

# Diagnosis for Reconfigurable Single-Electron Transistor Arrays with a More Generalized Defect Model

CHIA-CHENG WU, National Tsing Hua University

YI-HSIANG HU, National Chiao Tung University

CHIA-CHUN LIN, National Tsing Hua University

YUNG-CHIH CHEN, Yuan Ze University

JUINN-DAR HUANG, National Chiao Tung University

CHUN-YAO WANG, National Tsing Hua University

---

Single-Electron Transistor (SET) is considered as a promising candidate of low-power devices for replacement or co-existence with Complementary Metal-Oxide-Semiconductor (CMOS) transistors/circuits. In this work, we propose a diagnosis approach for SET array under a more generalized defect model. With the more generalized defect model, the diagnosis approach will become more practical but complicated. We conducted experiments on a set of SET arrays with different dimensions and defect rates. The experimental results show that our approach only has 3.8% false-negative rate and 0.7% misjudged-category rate on average without reporting any false-positive edge when the defect rate is 4%. Therefore, the proposed diagnosis approach can diagnose the defective SET arrays and elevate the reliability of the SET arrays in the synthesis flow.

CCS Concepts: • **Hardware** → **Single electron devices**;

Additional Key Words and Phrases: Single-electron transistor array, diagnosis, generalized defect model, clustering-defect

## ACM Reference format:

Chia-Cheng Wu, Yi-Hsiang Hu, Chia-Chun Lin, Yung-Chih Chen, Juinn-Dar Huang, and Chun-Yao Wang. 2021. Diagnosis for Reconfigurable Single-Electron Transistor Arrays with a More Generalized Defect Model. *J. Emerg. Technol. Comput. Syst.* 17, 2, Article 15 (January 2021), 23 pages.

<https://doi.org/10.1145/3444751>

---

## 1 INTRODUCTION

Reducing power consumption in Complementary Metal-Oxide-Semiconductor (CMOS) circuits has been considered as one of the main challenges to meeting Moore's Law. To deal with this issue, many emerging low-power devices have been proposed. Among them, some demonstrations

---

This work is supported in part by the Ministry of Science and Technology of Taiwan under MOST 107-2221-E-155-046, MOST 108-2221-E-155-047, MOST 106-2221-E-007-111-MY3 and MOST 108-2218-E-007-061.

Authors' addresses: C.-C. Wu, C.-C. Lin, and C.-Y. Wang, Department of Computer Science, National Tsing Hua University, Hsinchu, Taiwan 30013; emails: orange392817@gmail.com, chiachunlin@gapp.nthu.edu.tw, wcyao@cs.nthu.edu.tw; Y.-H. Hu and J.-D. Huang, Department of Electronics Engineering and the Institute of Electronics, National Chiao Tung University, Hsinchu, Taiwan 30013; emails: p8306271636@gmail.com, jdhuang@g2.nctu.edu.tw; Y.-C. Chen, Department of Computer Science and Engineering, Yuan Ze University, Taoyuan, Taiwan 32003; email: ycchen.cse@saturn.yzu.edu.tw. Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from [permissions@acm.org](mailto:permissions@acm.org).

© 2021 Association for Computing Machinery.

1550-4832/2021/01-ART15 \$15.00

<https://doi.org/10.1145/3444751>

of Single-Electron Transistor (SET) operations at room temperature proved that SET devices can be the promising candidate for replacement or co-existence with CMOS transistors/circuits in the future [Liu et al. 2015b, 2011; Postma et al. 2001; Tan et al. 2003; Zhuang et al. 1998].

Since only a few electrons are involved in the switching process of the SET operations, SETs suffer from low transconductance such that CMOS-based logic architecture is not applicable. To this end, a Binary Decision Diagram (BDD)-based platform was proposed in Asahi et al. [1997] to implement logic functions with SET devices. With this BDD-based platform, any Boolean function can be mapped onto an SET array.

However, the BDD-based SET arrays in Asahi et al. [1997] are not amendable. If there exists a defective nanowire segment or SET device in the array, the SET array cannot achieve its function and the entire circuit becomes useless. This causes a low yield of SET arrays since nanowires and nanodevices have a high defect rate. Fortunately, a reconfigurable SET array was proposed to improve the reliability and increase the flexibility of the SETs [Eachempati et al. 2008]. Such flexible reconfigurable SET arrays promote the research of synthesis and verification of SET arrays [Chen et al. 2011, 2013, 2014, 2015; Chiang et al. 2013; Liu et al. 2014, 2015a; Zhao et al. 2014]. Although these works were effective, they assumed that the SET arrays are defect-free. Therefore, these approaches might fail once the SET arrays are defective. As a result, a defect-aware synthesis algorithm was proposed in Huang et al. [2015]. This method successfully detoured or reused the defects for mapping a function onto a defective SET array.

