IEICE TRANS. FUNDAMENTALS, VOL.E105–A, NO.6 JUNE 2022

PAPER

An Improved Adaptive Algorithm for Locating Faulty Interactions in Combinatorial Testing

Qianqian YANG[†,††], ***Nonmember and*** **Xiao-Nan LU**[††a)], ***Member***

SUMMARY Combinatorial testing is an effective testing technique for detecting faults in a software or hardware system with multiple factors using combinatorial methods. By performing a test, which is an assignment of possible values to all the factors, and verifying whether the system functions as expected (pass) or not (fail), the presence of faults can be detected. The failures of the tests are possibly caused by combinations of multiple factors assigned with specific values, called faulty interactions. Martínez et al. [1] proposed the first deterministic adaptive algorithm for discovering faulty interactions involving at most two factors where each factor has two values, for which graph representations are adopted. In this paper, we improve Martínez et al.'s algorithm by an adaptive algorithmic approach for discovering faulty interactions in the so-called "non-2-locatable" graphs. We show that, for any system where each "non-2-locatable factor-component" involves two faulty interactions (for example, a system having at most two faulty interactions), our improved algorithm efficiently discovers all the faulty interactions with an extremely low mistaken probability caused by the random selection process in Martínez et al.'s algorithm. The effectiveness of our improved algorithm are revealed by both theoretical discussions and experimental evaluations.

key words: *combinatorial testing, covering array, adaptive testing, faulty interaction, testing-equivalence, factor-component*

1. Introduction

Combinatorial testing (CT), also called combinatorial interaction testing (CIT), is an effective testing technique for detecting multi-factor faults in a software or hardware system using combinatorial methods. A system under test (SUT) for CT is basically modeled by multiple *factors* (or parameters) and their associated *values* (or levels) taken from a finite alphabet. A test suite is a collection of test cases for the given SUT. The presence of failures can be detected from the outcomes of tests. In a large software or hardware system, the factors are usually interrelated with some others, and specific combinations of values may cause unexpected or incorrect test outcomes, called *failures*. The combinations of factors assigned with specific values causing failures are called *faulty interactions*.

Pioneer studies on CT can be traced back to 1980s [2], [3]. Theories and applications on CT have been extensively studied in the recent decades for software testing and hardware testing [4], [5]. However, most studies on theories of CT concentrated on test suite generation. Whereas, there is only a limited number of literature on the issue of discovering the reasons once a failure is encountered.

Let us give an example for CT. An SUT model for a printing system with four factors is shown in Table 1, where each factor has two or three possible values. Suppose that a failure occurs if one of the following is specified: "PDF files with double-sided printing", "JPEG files with wireless connection", and "PS files with color printing". By using 1, 2, 3, 4 for the factors and 0, 1, 2 for the corresponding values as given in Table 1, these faulty interactions can be written as $\{(1,0),(2,1)\}$, $\{(2,0),(3,1)\}$, and $\{(2,2),(4,1)\}$, respectively. In order to detect the failures, a trivial way is to perform an exhaustive testing for all the possible combinations, which requires $2 \times 3 \times 2 \times 2 = 24$ test cases. However, using the test suite provided in Table 2, all the failures can be verified from the failed test outcomes, for which only six tests are performed. The combinations causing failures are marked in bold in Table 2. In fact, it is not hard to see in Table 2 that, for each pair of factors, every possible combination appears at least once. The underlying array of such test suites are called *(mixed) covering arrays*, for which the formal definitions will be given in Sect. 2.2.

Although the presence of failures can be successfully detected by using (mixed) covering arrays, it is not easy to identify which are the faulty interactions causing the failures.

Table 1 An SUT model for a printing system.

Factors	Values
(1) Double-sided or not	0 = double-sided 1 = single-sided
(2) File format	0 = JPEG 1 = PDF 2 = PS
(3) Connection	0 = USB cable 1 = wireless
(4) Colors	0 = black & white 1 = color

Table 2 A test suite for the printing system.

Tests	Factors				Outcomes
	(1)	(2)	(3)	(4)	
Test 1	0	0	0	0	pass
Test 2	1	**0**	**1**	1	fail
Test 3	1	1	0	1	pass
Test 4	**0**	**1**	1	0	fail
Test 5	1	2	0	0	pass
Test 6	0	**2**	1	**1**	fail

Manuscript received June 23, 2021.
Manuscript revised September 27, 2021.
Manuscript publicized November 29, 2021.

[†]The author is with College of Computer Science and Technology, Hangzhou Dianzi University, Hangzhou 310018, China.
[††]The authors are with Department of Computer Science and Engineering, University of Yamanashi, Kofu-shi, 400-8510 Japan.
a) E-mail: xnlu@yamanashi.ac.jp

DOI: 10.1587/transfun.2021EAP1071

For instance, only using the outcomes given in Table 2, one cannot determine the real reason that makes Test 2 failed. It might be because we printed a JPEG file with single-sided mode ($\{(2,0),(1,1)\}$), or we printed a JPEG file by wireless connection ($\{(2,0),(3,1)\}$), or we used the wireless connection for color printing ($\{(3,1),(4,1)\}$).

There are two kinds of approaches for discovering faulty interactions: *non-adaptive* testing and *adaptive* testing. For non-adaptive testing, the full test suite should be prepared in advance without prior information on the outcome of any test. Whereas, for adaptive testing, test cases can be generated based on the previous outcomes. The most significant contributions for non-adaptive testing are (d,t)-detecting arrays (DAs) and (d,t)-locationg arrays (LAs) introduced by Colbourn and McClary [6]. While, the parameters d for the number of faulty interactions and t for the size of each interaction are required to be determined in advance. There are a few work on DAs and LAs following [6], and the interested reader is referred to [7]–[9] and their references.

The first deterministic adaptive algorithm was proposed by Martínez et al. [1] for discovering faulty interactions of size at most two where each factor has two values. However, there are a few classes of faulty interactions with specific structural constraints that their approach cannot deal with.

In this paper, we improve Martínez et al.'s adaptive algorithm [1, Sect. 6] for the above issues and show the effectiveness for our improvements. In particular, our improved algorithm is the first algorithm that can efficiently discover all the faulty interactions whenever there are at most two pairwise faulty interactions (i.e., faulty interactions involving two factors).

At the system testing phase, it could be assumed that each factor has been tested in isolation, and the integrated system is going to be tested as a whole [10]. Moreover, it is shown by several empirical results [11]–[13] that testing all pairwise interactions in a software system is sufficient to discover most of its faults. Hence, we restrict our attention on only pairwise interactions.

The remaining of this paper is organized as follows. In Sect. 2, formal definitions for combinatorial testing and covering arrays are reviewed. Section 3 gives an intensive introduction on the graph representations for faulty interactions, together with some novel concepts and lemmas for our further discussions. Moreover, we clarify which kind of faulty interactions cannot be discovered by the adaptive algorithm in [1] by a careful investigation on graph structures. In Sect. 4, we propose the improved adaptive algorithm for the issues clarified in Sect. 3 and prove their correctness. Section 5 shows the effectiveness of our improved algorithm by both theoretical discussions and experimental evaluations. Conclusions and future work are summarized in Sect. 6.

2. Definitions and Notation

In this section, we introduce the formal notions and terminologies in combinatorial testing, and then summarize some basic facts on covering arrays.

2.1 Basic Notions in Combinatorial Testing

Consider a system under test (SUT) consisting of k factors denoted by $1,2,\ldots,k$. Throughout this paper, we use the notation $[1,k]$ to denote the set $\{1,2,\ldots,k\}$. Without loss of generality, the values are taken on $\Omega_i = \{0,1,\ldots,g_i-1\}$ for factor $i \in [1,k]$ A *test case* (simply, a *test*) is an assignment of values to all the factors, which can be represented by a k-tuple in $\Omega_1 \times \Omega_2 \times \cdots \times \Omega_k$. A *$t$-way interaction* is an assignment of values to t factors, which can be represented by a set of factor–value pairs involving t distinct factors, namely $I = \{(f_1,v_1),(f_2,v_2),\ldots,(f_t,v_t)\}$ with distinct $f_j \in [1,k]$ and $v_j \in \Omega_j$ for $1 \le j \le t$, where the parameter t is called the *strength* of I. A t-way interaction $I = \{(f_1,v_1),(f_2,v_2),\ldots,(f_t,v_t)\}$ is said to be *covered* by a test case $T = (T_1,T_2,\ldots,T_k)$ if $T_{f_j} = v_j$ for every $1 \le j \le t$. By regarding the test T as a k-way interaction, we say the t-way interaction I is covered by T if and only if $I \subseteq T$.

The execution of a test gives an outcome, *pass* or *fail*. We assume that a failure of a test is caused by some interactions (including 1-way interactions, which is simply a factor assigned with some value). In other words, the execution of a test T fails if and only if T covers one or more faulty interactions.

2.2 Covering Arrays

A *covering array* (CA) $\mathcal{A}$, denoted by $\mathrm{CA}(N;t,k,g)$, is an $N \times k$ array whose entries are taken from the alphabet $[0,g-1]$, such that every t-way interaction is covered by some row of $\mathcal{A}$. In $\mathcal{A}$, each column of corresponds to a factor and each row represents a test case. In other words, in a CA $\mathcal{A} = (a_{i,j})$, for any set of t factors $\{f_1,\ldots,f_t\}$ and any choice of t values $(v_1,\ldots,v_t) \in [0,g-1]^t$, there exists some test r such that $a_{r,f_i} = v_i$ for each $1 \le i \le t$. The parameter t is called the *strength* of $\mathcal{A}$. If each factor is allowed to take different values, the notion of covering arrays can be generalized to *mixed covering arrays*, denoted by $\mathrm{MCA}(N;t,k,(g_1,g_2,\ldots,g_k))$, where the i-th factor takes values from $[0,g_i-1]$ (see [14]). For example, the test suite used in Table 2 is an $\mathrm{MCA}(6;2,4,(2,3,2,2))$.

