (ICSE)*. IEEE, 2013, pp. 182–191.



Chang Rao is a PhD candidate in the School of Information Science and Technology, Southwest Jiaotong University. He has 7 publications in journals and proceedings, including INT J COMPUT MATH, IWCT and QRS. His current research interests include the software testing and railway signaling software safety and reliability, especially combinatorial testing, model-based testing and applications of software testing in railway signaling system.



Raghu N. Kacker is currently a Mathematical Statistician with the Applied and Computational Mathematics Division, Information Technology Laboratory, National Institute of Standards and Technology, Gaithersburg, MD, USA. He has coauthored more than 125 refereed publications and one book. His research interests include development and use of combinatorial methods for testing software and systems.



Nan Li is an engineering manager at Dassault Systems, where he is leading mobile development and testing. He serves on program committees for ICST and FSE. He has been serving as a reviewer for multiple journals and conferences including TSE, CSUR, JSS, IST, SoSyM. He won the 2015 Medidata Innovator Award, and holds two U.S. and Canada patents. Li received a BE in Software Engineering from Beihang University and received the PhD in Information Technology from George Mason University.



Yu Lei is a full professor in the Department of Computer Science and Engineering at the University of Texas, Arlington. He received his PhD degree from North Carolina State University. He was a Member of Technical Staff in Fujitsu Network Communications, Inc. for about three years. His research is in the area of automated software analysis, testing and verification, with a special interest in software security assurance at the implementation level.



D. Richard Kuhn is currently a Computer Scientist with the Computer Security Division, National Institute of Standards and Technology (NIST), Gaithersburg, MD, USA. He has authored three books and more than 150 conference or journal publications on information security, empirical studies of software failure, and software assurance. His research interests include software failure analysis and applications of combinatorial methods to software assurance.



Jin Guo is a full professor in the School of Information Science and Technology, Southwest Jiaotong University. He received his PhD degree from Southwest Jiaotong University in 2006. He is a Senior Fellow of China Railway Society. His research interest is in the area of automatic train control methods, railway signaling system safety and reliability, railway signaling system modeling, simulation and testing.



Yadong Zhang is a teacher in the School of Information Science and Technology, Southwest Jiaotong University. He received his PhD degree from Southwest Jiaotong University in 2013. His research interest is in the area of system safety analysis and evaluation, railway signaling system modeling, simulation and testing.