

Constrained Detecting Arrays: Mathematical Structures for Fault Identification in Combinatorial Interaction Testing

Hao Jin^{a,*}, Ce Shi^b and Tatsuhiro Tsuchiya^a

^aOsaka University, Yamadaoka 1-5, Suita City, Osaka, Japan

^bShanghai Lixin University of Accounting and Finance, 2800 Wenxiang Road, Shanghai, China

ARTICLE INFO

Keywords:

Combinatorial interaction testing
Detecting arrays
Constraint handling
Fault Identification

ABSTRACT

Context: Detecting arrays are mathematical structures aimed at fault identification in combinatorial interaction testing. However, they cannot be directly applied to systems that have constraints among test parameters. Such constraints are prevalent in real-world systems.

Objectives: This paper proposes Constrained Detecting Arrays (CDAs), an extension of detecting arrays, which can be used for systems with constraints.

Methods: The paper examines the properties and capabilities of CDAs with rigorous arguments. The paper also proposes two algorithms for constructing CDAs: One is aimed at generating minimum CDAs and the other is a heuristic algorithm aimed at fast generation of CDAs. The algorithms are evaluated through experiments using a benchmark dataset.

Results: Experimental results show that the first algorithm can generate minimum CDAs if a sufficiently long generation time is allowed, and the second algorithm can generate minimum or near-minimum CDAs in a reasonable time.

Conclusion: CDAs enhance detecting arrays to be applied to systems with constraints. The two proposed algorithms have different advantages with respect to the array size and generation time.

1. Introduction

Combinatorial Interaction Testing (CIT) is a testing approach that aims to exercise interactions among test parameters. The basic strategy of CIT is to test all interactions among a specified number (usually a small integer such as 2 or 3) of parameters. Empirical results suggest that it is sufficient to only test those interactions involving a small number of parameters to reveal most of the latent faults [1, 2]. Using CIT can cut off testing cost significantly when compared to exhaustive testing. The test suites used in CIT are usually modeled as *arrays* where each row represents a test case and each column corresponds to each test parameter. The most typical class of arrays used for CIT is *t-way Covering Arrays* (*t*-CAs). In a *t*-CA every interaction involving *t* parameters appears in at least one test case; thus the use of a *t*-CA ensures exercising all *t*-way interactions.

There are many directions to expand the capability of CIT. One of the directions is to add fault localization capability to test suites. *(d, t)-Locating Arrays* (LAs) and *(d, t)-Detecting Arrays* (DAs) proposed in [3] represent test suites that can not only detect but also identify faulty interactions. The integers *d* and *t* are predefined parameters: *d* represents the number of faulty interactions that can be identified and *t* represents the number of parameters involved in the faulty interactions. LAs and DAs add this capability to CAs at the cost of an increased number of test cases.

Another direction of expanding CIT is to incorporate constraints. Real-world systems usually have constraints on the input space. These constraints are originated from, for example, user-defined requirements or running environment restrictions. In order to test systems with constraints correctly, proper handling of the constraints is necessary. For example, all test cases must satisfy the constraints. In addition, constraints may make some interactions no longer testable. These *invalid* interactions require additional handling. *Constrained Covering Arrays* (CCAs) are an extension of CAs in which such constraints are incorporated into the definition. Many previous studies on CIT have tackled the problem of generating CCAs of small sizes [4, 5, 6].

In [7] we proposed the concept of *Constrained Locating Arrays* (CLAs) which extends LAs by incorporating constraints. CLAs inherit basic properties of LAs and at the same time can be used as test suites for systems that have constraints on the input space. In this paper, we further develop this line of research: We propose a new mathematical structure called *Constrained Detecting Arrays* (CDAs). As the name suggests, CDAs extend DAs by incorporating

* k-kou@ist.osaka-u.ac.jp (H. Jin); shice060@lixin.edu.cn (C. Shi); t-tutiya@ist.osaka-u.ac.jp (T. Tsuchiya)
ORCID(s):

Table 1

SUT: an online shopping mobile application [8]

\mathcal{F}	F_1 (Total Price)	F_2 (Shipping Address)	F_3 (Shipping Method)	F_4 (Payment Method)
S	0: \$50	0: Domestic	0: Same-Day Delivery	0: Visa
	1: \$500	1: International	1: 2-Day Delivery	1: Mastercard
	2: \$1000	--	2: 7-Day Delivery	2: Paypal
	--	--	--	3: Gift Card
ϕ	$\phi_1 : \text{Shipping Address} = \text{International} \Rightarrow \text{Shipping Method} \neq \text{Same-Day Delivery}$			
	$\phi_2 : \text{Payment} = \text{Gift Card} \Rightarrow \text{Shipping Address} = \text{Domestic} \wedge \text{Shipping Method} = \text{Same-Day Delivery}$			

constraints. Compared to LAs, DAs allow more accurate localization of faulty interactions. Specifically, DAs prevent faulty interactions from being erroneously identified as non-faulty, even when there are more faulty interactions than assumed. CDAs extend DAs to adapt to constraints while inheriting the good property of DAs.

The remainder of the paper is organized as follows. In Section 2 we explain CAs, LAs, and DAs, and then describe CCAs and CLAs with examples. In Section 3 we introduce the notion of CDAs. We also provide some basic properties of CDAs. In Section 4 we propose two algorithms for constructing CDAs: One leverages a solver for satisfiability problems and the other is a fast heuristic algorithm. The results of experiments using these algorithms are shown in Section 5. In Section 6 we discuss threats to validity. In Section 7 we summarize related work. Finally, we conclude this paper in Section 8.

2. Preliminaries

2.1. SUT models and basic notions of CIT

A *System Under Test* (SUT) is modeled as a 3-tuple $\mathcal{M} = \{\mathcal{F}, \mathcal{S}, \phi\}$. $\mathcal{F} = \{F_1, F_2, \dots, F_k\}$ is a set of parameters in the system where k is the total number of these parameters. $\mathcal{S} = \{S_1, S_2, \dots, S_k\}$ is a set of domains for all the parameters in \mathcal{F} , where S_i is the domain for the parameter F_i . Each domain S_i consists of two or more integers ranging from 0 to $|S_i| - 1$, i.e., $S_i = \{0, 1, \dots, |S_i| - 1\}$. Different values of S_i represent different values for the parameter F_i . $\phi : S_1 \times S_2 \times \dots \times S_k \rightarrow \{\text{true}, \text{false}\}$ is a mapping representing the system constraints. We assume that $\phi(\sigma) = \text{true}$ for some σ .

A *test case* is an element of $S_1 \times S_2 \times \dots \times S_k$; that is, a test case is a k -tuple $\langle \sigma_1, \dots, \sigma_i, \dots, \sigma_k \rangle$ such that $\sigma_i \in S_i$. A *test suite* is a set of test cases. We view a test suite as an $N \times k$ array where each row represents a test case and N is the number of test cases. The *size* of a test suite (i.e., array) is the number of test cases (rows) in it.

An *interaction* is a set of parameter-value pairs such that no parameters are overlapped. The *strength* of an interaction is the number of parameter-value pairs in the interaction. That is, $\{(F_{i_1}, \sigma_1), \dots, (F_{i_t}, \sigma_t)\}$ is an interaction of strength t if and only if (iff) $F_{i_j} \neq F_{i_k}$ for any i_j, i_k ($i_j \neq i_k$) and $\sigma_j \in S_{i_j}$ for all $i_j \in \{i_1, \dots, i_t\}$. We let λ , instead of \emptyset , denote the interaction of strength 0 which is an empty set. We say that an interaction is t -way iff the strength is t .

An interaction T_1 covers another interaction T_2 iff $T_2 \subseteq T_1$. We say that a set \mathcal{T} of interactions are *independent* iff $T_1 \not\subseteq T_2$ for any $T_1, T_2 \in \mathcal{T}, T_1 \neq T_2$.

We also say that a test case covers an interaction iff the value matches between the test case and the interaction on every parameter involved in the interaction. Formally, $\sigma = \langle \sigma_1, \dots, \sigma_i, \dots, \sigma_k \rangle$ covers an interaction $T = \{(F_{i_1}, \sigma'_1), \dots, (F_{i_t}, \sigma'_t)\}$ iff $\sigma_{i_j} = \sigma'_j$ for all $j \in \{i_1, \dots, i_t\}$. Given an array (i.e., test suite) A , we express the set of rows (i.e., test cases) that cover the interaction T as $\rho_A(T)$. Also we let $\rho_A(\mathcal{T}) = \bigcup_{T \in \mathcal{T}} \rho_A(T)$. In words, $\rho_A(\mathcal{T})$ is the set of rows that cover at least one interaction in \mathcal{T} . Note that $\rho_A(\emptyset) = \emptyset$ and $\rho_A(\lambda) = A$.

A test case σ is *valid* iff it satisfies the constraints ϕ , i.e., $\phi(\sigma) = \text{true}$; otherwise, *invalid*. The set of all valid test cases is denoted as $\mathcal{R} (\subseteq S_1 \times S_2 \times \dots \times S_k)$. Hence \mathcal{R} is also regarded as the *exhaustive test suite* which consists of all valid test cases.

The valid/invalid distinction also applies to interactions: Interactions that no valid test cases can cover are *invalid*; the other interactions, i.e., those that are covered by at least one valid test case are *valid*. We let I_t and \mathcal{VI}_t denote the set of all t -way interactions and the set of all valid t -way interactions, respectively. Similarly we let \overline{I}_t and $\overline{\mathcal{VI}}_t$ be the set of all interactions of strength at most t and the set of all valid interactions of strength at most t .

A valid interaction is either *faulty* or *non-faulty*. The outcome of execution of a valid test case is either *PASS* or *FAIL*. The outcome is *FAIL* iff the test case covers at least one faulty interaction; the outcome is *PASS* otherwise. The

test outcome of an array is the collection of the outcomes of all rows.

Table 1 shows an SUT model which represents an online shopping mobile application. This example, which is a modification of one in [8], serves as a running example throughout the paper. This model is formally represented as $\mathcal{M} = \{\mathcal{F}, \mathcal{S}, \phi\}$ where $\mathcal{F} = \{F_1, F_2, F_3, F_4\}$, $\mathcal{S} = \{S_1, S_2, S_3, S_4\}$, $S_1 = S_3 = \{0, 1, 2\}$, $S_2 = \{0, 1\}$, $S_4 = \{0, 1, 2, 3\}$ and $\phi = \phi_1 \wedge \phi_2 = (F_2 = 1 \Rightarrow F_3 \neq 0) \wedge (F_4 = 3 \Rightarrow (F_2 = 0 \wedge F_3 = 0))$. An example of a valid test case is $\langle 0, 0, 0, 0 \rangle$, representing $\langle \$50, \text{Domestic, Same-Day Delivery, Visa} \rangle$. On the other hand, $\langle 0, 1, 0, 0 \rangle$ (i.e., $\langle \$50, \text{International, Same-Day Delivery, Visa} \rangle$) is an invalid test case. Invalid interactions include, for example, $\{(F_2, 1), (F_3, 0)\}, \{(F_2, 1), (F_4, 3)\}$, etc.

2.2. Covering arrays, locating arrays, and detecting arrays

Covering arrays (CAs), *locating arrays* (LAs), and *detecting arrays* (DAs) are mathematical structures that can be implemented as test suites. They are usually used to detect or locate faulty interactions for unconstrained SUTs (i.e., $\phi = \text{true}$).

A t -way covering array, t -CA for short, is defined as follows:

$$t\text{-CA} \quad \forall T \in \mathcal{I}_t : \rho_A(T) \neq \emptyset$$

The condition requires that all interactions T in \mathcal{I}_t be covered by at least one row in the array. In other words, when the test cases in A are executed, all t -way interactions are examined or executed at least once. This condition is sufficient to reveal the existence of a t -way faulty interaction; but it is generally not possible to identify the faulty interaction from the test outcome. Figure 1a shows a 2-CA for the running example.