The previous work in Huang et al. [2015] assumed that all the defect information, including the defect types and locations, has been obtained before mapping. Unfortunately, this assumption is optimistic. Since defects on the SET arrays can affect the mapping result, it is important to have a diagnosis algorithm to recognize the defects in a defective SET array before mapping. Therefore, the first diagnosis approach to identify the defects in a reconfigurable SET array was proposed in Huang et al. [2016]. This diagnosis algorithm used a *diagnosis sequence*, which is composed of input patterns and configurations, to diagnose an SET array effectively. Since the diagnosis sequence is fixed, the diagnosis process only terminates when the entire SET array is traversed completely. This diagnosis approach is considered as a *static* diagnosis approach. Since this static approach spent a lot of redundant efforts on the part that has been identified, it is inefficient. To improve the efficiency of the diagnosis process, a *dynamic* diagnosis approach was proposed in Li et al. [2017]. The dynamic approach adjusts the diagnosis sequence based on the feedback of its previous diagnosis process. This approach can achieve 100% coverage as well while spending much less CPU time than the static approach.

However, these two previous diagnosis methods [Huang et al. 2016; Li et al. 2017] have some optimistic assumptions about the defect distribution, i.e., (1) open defects and short defects do not occur on the two edges of a node device simultaneously; (2) two defective nodes are not adjacent to each other. When the size of a SET node becomes smaller, these two assumptions are not reasonable. This is because the size of a particle might be larger than an SET node such that adjacent nodes could be defective simultaneously. Therefore, in this work we allow that defective nodes could be adjacent and clustered together to form a defective area in an SET array. We call such defective area a *clustering-defect*, which will be further discussed in Section 2.

In this work, we propose a diagnosis approach for SET array under a more generalized defect model, i.e., a clustering-defect is possible. With the more generalized defect model, the diagnosis approach will become more practical but complicated. We conducted experiments on a set of SET arrays with different dimensions. The experimental results show that our approach can achieve 3.8% false-negative rate and 0.7% misjudged-category rate on average without reporting any false-positive edge when the defect rate is 4%.

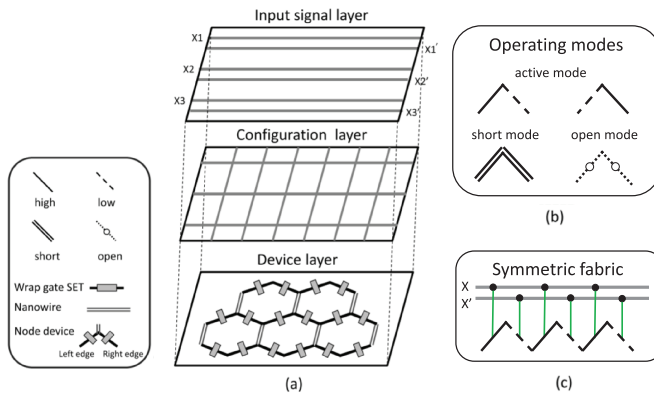


Fig. 1. (a) Physical architecture of a reconfigurable SET array [Chen et al. 2014]. (b) Operating modes. (c) The symmetric fabric architecture for input wiring.

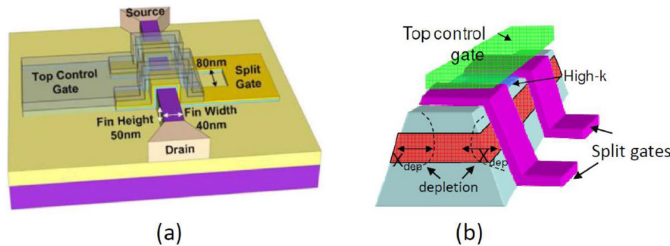


Fig. 2. (a) The structure of a reconfigurable SET device [Liu et al. 2011]. (b) The formulations of wrap-around Schottky split gates and the top control gate [Eachempati et al. 2008].

The rest of the article is organized as follows. Section 2 describes the background of SET arrays. Section 3 presents the proposed diagnosis approach for defective SET arrays. Section 4 shows the experimental results, and Section 5 concludes the work.