The most essential problem on CAs is to find a $\mathrm{CA}(N;t,k,g)$ with the smallest N when the other parameters are given. Let $\mathrm{CAN}(t,k,g)$ denote the smallest integer such that a $\mathrm{CA}(N;t,k,g)$ exists. A $\mathrm{CA}(N;t,k,g)$ has exacly $N = \mathrm{CAN}(t,k,g)$ rows is said to be *optimal*. For $t = g = 2$, Katona [15], Kleitman and Spencer [16] independently proved that $\mathrm{CAN}(2,k,2)$ is equal to the smallest N satisfying $k \le \binom{N-1}{\lceil N/2 \rceil}$. The construction for such optimal $\mathrm{CA}(N;2,k,2)$ is quite simple and elegant, as follows.

Theorem 2.1 (see [15], [16]): Let $\mathcal{X}$ be the set of all the N-dimensional binary vectors such that the Hamming weight of $\mathbf{x}$ is $\lceil N/2 \rceil$ and the first entry of $\mathbf{x}$ is 0 for every $\mathbf{x} \in \mathcal{X}$. Then, the $N \times k$ array whose columns are the vectors of $\mathcal{X}$ is an optimal $\mathrm{CA}(N;2,k,2)$, where $k = |\mathcal{X}| = \binom{N-1}{\lceil N/2 \rceil}$.

An example of optimal CA(6; 2, 10, 2) obtained by Theorem 2.1 is given as follows:

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 \end{bmatrix}.$$

It is well-known that $\mathrm{CAN}(t,k,g) \in \Theta(\log k)$ for fixed t and g. For $t = 2$, Gargano et al. [17] showed that $\mathrm{CAN}(2,k,g) = \frac{g}{2}\log k(1+o(1))$. For $t > 2$, the determination of $\mathrm{CAN}(t,k,g)$ is still a widely open question. Moreover, Colbourn maintains a comprehensive repository [18] on the best known values of $\mathrm{CAN}(t,k,g)$ for $2 \le t \le 6$ and $2 \le g \le 25$. The mathematical constructions for optimal and nearly optimal CAs have been intensively studied in the recent decades; see the surveys [19]–[22] for details.

3. Graph Representation for Faulty Interactions of Strength at Most 2

In what follows, the strength of all the faulty interactions are supposed to be *at most* 2. In this section, we give a brief introduction on the graph structures for representing faulty interactions with the notions in [1]. Moreover, we introduce two novel concepts, "testing-equivalence" between two graphs and "factor-components" of the graphs, which play an important role in our improved locating algorithms.

3.1 Graph Representation and Characterization

If we restrict the strength to be *exactly* two, then each faulty interaction can be represented by an edge, and then all the faulty interactions form a simple graph, that is, an undirected graph without loops and multiple edges. Notably, there may be self-loops in the graph if 1-way interactions are allowed.

Suppose that the SUT consists of the factors in $[1,k]$, and each factor i has g_i values in $\Omega_i = \{0,1,\ldots,g_i-1\}$. Let $V_i = \{(i,0),(i,1),\ldots,(i,g_i-1)\}$ and $V = \bigcup_{i=1}^{k} V_i$. Let E be the set of all 2-way faulty interactions of the SUT. Then the graph $G = (V,E)$ visually represents the faulty interactions for the SUT. The graph G is a subgraph of $K_{g_1,g_2,\ldots,g_k}$, the complete k-partite graph where the i-th partite set V_i consists of g_i vertices for each $1 \le i \le k$. Clearly, G is also a k-partite graph. In what follows, the term "graphs" means the graphs representing the faulty interactions, unless stated otherwise.

Example 3.1: The graph shown in Fig. 1 represents the faulty interactions with respect to the SUT given in Table 1.

Let T be a test and let $\textsc{Test}(G,T)$ be the outcome of T for (the SUT represented by) the graph G, which is either *pass* or *fail*. More precisely, $\textsc{Test}(G,T) = \text{fail}$ if and only if T covers some $e \in E(G)$. For simplicity, we will write $\textsc{Test}(T)$ instead of $\textsc{Test}(G,T)$ if no confusion occurs.

Definition 3.2: Let G and G' be two different graphs (not necessarily simple, in other words, may having self-loops) with the same vertex set. Then G and G' are *testing-equivalent* if $\textsc{Test}(G,T) = \textsc{Test}(G',T)$ for any test case $T \in \Omega_1 \times \Omega_2 \times \cdots \times \Omega_k$.

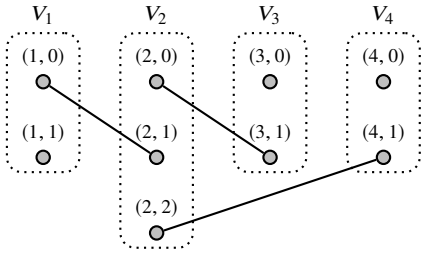


Fig. 1 The graph for the SUT given in Table 1.

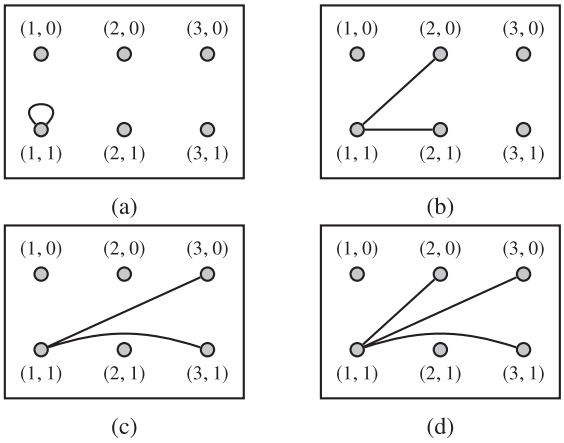


Fig. 2 Examples of testing-equivalent graphs.

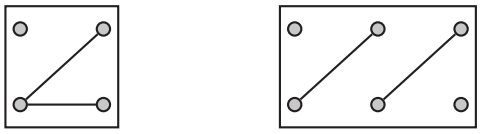
Definition 3.3: A graph G is said to be *minimal* if there does not exist a graph G' testing-equivalent to G such that $E(G')$ is a proper subset of $E(G)$.

Example 3.4: The graphs shown in Fig. 2 are mutually testing-equivalent. It is easily seen that for each G of the four graphs, we have $\textsc{Test}(G,T) = \text{fail}$ for $T = (1,a,b)$ with any $a,b \in \{0,1\}$ and $\textsc{Test}(G,T') = \text{pass}$ for any other T'. The graph in Fig. 2(d) is not minimal, since the graph in Fig. 2(c) is its (minimal) subgraph. It is remarkable that the graph in Fig. 2(a) has a self-loop at $(1,1)$, representing a 1-way interaction, so it is not a simple graph.

In general, we have the following lemma describing the testing-equivalence for the graphs with self-loops.

Lemma 3.5: Let G be a graph containing a self-loop at (f_ℓ, v_ℓ). Let G' be the graph obtained by removing the self-loop at (f_ℓ, v_ℓ) and adding the edges $\{(f_\ell, v_\ell),(f,s)\}$ for all $0 \le s \le g_i - 1$, where $f \in [1,k] \setminus \{f_\ell\}$. Then, G' is testing-equivalent to G. Furthermore, if G is minimal, then the resulting G' is minimal as well.

Proof. For any test T with $T_{f_\ell} = v_\ell$, $\textsc{Test}(G,T) = \textsc{Test}(G',T) = \text{fail}$. Conversely, for any test T with $T_{f_\ell} \neq v_\ell$, since the induced subgraph $G[X]$ is exactly the same graph with $G'[X]$, where $X = V(G) \setminus \{(f_\ell, v_\ell)\}$, it is clear that



(a) type-a (b) type-b

Fig. 3 The forbidden configurations for 2-locatable graphs.

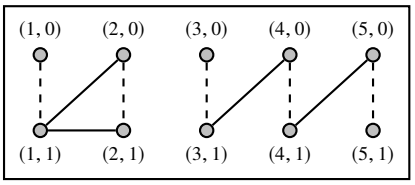


Fig. 4 The auxiliary graph G^* in Example 3.8.

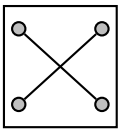


Fig. 5 The factor-component of parallel edges.

$\textsc{Test}(G,T)$ and $\textsc{Test}(G',T)$ are identical. The minimality is obvious by noting that $G'[X] = G[X]$ is minimal, and $(V(G'), E(G') \setminus E(G))$ is also minimal. □

For a test $T = (T_1, T_2, \ldots, T_k) \in \Omega_1 \times \Omega_2 \times \cdots \times \Omega_k$, where $\Omega_i = \{0, 1, \ldots, g_i - 1\}$, we say T *avoids* the graph G if for any $J \subseteq [1,k]$ we have $\{(i, T_i) : i \in J\} \notin E(G)$. Here, it suffices to consider the case $|J| \leq 2$, since we only consider the interactions of strength at most 2. A 2-way interaction $I = \{(f_1, v_1), (f_2, v_2)\}$ is *locatable* with respect to G if there exists a k-tuple $T \in \Omega_1 \times \Omega_2 \times \cdots \times \Omega_k$ with $T_{f_i} = v_i$ for $i \in \{1,2\}$ that avoids $G - e_I$, where e_I denotes the edges in G corresponding to I and

$$G - e_I = \begin{cases} G, & \text{if } e_I \notin E(G), \\ (V(G), E(G) \setminus \{e_I\}), & \text{if } e_I \in E(G), \end{cases}$$

i.e., $G - e_I$ is the graph obtained by removing e_I from G if $e_I \in E(G)$. A graph G is 2-*locatable* (resp., $\bar{2}$-*locatable*) if every t-way interaction with $t = 2$ (resp., $t \leq 2$) is locatable with respect to G.

Martínez et al. [1] proposed the following characterization of 2-locatable graphs where each factor has exactly two values, which will be called *binary graphs* in what follows.

Lemma 3.6 ([1] Corollary 6.2): A binary graph G is not 2-locatable if and only if it contains a subgraph isomorphic to one of the two forbidden configurations (type-a or type-b) shown in Fig. 3.

In this paper, we focus on binary graphs. Then, $V_i = \{(i,0),(i,1)\}$ and $V(G) = \bigcup_{i=1}^{k} V_i$. We introduce the notion of *factor-components* of G by using the the auxiliary graph G^* of G defined as $V(G^*) = V(G)$ and $E(G^*) = E(G) \cup \{\{(i,0),(i,1)\} : 1 \leq i \leq k\}$.