On the other hand, LAs and DAs can be used to not only detect the existence of faulty interactions but also locate them. LAs and DAs were first proposed by Colbourn and McClary in [3]. They introduced a total of six types for both LAs and DAs according to fault locating capability. Two types of them exist only in extreme cases. The rest four types, namely, (d, t) -, (\bar{d}, t) -, (d, \bar{t}) -, (\bar{d}, \bar{t}) -LA (and DA), are as follows ($d \geq 0, 0 \leq t \leq k$).

$$(d, t)\text{-LA} \quad \forall \mathcal{T}_1, \mathcal{T}_2 \subseteq \mathcal{I}_t \text{ such that } |\mathcal{T}_1| = |\mathcal{T}_2| = d :$$

$$\rho_A(\mathcal{T}_1) = \rho_A(\mathcal{T}_2) \Leftrightarrow \mathcal{T}_1 = \mathcal{T}_2$$

$$(\bar{d}, t)\text{-LA} \quad \forall \mathcal{T}_1, \mathcal{T}_2 \subseteq \mathcal{I}_t \text{ such that } 0 \leq |\mathcal{T}_1| \leq d, 0 \leq |\mathcal{T}_2| \leq d :$$

$$\rho_A(\mathcal{T}_1) = \rho_A(\mathcal{T}_2) \Leftrightarrow \mathcal{T}_1 = \mathcal{T}_2$$

$$(d, \bar{t})\text{-LA} \quad \forall \mathcal{T}_1, \mathcal{T}_2 \subseteq \overline{\mathcal{I}_t} \text{ such that } |\mathcal{T}_1| = |\mathcal{T}_2| = d \text{ and } \mathcal{T}_1, \mathcal{T}_2 \text{ are independent} :$$

$$\rho_A(\mathcal{T}_1) = \rho_A(\mathcal{T}_2) \Leftrightarrow \mathcal{T}_1 = \mathcal{T}_2$$

$$(\bar{d}, \bar{t})\text{-LA} \quad \forall \mathcal{T}_1, \mathcal{T}_2 \subseteq \overline{\mathcal{I}_t} \text{ such that } 0 \leq |\mathcal{T}_1| \leq d, 0 \leq |\mathcal{T}_2| \leq d \text{ and } \mathcal{T}_1, \mathcal{T}_2 \text{ are independent} :$$

$$\rho_A(\mathcal{T}_1) = \rho_A(\mathcal{T}_2) \Leftrightarrow \mathcal{T}_1 = \mathcal{T}_2$$

$$(d, t)\text{-DA} \quad \forall \mathcal{T} \subseteq \mathcal{I}_t \text{ such that } |\mathcal{T}| = d, \forall T \in \mathcal{I}_t :$$

$$\rho_A(T) \subseteq \rho_A(\mathcal{T}) \Leftrightarrow T \in \mathcal{T}$$

$$(\bar{d}, t)\text{-DA} \quad \forall \mathcal{T} \subseteq \mathcal{I}_t \text{ such that } 0 \leq |\mathcal{T}| \leq d, \forall T \in \mathcal{I}_t :$$

$$\rho_A(T) \subseteq \rho_A(\mathcal{T}) \Leftrightarrow T \in \mathcal{T}$$

$$(d, \bar{t})\text{-DA} \quad \forall \mathcal{T} \subseteq \overline{\mathcal{I}_t} \text{ such that } |\mathcal{T}| = d, \forall T \in \overline{\mathcal{I}_t} \text{ and } \mathcal{T} \cup \{T\} \text{ is independent} :$$

$$\rho_A(T) \subseteq \rho_A(\mathcal{T}) \Leftrightarrow T \in \mathcal{T}$$

$$(\bar{d}, \bar{t})\text{-DA} \quad \forall \mathcal{T} \subseteq \overline{\mathcal{I}_t} \text{ such that } 0 \leq |\mathcal{T}| \leq d, \forall T \in \overline{\mathcal{I}_t} \text{ and } \mathcal{T} \cup \{T\} \text{ is independent} :$$

$$\rho_A(T) \subseteq \rho_A(\mathcal{T}) \Leftrightarrow T \in \mathcal{T}$$

The parameter d of these arrays stands for the number of faulty interactions that the array can correctly locate, while t represents the strength of the target interactions. Writing \bar{d} or \bar{t} in place of d or t means that the array permits

Figure 1: CA, LA, and DA for the running example (constraint ignored)

(a) 2-CA				(b) (1,2)-LA				(c) (1,2)-DA				
	F_1	F_2	F_3		F_1	F_2	F_3		F_1	F_2	F_3	F_4
σ_1	0	0	0	0	σ_1	0	0	0	0	0	0	0
σ_2	0	0	2	2	σ_2	0	0	1	1	0	0	2
σ_3	0	1	0	3	σ_3	0	0	1	2	0	0	1
σ_4	0	1	1	1	σ_4	0	1	0	3	0	0	2
σ_5	1	0	0	1	σ_5	0	1	2	0	0	1	0
σ_6	1	0	2	3	σ_6	0	1	2	2	0	1	1
σ_7	1	1	0	2	σ_7	1	0	0	1	0	1	2
σ_8	1	1	1	0	σ_8	1	0	1	0	0	1	2
σ_9	2	0	0	3	σ_9	1	0	2	1	1	0	0
σ_{10}	2	0	1	2	σ_{10}	1	1	0	0	1	0	1
σ_{11}	2	0	2	0	σ_{11}	1	1	0	2	1	0	2
σ_{12}	2	1	1	3	σ_{12}	1	1	1	1	0	2	2
σ_{13}	2	1	2	1	σ_{13}	1	1	2	3	1	1	0
σ_{14}					σ_{14}	2	0	0	2	1	1	1
σ_{15}					σ_{15}	2	0	0	3	1	1	1
σ_{16}					σ_{16}	2	0	1	1	1	2	1
σ_{17}					σ_{17}	2	0	2	3	2	0	0
σ_{18}					σ_{18}	2	1	0	0	2	0	1
σ_{19}					σ_{19}	2	1	1	3	2	0	1
					σ_{20}	2	0			2	0	2
					σ_{21}	2	1			1	0	0
					σ_{22}	2	1			1	0	2
					σ_{23}	2	1			1	1	1
					σ_{24}	2	1			1	2	3

at most of d faulty interactions or strength at most t . For example, a (1, 2)-LA (or DA) can locate one 2-way faulty interaction, while a ($\bar{2}, \bar{2}$)-LA (or DA) can locate at most two faulty interactions that have strength not greater than $\underline{2}$. The reason why when dealing with $\bar{\mathcal{I}}_t$ it is required that $\mathcal{T}_1, \mathcal{T}_2$ or $\mathcal{T} \cup \{T\}$ be independent is that if there are $T_1, T_2 \in \bar{\mathcal{I}}_t$ such that $T_1 \subset T_2$, whether T_2 is faulty or not cannot be determined when T_1 is faulty. Figure 1b and Figure 1c show a (1, 2)-LA and a (1, 2)-DA for the running SUT.

In [3] it is proved that a (d, t) -DA is a (d, t) -LA and a (\bar{d}, t) -LA and that a (\bar{d}, t) -LA is a $(d-1, t)$ -DA (Lemma 7.1). It is also proved that a (d, t) -DA is equivalent to a (\bar{d}, t) -DA and that a (d, \bar{t}) -DA is equivalent to a (\bar{d}, \bar{t}) -DA (Lemma 7.2). We will later provide theorems for CDAs, namely Theorems 2 and 5, that are parallel to these lemmas.

How to identify faulty interactions using these arrays is the same as their constrained versions, namely, CLAs and CDAs.

2.3. Constrained versions of covering arrays and locating arrays

CAs, LAs, and DAs do not take constraints into account. However real-world systems usually have complicated constraints that must be satisfied by all test cases.

2.3.1. Constrained covering arrays

Constrained Covering Arrays (CCAs) are the constrained version of CAs. CCAs are the most common form of test suites used in CIT: Most of test generation tools for CIT are in effect generators of CCAs.

Definition 1 (CCA). An array A that consists of valid test cases is a t -CCA iff the following condition holds.

$$t\text{-CCA} \quad \forall T \in \mathcal{VI}_t : \rho_A(T) \neq \emptyset$$

The definition of CCAs requires that all valid t -way interactions be covered by at least one test case in the test suite. This condition implies that every valid interaction of strength $< t$ is covered by at least one test case. That is, a t -CCA

Figure 2: CCA and CLA for the running example

	(a) 2-CCA				(b) (1,2)-CLA				
	F_1	F_2	F_3	F_4		F_1	F_2	F_3	F_4
σ_1	0	0	0	0	σ_1	0	0	0	0
σ_2	0	0	0	3	σ_2	0	0	0	3
σ_3	0	1	1	1	σ_3	0	0	1	1
σ_4	0	1	2	2	σ_4	0	1	1	2
σ_5	1	0	0	2	σ_5	0	1	2	0
σ_6	1	0	0	3	σ_6	1	0	0	2
σ_7	1	0	2	1	σ_7	1	0	0	3
σ_8	1	1	1	0	σ_8	1	0	1	2
σ_9	2	0	0	1	σ_9	1	1	1	1
σ_{10}	2	0	0	3	σ_{10}	1	1	2	0
σ_{11}	2	0	1	2	σ_{11}	1	1	2	2
σ_{12}	2	1	2	0	σ_{12}	2	0	0	1

	F_1	F_2	F_3	F_4
σ_1	0	0	0	0
σ_2	0	0	0	3
σ_3	0	0	1	1
σ_4	0	1	1	2
σ_5	0	1	2	0
σ_6	1	0	0	2
σ_7	1	0	0	3
σ_8	1	0	1	2
σ_9	1	1	1	1
σ_{10}	1	1	2	0
σ_{11}	1	1	2	2
σ_{12}	2	0	0	1
σ_{13}	2	0	0	3
σ_{14}	2	0	2	0
σ_{15}	2	1	1	0
σ_{16}	2	1	1	2
σ_{17}	2	1	2	1

is also a $(t - 1)$ -CCA when $t > 0$. Thus, a t -CCA can also be defined as follows.

$$t\text{-CCA} \quad \forall T \in \overline{\mathcal{VI}_t} : \rho_A(T) \neq \emptyset$$

Figure 2a shows a 2-CCA for the running example. All four invalid 2-way interactions which violate the constraints are listed as follows.

$$\begin{array}{ll} \{(F_2, 1), (F_3, 0)\} & \{(F_2, 1), (F_4, 3)\} \\ \{(F_3, 2), (F_4, 3)\} & \{(F_3, 1), (F_4, 3)\} \end{array}$$

It is easy to observe that none of the invalid interactions appears in any rows in Figure 2a.

2.3.2. Constrained locating arrays

LAs allow us to identify the set of faulty interactions using the test outcome. This becomes possible because for any LA A , $\rho_A(\cdot)$ injectively maps an interaction set to a test outcome (i.e., $\rho_A(\mathcal{T}_1) = \rho_A(\mathcal{T}_2) \Rightarrow \mathcal{T}_1 = \mathcal{T}_2$). Since a test outcome corresponds to at most one interaction set, the set of faulty interactions can be uniquely inferred from the test outcome.

When incorporating constraints into LAs, it is necessary to handle the situation where constraints may prevent some sets of faulty interactions from being identified. For the running example, when either one of the two interaction sets shown below is faulty, it is impossible to determine which is indeed faulty.

$$\mathcal{T}_1 = \{\{(F_2, 0), (F_4, 3)\}\} \quad \mathcal{T}_2 = \{\{(F_3, 0), (F_4, 3)\}\}$$

This is because the two interactions always appear simultaneously in any valid test case and thus no valid test case exists that yields different outcomes for the two faulty interaction sets. We say that two sets of valid interactions, \mathcal{T} and \mathcal{T}' , are not *distinguishable* or *indistinguishable* iff $\rho_A(\mathcal{T}) = \rho_A(\mathcal{T}')$ for any array A that consists of valid tests.

The definition of CLAs adapts LAs to the presence of indistinguishable interaction sets by exempting them from fault identification.

Definition 2 (CLA). Let $d \geq 0$ and $0 \leq t \leq k$. An array A that consists of valid tests is a (d, t) -, (\bar{d}, t) -, (d, \bar{t}) - or

(\bar{d}, \bar{t}) -CLA iff the corresponding condition shown below holds.

- | | |
|---------------------------|--|
| (d, t) -CLA | $\forall \mathcal{T}_1, \mathcal{T}_2 \subseteq \mathcal{VI}_t$ such that $ \mathcal{T}_1 = \mathcal{T}_2 = d$ and $\mathcal{T}_1, \mathcal{T}_2$ are distinguishable :
$\rho_A(\mathcal{T}_1) \neq \rho_A(\mathcal{T}_2)$ |
| (\bar{d}, t) -CLA | $\forall \mathcal{T}_1, \mathcal{T}_2 \subseteq \mathcal{VI}_t$ such that $0 \leq \mathcal{T}_1 \leq d, 0 \leq \mathcal{T}_2 \leq d$ and $\mathcal{T}_1, \mathcal{T}_2$ are distinguishable :
$\rho_A(\mathcal{T}_1) \neq \rho_A(\mathcal{T}_2)$ |
| (d, \bar{t}) -CLA | $\forall \mathcal{T}_1, \mathcal{T}_2 \subseteq \overline{\mathcal{VI}_t}$ such that $ \mathcal{T}_1 = \mathcal{T}_2 = d$ and $\mathcal{T}_1, \mathcal{T}_2$ are independent and distinguishable :
$\rho_A(\mathcal{T}_1) \neq \rho_A(\mathcal{T}_2)$ |
| (\bar{d}, \bar{t}) -CLA | $\forall \mathcal{T}_1, \mathcal{T}_2 \subseteq \overline{\mathcal{VI}_t}$ such that $0 \leq \mathcal{T}_1 \leq d, 0 \leq \mathcal{T}_2 \leq d$ and $\mathcal{T}_1, \mathcal{T}_2$ are independent and distinguishable :
$\rho_A(\mathcal{T}_1) \neq \rho_A(\mathcal{T}_2)$ |

A (1,2)-CLA for the running example is shown in Figure 2b. From the array, one can see that none of the invalid interactions shown above appears in any rows of the CLA. In addition, for any pair of 2-way interactions except the above indistinguishable pair, the rows that cover one of them are different from those that cover the other. The exception is the rows where the indistinguishable pair of interaction sets appear, namely σ_2 , σ_7 and σ_{13} . This occurs because of the second constraint ϕ_2 in the SUT. ϕ_2 enforces all test cases that have $(F_4, 3)$ to contain $(F_2, 0)$ and $(F_3, 0)$ at the same time. Thus, the interaction sets $\mathcal{T}_1 = \{(F_2, 0), (F_4, 3)\}$ and $\mathcal{T}_2 = \{(F_3, 0), (F_4, 3)\}$ appear in the same test cases as long as the test cases are valid.

3. Constrained Detecting Arrays

In this section, we propose Constrained Detecting Arrays (CDAs). First we present the definition of CDAs; then we show how one can identify faulty interactions with CDAs. In addition, we provide some theorems that relate CDAs to CCAs and CLAs.

3.1. Definition

For an array A to be a DA, A must satisfy $\rho_A(T) \subseteq \rho_A(\mathcal{T}) \Leftrightarrow T \in \mathcal{T}$ for any pair of an interaction T and an interaction set \mathcal{T} . However, this may be impossible if A consists only of valid test cases. Here we introduce the concept of *masking* to capture such a situation.

Definition 3 (Masking). A set \mathcal{T} of valid interactions *masks* a valid interaction T iff $T \notin \mathcal{T}$ and

$$\forall \sigma \in \mathcal{R} : T \subseteq \sigma \Rightarrow (\exists T' \in \mathcal{T} : T' \subseteq \sigma).$$

If \mathcal{T} masks T , we write $\mathcal{T} > T$; otherwise we write $\mathcal{T} \not> T$. By definition, $\mathcal{T} \not> T$ iff $T \in \mathcal{T}$ or

$$\exists \sigma \in \mathcal{R} : T \subseteq \sigma \wedge (\forall T' \in \mathcal{T} : T' \not\subseteq \sigma).$$

In words, when \mathcal{T} masks T , T always appears together with some interaction T' in \mathcal{T} in any valid test case σ . In this case, $T \notin \mathcal{T}$ but $\rho_A(T) \subseteq \rho_A(\mathcal{T})$ always holds for any A that meets the constraints. In the running example, such T - \mathcal{T} pairs include:

$$\begin{array}{ll} \mathcal{T}_1 = \{(F_1, 0), (F_2, 0)\} > T_a = \{(F_1, 0), (F_3, 0)\} & \mathcal{T}_1 = \{(F_1, 0), (F_2, 0)\} > T_b = \{(F_1, 0), (F_4, 3)\} \\ \mathcal{T}_2 = \{(F_1, 1), (F_2, 0)\} > T_c = \{(F_1, 1), (F_3, 0)\} & \mathcal{T}_3 = \{(F_2, 0), (F_3, 0)\} > T_b = \{(F_1, 0), (F_4, 3)\} \\ \mathcal{T}_4 = \{(F_3, 0), (F_4, 3)\} > T_d = \{(F_1, 2), (F_4, 3)\} & \mathcal{T}_4 = \{(F_3, 0), (F_4, 3)\} > T_e = \{(F_2, 0), (F_4, 3)\} \\ & \dots \dots \text{(31 pairs in total)} \end{array}$$

When \mathcal{T} masks T , the failure caused by T cannot be inherently distinguished from that caused by \mathcal{T} . The idea at the core of CDAs is to relax the condition of DAs by exempting T - \mathcal{T} pairs such that $\mathcal{T} > T$ from fault identification.

Figure 3: CDAs for the running example

(a) (1,1)-CDA				(b) (2,1)-CDA				(c) (1,2)-CDA				
	F_1	F_2	F_3		F_1	F_2	F_3		F_1	F_2	F_3	F_4
σ_1	0	0	0	0	σ_1	0	0	0	0	0	0	0
σ_2	0	1	2	1	σ_2	0	0	0	3	0	0	0
σ_3	1	0	0	3	σ_3	0	0	2	1	0	0	3
σ_4	1	0	1	1	σ_4	0	1	1	2	0	0	1
σ_5	1	1	1	2	σ_5	0	1	2	0	0	0	2
σ_6	2	0	0	3	σ_6	1	0	1	1	0	1	1
σ_7	2	0	2	2	σ_7	1	0	0	1	0	1	2
σ_8	2	1	1	0	σ_8	1	0	0	2	0	1	2
					σ_9	1	0	0	3	1	0	0
					σ_{10}	1	1	1	0	1	0	0
					σ_{11}	1	1	2	1	1	0	1
					σ_{12}	2	0	0	3	1	0	2
					σ_{13}	2	0	1	2	1	1	1
					σ_{14}	2	0	2	0	1	1	2
					σ_{15}	2	1	1	1	1	1	2
					σ_{16}	2	1	2	2	2	0	0
									σ_{17}	2	0	0
									σ_{18}	2	0	0
									σ_{19}	2	0	0
									σ_{20}	2	0	1
									σ_{21}	2	0	2
									σ_{22}	2	1	1
									σ_{23}	2	1	1
									σ_{24}	2	1	2

Definition 4 (CDA). Let $d \geq 0$ and $0 \leq t \leq k$. An array A that consists of valid test cases or no rows is a (d, t) -, (\bar{d}, t) -, (d, \bar{t}) - or (\bar{d}, \bar{t}) -CDA iff the corresponding condition shown below holds.

- | | |
|---------------------------|---|
| (d, t) -CDA | $\forall \mathcal{T} \subseteq \mathcal{VI}_t$ such that $ \mathcal{T} = d, \forall T \in \mathcal{VI}_t :$
$\mathcal{T} \succ T \Rightarrow (T \in \mathcal{T} \Leftrightarrow \rho_A(T) \subseteq \rho_A(\mathcal{T}))$ |
| (\bar{d}, t) -CDA | $\forall \mathcal{T} \subseteq \mathcal{VI}_t$ such that $0 \leq \mathcal{T} \leq d, \forall T \in \mathcal{VI}_t :$
$\mathcal{T} \succ T \Rightarrow (T \in \mathcal{T} \Leftrightarrow \rho_A(T) \subseteq \rho_A(\mathcal{T}))$ |
| (d, \bar{t}) -CDA | $\forall \mathcal{T} \subseteq \overline{\mathcal{VI}_t}$ such that $ \mathcal{T} = d, \forall T \in \overline{\mathcal{VI}_t}$ and $\mathcal{T} \cup \{T\}$ is independent :
$\mathcal{T} \succ T \Rightarrow (T \in \mathcal{T} \Leftrightarrow \rho_A(T) \subseteq \rho_A(\mathcal{T}))$ |
| (\bar{d}, \bar{t}) -CDA | $\forall \mathcal{T} \subseteq \overline{\mathcal{VI}_t}$ such that $0 \leq \mathcal{T} \leq d, \forall T \in \overline{\mathcal{VI}_t}$ and $\mathcal{T} \cup \{T\}$ is independent :
$\mathcal{T} \succ T \Rightarrow (T \in \mathcal{T} \Leftrightarrow \rho_A(T) \subseteq \rho_A(\mathcal{T}))$ |

Figure 3a, Figure 3b, and Figure 3c respectively show a (1,1)-CDA, a (2,1)-CDA, and a (1,2)-CDA for the running example. Now let us take the (1,2)-CDA in Figure 3c and the pair of $\mathcal{T}_3 = \{\{(F_2, 0), (F_3, 0)\}\}$ and $T_b = \{(F_1, 0), (F_4, 3)\}$ as examples. Let A denote the (1,2)-CDA for now; then $\rho_A(\mathcal{T}_3) = \{\sigma_1, \sigma_2, \sigma_3, \sigma_9, \sigma_{10}, \sigma_{16}, \sigma_{17}, \sigma_{18}, \sigma_{19}\}$, whereas $\rho_A(T_b) = \{\sigma_3\}$. Hence, $T_b \in \mathcal{T}_3 \Leftrightarrow \rho_A(T_b) \subseteq \rho_A(\mathcal{T}_3)$ does not hold. This is prohibited by the definition of DAs but is allowed by CDAs, because $\mathcal{T}_3 \succ T_b$, which means that no array can satisfy it unless the constraints are violated.

By definition, it is straightforward to see that the following observations hold.

Observation 1. A (\bar{d}, \bar{t}) -CDA is a (\bar{d}, t) -CDA and a (d, \bar{t}) -CDA. A (\bar{d}, t) -CDA and a (d, \bar{t}) -CDA are both a (d, t) -CDA. When $d > 0$, a (\bar{d}, \bar{t}) -CDA and a (\bar{d}, t) -CDA are a $(\bar{d}-1, \bar{t})$ -CDA and a $(\bar{d}-1, t)$ -CDA, respectively. When $t > 0$, a (\bar{d}, \bar{t}) -CDA and a (d, \bar{t}) -CDA are a $(\bar{d}, t-1)$ -CDA and a $(d, t-1)$ -CDA, respectively.

Figure 4: The relationships among CDA variants (when marked with +, one requires that $d \leq \tau_t$; when marked with \dagger , $d \leq \tau_1$)

$$\begin{array}{ccc} (d, \bar{t})\text{-CDA} & \Rightarrow & (d, t)\text{-CDA} \\ \uparrow \Downarrow^+ & & \uparrow \Downarrow^\dagger \\ (\bar{d}, \bar{t})\text{-CDA} & \Rightarrow & (\bar{d}, t)\text{-CDA} \end{array}$$

Observation 2. Suppose that the SUT has no constraints, i.e., $\phi(\sigma) = \text{true}$ for all $\sigma \in \mathcal{R} = V_1 \times V_2 \times \dots \times V_k$ and that a (d, t) -DA A exists. Then A is a (d, t) -CDA. This also applies when d or t is replaced with \bar{d} or \bar{t} , respectively.

According to the above definition, if $|\mathcal{T}|$ is very large, then $\rho_A(\mathcal{T}) = A$ for any array A , in which case all interactions are masked by \mathcal{T} . In order to avoid such cases of no practical interest, here we introduce an upper bound, denoted τ_t , on d . We let $\tau_t \geq 0$ be the greatest integer that satisfies the condition as follows:

$$\forall \mathcal{T} \subseteq \mathcal{VI}_t \text{ such that } |\mathcal{T}| \leq \tau_t : \mathcal{R} - \rho_{\mathcal{R}}(\mathcal{T}) \neq \emptyset$$

In words, given τ_t interactions of strength t , there is always a test case in \mathcal{R} in which none of the given interactions appears. Note that $\tau_0 = 0$.

Theorem 1. For $d \leq \tau_t$, a (d, t) -CDA is equivalent to a (\bar{d}, t) -CDA.

PROOF. Trivially a $(0, t)$ -CDA is a $(\bar{0}, t)$ -CDA. Let A be a (d, t) -CDA such that $1 \leq d \leq \tau_t$ and $t > 0$. We will show that A is a $(d-1, t)$ -CDA. Let \mathcal{T} and T be a set of $d-1$ valid interactions of strength t and a t -way valid interaction, respectively. If $\mathcal{T} > T$ or $T \in \mathcal{T}$, then $\mathcal{T} \not> T \Rightarrow (T \in \mathcal{T} \Leftrightarrow \rho_A(T) \subseteq \rho_A(\mathcal{T}))$ trivially holds. The rest of the proof considers the case where $T \notin \mathcal{T}$ and $\mathcal{T} \not> T$. In this case, there is some $\sigma \in \mathcal{R}$ such that $T \subseteq \sigma$ and $\sigma \notin \rho_{\mathcal{R}}(\mathcal{T})$. Since $|\mathcal{T} \cup \{T\}| \leq \tau_t$, $\mathcal{R} - \rho_{\mathcal{R}}(\mathcal{T} \cup \{T\})$ is not empty. Let T' be any t -way interaction that appears in a test case in $\mathcal{R} - \rho_{\mathcal{R}}(\mathcal{T} \cup \{T\})$ and has exactly the same t parameters as T . Note that T and T' cannot appear in any test case simultaneously. Let $\mathcal{T}' = \mathcal{T} \cup \{T'\}$. $\mathcal{T}' \not> T$ since $T \subseteq \sigma$, $\sigma \notin \rho_{\mathcal{R}}(\mathcal{T})$, and $\sigma \notin \rho_{\mathcal{R}}(T')$. Because A is a (d, t) -CDA and $\mathcal{T}' \not> T$, $\rho_A(T) \not\subseteq \rho_A(\mathcal{T}')$. Hence $\rho_A(T) \not\subseteq \rho_A(\mathcal{T})$, which means $\mathcal{T} \not> T \Rightarrow (T \in \mathcal{T} \Leftrightarrow \rho_A(T) \subseteq \rho_A(\mathcal{T}))$. By induction, A is a (d', t) -CDA for any $0 \leq d' \leq d$ and thus is a (\bar{d}, t) -CDA. \square

A similar argument applies to (d, \bar{t}) -CDAs and (\bar{d}, \bar{t}) -CDAs.