Definition 3.7: Let F be a subset of $[1,k]$, and let $X_F = \bigcup_{f \in F} V_f$. The induced subgraph $G[X_F]$ is said to be a *factor-component* of G if $G^*[X_F]$ is a component in G^*. The *size* of a factor-component $G[X_F]$ is the number of edges in $G[X_F]$.

Example 3.8: As illustrated in Fig. 4, the auxiliary graph G^* is obtained by adding the dashed edges inside each factor in G, where G is represented by the solid edges. There are two factor-components in G, namely, $G[V_1 \cup V_2]$ and $G[V_3 \cup V_4 \cup V_5]$.

3.2 Preliminaries for the Locating Algorithms

To investigate the graph structures in a comprehensive and systematic manner, following the notation in [1], the factors are partitioned into subsets according to their incident edges. The partition of factors is associated with a passing test. Without loss of generality, we assume that the test where all the factors assigned with 0 is a passing test.

We define the sets A, B, and C to partition the set of factors $[1,k]$. The set A consists of the factors where 1 is an endpoint of a 1-0 edge, i.e., $A = \{f \in [1,k] :$ there exists $j \in [1,k]$ such that $\{(f,1),(j,0)\} \in E(G)\}$. The set B contains the factors where 1 is an endpoint of a 1-1 edge and neither endpoints are in A, i.e., $B = \{f \in [1,k] \setminus A :$ there exists $j \in [1,k] \setminus A$ such that $\{(f,1),(j,1)\} \in E(G)\}$. All the remaining factors form the set C, i.e., $C = [1,k] \setminus (A \cup B)$.

The set A is further partitioned into the sets A^L, A^P, and A^S, where the set A^L consists of the factors $f \in A$ such that $\{(f,1),(j,0)\}, \{(f,1),(j,1)\} \in E(G)$ for some $j \in [1,k]$ (see Fig. 3(a)), the set A^P consists of the factors $f \in A \setminus A^L$ such that $\{(f,1),(f',0)\} \in E(G)$ for some $f' \in A$ (see Fig. 5), and $A^S = A \setminus (A^P \cup A^L)$. The set A^P is called the set of endpoints of "parallel edges" and A^S is called the set of "single factors" in A.

Remark 3.9: As shown in Lemma 3.5, by removing $\{(f,1),(j,0)\}, \{(f,1),(j,1)\}$ from G and adding a self-loop at $(f,1)$, the resulting graph is test-equivalent to G. So we simply say A^L is the set of "loops".

As subsets of the set C, let C^0 (resp., C^1) be the set of factors $f \in C$ such that $\{(f,0),(f',1)\} \in E(G)$ (resp., $\{(f,1),(f',1)\} \in E(G)$) for some $f' \in A^S$, and $C^I = C \setminus (C^0 \cup C^1)$.

Remark 3.10: As indicated in [1], C^0 and C^1 must be disjoint if G is 2-locatable; otherwise, a type-b forbidden configuration (e.g., $\{\{(f_3,0),(f_1,1)\}, \{(f_3,1),(f_2,1)\}\}$ where $f_3 \in C^0 \cap C^1$ as in Fig. 6) would be present. But, C^0 and C^1 may not be disjoint in general. However, in this case, the factor-component has at least three edges (see Fig. 6). Hence, C^0 and C^1 must be disjoint if every factor-component is either 2-locatable or of size 2.

Claim 3.11: The possible edges can be classified into the following three categories.

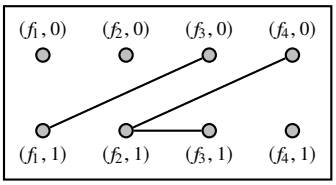
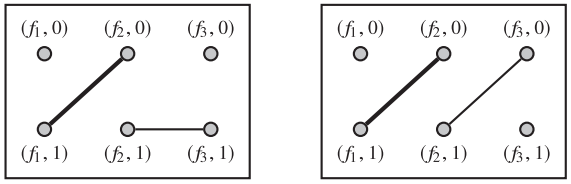


Fig. 6 A factor-component involving $f_3 \in C^0 \cap C^1$.



(a) G_{b1} (b) G_{b2}

Fig. 7 The type-b forbidden configurations for cases (II.i) and (II.ii). The thick lines indicate the corresponding edges.

(I) The edges which may appear in a 2-locatable factor-component:

- (I.i) 1-0 edges inside A^P,
- (I.ii) 1-1 edges inside A^S,
- (I.iii) 1-1 edges inside B,
- (I.iv) 1-1 edges between A^S and B,
- (I.v) 1-0 edges between A^S and C^0,
- (I.vi) 1-1 edges between A^S and C^1.

(II) The edges which is unavailable for a 2-locatable factor-component but may be contained in a factor-component of size 2:

- (II.i) 1-0 edges between A^S and B (which form type-b forbidden configurations as shown in Fig. 7(a)),
- (II.ii) 1-0 edges between A^P and A^S (which form type-b forbidden configurations as shown in Fig. 7(b)),
- (II.iii) 1-0 and 1-1 edges involving factors in A^L (which form type-a forbidden configurations).

(III) The edges which is impossible to be contained in a factor-component of size 2:

- (III.i) 1-1 edges with one of the endpoints in A^P.

The edges in category (I) can be correctly discovered by the algorithm proposed in [1, Sect. 6][†], which will be reviewed in Sect. 4.1. For category (III), if we assume that every factor-component of G is of size 2, then G has no 1-1 edge with an endpoint in A^P; otherwise, the factor-component involving some factor in A^P would contain at least three edges. In Sect. 4.3, we will propose adaptive algorithmic approaches for the category (II) edges.

[†]More precisely, some edges between A^S and $B \cup C$ may be missed with a low probability due to a random selection process in the algorithm.

4. Improved Adaptive Algorithms for Locating Binary Graphs

Martínez et al. [1] proposed an adaptive algorithm for locating faulty interactions in a $\bar{2}$-*locatable* graph, which will be simply called "MMPS algorithm" (named after the authors' names) hereafter. Based on MMPS algorithm, we give improved adaptive algorithms for locating faulty interactions in a graph G such that each factor-component of G is either 2-locatable or of size 2.

4.1 A Brief Review on MMPS Algorithm

MMPS algorithm is proposed for $\bar{2}$-locatable graphs, and clearly it also works for 2-locatable graphs, which are merely $\bar{2}$-locatable graphs without self-loops. In the original setting of MMPS algorithm, the set A^L is defined to be the factors involving self-loops. While, in Sect. 3.2 the definition of A^L is changed to be the factor of the common vertex of two edges in a type-a forbidden configuration (cf. Remark 3.9).

The framework of MMPS algorithm is given in Algorithm 1. The correctness is stated as follows.

Theorem 4.1 ([1] Theorem 6.4): For a $\bar{2}$-locatable graph G, MMPS algorithm correctly outputs the set A^P, $\tilde{A}^S \subseteq A^S$, $\tilde{A}^L \supseteq A^L$, B, C, and a set of edges $\tilde{E} \subseteq E(G)$ such that the only possible edges in $E(G) \setminus \tilde{E}$ are the 1-0 edges with one endpoint in $\tilde{A}^L \setminus A^L \subseteq A^S$. The probability that a factor f is wrongly classified into $\tilde{A}^L \setminus A^L$ is at most $\exp(-M/2^{\delta_f})$, where δ_f denotes the degree of the vertex $(f,1)$ in G, and $M > 2^{\delta_f}$ is a manually decided parameter for the number of iterations of a random selection process in the algorithm.

Now, we give a brief review on MMPS algorithm and investigate what happens if the graph is simple but not 2-locatable. As assumed in Sect. 3.2, the test $(0,0,\ldots,0)$ is a passing test. A covering array $\mathcal{A}$ of strength 2 is utilized as the initial test suite, which can be obtained from Theorem 2.1.

MMPS algorithm is divided into five steps as in Algorithm 1. The procedure in step 4 plays an essential role in our further discussions, so it is listed in Algorithm 2. While, the detailed procedures of the other steps are omitted and the reader is referred to the original work [1].

In step 1, SearchEndPoint(T, D), where T is a failing test and D is a set of factors, is a procedure for finding f such that $(f,1)$ is an endpoint of some edge in G. SearchEndPoint is basically a binary search algorithm starting from each failing test with its possible "fault-causing" factors. Combining with the 2-coverage property of a covering array, it is easily shown that SearchEndPoint correctly outputs the set A for any simple graph.

Step 2 is devoted to discovering parallel edges in E^P and the corresponding factors in A^P. Note that there is only one possibility (up to isomorphism) for a factor-component involving parallel edges in a 2-locatable graph, as illustrated

Algorithm 1 The framework of MMPS algorithm

1: **function** DiscoverEdges($\mathcal{F}$)
2: ▷ Precondition: $(0, 0, \ldots, 0)$ is a passing test; $\mathcal{F}$ consists of all failing test cases in $\mathcal{A}$
3: ▷ Step 1. Discover the set A.
4: $A \leftarrow \emptyset$
5: **for each** $T \in \mathcal{F}$ **do**
6: $F_1(T) \leftarrow \{f \in [1, k] : T_f = 1\}$
7: $A \leftarrow A \cup$ SearchEndPoint$(T, F_1(T))$
8: **end for**
9: ▷ Step 2. Discover the edge set E^P and the set A^P.
10: $(A^P, E^P) \leftarrow$ FindParallelEdges(A)
11: ▷ Step 3. Discover the set B.
12: Define T by $T_f = 0$ for $f \in A$ and $T_w = 1$ for $w \in [1, k] \setminus A$
13: $E^B \leftarrow$ LocateErrorsInTest$(s = (0, 0, \ldots, 0), T, [1, k] \setminus A)$
14: $B \leftarrow \bigcup_{e \in E^B} \{f_1, f_2 : e = \{(f_1, 1), (f_2, 1)\}\}$
15: ▷ Step 4. Find $\tilde{A}^S$ and $\tilde{A}^L$ that approximate A^S and A^L.
16: $(\tilde{A}^S, \tilde{A}^L, \tilde{C}^0, \tilde{C}^1, E^S) \leftarrow$ DiscoverAsAndAl(A, A^P, B)
17: ▷ Step 5. Find E^{SS} within $\tilde{A}^S$.
18: $E^{SS} \leftarrow$ FindEdgesWithinAs$(\tilde{A}^S, \tilde{C}^0)$
19: **return** $(A, B, A^P, \tilde{A}^L, \tilde{A}^S, \tilde{C}^0, \tilde{C}^1, E^P, E^B, E^S, E^{SS})$
20: **end function**

in Fig. 5. But for non-2-locatable graphs, there may be other types of factor-components containing parallel edges, such that MMPS algorithm may miss some 1-0 edges with both endpoints in A. The solution for this issue will be given in Sect. 4.3.

Similarly to step 1, LocateErrorsInTest in step 3 is a procedure for searching 1-1 edges, namely the edges internal to B, via binary search. As an initial test, the test T declared in line 12 of Algorithm 1 avoids all the edges having an endpoint $(f, 1)$ with $f \in A$. Hence, the set B and its internal edges in E^B can be correctly discovered in step 3 for any graph, regardless of the 2-locatable property.

The details of step 4 are given in Algorithm 2, which requires the results of the sets A, $A^P \subseteq A$, and B obtained in the previous steps. This step is aimed to distinguish A^S and A^L, which partition $A \setminus A^P$, and discover all the edges involving the factors in A^S. For finding A^L, a random selection process is employed, so that a factor $f \in A^S$ may be wrongly classified into $\tilde{A}^L \setminus A^L$ with a small probability that is exponentially decreasing with the iteration number M. To discover the 1-0 edges between $\tilde{A}^S$ (the approximate output for A^S) and B, new tests are exhaustively generated for all possible pairs in $\tilde{A}^S \times B$. For the edges between $\tilde{A}^S$ and C, the binary search algorithm SearchEndPoint is adopted. For non-2-locatable factor-components, even if the iteration number M tends to ∞, there is still some factor $f_s \in A^S$ that would be wrongly classified into $\tilde{A}^L$, and thus the edges involving f_s cannot be discovered. Detailed discussion on this issue will be held in Sect. 4.3.

Lastly, the purpose of step 5 is to find the set E^{SS} of all edges with both endpoints in $\tilde{A}^S$, based on a full exploration on all the pairs within $\tilde{A}^S$.

In summary, MMPS algorithm is able to correctly discover all the edges of category (I) in Claim 3.11, regardless of the 2-locatable property. As a direct consequence, MMPS algorithm correctly works for all the 2-locatable factor-components.

Algorithm 2 Procedure in step 4 of MMPS algorithm [1]

1: **function** DiscoverAsAndAl(A, A^P, B)
2: $\tilde{A}^S \leftarrow \emptyset$; $\tilde{A}^L \leftarrow \emptyset$
3: $E^S \leftarrow \emptyset$; $C \leftarrow [1, k] \setminus (A \cup B)$
4: **for each** $f \in A \setminus A^P$ **do**
5: Fix $T_f = 1$ and $T_i = 0$ for all $i \in (A \setminus \{f\}) \cup B$
6: $iter \leftarrow 0$
7: **repeat**
8: $iter++$
9: Randomly pick the values of T_j for $j \in C$
10: **until** (Test(T) = pass) or $(iter > M)$
11: **if** Test(T) = fail **then**
12: $\tilde{A}^L \leftarrow \tilde{A}^L \cup \{f\}$
13: **else**
14: $\tilde{A}^S \leftarrow \tilde{A}^S \cup \{f\}$
15: **for each** $b \in B$ **do**
16: $T' \leftarrow T$; $T'_b \leftarrow 1$
17: **if** Test(T') = fail **then**
18: $E^S \leftarrow E^S \cup \{\{v_{f,1}, v_{b,1}\}\}$
19: **end if**
20: **end for**
21: ▷ Binary search for mates of f in C
22: Set T'' with $T''_i = T_i$ for $i \in A \cup B$, $T''_i = \neg T_i$ for $i \in C$
23: **if** Test(T'') = fail **then**
24: $L \leftarrow$ SearchEndPoint(T'', C)
25: **end if**
26: $E^S \leftarrow E^S \cup \{\{v_{f,1}, v_{c,T''_c}\} : c \in L\}$
27: $\tilde{C}^0 \leftarrow \{c \in L : T''_c = 0\}$; $\tilde{C}^1 \leftarrow \{c \in L : T''_c = 1\}$
28: **end if**
29: **end for**
30: **return** $(\tilde{A}^S, \tilde{A}^L, \tilde{C}^0, \tilde{C}^1, E^S)$
31: **end function**

4.2 Framework of Our Improvements

Recall Claim 3.11 that, there are three types of edges unavailable for a 2-locatable factor-component but they are possibly contained in a factor-component of size 2, listed as follows:

(II.i) 1-0 edges between A^S and B,

(II.ii) 1-0 edges between A^P and A^S,

(II.iii) 1-0 and 1-1 edges involving factors in A^L.

Here, categories (II-i) and (II-ii) correspond to type-b forbidden configurations, and category (II-iii) corresponds to type-a forbidden configurations.

The framework of our improved algorithm is given in Algorithm 3. In Sect. 4.3, for finding the missed 1-0 edges in non-2-locatable factor-components of size 2, we will propose the algorithms FindBetweenAsAndB and FindBetweenAsAndAs for (II-i) and (II-ii), respectively. Moreover, in order to deal with category (II-iii) edges, in Sect. 4.4, we will explain the practical solution we provided in lines 10–17 in Algorithm 3.

4.3 Discovering Missed 1-0 Edges in Factor-Components of Size 2

Before going deeply in algorithms and proofs, let us start by

Algorithm 3 The framework of our improved algorithm

1: **function** ImprovedDiscoverEdges($\mathcal{F}$)
2: ▷ Precondition: $(0, 0, \ldots, 0)$ is a passing test; $\mathcal{F}$ consists of all failing test cases in $\mathcal{A}$
3: ▷ Discover the sets and edges by MMPS algorithm
4: $(A, B, A^P, \tilde{A}^L, \tilde{A}^S, \tilde{C}^0, \tilde{C}^1, E^P, E^B, E^S, E^{SS}) \leftarrow$ DiscoverEdges($\mathcal{F}$)
5: $E \leftarrow E^P \cup E^B \cup E^S \cup E^{SS}$
6: ▷ Discover missed 1-0 edges between A^S and B (see Algorithm 4)
7: $E \leftarrow E \cup$ FindBetweenAsAndB($\tilde{A}^L, \tilde{A}^S, E^B$)
8: ▷ Discover missed 1-0 edges between A^P and A^S (see Algorithm 5)
9: $E \leftarrow E \cup$ FindBetweenApAndAs($A^P, \tilde{A}^L, E^S$)
10: ▷ Output the solution set containing testing-equivalent subgraphs
11: ▷ Suppose $\tilde{A}^L = \{a_1, a_2, \ldots, a_\ell\}$, where $\ell = |\tilde{A}^L|$.
12: **if** $\ell > 0$ **then**
13: $\mathcal{E} \leftarrow \{\bigcup_{i=1}^{\ell} \{\{(a_i, 1), (f_i, 0)\}, \{(a_i, 1), (f_i, 1)\}\} \cup E : f_i \in [1, k] \setminus \{a_i\}$ for $1 \le i \le \ell\}$
14: **return** $\mathcal{E}$
15: **else**
16: **return** $\{E\}$
17: **end if**
18: **end function**

observing what would happen if a non-2-locatable graph is inputted into MMPS algorithm.

First, consider graph G_{b1} shown in Fig. 7(a). By the discussions in Sect. 4.1, the sets A and B can be correctly obtained by MMPS algorithm as $A = \{f_1\}$ and $B = \{f_2, f_3\}$. However, step 4 (Algorithm 2) gives the output $E_{\text{out}} = \{\{(f_2, 1), (f_3, 1)\}\}$ with $\tilde{A}^L = \{f_1\}$, but the edge $\{(f_1, 1), (f_2, 0)\}$ is missed.

Next, consider graph G_{b2} shown in Fig. 7(b). Then, the definitions tell us $f_1 \in A^P$, $f_2 \in A^S$, $f_3 \in C^0$. By the discussion in Sect. 4.1, the sets A and C can be correctly obtained by MMPS algorithm as $A = \{f_1, f_2\}$ and $C = \{f_1, f_2, f_3\} \setminus A = \{f_3\}$. However, step 4 (Algorithm 2) gives the output $E_{\text{out}} = \{\{(f_2, 1), (f_3, 0)\}\}$ with $\tilde{A}^L = \{f_1\}$, $\tilde{A}^S = \{f_2\}$, and $\tilde{C}^0 = \{f_3\}$, but the edge $\{(f_1, 1), (f_2, 0)\}$ is missed.

In step 4 of MMPS algorithm (Algorithm 2), in both G_{b1} and G_{b2}, for identifying whether f_1 is in A^L, the test cases $T = (1, 0, *)$ ($* \in \{0, 1\}$) are generated (see line 5 of Algorithm 2). Clearly, Test(T) always fails because of the unexpected edge $\{(f_1, 1), (f_2, 0)\}$. This gives rise to the resulting output with $f_1 \in \tilde{A}^L$. Note that, in both G_{b1} and G_{b2}, the other edge can be successfully discovered.

Now, we propose Algorithm 4 for discovering 1-0 edges between A^S and B in the factor-components isomorphic to G_{b1} (see Fig. 7(a)).

Lemma 4.2: If in graph G each non-2-locatable factor-component is of size 2, then Algorithm 4 is correct, that is, it correctly discovers the 1-0 edges between A^S and B in non-2-locatable factor-components of G.