Theorem 2. For $d = t = 0$ or $d \leq \tau_1$ and $t > 0$, a (d, \bar{t}) -CDA is equivalent to a (\bar{d}, \bar{t}) -CDA.

PROOF. Trivially $(0, \bar{t})$ -CDA is a $(\bar{0}, \bar{t})$ -CDA. Let A be a (d, \bar{t}) -CDA such that $1 \leq d \leq \tau_1$ and $t > 0$. Below we will show that A is a $(d-1, \bar{t})$ -CDA. Let $\mathcal{T} \subseteq \overline{\mathcal{VI}_t}$ such that \mathcal{T} is independent and $|\mathcal{T}| = d-1$. Let T be a valid interaction of strength at most t . If $\mathcal{T} > T$ or $T \in \mathcal{T}$, then $\mathcal{T} \not> T \Rightarrow (T \in \mathcal{T} \Leftrightarrow \rho_A(T) \subseteq \rho_A(\mathcal{T}))$ trivially holds.

Consider the remaining case where $\mathcal{T} \not> T$ and $T \notin \mathcal{T}$. In this case, there is some $\sigma \in \mathcal{R}$ such that $T \subseteq \sigma$ and $\sigma \notin \rho_{\mathcal{R}}(\mathcal{T})$.

Case $|T| > 0$. Since $|\mathcal{T} \cup \{T\}| = d \leq \tau_1$ and every interaction in $\mathcal{T} \cup \{T\}$ has strength at least one, $\mathcal{R} - \rho_{\mathcal{R}}(\mathcal{T} \cup \{T\})$ is not empty. Let T' be an interaction of strength t that appears in a test case in $\mathcal{R} - \rho_{\mathcal{R}}(\mathcal{T} \cup \{T\})$ and has a different value on at least one parameter from T . Let $\mathcal{T}' = \mathcal{T} \cup \{T'\}$. \mathcal{T}' is independent because \mathcal{T} is independent and $\hat{T} \notin T'$ for any $\hat{T} \in \mathcal{T}$. (Note that if $\hat{T} \subset T'$, \hat{T} would occur in the test case with T' .) Also $\mathcal{T}' \not> T$ since $T \subseteq \sigma$, $\sigma \notin \rho_{\mathcal{R}}(\mathcal{T})$, and $\sigma \notin \rho_{\mathcal{R}}(T')$. Because A is a (d, \bar{t}) -CDA, \mathcal{T}' is independent, and $\mathcal{T}' \not> T$, we have $\rho_A(T) \not\subseteq \rho_A(\mathcal{T}')$. Hence $\rho_A(T) \not\subseteq \rho_A(\mathcal{T})$.

Case $T = \lambda$ (i.e., $|T| = 0$). As $\lambda \notin \mathcal{T}$, every interaction in \mathcal{T} has strength at least one. Since $|\mathcal{T}| = d-1 < \tau_1$, $\mathcal{R} - \rho_{\mathcal{R}}(\mathcal{T})$ is not empty. Let T' be any t -way interaction that appears in a test case in $\mathcal{R} - \rho_{\mathcal{R}}(\mathcal{T})$. Let $\mathcal{T}' = \mathcal{T} \cup \{T'\}$. Because of the same argument as in the case $|T| > 0$, \mathcal{T}' is independent and $\mathcal{T}' \not> T$ and thus $\rho_A(T) \not\subseteq \rho_A(\mathcal{T}')$. Hence $\rho_A(T) \not\subseteq \rho_A(\mathcal{T})$.

As a result, $\mathcal{T} \not> T \Rightarrow (T \in \mathcal{T} \Leftrightarrow \rho_A(T) \subseteq \rho_A(\mathcal{T}))$. By induction, A is a (d', \bar{t}) -CDA for any $0 \leq d' \leq d$ and thus is a (\bar{d}, \bar{t}) -CDA. \square

These theorems lead to the relationships illustrated in Figure 4. Because of these results, we henceforth focus on (\bar{d}, t) -CDAs and (\bar{d}, \bar{t}) -CDAs.

Figure 5: 2-CCA, $(\bar{1}, 2)$ -CLA, and test outcomes in Case 1 and Case 2.

(a) Test outcomes of 2-CCA.					(b) Test outcomes of $(\bar{1}, 2)$ -CLA.																														
	F_1	F_2	F_3	F_4		F_1	F_2	F_3	F_4		F_1	F_2	F_3	F_4		F_1	F_2	F_3	F_4		F_1	F_2	F_3	F_4		F_1	F_2	F_3	F_4		F_1	F_2	F_3	F_4	
σ_1	0	0	0	0		Fail	Fail				σ_1	0	0	0	0		Fail	Fail				σ_1	0	0	0	0		Fail	Fail						
σ_2	0	0	0	3		Fail	Fail				σ_2	0	0	0	3		Fail	Fail				σ_2	0	0	0	3		Fail	Fail						
σ_3	0	1	1	1		Pass	Fail				σ_3	0	0	1	1		Fail	Fail				σ_3	0	0	1	1		Fail	Fail						
σ_4	0	1	2	2		Pass	Pass				σ_4	0	1	1	2		Pass	Pass				σ_4	0	1	1	2		Pass	Pass						
σ_5	1	0	0	2		Pass	Pass				σ_5	0	1	2	0		Pass	Pass				σ_5	0	1	2	0		Pass	Pass						
σ_6	1	0	0	3		Pass	Pass				σ_6	1	0	0	2		Pass	Pass				σ_6	1	0	0	2		Pass	Pass						
σ_7	1	0	2	1		Pass	Pass				σ_7	1	0	0	3		Pass	Pass				σ_7	1	0	0	3		Pass	Pass						
σ_8	1	1	1	0		Pass	Pass				σ_8	1	0	1	2		Pass	Pass				σ_8	1	0	1	2		Pass	Pass						
σ_9	2	0	0	1		Pass	Pass				σ_9	1	1	1	1		Pass	Pass				σ_9	1	1	1	1		Pass	Pass						
σ_{10}	2	0	0	3		Pass	Pass				σ_{10}	1	1	2	0		Pass	Pass				σ_{10}	1	1	2	0		Pass	Pass						
σ_{11}	2	0	1	2		Pass	Pass				σ_{11}	1	1	2	2		Pass	Pass				σ_{11}	1	1	2	2		Pass	Pass						
σ_{12}	2	1	2	0		Pass	Pass				σ_{12}	2	0	0	1		Pass	Pass				σ_{12}	2	0	0	1		Pass	Pass						

Theorem 3. A (\bar{d}, t) -CDA is a t -CCA. A (\bar{d}, \bar{t}) -CDA is a t -CCA.

PROOF. Let $T \in \mathcal{VI}_t$. Let A be a (\bar{d}, t) -CDA or a (\bar{d}, \bar{t}) -CDA. Then $T \not\succ T \Rightarrow (T \in \mathcal{T} \Leftrightarrow \rho_A(T) \subseteq \rho_A(\mathcal{T}))$ for any $\mathcal{T} \subseteq \mathcal{VI}_t$ such that $|\mathcal{T}| \leq d$. If $|\mathcal{T}| = 0$, then $\mathcal{T} = \emptyset$, in which case $\mathcal{T} \not\succ T$, $T \notin \mathcal{T}$, and $\rho_A(\mathcal{T}) = \emptyset$. Hence $\rho_A(T) \neq \emptyset$.

□

3.2. Identification of faulty interactions

Using a CDA as a test suite, faulty interactions are identified as follows. The execution of a test suite yields a test outcome which is a set of failed test cases and a set of passing test cases. Every interaction of the target strength is identified as faulty iff it appears in at least one of the failed test cases but none of the passing ones. For a (\bar{d}, t) -CDA or a (\bar{d}, \bar{t}) -CDA, the target strength is t or $0, 1, \dots, \bar{t}$, respectively. Let \mathcal{T} be the set of the faulty interactions. In other words, for a CDA A and an interaction T of the target strength, T is identified as faulty iff $\rho_A(T) \subseteq \rho_A(\mathcal{T})$. Note that we do not know \mathcal{T} in advance but know the set of failed test cases $\rho_A(\mathcal{T})$.

Suppose that a (d, t) -CDA A is used as a test suite assuming that the number of faulty interactions is at most d and that they are all t -way.

First let us consider the case where the assumptions are indeed true. For an interaction $T \in \mathcal{VI}_t$, if $\mathcal{T} \not\succ T$, then $T \in \mathcal{T} \Leftrightarrow \rho_A(T) \subseteq \rho_A(\mathcal{T})$ holds; i.e., T is faulty iff test cases in which T appears all failed. Hence T is accurately determined to be faulty or not faulty, except when $\mathcal{T} > T$, in which case T is identified as faulty.

Next consider the case where the assumption on the number of faulty interactions is false. That is, there are more than d faulty interactions. In this case, interactions T can be falsely determined to be faulty even if $\mathcal{T} \not\succ T$. However, all faulty interactions are correctly identified as faulty, because all valid t -way interactions appear in A (Theorem 3). When the assumption on the strength is false, it is not possible to identify all faulty interactions. This is because in that case some faulty interaction may not appear in any test case, unless an exhaustive test suite is used.

The situation is similar when A is a (\bar{d}, \bar{t}) -CDA. In this case, the assumptions are: $|\mathcal{T}| \leq d$ and the strength of the faulty interactions is at most t .

When these assumptions are true, $T (\in \overline{\mathcal{VI}}_t)$ is accurately determined to be faulty or not faulty unless $\mathcal{T} > T$ or $\mathcal{T} \cup \{T\}$ is not independent, since $T \in \mathcal{T} \Leftrightarrow \rho_A(T) \subseteq \rho_A(\mathcal{T})$.

But even when $\mathcal{T} \cup \{T\}$ is not independent, accurate identification is still possible if \mathcal{T} contains neither proper subsets nor proper supersets of T and $\mathcal{T} \not\succ T$. In that case, if we let $\mathcal{T}_{\min} = \{T' \in \mathcal{T} : T'' \notin T' \text{ for all } T'' \in \mathcal{T}\}$ (i.e., $\mathcal{T}_{\min} (\subseteq \mathcal{T})$ is the set of minimal interactions in \mathcal{T}), then $\mathcal{T}_{\min} \not\succ T$ and $\mathcal{T}_{\min} \cup \{T\}$ becomes independent, and thus

Figure 6: $(\bar{1}, 2)$ -CDA and test outcomes in Case 1 and Case 2.

	F_1	F_2	F_3	F_4	Case1	Case2
σ_1	0	0	0	0	Fail	Fail
σ_2	0	0	0	1	Fail	Fail
σ_3	0	0	0	3	Fail	Fail
σ_4	0	0	1	1	Fail	Fail
σ_5	0	0	2	2	Fail	Pass
σ_6	0	1	1	2	Pass	Pass
σ_7	0	1	2	0	Pass	Pass
σ_8	0	1	2	1	Pass	Fail
σ_9	1	0	0	2	Pass	Pass
σ_{10}	1	0	0	3	Pass	Pass
σ_{11}	1	0	1	0	Pass	Pass
σ_{12}	1	0	2	1	Pass	Pass
σ_{13}	1	1	1	1	Pass	Pass
σ_{14}	1	1	2	0	Pass	Pass
σ_{15}	1	1	2	2	Pass	Pass
σ_{16}	2	0	0	0	Pass	Pass
σ_{17}	2	0	0	1	Pass	Pass
σ_{18}	2	0	0	2	Pass	Pass
σ_{19}	2	0	0	3	Pass	Pass
σ_{20}	2	0	1	2	Pass	Pass
σ_{21}	2	0	2	0	Pass	Pass
σ_{22}	2	1	1	0	Pass	Pass
σ_{23}	2	1	1	1	Pass	Pass
σ_{24}	2	1	2	2	Pass	Pass

$T \in \mathcal{T}_{\min} \Leftrightarrow \rho_A(T) \in \rho_A(\mathcal{T}_{\min})$. Also $T \in \mathcal{T} \Leftrightarrow T \in \mathcal{T}_{\min}$ and $\rho_A(\mathcal{T}) = \rho_A(\mathcal{T}_{\min})$. Consequently $T \in \mathcal{T} \Leftrightarrow \rho_A(T) \subseteq \rho_A(\mathcal{T})$.

In sum, faulty interactions are all identified as faulty and a non-faulty interaction may be identified as faulty only if the set of the faulty interactions masks the non-faulty interaction or contains its proper subsets or supersets.

When $|\mathcal{T}| > d$, non-faulty interactions might be falsely identified as faulty; but all faulty interactions are correctly identified as faulty. The faulty interactions can be correctly identified because they only appear in the failed test cases.

When the assumption on the strength is not false, it is not possible to identify all faulty interactions.