Proof. Consider a factor-component isomorphic to G_{b1}. Similarly to the previous discussions, the factor f_1 must be wrongly classified into $\tilde{A}^L$ by MMPS algorithm. While, the other two factors f_2 and f_3 can be correctly classified into B. It suffices to check whether an "unexpected" edge (like $\{(f_1, 1), (f_2, 0)\}$ in G_{b1}) exists for each pair of factor $f \in \tilde{A}^L$ and edge $\{(b_1, 1), (b_2, 1)\} \in E^B$. Let T' and T'' be the tests

Algorithm 4 Discover missed 1-0 edges between A^S and B

1: ▷ E^B is the edge set discovered by MMPS algorithm containing all the 1-1 edges inside B
2: **function** FindBetweenAsAndB($\tilde{A}^L, \tilde{A}^S, E^B$)
3: $E \leftarrow \emptyset$
4: **for each** $f \in \tilde{A}^L$ **do**
5: **for each** edge $\{(b_1, 1), (b_2, 1)\} \in E^B$ **do**
6: Define T' by $T'_f = 1$, $T'_{b_1} = 1$, and $T'_w = 0$ for $w \in [1, k] \setminus \{f, b_1\}$
7: Define T'' by $T''_f = 1$, $T''_{b_2} = 1$, and $T'_w = 0$ for $w \in [1, k] \setminus \{f, b_2\}$
8: **if** Test(T') = pass and Test(T'') = fail **then**
9: $E \leftarrow E \cup \{(f, 1), (b_1, 0)\}$
10: $\tilde{A}^L \leftarrow \tilde{A}^L \setminus \{f\}$; $\tilde{A}^S \leftarrow \tilde{A}^S \cup \{f\}$
11: **end if**
12: **if** Test(T') = fail and Test(T'') = pass **then**
13: $E \leftarrow E \cup \{(f, 1), (b_2, 0)\}$
14: $\tilde{A}^L \leftarrow \tilde{A}^L \setminus \{f\}$; $\tilde{A}^S \leftarrow \tilde{A}^S \cup \{f\}$
15: **end if**
16: **end for**
17: **end for**
18: **return** E
19: **end function**

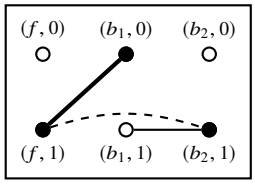


Fig. 8 The case when T'' (the solid vertices) is failed and T' is passed.

as defined lines 6 and 7 in Algorithm 4, respectively.

Consider the case when Test(T') = pass and Test(T'') = fail. Note that, since $b_1 \in B$, the vertex $(b_1, 1)$ cannot be adjacent to $(j, 0)$ for $j \in [1, k] \setminus \{f, b_2\}$; otherwise, the 1-0 edge would force factor b_1 into the set A. Then, Test(T'') is failed if and only if

- $\{(f, 1), (b_2, 1)\} \in E(G)$, or
- $\{(f, 1), (j, 0)\} \in E(G)$ for some $j \in [1, k] \setminus \{f, b_2\}$.

In the same manner, Test(T') is passed if and only if

- $\{(f, 1), (b_1, 1)\} \notin E(G)$, and
- $\{(f, 1), (j, 0)\} \notin E(G)$ for any $j \in [1, k] \setminus \{f, b_1\}$.

Therefore, combing the above conditions, as illustrated in Fig. 8, we have Test(T') = pass and Test(T'') = fail if and only if

(a) $\{(f, 1), (b_2, 1)\} \in E(G)$ and $\{(f, 1), (b_1, 1)\} \notin E(G)$ and $\{(f, 1), (j, 0)\} \notin E(G)$ for any $j \in [1, k] \setminus \{f, b_1\}$, or

(b) $\{(f, 1), (b_1, 0)\} \in E(G)$ and $\{(f, 1), (b_1, 1)\} \notin E(G)$ and $\{(f, 1), (b_2, 0)\} \notin E(G)$.

Case (a) can be further divided into two sub-cases:

(a-1) both $\{(f, 1), (b_2, 1)\}$ and $\{(f, 1), (b_1, 0)\}$ are edges in G, but there is no other 1-0 edge incident with $(f, 1)$;

(a-2) $\{(f, 1), (b_2, 1)\} \in E(G)$ and there does not exist any

1-0 edge incident with $(f, 1)$.

Sub-case (a-1) violates the assumption that each non-2-locatable factor-component is of size 2. While, for sub-case (a-2), the factor f should be classified into the set B, which contradicts the fact that $f \in \tilde{A}^L$. So, we can rule out case (a). In other words, the only possible edge that makes $\textsc{Test}(T') = \text{pass}$ and $\textsc{Test}(T'') = \text{fail}$ is $\{(f, 1), (b_1, 0)\}$. This completes the proof of the correctness for the former half part of Algorithm 4 (up to line 11).

The proof for the latter half part is totally the same to the former part, just by exchanging b_1 and b_2 with each other. □

Remark 4.3: Sub-case (a-1) possibly occurs if the graph G does not fulfill the assumption that each non-2-locatable factor-component is of size 2. In this case, Algorithm 4 successfully finds $\{(f, 1), (b_1, 0)\}$ (the thick edge in Fig. 8), but misses $\{(f, 1), (b_2, 1)\}$ (the dashed edge in Fig. 8). However, it can be directly verified that the factor-component shown in Fig. 8 with the dashed edge is testing-equivalent to that without the dashed edge. Hence, one can improve Algorithm 4 by outputting the dashed edge as a candidate solution.

Next, we consider the general case of G_{b2} that a graph G contains, as factor-components, subgraphs isomorphic to G_{b2}, i.e., type-b forbidden configurations in case (II.ii) with 1-0 edges between A^P and A^S. The algorithm needs the following passing test.

Lemma 4.4: Define T^{C0} by $T^{C0}_f = 1$ for $f \in C^0$ and $T^{C0}_w = 0$ for $w \in [1, k] \setminus C^0$. Then, T^{C0} is a passing test.

Proof. It follows from the definition of the set C that there is no edge within C; otherwise, the corresponding factor would be assigned to the set A or B. For the same reason, there does not exist an edge $\{(f, 1), (w, 0)\}$ for any $f \in C$ and $w \in [1, k] \setminus C$. Recall that $(0, 0, \ldots, 0)$ is a passing test. Then, the test T^{C0} avoids all the possible edges, which is equivalent to saying that T^{C0} is a passing test. □

Then, we propose Algorithm 5 for discovering the "unexpected" 1-0 edges between A^P and A^S in the factor-components isomorphic to G_{b2} (see Fig. 7(b)).

Lemma 4.5: If in graph G each non-2-locatable factor-component is of size 2, then Algorithm 5 is correct, that is, it correctly discovers the 1-0 edges between A^P and A^S in non-2-locatable factor-components of G.

Proof. Consider a factor-component isomorphic to G_{b2}. The factor f_1 is wrongly classified into $\tilde{A}^L$, and the other two factors f_2 and f_3, can be correctly classified into $\tilde{A}^S$ and $\tilde{C}^0$, respectively. We need to check whether an edge (like $\{(f_1, 1), (f_2, 0)\}$ in G_{b2}) exists for any factor $f \in \tilde{A}^L$ and any edge $\{(a_s, 1), (c_0, 0)\} \in E^S$.

It follows from Lemma 4.4 that T^{C0} is a passing test. Let T' and T'' be the tests as defined in lines 7 and 8 in Algorithm 5, respectively. Consider the case when $\textsc{Test}(T') = \text{pass}$ and $\textsc{Test}(T'') = \text{fail}$. Note that T'' and the passing test

Algorithm 5 Discover missed 1-0 edges between A^P and A^S

```
1: ▷ E^S is the edge set discovered in Algorithm 2 of MMPS algorithm
   containing all the 1-0 edges between A^S and C
2: function FindBetweenApAndAs(A^P, Ã^L, E^S)
3:     E ← ∅
4:     Define T^C0 by T^C0_f = 1 for f ∈ C^0 and T^C0_w = 0 for w ∈
   [1, k] \ C^0
5:     for each f ∈ Ã^L do
6:         for each edge {(a_s, 1), (c_0, 0)} ∈ E^S do
7:             Define T' by T'_f = 1, T'_{a_s} = 1, and T'_w = T^C0_w for w ∈
   [1, k] \ {a_s, c_0}
8:             Define T'' by T''_f = 1 and T''_w = T^C0_w for w ∈ [1, k] \ {f}
9:             if Test(T') = pass and Test(T'') = fail then
10:                E ← E ∪ {(f, 1), (a_s, 0)}
11:                Ã^L ← Ã^L \ {f}; A^P ← A^P ∪ {f}
12:            end if
13:        end for
14:    end for
15:    return E
16: end function
```

T^{C0} only differ in the values for the factor f. So $\textsc{Test}(T'')$ is failed if and only if

- $\{(f, 1), (j_0, 0)\} \in E(G)$ for some $j_0 \in [1, k] \setminus (C^0 \cup \{f\})$, or
- $\{(f, 1), (j_1, 1)\} \in E(G)$ for some $j_1 \in C^0$.

$\textsc{Test}(T')$ is passed only if

- $\{(f, 1), (j_0, 0)\} \notin E(G)$ for any $j_0 \in [1, k] \setminus (C^0 \cup \{f, a_s\})$, and
- $\{(f, 1), (j_1, 1)\} \notin E(G)$ for any $j_1 \in C^0 \cup \{f, a_s\}$.

Therefore, $\textsc{Test}(T') = \text{pass}$ and $\textsc{Test}(T'') = \text{fail}$ imply that $\{(f, 1), (a_s, 0)\} \in E(G)$. □

As a direct consequence of Lemmas 4.2 and 4.5, we have the following corollary.

Corollary 4.6: If in graph G each non-2-locatable factor-component is of size 2, then all the 1-0 edges in non-2-locatable factor-components of G can be correctly discovered by Algorithms 4 and 5, provided that $\tilde{A}^S = A^S$.

The only possible edges that cannot be discovered are that of the form $\{(f_1, 1), (f_2, 0)\}$ in a type-b forbidden configuration (see Fig. 7(b)), where $f_1 \in A^P \cap \tilde{A}^L$ and $f_2 \in \tilde{A}^L \setminus A^L$, due to the mistaken probability on f_2 of MMPS algorithm.