3.2.1. Examples

Here using the running example, we illustrate how CCA, CLA and CDA arrays are used to detect and locate faulty interactions. We consider the cases $d = 1$ and $t = 2$. Suppose that the 2-way CCA, the $(1, 2)$ -CLA, and the $(1, 2)$ -CDA shown in Figures 2a, 2b, and Figure 3c are used as test suites. In fact the CLA and the CDA are a $(\bar{1}, 2)$ -CLA and a $(\bar{1}, 2)$ -CDA. Figure 5 and Figure 6 summarize the results of test cases when executed in the two cases below.

Case 1 The only faulty interaction is $T_\alpha = \{(F_1, 0), (F_2, 0)\}$.

Case 2 There are two faulty interactions $T_\beta = \{(F_1, 0), (F_3, 0)\}$ and $T_\gamma = \{(F_1, 0), (F_4, 1)\}$.

CCA In *Case 1*, within the test cases in the 2-CCA (Figure 2a), only the test cases σ_1 and σ_2 fail. The two-way interactions that appear only in those failed test cases are as follows (the faulty interaction is indicated by underline.)

$$\begin{array}{lll} \underline{\{(F_1, 0), (F_2, 0)\}} & \{(F_1, 0), (F_3, 0)\} & \{(F_1, 0), (F_4, 0)\} \\ \{(F_2, 0), (F_4, 0)\} & \{(F_3, 0), (F_4, 0)\} & \{(F_1, 0), (F_4, 3)\} \end{array}$$

In *Case 2*, the failed test cases are σ_1 , σ_2 , and σ_3 ; thus the candidates for faulty interactions are:

$$\begin{array}{lll} \{(F_1, 0), (F_2, 0)\} & \underline{\{(F_1, 0), (F_3, 0)\}} & \{(F_1, 0), (F_4, 0)\} \\ \{(F_2, 0), (F_4, 0)\} & \underline{\{(F_3, 0), (F_4, 0)\}} & \{(F_1, 0), (F_4, 3)\} \\ \{(F_1, 0), (F_3, 1)\} & \underline{\{(F_1, 0), (F_4, 1)\}} & \{(F_2, 1), (F_4, 1)\} \\ \hline & \{(F_3, 1), (F_4, 1)\} & \end{array}$$

For both cases it is impossible to further reduce the candidates of faulty interactions.

CLA Suppose that the $(1, 2)$ -CLA ($(\bar{1}, 2)$ -CLA) shown in Figure 2b is used. In *Case 1*, the test cases σ_1 , σ_2 and σ_3 fail and all the other test cases pass. The interactions that appear only in the failed test cases are as follows.

$$\begin{array}{lll} \underline{\{(F_1, 0), (F_2, 0)\}} & \{(F_1, 0), (F_3, 0)\} & \{(F_3, 0), (F_4, 0)\} \\ \{(F_1, 0), (F_4, 3)\} & \underline{\{(F_1, 0), (F_4, 1)\}} & \end{array}$$

The core idea of CLAs is that it allows a test outcome to be uniquely associated with a set of faulty interactions, which is mathematically represented as $T_1 = T_2 \Leftrightarrow \rho_A(T_1) = \rho_A(T_2)$. In this case, $\rho_A(\mathcal{T}) = \rho_A(\{T_\alpha\}) = \{\sigma_1, \sigma_2, \sigma_3\}$ holds only for $\mathcal{T} = \{\{(F_1, 0), (F_2, 0)\}\}$, provided that $\mathcal{T} \subseteq \mathcal{VI}_2$ and $|\mathcal{T}| \leq 1$. Thus, we correctly locate the faulty interaction.

Now consider *Case 2*. The failed test cases are the same as in *Case 1*, i.e., σ_1 , σ_2 and σ_3 . Hence the conclusion that T_α is the only faulty interaction is also the same. This incorrect result is caused by that the number of faulty interactions does not coincide with the assumption (namely $d = 1$). In general, if faulty interactions exceed the number assumed, CLAs may identify non-faulty interactions as faulty but also identify faulty interactions as non-faulty.

CDA Suppose that the $(1, 2)$ -CDA ($(\bar{1}, 2)$ -CDA) shown in Figure 3c is used to locate faulty interactions. For *Case 1*, the failed test cases are σ_1 , σ_2 , σ_3 , σ_4 and σ_5 . The interactions occurring only in the failed test cases are all identified as faulty. In this case these interactions are:

$$\underline{\{(F_1, 0), (F_2, 0)\}} \quad \{(F_1, 0), (F_3, 0)\} \quad \{(F_1, 0), (F_4, 3)\}$$

T_α is correctly identified as faulty, whereas $\{(F_1, 0), (F_3, 0)\}$ and $\{(F_1, 0), (F_4, 3)\}$ are incorrectly identified as faulty. Since $\{T_\alpha\}$ masks $\{(F_1, 0), (F_3, 0)\}$ and $\{(F_1, 0), (F_4, 3)\}$ ($\{T_\alpha\} > \{(F_1, 0), (F_3, 0)\}$ and $\{T_\alpha\} > \{(F_1, 0), (F_4, 3)\}$), it is inherently impossible to determine that $\{(F_1, 0), (F_3, 0)\}$ and $\{(F_1, 0), (F_4, 3)\}$ are not faulty when T_α is faulty. However, it should be noted that if we relied on the assumption that the number of faulty interactions is $d = 1$, just as in the case of the CLA above, we could correctly identify only T_α as faulty. In fact, we will show later that any CDA is a CLA.

For *Case 2*, the failed test cases are σ_1 , σ_2 , σ_3 , σ_4 , and σ_8 . The interactions that are identified as faulty are:

$$\underline{\{(F_1, 0), (F_3, 0)\}} \quad \underline{\{(F_1, 0), (F_4, 1)\}} \quad \{(F_1, 0), (F_4, 3)\}$$

Although the last interaction is in fact not faulty, all the faulty ones are correctly identified. In general, when using a CDA, non-faulty interactions are never wrongly identified as faulty even if the number of faulty interactions exceeds the assumed number d .

3.3. Properties of CDAs

In the rest of the section we provide some theorems on the properties of CDAs.

Theorem 4. \mathcal{R} , the exhaustive test suite, is a (d, t) -, (d, \bar{t}) -, (\bar{d}, t) - and (\bar{d}, \bar{t}) -CDA for any d and t .

PROOF. Let T be a valid interaction and \mathcal{T} be a set of valid interactions. Below we will show $\mathcal{T} \not\succ T \Rightarrow (T \in \mathcal{T} \Leftrightarrow \rho_{\mathcal{R}}(T) \subseteq \rho_{\mathcal{R}}(\mathcal{T}))$. If $\mathcal{T} \not\succ T$ and $T \notin \mathcal{T}$, then there is some $\sigma \in \mathcal{R}$ such that $T \subseteq \sigma$ and $\forall T' \in \mathcal{T} : T' \not\subseteq \sigma$, in which case $\sigma \in \rho_{\mathcal{R}}(T) - \rho_{\mathcal{R}}(\mathcal{T})$. That is, $\mathcal{T} \not\succ T \Rightarrow (T \notin \mathcal{T} \Rightarrow \rho_{\mathcal{R}}(T) \not\subseteq \rho_{\mathcal{R}}(\mathcal{T}))$. In addition $T \in \mathcal{T} \Rightarrow \rho_{\mathcal{R}}(T) \subseteq \rho_{\mathcal{R}}(\mathcal{T})$ trivially holds. As a result, the theorem follows. \square

Theorem 5. A (d, t) -CDA is also a (d, t) -CLA; a (\bar{d}, t) -CDA is also a (\bar{d}, t) -CLA; A (d, \bar{t}) -CDA is also a (d, \bar{t}) -CLA and a (\bar{d}, \bar{t}) -CDA is also a (\bar{d}, \bar{t}) -CLA.

PROOF. Let A be a (d, t) -CDA. Let \mathcal{T}_1 and \mathcal{T}_2 be different sets of t -way interactions of size d and mutually distinguishable. $\rho_R(\mathcal{T}_1) \neq \rho_R(\mathcal{T}_2)$ by the definition of distinguishability. Without loss of generality, we assume that $\mathcal{T}_1 \not\subseteq \mathcal{T}_2$. Denote the test case that exists in $\rho_R(\mathcal{T}_1)$ but not in $\rho_R(\mathcal{T}_2)$ as σ_e . There exists at least one valid interaction T in \mathcal{T}_1 that is covered by σ_e . \mathcal{T}_2 does not mask T and $T \notin \mathcal{T}_2$ because $\sigma_e \in \rho_R(T) \wedge \sigma_e \notin \rho_R(\mathcal{T}_2)$. Since A is a (d, t) -CDA, $\mathcal{T}_2 \not\nearrow T$, and $T \notin \mathcal{T}_2$, we have $\rho_A(T) \not\subseteq \rho_A(\mathcal{T}_2)$. Hence there exists a row σ'_e in A such that σ'_e covers T but does not cover any interactions in \mathcal{T}_2 , that is, $\sigma'_e \in \rho_A(\mathcal{T}_1) \wedge \sigma'_e \notin \rho_A(\mathcal{T}_2)$ holds. Thus $\rho_A(\mathcal{T}_1) \neq \rho_A(\mathcal{T}_2)$ holds; hence A is a (d, t) -CLA. The same argument holds when $|\mathcal{T}_1|$ and $|\mathcal{T}_2|$ are not exactly d but at most d . Therefore it follows that a (\bar{d}, t) -CDA is a (\bar{d}, t) -CLA.

Next let A be a (d, \bar{t}) -CDA or a (\bar{d}, \bar{t}) -CDA. Let \mathcal{T}_1 and \mathcal{T}_2 be different sets consisting of exactly d interactions of strength at most t or at most d interactions of strength at most t , respectively. Then the same argument for the case of (d, t) -CDAs and (\bar{d}, t) -CDAs holds. As a result, the theorem follows. \square

Theorem 6. A $(t + d)$ -CCA is a (\bar{d}, \bar{t}) -CDA.

PROOF. Suppose that A is a $(t + d)$ -CCA. The theorem holds if $\rho_A(T) \not\subseteq \rho_A(\mathcal{T})$ for any $T \in \overline{\mathcal{VI}_t}$ and $\mathcal{T} \subseteq \overline{\mathcal{VI}_t}$ such that $0 \leq |\mathcal{T}| \leq d$, $T \notin \mathcal{T}$, $\mathcal{T} \not\nearrow T$, and $\{T\} \cup \mathcal{T}$ is independent. We show this by constructing a valid interaction \hat{T} of strength $\leq d + t$ that covers T but cannot appear with any interaction in \mathcal{T} in the same row. If such \hat{T} exists, some row of A contains it because A is a $(t + d)$ -CCA. This row is in $\rho_A(T)$ but not in $\rho_A(\mathcal{T})$; thus $\rho_A(T) \not\subseteq \rho_A(\mathcal{T})$.

Since $\mathcal{T} \not\nearrow T$, there must be a valid test case σ that covers T but does not cover any $T' \in \mathcal{T}$. Let $\sigma = \langle s_1, s_2, \dots, s_k \rangle$. We regard σ as k -way interaction $\{(F_1, s_1), (F_2, s_2), \dots, (F_k, s_k)\}$. \hat{T} is constructed by starting from $\hat{T} = T$ and gradually expanding it by applying the following process for all $T' \in \mathcal{T}$: Select any $(F_i, v) \in T'$ such that $s_i \neq v$. This can be done because T' is not covered by σ (and thus $\lambda \notin \mathcal{T}$). Add (F_i, s_i) to \hat{T} . Finally \hat{T} becomes the desired interaction. \square

4. Generation Algorithms

In this section, we present two algorithms for generating CDAs: the satisfiability-based algorithm and the two-step heuristic algorithm. In this section we limit ourselves to (d, t) -CDAs because (d, t) -CDAs are (\bar{d}, t) -CDAs except in extreme cases (Theorem 2). Also it is straightforward to adjust the algorithms to (d, \bar{t}) -CDAs and (\bar{d}, \bar{t}) -CDAs.

4.1. The satisfiability-based algorithm

The first algorithm leverages a satisfiability solver. We reduce the problem of generating a CDA of a given size to the satisfiability problem of a logical (i.e., Boolean-valued) expression. A logical expression is satisfiable iff it evaluates to true for some *valuation*, i.e., assignment of values to the variables. The algorithm first estimates the upper bound on the minimum size of a CDA and uses it as the initial size of a CDA. Then it creates a logical expression that is satisfiable iff a CDA of the initial size exists. The logical expression is in turn evaluated by a satisfiability solver. We design the logical expression so that the valuation that satisfies it directly represents a CDA. Satisfiability solvers can produce such a satisfying valuation when the expression is satisfiable; hence a CDA can be obtained from the output of the solver. Repeating the process while decreasing the CDA size, the algorithm can obtain the smallest CDA.

4.1.1. The logic expression

To represent an array with a collection of variables, we adopt the *naïve matrix model* which is used by Hnich et al. [9] in their study to find CAs. In this model, an $N \times k$ array is represented as an $N \times k$ matrix of integer variables as follows.

$$A = \begin{pmatrix} p_1^1 & \cdots & p_k^1 \\ \vdots & \ddots & \vdots \\ p_1^N & \cdots & p_k^N \end{pmatrix}$$

The variable p_i^n represents the value on the parameter F_i in the n -th test case. The domain of p_i^n is $S_i = \{0, 1, \dots, |S_i| - 1\}$.

In order for the array A to become a (d, t) -CDA, we impose the following conditions on A using logical expressions.

1. The rows of A represent valid test cases.
2. $\forall \mathcal{T} \subseteq \mathcal{VI}_t$ such that $|\mathcal{T}| = d, \forall T \in \mathcal{VI}_t : \mathcal{T} \not\nearrow T \Rightarrow (T \in \mathcal{T} \Leftrightarrow \rho_A(T) \subseteq \rho_A(\mathcal{T}))$

Below we present logical expressions that represent the above two conditions. By conjuncting all the expressions, we obtain a single logical expression to be checked for satisfiability.

Condition 1 In A , the n -th row is expressed as a tuple of k variables $(p_1^n, p_2^n, \dots, p_k^n)$. As defined in Section 2, a test case is valid iff it satisfies the constraints and the constraints are represented by ϕ , a Boolean-valued formula over parameters F_1, \dots, F_k . We let $\phi|_{p_1^n, p_2^n, \dots, p_k^n}$ denote ϕ with each F_i being replaced with p_i^n . Then, the following expression enforces A to only contain valid test cases.

$$\text{Valid} := \bigwedge_{n=1}^N \phi|_{p_1^n, p_2^n, \dots, p_k^n}$$

Condition 2 It is important to note that $\mathcal{T} \not\succ T \Rightarrow (T \in \mathcal{T} \Leftrightarrow \rho_A(T) \subseteq \rho_A(\mathcal{T}))$ is equivalent to:

$$(\mathcal{T} \not\succ T \wedge T \notin \mathcal{T}) \Rightarrow \rho_A(T) \not\subseteq \rho_A(\mathcal{T})$$

because $T \in \mathcal{T} \Rightarrow \rho_A(T) \subseteq \rho_A(\mathcal{T})$ trivially holds. Hence we can focus on the case where $\mathcal{T} \not\succ T$ and $T \notin \mathcal{T}$. The right part of this formula, that is, $\rho_A(T) \not\subseteq \rho_A(\mathcal{T})$ holds iff there is a row in A that covers T but none of the interactions in \mathcal{T} . This condition is represented by a logical expression as follows:

$$\text{Locating}(\mathcal{T}, T) := \bigvee_{n=1}^N \left(\bigwedge_{j=1}^t (p_{x_j}^n = v_{x_j}) \wedge \neg \left(\bigvee_{L=1}^d \bigwedge_{l=1}^t (p_{y_{L_l}}^n = v_{y_{L_l}}) \right) \right)$$

where $\mathcal{T} = \{(F_{y_{1_1}}, v_{y_{1_1}}), \dots, (F_{y_{1_t}}, v_{y_{1_t}})\}, \dots, \{(F_{y_{d_1}}, v_{y_{d_1}}), \dots, (F_{y_{d_t}}, v_{y_{d_t}})\}\}$ and $T = \{(F_{x_1}, v_{x_1}), \dots, (F_{x_t}, v_{x_t})\}$. For given \mathcal{T} and T , $\rho_A(T) \not\subseteq \rho_A(\mathcal{T})$ holds iff $\text{Locating}(\mathcal{T}, T)$ is satisfiable.

Let us define \mathcal{U} as follows:

$$\mathcal{U} := \{(\mathcal{T}, T) \mid \mathcal{T} \subseteq \mathcal{VI}_t, |\mathcal{T}| = d, T \in \mathcal{VI}_t, \mathcal{T} \not\succ T, T \notin \mathcal{T}\}$$

By ANDing $\text{Locating}(\mathcal{T}, T)$ for all $(\mathcal{T}, T) \in \mathcal{U}$, we obtain an expression that represents the second condition.

The whole expression The whole expression that will be checked for satisfiability is obtained by conjuncting the expressions defined above as follows:

$$\text{existCDA} := \text{Valid} \wedge \bigwedge_{(\mathcal{T}, T) \in \mathcal{U}} \text{Locating}(\mathcal{T}, T)$$

By checking the satisfiability of this expression, whether a (d, t) -CDA of size N exists or not can be determined. If it is satisfiable, then a CDA of size N exists. In this case, the satisfying valuation for the $N \times k$ variables p_i^n represents all the entries of one such CDA. On the other hand, if the expression is unsatisfiable, then it can be concluded that no (d, t) -CDA of size N exists.

The satisfiability of the above expression can be checked using Constraint Satisfaction Problem (CSP) solvers, Satisfiability Modulo Theories (SMT) solvers, or Boolean Satisfiability (SAT) solvers with a Boolean encoding of integers.

4.1.2. Computing \mathcal{U}

In order to construct the above logical expression existCDA , we need to obtain \mathcal{U} (see the subscript of the \bigwedge in the expression). Computing \mathcal{U} requires \mathcal{VI}_t . We will show how to compute \mathcal{VI}_t later. Here we describe how one can compute \mathcal{U} when \mathcal{VI}_t is available.

Now consider enumerating all \mathcal{T} - T pairs such that $T \in \mathcal{VI}_t$, $\mathcal{T} \subseteq \mathcal{VI}_t$, $|\mathcal{T}| = d$, $T \notin \mathcal{T}$, and $\mathcal{T} \not\succ T$. The problem here is how to decide whether or not $\mathcal{T} (\subseteq \mathcal{VI}_t)$ masks $T (\in \mathcal{VI}_t)$ when \mathcal{T} and $T \notin \mathcal{T}$ are given. This too is possible by making use of satisfiability solving. We let integer variables p_1, p_2, \dots, p_k to symbolically represent a test case σ ; that is,

$$\sigma = (p_1, p_2, \dots, p_k)$$

Algorithm 1: The satisfiability-based algorithm

```

Input: SUT  $\mathcal{M} = \langle \mathcal{F}, \mathcal{S}, \phi \rangle$ ; integers  $d, t$ 
Output:  $(d,t)$ -CDA  $A$ 
// construct a  $(d + t)$ -CCA for the input SUT
1  $S \leftarrow generateCCA(\mathcal{M}, d + t)$ 
// get all valid  $t$ -way interactions from the  $(d + t)$ -CCA
2  $\mathcal{VI}_t \leftarrow getAllInteractions(S, t)$ 
// get all non-masking pairs  $\mathcal{U}$  of interaction sets and interactions
3  $\mathcal{U} \leftarrow getU(\mathcal{VI}_t, d, t)$ 
// get the initial size for the CDA to be generated
4  $N \leftarrow$  The size of  $S - 1$ 
5  $nextA \leftarrow S$ 
6 do
    // reserve the current test suite instance
7    $A \leftarrow nextA$ 
    // SAT checking; the solver returns an instance if satisfiable; an emptyset, otherwise
8    $nextA \leftarrow generateCDA(\mathcal{M}, d, t, N, \mathcal{U})$ 
    // decrease the size by one
9    $N \leftarrow N - 1$ 
10 while  $nextA \neq \perp$ 
11 return  $A$ 

```

The domain of p_i is $\{0, 1, \dots, |S_i| - 1\}$. Note that S_i is the domain of parameter F_i .

By the definition of masking, given such a \mathcal{T} - T pair, \mathcal{T} does not mask T iff the following condition holds:

$$\exists \sigma \in \mathcal{R} : T \subseteq \sigma \wedge \neg(\exists T' \in \mathcal{T} : T' \subseteq \sigma)$$

In words, the condition holds if there is a valid test case that covers the interaction T but does not cover any interactions in the interaction set \mathcal{T} . Hence, given \mathcal{T} and T , $\mathcal{T} \not\succ T$ holds iff the following expression evaluates to true.

$$checkUnMasking(\mathcal{T}, T) := \bigwedge_{j=1}^t (p_{x_j} = v_{x_j}) \wedge \neg \left(\bigvee_{L=1}^d \bigwedge_{l=1}^t (p_{y_{L_l}} = v_{y_{L_l}}) \right) \wedge \phi|_{p_1, p_2, \dots, p_k}$$

where $\mathcal{T} = \{(F_{y_{1_1}}, v_{y_{1_1}}), \dots, (F_{y_{1_t}}, v_{y_{1_t}})\}, \dots, \{(F_{y_{d_1}}, v_{y_{d_1}}), \dots, (F_{y_{d_t}}, v_{y_{d_t}})\}\}$ and $T = \{(F_{x_1}, v_{x_1}), \dots, (F_{x_t}, v_{x_t})\}$.

\mathcal{U} is obtained by, for every \mathcal{T} - T pair, checking the satisfiability of $checkUnMasking(\mathcal{T}, T)$ and keeping the pair in \mathcal{U} if the expression is satisfiable.

4.1.3. The algorithm

The CDA generation algorithm that uses satisfiability solving is shown as Algorithm 1. The algorithm repeatedly solves the problem of finding a (d, t) -CDA while varying the array size N . The array size N starts with a value large enough to ensure the existence of a CDA and is gradually decreased until no existence of a CDA of size N is proved. To obtain the initial value of N , the algorithm creates a $(d + t)$ -CCA using an off-the-shelf algorithm (line 1), where the CCA generation algorithm is represented as function $generateCCA(\mathcal{M}, x)$ which returns an x -CCA. Our algorithm uses the size of the CCA minus one as the initial N , as any $(d + t)$ -CCA is a (d, t) -CDA. The $(d + t)$ -CCA is also used for computing \mathcal{VI}_t , since all valid t -way interactions appear in the CCA: The algorithm enumerates all t -way interactions occurring in the array, thus obtaining \mathcal{VI}_t .

In the algorithm, $generateCDA(\mathcal{M}, d, t, N, \mathcal{U})$ in line 8 represents a function that produces a (d, t) -CDA of size N by checking the satisfiability of the expression $existCDA$. If the expression is satisfiable, then the SMT solver returns the satisfying valuation, in which case a (d, t) -CDA of size N is obtained, since the valuation represents the (d, t) -CDA. The size N is decreased by one and the same process is repeated. If the result of satisfiability check is UNSAT

Algorithm 2: The two-step heuristic generation algorithm

```

Input: SUT  $\mathcal{M} = \langle \mathcal{F}, \mathcal{S}, \phi \rangle$ ; integers  $d, t$ 
Output:  $(d, t)$ -CDA  $A$ 
// construct a  $(d + t)$ -CCA for the input SUT
1  $S \leftarrow \text{generateCCA}(\mathcal{M}, d + t)$ 
// get all t-way interactions from the  $(d + t)$ -CCA
2  $\mathcal{VI}_t \leftarrow \text{getAllInteractions}(S, t)$ 
//  $\text{Rows}[T] = \rho_S(T)$  for  $T \in \mathcal{VI}_t$ 
3  $\text{Rows}[] \leftarrow \text{mapInteractionToRows}(\mathcal{VI}_t, S)$ 
//  $\text{DiffRows}[\mathcal{T}][T] = \rho_S(T) - \rho_S(\mathcal{T})$  for  $\mathcal{T} \subseteq \mathcal{VI}_t, |\mathcal{T}| = d$ 
4  $\text{DiffRows}[][] \leftarrow \text{getDiffRows}(\mathcal{VI}_t, S, d)$ 
5  $A \leftarrow S$ 
6 while  $A \neq \emptyset$  do
7    $\sigma \leftarrow \text{getRandomTestcase}(S)$ 
8    $S \leftarrow S - \{\sigma\}$ 
9    $A \leftarrow A - \{\sigma\}$ 
10   $\text{DiffRows}'[][] \leftarrow \text{update}(\text{DiffRows}[], \sigma)$ 
11  if  $\exists \mathcal{T}, T : \text{DiffRows}[\mathcal{T}][T] \neq \emptyset$  and  $\text{DiffRows}'[\mathcal{T}][T] = \emptyset$  then
    // the test case  $\sigma$  is unremovable
12     $S \leftarrow S + \{\sigma\}$ 
13  else
    // the test case  $\sigma$  is removable
14     $\text{DiffRows}[][] \leftarrow \text{DiffRows}'[][]$ 
15 return  $A$ 

```

(unsatisfiable), no CDA of size N exists (denoted as \perp in the algorithm). Then the algorithm returns the CDA of size $N + 1$ and stops its execution.

One might think that binary search could work better to vary N than the linear search adopted by the algorithm. In fact, this is not the case because showing unsatisfiability, that is, the nonexistence of a CDA, usually takes much longer time than showing satisfiability, that is, the existence of a CDA. The linear search delays solving an unsatisfiable expression until all possible sizes are checked, avoiding getting trapped in a long computation required for the unsatisfiable problem instance.