The time cost for fault location is evaluated by the number of tests needed to be performed. The total number of tests performed in Algorithms 4 and 5 is $2\left|\tilde{A}^L\right|(m_1 + m_2) \in O(d^2)$, where m_1 and m_2 are the number of 1-1 edges inside B and the number of 1-0 edges between $\tilde{A}^S$ and C, respectively. Here, d is the number of edges in the graph. It is remarkable that, the number of tests executed in MMPS algorithm is $O(d^2 + d \log k + d(\log k)^c)$ if a passing test is found in the covering array $\mathcal{A}$ and the maximum degree of G is upper bounded by $c \log \log k$ (see [1, Theorem 6.5]). This means that, by appending our proposed Algorithms 4 and 5 after MMPS algorithm, the complexity order in fault location does not change.

4.4 Generating Minimal Testing-Equivalent Solutions for Graphs Containing Type-a Forbidden Configurations

As shown in Algorithm 3, we suppose $\ell = \left|\tilde{A}^L\right| > 0$ and let $\tilde{A}^L = \{a_1, a_2, \ldots, a_\ell\}$. According to the definition of A^L (see Sect. 3.2), each $a_i \in \tilde{A}^L$ corresponds to a type-a forbidden configuration, that is a subgraph $G_{i,f_i} = (V(G), E_{i,f_i})$ with

$$E_{i,f_i} = \left\{\{(a_i,1),(f_i,0)\},\{(a_i,1),(f_i,1)\}\right\}$$

where $f_i \in [1,k]\setminus\{a_i\}$. It follows from Lemma 3.5 that G_{i,f_i} is testing-equivalent to $G_{i,f_i'}$ for any $f_i, f_i' \in [1,k]\setminus\{a_i\}$. In other words, by using combinatorial testing methods, no one can distinguish G_{i,f_i} and $G_{i,f_i'}$. However, in practical applications, even if the output graph is testing-equivalent to the real graph for the faulty interactions, it would be useless as long as they are not exactly the same. Hence, it can be considered that *a solution set which contains the true answer* is better than an "equivalently true" answer. In Algorithm 3, the solution set $\mathcal{E}$ is defined as follows:

$$\mathcal{E} = \left\{\bigcup_{i=1}^{\ell} E_{i,f_i} \cup E : f_i \in [1,k]\setminus\{a_i\} \text{ for } 1 \le i \le \ell\right\}. \quad (1)$$

Note that all the candidates in $\mathcal{E}$ are minimal by Lemma 3.5. Generally, it is possible that for each $a_i \in \tilde{A}^L$ there are more than two edges involving a_i in the real graph. However, if all the possibilities are considered, it would cause an exponential explosion for the cardinality of the solution set. This is the primary reason why we only consider minimal graphs.

Every non-minimal graph contains a minimal one as its subgraph. In practice, a minimal subgraph can provide most information on the faulty interactions. For example, suppose the real graph G is as shown in Fig. 2(d), which is not minimal. The algorithm tells us that $\tilde{A}^L = \{1\}$ and $\mathcal{E}$ has two members, as in Fig. 2(b) and 2(c). In this case, no candidate in $\mathcal{E}$ meets the true answer $E(G)$. However, as the minimal testing-equivalent subgraph of G, the graph in Fig. 2(c), which contains most faulty edges, is provided as a candidate solution.

To end this section, we show the correctness of the entire improved algorithm (Algorithm 3). Here, the output $\mathcal{E}$ is considered to be correct if there exists $E \in \mathcal{E}$ coincides with $E(G)$ of the real graph G.

Theorem 4.7: Let G be a graph where each non-2-locatable factor-component is of size 2. Then Algorithm 3 is correct, that is, it correctly gives a solution set $\mathcal{E}$ such that $E(G) \in \mathcal{E}$, provided that $\tilde{A}^S = A^S$.

Moreover, the probability that a factor f is wrongly classified into $\tilde{A}^L \setminus A^L$ is at most $\exp(-M/2^{\delta_f})$, due to the mistaken probability of MMPS algorithm as in Theorem 4.1. Let $f \in \tilde{A}^L \setminus A^L$ be such a factor. Then, there exists $\tilde{E} \in \mathcal{E}$ such that $E(G) \subseteq \tilde{E}$ and $\tilde{E} \setminus E(G) = \left\{\{(f,1),(c_0,1)\},\{(f,1),(a_p,1)\}\right\}$, where c_0 is the factor in C^0 satisfying $\{(f,1),(c_0,0)\} \in E(G)$ and a_p is the factor in $A^P \cap \tilde{A}^L$ satisfying $\{(a_p,1),(f,0)\} \in E(G)$.

Proof. Since each non-2-locatable factor-component of G is of size 2, it is obvious that each factor-component containing a type-a forbidden configuration must be minimal. Combining with Corollary 4.6, the statement clearly holds when $\tilde{A}^S = A^S$.

For the case when MMPS Algorithm wrongly gives $f \in \tilde{A}^L \setminus A^L$, the following two edges in $E(G)$ cannot be discovered before line 10 of Algorithm 3:

- the edge of the form $\{(f,1),(c_0,0)\}$ where $c_0 \in C_0$;
- the edge of the form $\{(a_p,1),(f,0)\}$ in a type-b forbidden configuration (cf. the edge $\{(f_1,1),(f_2,0)\}$ in Fig. 7(b)), where $a_p \in A^P \cap \tilde{A}^L$.

The former is missing in MMPS Algorithm because of the mistake of f. Consequently, this causes the latter missing in Algorithm 5. Eventually, both f and a^p are wrongly classified into $\tilde{A}^L \setminus A^L$. Then, in the last step of Algorithm 3, four edges $\{(f,1),(g,0)\}$, $\{(f,1),(g,1)\}$, $\{(a_p,1),(h,0)\}$, and $\{(a_p,1),(h,1)\}$ would be added to create a candidate in $\mathcal{E}$, where $g \in [1,k]\setminus\{f\}$ and $h \in [1,k]\setminus\{a_p\}$. Let $\tilde{E}$ be the candidate in $\mathcal{E}$ with $g = c_0$ and $h = f$. Then, we have $E(G) \subseteq \tilde{E}$ and the proof is completed. □

5. Discussions and Experimental Evaluations on the Effectiveness of our Improved Algorithm

In this section, we make a comparison on the numbers of graphs that can be correctly recovered between MMPS algorithm and our improved algorithm, which shows the effectiveness of our improved algorithm.

Let $\mathcal{G}_k^{(d)}$ be the set of all graphs with d edges representing the faulty interactions in an SUT with k factors where each factor takes two values. Let $\rho_d(k)$ denote the proportion of 2-locatable graphs in $\mathcal{G}_k^{(d)}$, which is also the the proportion of graphs that can be correctly recovered by using MMPS algorithm in $\mathcal{G}_k^{(d)}$. Moreover, let $\rho_d^*(k)$ denote the proportion of graphs that can be correctly recovered by our improved algorithm in $\mathcal{G}_k^{(d)}$.

5.1 Explicit Formulas for Graph Enumerating

In this subsection, we are devoted to theoretical analysis of $\rho_d(k)$ and $\rho_d'(k)$, where $\rho_d'(k)$ denotes the proportion of the graphs where each factor-component is 2-locatable or of size 2 in $\mathcal{G}_k^{(d)}$. Here, $\rho_d'(k)$ is considered to be a good lower bound for the "success proportion" $\rho_d^*(k)$ of our improved algorithm.

Let $N_d(k) = \left|\mathcal{G}_k^{(d)}\right|$. Then, $N_d(k)$ is equal to the number of subgraphs of the complete k-partite graph $K_{2,2,\ldots,2}$ of size d, that is $N_d(k) = \binom{e_k}{d}$, where $e_k := 4\binom{k}{2} = 2k(k-1)$ is the number of edges in $K_{2,2,\ldots,2}$. Thus, it suffices to find the number of non-2-locatable graphs of size d for given k, denoted by $n_d(k)$. Similarly, for given k, let $n_d'(k)$ denote

the number of size-d graphs containing at least one non-2-locatable factor-component with more than two edges. Then,

$$\rho_d(k) = 1 - \frac{n_d(k)}{N_d(k)} \quad \text{and} \quad \rho'_d(k) = 1 - \frac{n'_d(k)}{N_d(k)}. \tag{2}$$

We firstly give the explicit formula of $n_2(k)$ with a combinatorial proof.

Proposition 5.1: The total number of non-2-locatable graphs of size 2 with respect to k factors is $n_2(k) = 2k(k-1)(2k-3)$.

Proof. By taking any two factors from $[1,k]$, say f_1 and f_2, and considering (f_i,u) (for some $i \in \{1,2\}$, $u \in \{0,1\}$) as a "loop" vertex, one can obtain a subgraph isomorphic to a type-a forbidden configuration. Hence, the number of size-2 graphs containing a subgraph isomorphic to the type-a forbidden configuration is $n_2^{(a)}(k) := 4\binom{k}{2} = 2k(k-1)$. Next, consider the similar process for creating a type-b forbidden configuration as follows. Firstly, take a factor f from $[1,k]$ to be the factor which has edges incident to another two factors. Secondly, sequentially take f_0, f_1 from $[1,k] \setminus \{f\}$ to be the two factors incident with $(f,0)$ and $(f,1)$, respectively. Thirdly, choose $\{(f_0,u_0),(f,0)\}$ and $\{(f_1,u_1),(f,1)\}$ with $u_0,u_1 \in \{0,1\}$. Then, the number of size-2 graphs containing a subgraph isomorphic to the type-b forbidden configuration is $n_2^{(b)}(k) := 4k(k-1)(k-2)$. Therefore, $n_2(k) = n_2^{(a)}(k) + n_2^{(b)}(k) = 2k(k-1)(2k-3)$, which completes the proof. □

Next, we propose a general way to calculate $n_d(k)$ by the following lemma. For the sake of convenience, in graph G, the factors involved in the edges of G are called *fault-causing factors*.

Lemma 5.2: The number of non-2-locatable graphs of size d with respect to k factors is

$$n_d(k) = \sum_{l=l_0}^{2d-1} m_l \binom{k}{l}, \tag{3}$$

where the term $m_l\binom{k}{l}$ represents the number of non-2-locatable graphs of size d with l fault-causing factors, and l_0 is the smallest positive integer such that $d \le 4\binom{l_0}{2}$.

Proof. The number of non-2-locatable graphs of size d with given $F \subseteq [1,k]$ as the set of its fault-causing factors, denoted by m_l, is invariant under the choices of F. Moreover, for any two different subsets $F_1, F_2 \subseteq [1,k]$, let G_1 and G_2 be two arbitrary graphs where F_i is the set of fault-causing factors of G_i for $i \in \{1,2\}$. Then, it is easily seen that G_1 and G_2 must be different graphs (not necessarily non-isomorphic), which implies that the above enumeration for m_l graphs does not involve duplicated graphs. Therefore, $n_d(k)$ can be expressed as the sum of $m_l\binom{k}{l}$ for positive integers $l \in \mathbb{N}$.

The range of l is determined as follows. On the one hand, at least l_0 factors is required to accommodate d edges, where l_0 is defined as in the statement of the lemma. On the other hand, in a non-2-locatable graph having d edges, the number of fault-causing factors cannot exceed $2d$, since each edge involves exactly two factors. Moreover, a size-d graph with exactly $2d$ fault-causing factors is nothing but a graph consisting of d isolated edges as its connected components, which is clearly 2-locatable. So, we set $l \le 2d-1$ for the sum of (3). □

The coefficient m_l in (3) can be obtained by a tedious mathematical enumeration or a computer program. With the aid of computers, we have the following formulas.

Proposition 5.3: The numbers of non-2-locatable graphs in $\mathcal{G}_k^{(d)}$ for $d \in \{3,4,5\}$ are, respectively,

$$\begin{aligned}
n_3(k) =& 4\binom{k}{2} + 200\binom{k}{3} + 864\binom{k}{4} + 960\binom{k}{5},\\
n_4(k) =& \binom{k}{2} + 492\binom{k}{3} + 8400\binom{k}{4} + 38400\binom{k}{5} + 66240\binom{k}{6}\\
&+ 40320\binom{k}{7},\\
n_5(k) =& 792\binom{k}{3} + 39240\binom{k}{4} + 446144\binom{k}{5} + 2025600\binom{k}{6}\\
&+ 4394880\binom{k}{7} + 4623360\binom{k}{8} + 1935360\binom{k}{9}.
\end{aligned}$$

It is possible to give a proof of Proposition 5.3 using purely combinatorial arguments, as in Proposition 5.1 for $d = 2$. However, the proofs by case analysis would be quite lengthy, and they are omitted. Even by using a computer program, the enumeration cannot be done with ease for large d, because of the huge number of candidate graphs. For example, when $d = 5$, in order to get the explicit formula for $n_5(k)$, one needs to generate all the 481,008,528 graphs having five edges with nine fault-causing factors, and check the 2-locatable property for each of them.

Proposition 5.4: The numbers of graphs in $\mathcal{G}_k^{(d)}$ containing at least one non-2-locatable factor-component with more than two edges, for $d \in \{3,4,5\}$ are, respectively,

$$\begin{aligned}
n'_3(k) =& 4\binom{k}{2} + 200\binom{k}{3} + 768\binom{k}{4},\\
n'_4(k) =& \binom{k}{2} + 492\binom{k}{3} + 8304\binom{k}{4} + 36000\binom{k}{5} + 46080\binom{k}{6},\\
n'_5(k) =& 792\binom{k}{3} + 39240\binom{k}{4} + 445824\binom{k}{5} + 2000640\binom{k}{6}\\
&+ 3978240\binom{k}{7} + 2580480\binom{k}{8}.
\end{aligned}$$

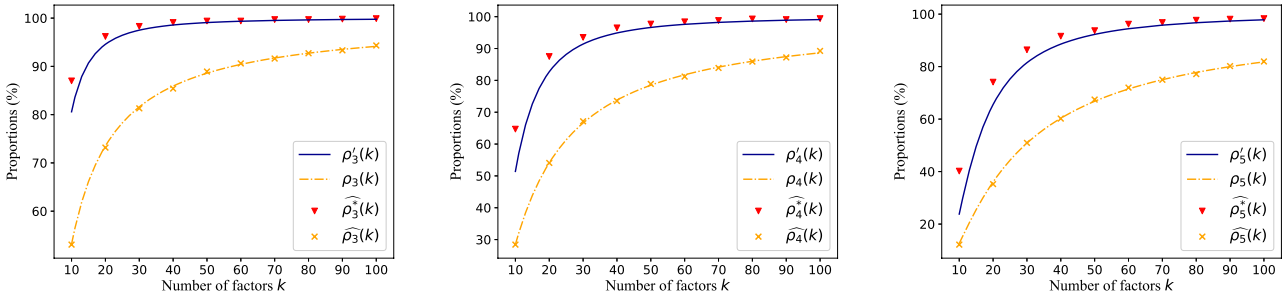
For the general case, which is similar to Lemma 5.2, we have

$$n'_d(k) = \sum_{l=l_0}^{2d-2} m'_l \binom{k}{l}, \tag{4}$$

where l_0 is as defined in Lemma 5.2, and $m'_l\binom{k}{l}$ represents

Table 3 The proportions that all the faulty edges can be correctly discovered by MMPS algorithm (denoted by $\widehat{\rho_d}(k)$) and our improved algorithm (denoted by $\widehat{\rho_d^*}(k)$).

		$k=10$	$k=20$	$k=30$	$k=40$	$k=50$	$k=60$	$k=70$	$k=80$	$k=90$	$k=100$
$d=2$	$\widehat{\rho_2}(k)$	81.0%	90.1%	93.7%	95.3%	96.1%	96.7%	97.2%	97.2%	97.5%	98.3%
	$\widehat{\rho_2^*}(k)$	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%
$d=3$	$\widehat{\rho_3}(k)$	53.7%	73.8%	81.1%	86.6%	89.0%	89.9%	91.8%	92.3%	93.7%	94.0%
	$\widehat{\rho_3^*}(k)$	87.0%	96.2%	98.3%	99.1%	99.4%	99.4%	99.7%	99.7%	99.8%	99.9%
$d=4$	$\widehat{\rho_4}(k)$	28.8%	54.6%	67.4%	73.7%	79.2%	81.6%	84.2%	86.2%	87.6%	88.9%
	$\widehat{\rho_4^*}(k)$	64.7%	87.5%	93.5%	96.5%	97.7%	98.4%	98.8%	99.3%	99.1%	99.4%
$d=5$	$\widehat{\rho_5}(k)$	13.3%	36.9%	52.1%	61.2%	68.0%	71.8%	75.7%	77.4%	80.3%	83.0%
	$\widehat{\rho_5^*}(k)$	40.2%	74.1%	86.4%	91.6%	93.7%	96.2%	96.8%	97.7%	98.1%	98.3%



(a) $d=3$ (b) $d=4$ (c) $d=5$

Fig. 9 The proportions $\rho_d(k)$, $\rho'_d(k)$, $\widehat{\rho_d}(k)$, and $\widehat{\rho_d^*}(k)$ for $d=3,4,5$.

the number of graphs in $\mathcal{G}_k^{(d)}$ having exacly l fault-causing factors such that there is at least one non-2-locatable factor-component with more than two edges. Remarkably, it must hold in (4) that $l \le 2d-2$, since a size-d graph with exactly $2d-1$ fault-causing factors must consist of $d-1$ factor-components, one of which is a size-2 factor-component involving three factors, and the other $d-2$ are isolated edges.

In summary, $n'_d(k)$ is a polynomial of degree $2d-2$, while $n_d(k)$ is of degree $2d-1$. In other words, for given k, among all the $O(k^{2d})$ graphs, there are $O(k^{2d-1})$ graphs for which MMPS algorithm is not applicable. While, by our improved algorithm, this number is reduced to $O(k^{2d-2})$.

5.2 Experimental Results for MMPS Algorithm and our Improved Algorithm

In this subsection, we propose experimental evaluation results for estimating the values of $\rho_d(k)$ and $\rho_d^*(k)$ for $2 \le d \le 5$.

We implemented MMPS algorithm and our proposed algorithm in Python 3. The performance of MMPS algorithm and the proposed algorithm are examined as follows. In each experiment, for fixed d and k, a graph with d edges is randomly generated as the input. More precisely, each edge of an input graph is independently randomly picked up from all the $2k(k-1)$ possible edges. The randomly generated graph represents the faulty interactions of an SUT. For each input graph G, MMPS algorithm outputs a set $\tilde{E}$ of edges. We say the edges are correctly discovered by MMPS algorithm if $E(G)=\tilde{E}$. While, our proposed algorithm outputs a set of candidates, say $\mathcal{E}$, where each candidate is a set of edges. We say the edges are correctly discovered by our improved algorithm if $E(G) \in \mathcal{E}$.

The iteration number for the experiments is set to be 10,000 for each pair of parameters d and k. By $\widehat{\rho_d}(k)$ and $\widehat{\rho_d^*}(k)$, we denote the proportions of graphs that are correctly discovered among the 10,000 randomly generated inputs by MMPS algorithm and our proposed algorithm, respectively. The parameter M in MMPS algorithm is set to be 50. Table 3 shows the values of $\widehat{\rho_d}(k)$ and $\widehat{\rho_d^*}(k)$ obtained from the experiments for $d \in \{2,3,4,5\}$ and $k \in \{10,20,\ldots,100\}$.

It can be verified from the results for $d=2$ in Table 3 that, our improved algorithm is able to correctly discover all the edges for all the graphs of size 2. However, some cases cannot be well dealt with by MMPS algorithm.

In Fig. 9, for $d \in \{3,4,5\}$, the graphs of the functions $\rho_d(k)$ and $\rho'_d(k)$, which are derived from (2) and Propositions 5.3 and 5.4, are plotted together with the data shown in Table 3. It can be observed that the values of $\rho_d(k)$ and $\widehat{\rho_d}(k)$ are almost identical, which provides strong numerical evidence for the correctness of the formulas in Propositions 5.3. Moreover, the values of $\rho'_d(k)$ and $\widehat{\rho_d^*}(k)$ are extremely close when k is large. Accordingly, it seems that $\rho'_d(k)$ can be used as a good lower bound for $\rho_d^*(k)$.