The size of the expression existCDA increases polynomially in k when $t, d, |S_i|$, and N are fixed. The expression can be expressed as a Boolean formula with a polynomial size increase, as $|S_i|$ is fixed. The Boolean satisfiability problem (SAT) is NP-complete in general and there is no reason that SAT can be solved in polynomial-time for this particular case. Hence the time complexity of the algorithm is likely to be exponential.

4.2. The two-step heuristic algorithm

In this subsection, we propose a two-step heuristic algorithm for the generation of (d, t) -CDAs which aims to generate (d, t) -CDAs that are not optimal but fairly small in reasonable time.

Theorem 6 shows that a $(d + t)$ -CCA is already a (d, t) -CDA. Based on the theorem, we devise a two-step heuristic algorithm (Algorithm 2). The algorithm generates a $(d + t)$ -CCA first. Then it repeatedly chooses a test case in it at random and checks whether it is removable. Here we say that a test case is removable from an array if a new array with the test case being removed would still be a (d, t) -CDA. If the test case is removable, then it is removed from the current array. Otherwise, a new test case is chosen and the check is performed again. This process is repeated until no test case is removable anymore.

The algorithm works in detail as follows. In line 1 the algorithm generates a $(d + t)$ -CCA S . At this point, S is already a (d, t) -CDA but contains many redundant test cases. Then the algorithm collects all valid t -way interactions and maps each interaction T to its covering test cases $\rho_S(T)$ in S (line 2). The map obtained here, denoted by $\text{Rows}[]$, is used to compute another map, $\text{DiffRows}[][]$, that associates each pair of an interaction set \mathcal{T} and a valid interaction

Table 2
Benchmark Information

ID	SUT	$ \mathcal{F} $	$ \phi $	$ \mathcal{VI}_2 $	$ \mathcal{I}_2 \setminus \mathcal{VI}_2 $	$ \mathcal{T} > T $
1	car	9	15	102	42	1,487
2	graph_product_line	20	45	499	261	37,212
3	real_fm	14	23	275	89	5,368
4	aircraft_fm	13	19	239	73	2,647
5	connector_fm	20	37	537	223	49,038
6	movies_app_fm	13	23	211	101	4,968
7	stack_fm	17	28	465	79	6,399
8	banking1	5	112	102	0	0
9	banking2	15	3	473	3	208
10	comm_protocol	11	128	285	35	2,177
11	concurrency	5	7	36	4	130
12	healthcare1	10	21	361	8	512
13	healthcare2	12	25	466	1	124
14	healthcare3	29	31	3,092	59	8,700
15	healthcare4	35	22	5,707	38	3,359
16	insurance	14	0	4,573	0	0
17	network_mgmt	9	20	1,228	20	189
18	processor_comm1	15	13	1,058	13	1,510
19	processor_comm2	25	125	2,525	854	35,156
20	services	13	388	1,819	16	1,088
21	storage1	4	95	53	18	112
22	storage2	5	0	126	0	0
23	storage3	15	48	1,020	120	3,400
24	storage4	20	24	3,491	24	0
25	storage5	23	151	5,342	246	10,095
26	system_mgmt	10	17	310	14	825
27	telecom	10	21	440	11	151

T with $\rho_S(T) = \rho_S(\mathcal{T})$. Note that $\rho_S(T) - \rho_S(\mathcal{T}) = \emptyset$ iff $\rho_S(T) \subseteq \rho_S(\mathcal{T})$. Since S is a CDA, $\text{DiffRows}[\mathcal{T}][T] = \emptyset$ if $\mathcal{T} > T$; $\text{DiffRows}[\mathcal{T}][T] \neq \emptyset$ otherwise.

Then the algorithm repeatedly chooses a test case at random and checks whether it is removable or not. To perform the check, the algorithm constructs a new interaction-to-row map $\text{DiffRows}[\cdot][\cdot]$ that would hold after the test case was removed (line 9). This can be done by simply removing σ from all $\text{DiffRows}[\mathcal{T}][T]$. Subsequently, the algorithm compares the two maps (line 10). If $\text{DiffRows}[\mathcal{T}][T] \neq \emptyset$ but $\text{DiffRows}'[\mathcal{T}][T] = \emptyset$, then $\rho_S(T) \subseteq \rho_S(\mathcal{T})$ and thus S is no longer a CDA. In this case, the algorithm reserves the test case (line 12). Otherwise, it deletes the test case and accordingly updates $\text{DiffRows}[\mathcal{T}][T]$ (line 14). When all test cases in the CCA are checked, the algorithm will terminate, yielding the resulting S .

Let $s = \max_{1 \leq i \leq k} |S_i|$. Outside the while loop, line 4 has the highest time complexity. It is $O((s^t k^t)^d s^t k^t n)$, since $|\mathcal{VI}_t| \leq s^t k^t$, $|\rho_S(\cdot)| \leq n$. Inside the while loop, line 10 and line 11 has the highest complexity $O((s^t k^t)^d s^t k^t n)$ for the same reason. And let n be the size of the initial CCA. As a result, the algorithm's time complexity is $O((s^t k^t)^d s^t k^t n^2)$. When s, t , and d are fixed, the complexity is polynomial in k and n .

5. Experiments

In this section we show the results of experiments to evaluate the two proposed algorithms presented in the previous section. We focus on generation of $(1, 2)$ -CDAs ($d = 1, t = 2$) for the following reasons. First, by nature of CDAs no interactions can be erroneously identified as non-faulty even when more than d interactions are faulty; thus it is natural to set a small value to d in practice. Second, the most common form of CIT targets two-way interactions (this form of CIT is called *pair-wise testing*.)

Table 3

Experimental results. Numbers with * indicate that the algorithm did not terminate within the time limit, in which case the CDAs obtained are not necessarily minimum.

ID	Time (second)				Size			
	SMT		Two-step		SMT		Two-step	
	avg.	max.	min.	avg.	avg.	max.	min.	avg.
1	2.58	0.12	0.07	0.08	12	12	12	12
2	—	0.18	0.13	0.15	—	26	26	26
3	806.21*	0.16	0.11	0.13	28*	31	30	30.40
4	268.75*	0.10	0.09	0.10	14*	18	18	18
5	88.06*	0.17	0.12	0.14	18*	18	18	18
6	6.41	0.12	0.09	0.10	9	10	10	10
7	357.50*	0.18	0.13	0.15	34*	33	33	33
8	1557.89*	0.15	0.10	0.13	25*	37	36	36.40
9	—	0.12	0.09	0.11	—	42	42	42
10	—	0.21	0.15	0.17	—	50	47	48.60
11	0.364	0.10	0.07	0.08	8	8	8	8
12	—	0.17	0.12	0.14	—	95	92	94
13	—	0.20	0.16	0.18	—	60	57	58.20
14	—	3.14	2.79	2.95	—	179	173	176
15	—	19.87	18.73	19.23	—	234	220	227.60
16	—	145.14	130.48	134.31	—	1997	1959	1971.40
17	—	1.56	1.48	1.52	—	405	394	399.40
18	—	0.67	0.59	0.61	—	114	111	112
19	—	3.16	2.94	3.03	—	122	118	120.20
20	—	4.82	4.76	4.79	—	430	413	422.40
21	2.53*	0.12	0.10	0.10	25*	25	25	25
22	—	0.08	0.07	0.07	—	51	45	47.60
23	—	0.64	0.58	0.61	—	189	185	187.60
24	—	16.28	15.61	15.88	—	517	500	506.20
25	—	130.25	120.02	124.86	—	860	843	851.60
26	—	0.12	0.09	0.11	—	53	49	51.80
27	—	0.19	0.14	0.16	—	102	98	99.8

5.1. Experiment settings

We wrote C++ programs that implement the two algorithms. Our implementation [10] of the IPOG algorithm [6] was used as a CCA generator for both algorithms, while the Z3 solver (version 4.8.1) was used in the satisfiability-based algorithms. We performed experiments with a total of 27 benchmark instances, numbered from 1 to 27. Benchmarks No. 1 to 7 are taken from [11] which are provided as part of the CitLab tool. Benchmarks No. 8 to 27 can be found in [12]. The detailed information of these benchmark instances is shown in Table 2. In Table 2, the columns labeled with $|\mathcal{F}|$ and $|\phi|$ show the number of parameters and the number of constraints (ϕ is the conjunction of the constraints). Columns $|\mathcal{VI}_2|$ and $|\mathcal{I}_2 \setminus \mathcal{VI}_2|$ show respectively the number of valid interactions and the number of invalid interactions. The last column labeled $|\mathcal{T} > T|$ shows the number of pairs of an interaction set \mathcal{T} and an interaction T such that \mathcal{T} masks T . For instance, the first line in the table shows that the benchmark *car* has 9 parameters with 15 constraints. In the test space there are 102 valid interactions and 42 invalid interactions. Among the valid interactions, there are 1,487 pairs of an interaction set and an interaction such that the interaction set masks the interaction. All experiments were conducted on a machine with Intel Core i7-8700 CPU, 64 GB memory and Ubuntu 18.04 LTS OS. For each benchmark instance, the two generation algorithms were executed five times. The timeout period for each run was set to 1800 seconds.

5.2. Experimental results

The results of the experiments are summarized in Table 3. The leftmost column shows the benchmark IDs. The rest of the table is divided into two parts representing the results of generation time and the results of sizes of the generated CDAs. Both parts have two sections describing the experiment results of the two proposed algorithms respectively.

For each problem instance, the average value is reported for the satisfiability-based algorithm as it is deterministic, while the maximum, minimum, and average values are reported for the two-step heuristic generation algorithm.

The numbers with asterisk (*) in the satisfiability-based algorithm's columns show that the generation did not terminate within the time limit. Because the algorithm repeatedly generates CDAs with sizes varying until the minimum one is found, CDAs that are not optimal are constructed during the course of execution. The values with asterisk (*) correspond to the smallest (not necessarily optimal) CDAs that were obtained within the time limit. For example, for benchmark No. 3, the algorithm took 806.21 seconds to generate a CDA of size 28. However, when it was trying to generate a CDA of size 27, the run of the algorithm exceeded the 1800 second time limit. There are also some benchmark instances that the algorithm did not find even one CDA within the time limit. We use the symbol “–” to indicate such a case. To compare the average consumed time of the two algorithms, the better results (i.e., the shorter time) are denoted in bold font. The smaller average sizes of generated CDAs are also denoted in bold font.

The satisfiability-based algorithm completed the generation process for three instances, namely, No. 1, 6, and 11. The CDAs obtained for these instances are all optimal. The algorithm was able to find small CDAs for some remaining instances (though it timed out), whereas it failed to find even a single CDA for others. In contrast, the two-step algorithms successfully generated CDAs for all benchmark instances. In addition, the execution time of the satisfiability-based algorithm was always much longer than the other algorithm, sometimes three orders of magnitude longer. There are two main reasons why the satisfiability-based algorithm is so slow. One reason is that the algorithm generates multiple CDAs in a single run. As stated in Section 4, it generates (d, t) -CDAs with sizes varying from the size of a $(d + t)$ -CCA. The CCA's size simply serves as the upper bound on the minimum CDA size: As it is not tight bound in general, to obtain an optimal (d, t) -CDA, the satisfiability solver is executed multiple times. The other reason, which is more obvious, is that satisfiability check may be time-consuming. The time required for the check becomes very long especially when the algorithm tries to find a CDA of minimum size minus 1, in which case the answer of the check is UNSAT (unsatisfiable). In the field of satisfiability, it is well known that UNSAT instances are usually more difficult than SAT instances.

The satisfiability-based algorithm is deterministic. As stated above, the CCA size affects the algorithm's execution time and, if timeout occurs, the resulting CDA size. In contrast, the two-step heuristic algorithm is inherently nondeterministic: it generates different CDAs for different runs. The algorithm decreases the array size by repeatedly removing from the current array a test case selected at random (line 7, Algorithm 2). A test case can be removed only if the array remains to be a CDA after its removal; thus which test case is removed depends strongly on earlier selections. Hence, different orders in which test cases are deleted lead to different CDAs.

Another observation is that the two-step heuristic algorithm generated smaller CDAs than the satisfiability-based algorithm for No. 7. For the case, the satisfiability-based algorithm ran out of time before searching for minimum or near minimum CDAs. In view of these, we conclude that the two-step heuristic algorithm has balanced capabilities with respect to running time and CDA sizes it generates.

6. Threats to Validity

The experimental results about the two CDA generation algorithms showed that the scalability of the satisfiability-based algorithm is substantially limited, especially when comparing to the heuristic algorithm. This conclusion heavily relies on the performance of the satisfiability solver used in the implementation. Although Z3, which we adopted in our implementation, is one of the best known and fastest SMT solvers, solvers of SMT or similar problems, such as SAT, have seen constant progress. Hence the difference between the two algorithms might narrow in near future.

The problem instances used in the experiments are well-known and have been used in many other studies; However, they do not necessarily capture the characteristics of all real-world problems. Although we believe that the algorithms' qualitative properties observed in the experiments are likely to hold in general, there can be new problems for which they do not hold.