As mentioned in the last paragraph of Sect. 5.1, by our improved algorithm, the number of inapplicable graphs is reduced to $O(k^{2d-2})$ from $O(k^{2d-1})$. Although one may notice that $\lim_{k\to\infty}\rho_d(k)=\lim_{k\to\infty}\rho'_d(k)=1$ and $\lim_{k\to\infty}(\rho'_d(k)-\rho_d(k))=0$, the gaps between $\rho'_d(k)$ and $\rho_d(k)$ are significant when k is not too huge, which can be intuitively verified from Fig. 9. Furthermore, it can be observed from Fig. 10 that, for fixed $k \ge 20$, the difference

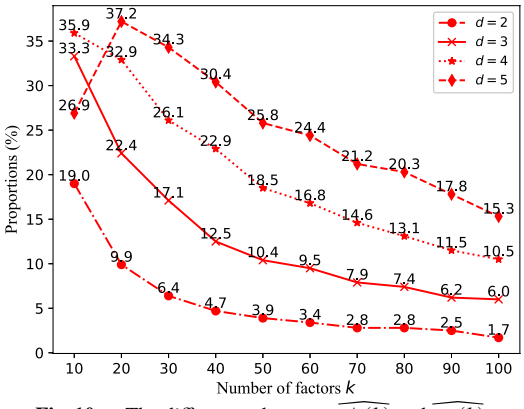
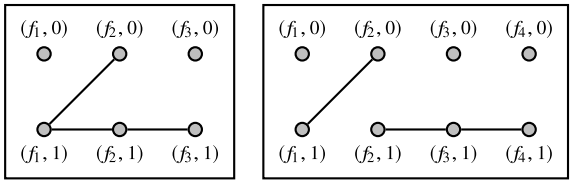


Fig. 10 The differences between $\widehat{\rho^*_d(k)}$ and $\widehat{\rho_d(k)}$.



(a) f_1, f_2 are involved in a type-a configuration

(b) f_1, f_2, f_3 are involved in a type-b configuration

Fig. 11 Factor-components of size 3 that is applicable to our improved algorithm.

$\rho^*_d(k) - \rho_d(k)$ becomes more significant for larger d.

5.3 Remarks on the Factor-Components with Three or More Edges

It can be observed from Fig. 9 that $\rho^*_d(k) \geq \rho'_d(k)$ holds in general. The main reason for the inequality can be considered as follows. In a factor-component H containing a type-a or type-b forbidden configuration with more than two edges, if the edges in the forbidden configuration are not involved in other forbidden configurations, then our improved algorithm can correctly discover all the edges in H. Two simple examples are illustrated in Fig. 11(a) and 11(b), for type-a and type-b forbidden configurations, respectively.

6. Conclusion and Future Work

In this paper, we studied adaptive algorithms for locating 2-way faulty interactions, continuing the work by Martínez et al. [1], which is called MMPS algorithm in this paper. By introducing two novel notions, *testing-equivalence* and *factor-component* for the graph structures representing the faulty interactions, we provided an adaptive algorithmic approach for discovering faulty interactions in a large number of graphs for which there was no locating algorithm in the literature. Such graphs are said to be not 2-locatable in [1]. However, our algorithm shows that they can be actually "located".

As the main result of this paper, we showed that, for any graph G where each non-2-locatable factor-component is of size 2, our improved algorithm can correctly give a solution set $\mathcal{E}$ such that $E(G) \in \mathcal{E}$ with an extremely low mistaken probability caused by the random selection process in MMPS algorithm. The proposed algorithm is realized by by a combination of MMPS algorithm and our proposed procedures. Notably, for a graph with d edges, only $O(d^2)$ tests are performed in the improved parts (i.e., Algorithms 4 and 5), such that the entire improved algorithm has the same order of time complexity with MMPS algorithm in fault location.

The effectiveness of our improved algorithm is discussed by both theoretical analysis and experimental evaluations. Among all the graphs with k factors and d edges, which is as many as $O(k^{2d})$, there are $O(k^{2d-1})$ graphs for which MMPS algorithm cannot deal with. While, by our improved algorithm, this number is reduced to $O(k^{2d-2})$. In particular, for practical applications when d and k are not that huge, it can be observed from Table 3 and Fig. 9 that, the locating ability is significantly improved by our proposed algorithm.

Potential directions for further extensions include adaptive algorithms for non-2-locatable factor-components with more than two edges, and the generalization of the algorithms for multi-value systems. As suggested by a reviewer, in addition to the simulation evaluation, it is an important future study to perform practical evaluation by using real software fault data, for instance, [23], [24].

Acknowledgments

The authors would like to thank the anonymous reviewers for their valuable comments in improving the quality of this work. This work was supported in part by JSPS Grants-in-Aid for Scientific Research Nos. 18H01133, 19K14585, and 20K03715. The corresponding author (X.-N. Lu) was supported by Leading Initiative for Excellent Young Researchers, MEXT, Japan.

References

[1] C. Martínez, L. Moura, D. Panario, and B. Stevens, "Locating errors using ELAs, covering arrays, and adaptive testing algorithms," SIAM J. Discrete Math., vol.23, no.4, pp.1776–1799, 2010.
[2] S. Satoh and H. Shimokawa, "Methods for setting software test parameters using the design of experiments method," Proc. 4th Symposium on Quality Control in Software, pp.1–8, Japanese Union of Scientists and Engineers (JUSE), 1984 (in Japanese).
[3] R. Mandl, "Orthogonal Latin squares: An application of experiment design to compiler testing," Commun. ACM, vol.28, no.10, pp.1054–1058, 1985.
[4] C. Nie and H. Leung, "A survey of combinatorial testing," ACM Comput. Surv., vol.43, no.2, pp.1–29, 2011.
[5] D.R. Kuhn, R.N. Kacker, and Y. Lei, Introduction to Combinatorial Testing, CRC Press, Boca Raton, 2013.
[6] C.J. Colbourn and D.W. McClary, "Locating and detecting arrays for interaction faults," J. Comb. Optim., vol.15, no.1, pp.17–48, 2008.
[7] C.J. Colbourn and V.R. Syrotiuk, "On a combinatorial framework for fault characterization," Math. Comput. Sci., vol.12, no.4, pp.429–451, 2018.
[8] X.N. Lu and M. Jimbo, "Arrays for combinatorial interaction testing:

A review on constructive approaches," Jpn. J. Stat. Data Sci., vol.2, no.2, pp.641–667, 2019.
[9] C. Shi, J. Fu, C. Wang, and J. Yan, "Upper bounds and constructions of locating arrays," IEICE Trans. Fundamentals, vol.E104-A, no.5, pp.827–833, May 2021.
[10] A.W. Williams and R.L. Probert, "A measure for component interaction test coverage," Proc. ACS/IEEE International Conference on Computer Systems and Applications, pp.304–311, IEEE, 2001.
[11] D.R. Kuhn and M.J. Reilly, "An investigation of the applicability of design of experiments to software testing," Proc. 27th Annual NASA Goddard/IEEE Software Engineering Workshop, pp.91–95, IEEE, 2002.
[12] D.R. Kuhn, D.R. Wallace, and A.M. Gallo, "Software fault interactions and implications for software testing," IEEE Trans. Softw. Eng., vol.30, no.6, pp.418–421, 2004.
[13] E.H. Choi, O. Mizuno, and Y. Hu, "Code coverage analysis of combinatorial testing," Proc. 4th International Workshop on Quantitative Approaches to Software Quality (QUASoQ 2016), in conjunction with APSEC, pp.34–40, 2016.
[14] L. Moura, J. Stardom, B. Stevens, and A. Williams, "Covering arrays with mixed alphabet sizes," J. Comb. Des., vol.11, no.6, pp.413–432, 2003.
[15] G.O. Katona, "Two applications (for search theory and truth functions) of sperner type theorems," Period. Math. Hung., vol.3, no.1-2, pp.19–26, 1973.
[16] D.J. Kleitman and J. Spencer, "Families of k-independent sets," Discrete Math., vol.6, no.3, pp.255–262, 1973.
[17] L. Gargano, J. Körner, and U. Vaccaro, "Sperner capacities," Graphs Combinatorics, vol.9, no.1, pp.31–46, 1993.
[18] C.J. Colbourn, "Covering array tables for $t = 2, 3, 4, 5, 6$," http://www.public.asu.edu/~ccolbou/src/tabby/catable.html, accessed at June 16th, 2021.
[19] C.J. Colbourn, "Covering arrays," Handbook of Combinatorial Designs, C.J. Colbourn and J.H. Dinitz, eds., ch. VI.10, pp.361–364, CRC Press, Boca Raton, 2006.
[20] C.J. Colbourn, "Combinatorial aspects of covering arrays," Le Matematiche, vol.59, no.1, 2, pp.125–172, 2004.
[21] A. Hartman, "Software and hardware testing using combinatorial covering suites," Graph Theory, Combinatorics and Algorithms, M.C. Golumbic and I.B.A. Hartman, eds., pp.237–266, Springer, New York, 2005.
[22] J. Lawrence, R.N. Kacker, Y. Lei, D.R. Kuhn, and M. Forbes, "A survey of binary covering arrays," Electron. J. Combin., vol.18, no.1, pp.1–30 (P84), 2011.
[23] H. Do, S. Elbaum, and G. Rothermel, "Supporting controlled experimentation with testing techniques: An infrastructure and its potential impact," Empir. Software Eng., vol.10, no.4, pp.405–435, 2005.
[24] L. Hu, W.E. Wong, D.R. Kuhn, and R.N. Kacker, "How does combinatorial testing perform in the real world: An empirical study," Empir. Software Eng., vol.25, no.4, pp.2661–2693, 2020.

Qianqian Yang received the B.S. degree in computer science and technology from Shandong Youth University of Political Science, Shandong, China, in 2019. She is currently pursuing the M.S. degree in computer science and technology with a joint program between Hangzhou Dianzi University, China, and the University of Yamanashi, Japan. Her research interests include deep learning and software engineering.

Xiao-Nan Lu received his M.S. degree and Ph.D. in Information Science from Nagoya University in 2014 and 2017, respectively. He is currently an assistant professor at University of Yamanashi. His research interests are in discrete mathematics and its applications to coding theory, cryptography, combinatorial algorithms, and statistics. His recent interests include combinatorial interaction testing, combinatorial group testing, and related searching problems.