7. Related Work

CIT has been widely used for many years. In the practice of CIT, constraint handling has always been a vital issue. Surveys about constraint handling for CIT include [13, 14, 15].

DAs, as well as LAs, were first introduced by Colbourn and McClary in [3]. They analyzed the mathematical properties of these arrays. As [3], most of the studies on DAs and LAs focus on their mathematical aspects [16,

17, 18, 19, 20, 21]. The application to screening experiments for TCP throughput in a mobile wireless network was reported in [22, 23]. Other types of arrays that are intended for fault location include *Error Locating Arrays* [24] and *Consecutive Detecting Arrays* [25].

The concept of CLAs was first introduced in [26]. Later a computational construction algorithm was proposed in [27]. In [7] the results of applying CLAs to fault localization for real-world programs were reported.

The present paper extends our earlier works: [28] and [29]. In [28] we introduced (d, t) -CDAs for the first time, together with a construction algorithm using an SMT-solver. The present paper introduces the other variants of CDAs, clarifies their relations, and refines the algorithm. The two-step heuristic algorithm was first proposed in [29]. In the current paper, we improved the implementation of the algorithm and conducted a new set of experiments to compare the two different algorithms with the new implementations.

There are many other approaches to faulty interaction localization without using CDAs or other related arrays. One of such approaches is the use of adaptive testing [30, 31, 32, 33, 34, 35, 36, 37]. In adaptive testing, when a failure is encountered, new test cases are adaptively generated and executed to narrow down possible causes. On the other hand, testing using CDAs is nonadaptive in the sense that test outcomes do not alter future test plans. A clear benefit of using nonadaptive testing is the execution of test suites, which is often the most time-consuming part of the whole testing process, can be parallelized.

CDAs and other arrays of similar kinds are intended to provide sufficient test outcomes to uniquely identify faulty interactions. On the other hand, some studies attempt to infer faulty interactions from insufficient information with, for example, machine learning. The studies in this line include [38, 39, 40].

For other approaches to identification of faulty interactions, readers are referred to a recent survey [41].

8. Conclusion

In this paper, we introduced the notion of Constrained Detecting Arrays (CDAs), which incorporates constraints among test parameters into Detecting Arrays (DAs). CDAs generalize DAs so that localization of faulty interactions can be performed for systems with constraints. We proved several properties of CDAs as well as those that relate CDAs with other array structures, such as Constrained Covering Arrays (CCAs) and Constrained Locating Arrays (CLAs). We then proposed two generation algorithms. The first algorithm generates optimal CDAs using an off-the-shelf satisfiability solver. The second algorithm is heuristic and generates near-optimal CDAs in a reasonable time. The experimental results of both algorithms indicated that the heuristic algorithm can scale to problems of practical sizes.

There are several possible directions for future work. One direction is to apply CDAs to the testing of real-world programs to identify faulty interactions. The development of new algorithms for CDA construction also deserves further study. We believe that both meta-heuristic search and greedy heuristics may be promising because they have proved to be useful for the construction of CCAs. A recent study attempts to provide a systematic framework to compare CCA generators [42]. Applying such a framework to compare different CDA construction algorithms is also of interest.

References

- [1] D. R. Kuhn, R. N. Kacker, Y. Lei, Introduction to combinatorial testing, CRC Press, 2013.
- [2] D. R. Kuhn, D. R. Wallace, Software fault interactions and implications for software testing, *IEEE Transactions on Software Engineering* 30 (2004) 418–421.
- [3] C. J. Colbourn, D. W. McClary, Locating and detecting arrays for interaction faults, *Journal of Combinatorial Optimization* 15 (2008) 17–48.
- [4] J. Lin, C. Luo, S. Cai, K. Su, D. Hao, L. Zhang, TCA: An efficient two-mode meta-heuristic algorithm for combinatorial test generation, in: Proc. of the 30th International Conference on Automated Software Engineering (ASE), ACM/IEEE, 2015, pp. 494–505.
- [5] T. Shiba, T. Tsuchiya, T. Kikuno, Using artificial life techniques to generate test cases for combinatorial testing, in: Proc. of 28th Annual International Computer Software and Applications Conference (COMPSAC '04), 2004, pp. 71–77.
- [6] Y. Lei, R. Kacker, D. R. Kuhn, V. Okun, J. Lawrence, Ipog/ipog-d: efficient test generation for multi-way combinatorial testing, *Software Testing, Verification and Reliability* 18 (2008) 125–148.
- [7] H. Jin, T. Tsuchiya, Constrained locating arrays for combinatorial interaction testing, *Journal of Systems and Software* 170 (2020) 110771.
- [8] L. Hu, W. E. Wong, D. R. Kuhn, R. N. Kacker, How does combinatorial testing perform in the real world: an empirical study, *Empirical Software Engineering* 25 (2020) 2661–2693.
- [9] B. Hnich, S. D. Prestwich, E. Selensky, B. M. Smith, Constraint models for the covering test problem, *Constraints* 11 (2006) 199–219.
- [10] T. Tsuchiya, Using binary decision diagrams for constraint handling in combinatorial interaction testing, *CoRR abs/1907.01779* (2019).
- [11] A. Gargantini, P. Vavassori, CitLab: A laboratory for combinatorial interaction testing, in: 2012 IEEE Fifth International Conference on Software Testing, Verification and Validation, 2012, pp. 559–568. doi:10.1109/ICST.2012.141.

- [12] I. Segall, R. Tzoref-Brill, E. Farchi, Using binary decision diagrams for combinatorial test design, in: Proc. of the 2011 International Symposium on Software Testing and Analysis (ISSTA), ACM, 2011, pp. 254–264.
- [13] B. S. Ahmed, K. Z. Zamli, W. Afzal, M. Bures, Constrained interaction testing: A systematic literature study, IEEE Access 5 (2017) 1–1.
- [14] H. Wu, C. Nie, J. Petke, Y. Jia, M. Harman, A survey of constrained combinatorial testing, 2019. arXiv:1908.02480.
- [15] H. Wu, N. Changhai, J. Petke, Y. Jia, M. Harman, Comparative analysis of constraint handling techniques for constrained combinatorial testing, IEEE Transactions on Software Engineering (2019) 1–1.
- [16] C. J. Colbourn, V. R. Syrotiuk, On a combinatorial framework for fault characterization, Mathematics in Computer Science 12 (2018) 429–451.
- [17] R. Compton, M. T. Mehari, C. J. Colbourn, E. De Poorter, V. R. Syrotiuk, Screening interacting factors in a wireless network testbed using locating arrays, in: 2016 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS), 2016, pp. 650–655. doi:10.1109/INFCOMW.2016.7562157.
- [18] S. A. Seidel, K. Sarkar, C. J. Colbourn, V. R. Syrotiuk, Separating interaction effects using locating and detecting arrays, in: C. Iliopoulos, H. W. Leong, W.-K. Sung (Eds.), Combinatorial Algorithms, Springer International Publishing, Cham, 2018, pp. 349–360.
- [19] C. Shi, Y. Tang, J. Yin, Optimal locating arrays for at most two faults, Science China Mathematics 55 (2012) 197–206.
- [20] C. Shi, Y. Tang, J. Yin, The equivalence between optimal detecting arrays and super-simple OAs, Designs, Codes and Cryptography 62 (2012) 131–142.
- [21] X.-N. Lu, M. Jimbo, Arrays for combinatorial interaction testing: a review on constructive approaches, Japanese Journal of Statistics and Data Science 2 (2019) 641–667.
- [22] A. N. Aldaco, C. J. Colbourn, V. R. Syrotiuk, Locating arrays: A new experimental design for screening complex engineered systems, SIGOPS Oper. Syst. Rev. 49 (2015) 31–40.
- [23] C. J. Colbourn, V. R. Syrotiuk, Coverage, location, detection, and measurement, in: 2016 IEEE Ninth International Conference on Software Testing, Verification and Validation Workshops (ICSTW), 2016, pp. 19–25. doi:10.1109/ICSTW.2016.38.
- [24] C. Martínez, L. Moura, D. Panario, B. Stevens, Locating errors using ELAs, covering arrays, and adaptive testing algorithms, SIAM Journal on Discrete Mathematics 23 (2010) 1776–1799.
- [25] C. Shi, L. Jiang, A. Tao, Consecutive detecting arrays for interaction faults, Graphs and Combinatorics 36 (2020) 1203–1218.
- [26] H. Jin, T. Kitamura, E.-H. Choi, T. Tsuchiya, A satisfiability-based approach to generation of constrained locating arrays, in: 2018 IEEE International Conference on Software Testing, Verification and Validation Workshops, 2018, pp. 285–294. doi:10.1109/ICSTW.2018.00062.
- [27] H. Jin, T. Tsuchiya, Deriving fault locating test cases from constrained covering arrays, in: 2018 IEEE 23rd Pacific Rim International Symposium on Dependable Computing (PRDC), 2018, pp. 233–240. doi:10.1109/PRDC.2018.00044.
- [28] H. Jin, C. Shi, T. Tsuchiya, Constrained detecting arrays for fault localization in combinatorial testing, in: Proceedings of the 35th Annual ACM Symposium on Applied Computing, SAC '20, Association for Computing Machinery, New York, NY, USA, 2020, p. 1971–1978. URL: <https://doi.org/10.1145/3341105.3373952>. doi:10.1145/3341105.3373952.
- [29] H. Jin, T. Tsuchiya, A two-step heuristic algorithm for generating constrained detecting arrays for combinatorial interaction testing, in: 2020 IEEE 29th International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE), 2020, pp. 219–224. doi:10.1109/WETICE49692.2020.00050.
- [30] Z. Wang, B. Xu, L. Chen, L. Xu, Adaptive interaction fault location based on combinatorial testing, in: 2010 10th International Conference on Quality Software, 2010, pp. 495–502. doi:10.1109/QSIC.2010.36.
- [31] Z. Zhang, J. Zhang, Characterizing failure-causing parameter interactions by adaptive testing, in: Proceedings of the 20th International Symposium on Software Testing and Analysis, ISSTA 2011, Toronto, ON, Canada, July 17–21, 2011, 2011, pp. 331–341. URL: <https://doi.org/10.1145/2001420.2001460>. doi:10.1145/2001420.2001460.
- [32] J. Li, C. Nie, Y. Lei, Improved delta debugging based on combinatorial testing, in: 2012 12th International Conference on Quality Software, Xi'an, Shaanxi, China, August 27–29, 2012, 2012, pp. 102–105. URL: <https://doi.org/10.1109/QSIC.2012.28>. doi:10.1109/QSIC.2012.28.
- [33] P. Arcaini, A. Gargantini, M. Radavelli, Efficient and guaranteed detection of t-way failure-inducing combinations, in: 2019 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW), 2019, pp. 200–209. doi:10.1109/ICSTW.2019.00054.
- [34] J. Bonn, K. Foegen, H. Lichter, A framework for automated combinatorial test generation, execution, and fault characterization, in: 2019 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW), 2019, pp. 224–233. doi:10.1109/ICSTW.2019.00057.
- [35] X. Niu, C. Nie, H. Leung, Y. Lei, X. Wang, J. Xu, Y. Wang, An interleaving approach to combinatorial testing and failure-inducing interaction identification, IEEE Trans. Softw. Eng. 46 (2020) 584–615.
- [36] X. Niu, C. Nie, Y. Lei, A. T. S. Chan, Identifying failure-inducing combinations using tuple relationship, in: 2013 IEEE Sixth International Conference on Software Testing, Verification and Validation Workshops, 2013, pp. 271–280. doi:10.1109/ICSTW.2013.38.
- [37] X. Niu, H. Wu, N. Changhai, Y. Lei, X. Wang, A theory of pending schemas in combinatorial testing, IEEE Transactions on Software Engineering (2021) 1–1.
- [38] C. Yilmaz, M. B. Cohen, A. A. Porter, Covering arrays for efficient fault characterization in complex configuration spaces, IEEE Trans. Software Eng. 32 (2006) 20–34.
- [39] K. Shakya, T. Xie, N. Li, Y. Lei, R. Kacker, D. R. Kuhn, Isolating failure-inducing combinations in combinatorial testing using test augmentation and classification, in: Fifth IEEE International Conference on Software Testing, Verification and Validation, ICST 2012, Montreal, QC, Canada, April 17–21, 2012, 2012, pp. 620–623. URL: <https://doi.org/10.1109/ICST.2012.149>. doi:10.1109/ICST.2012.149.
- [40] K. Nishiura, E. Choi, O. Mizuno, Improving faulty interaction localization using logistic regression, in: 2017 IEEE International Conference on Software Quality, Reliability and Security (QRS), 2017, pp. 138–149. doi:10.1109/QRS.2017.24.
- [41] T. Friedrichs, K. Fögen, H. Lichter, A comparison infrastructure for fault characterization algorithms, in: 2020 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW), 2020, pp. 201–210. doi:10.1109/ICSTW50294.2020.00042.
- [42] A. Bombarda, E. Crippa, A. Gargantini, An environment for benchmarking combinatorial test suite generators, in: 2021 IEEE International