## 2 PRELIMINARIES

### 2.1 Reconfigurable SET Array

The structure of the reconfigurable SET array in Chen et al. [2014] can be divided into three layers as shown in Figure 1(a). The three layers are the device layer, the configuration layer, and the input signal layer from the bottom to the top of the structure, respectively. The device layer is composed of the wrap gate SETs to form a hexagonal nanowire network. The configuration layer determines the operation modes of each SET node, which has two edges corresponding to two SETs. The input signal layer is an interface between the input signals and the SET nodes.

Figure 2 [Eachempati et al. 2008; Liu et al. 2011] illustrates the structure of a wrap gate SET in the device layer. A pair of Schottky gates, called split gates, are wrapped around the fin that connects the source and drain, and the top control gate is built upon the split gates. The split gates are connected to the configuration layer such that a voltage bias on the split gates sets the SET nodes in three operation modes: (1) active; (2) open; and (3) short, as shown in Figure 1(b). In the active mode, the voltage bias of the split gates is altered to make the tunneling resistance of the source and drain junctions exceed the resistance quantum. In the open mode, the split gate bias is adjusted to a sufficiently negative value to encroach and pinch off the nanodot island completely from both sides of depletion regions. On the contrary, in the short mode, a large positive split gate

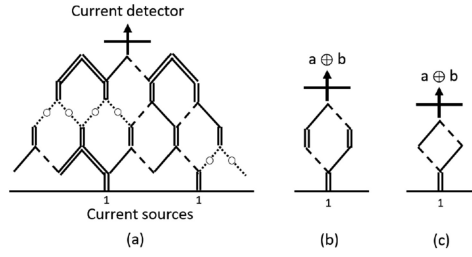


Fig. 3. (a) SET array. (b) Example of  $a \oplus b$ . (c) Simplified diamond-shaped network of  $a \oplus b$  [Chen et al. 2011].

bias is adopted to reduce the tunneling resistance significantly. In other words, the device behaves as a near ohmic conductor.

The top control gate of the wrapped gate SET is connected to the input layer of an SET array. When an SET node is in the active mode, the corresponding input signal controls the dot potential to block or permit electrons tunneling. Due to the limitation of SET array structure, the SET nodes in the same row are controlled by the same input signal.

A reconfigurable SET array can be represented as a hexagonal network as shown in Figure 3(a). At its bottom, there are current sources, represented as 1, while a current detector, which is at the top of it and is considered as the output of the SET array, measures the current coming from the current sources. When the electrons from the current source are detected by the current detector, the output value is 1; otherwise, it is 0.

Each pair of edges, a left-sloping edge and a right-sloping edge, represents an SET node and can be configured as high, low, short, or open. When an SET node is in the active mode, the corresponding pair of edges is (high, low) or (low, high). The current transports through the high edge when the input to the SET node is logic 1, and through the low edge when logic 0. When an SET node is in the short (open) mode, both edges are short (open) to represent electrical short (open). There are connections, which can be configured as short (open) for connection (disconnection), between the current sources and the SET array. Figure 3(b) is an example of an implementation of  $a \oplus b$  on an SET array. The output value will be 1 if the input pattern is either ( $ab = 01$ ) or ( $ab = 10$ ). For the other input patterns ( $ab = 00$  or  $11$ ), the output value will be 0. Figure 3(c) is a simplified version of Figure 3(b) by removing the vertical edge of the hexagons, since they are electrically short. In the rest of this article, only the sloping edges will be shown.

## 2.2 Symmetric Fabric Constraint

The *symmetric fabric constraint* proposed in Eachempati et al. [2008] was widely adopted in the related works for reducing the number of input wires, which are used to configure the node devices. The symmetric fabric constraint confines the configuration of an SET node to be only one of (high, low), (low, high), (short, short), or (open, open) as shown in Figure 1(c). Furthermore, the (high, low) and (low, high) configurations are not allowed to appear in the same row.

In this article, we only use the configuration of (high, low) for a node in the active mode for the ease of discussion since the determination for a node device to be configured either (high, low) or (low, high) is independent of the proposed diagnosis algorithm.

## 2.3 Defect Model

Three types of defects, single-stuck-at-open, double-stuck-at-open, and single-stuck-at-short, which are used in the previous works [Huang et al. 2016; Li et al. 2017], are considered in this work as well. Figure 4 illustrates the representations of the three defect types. If a node device is


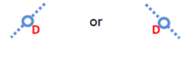






Defect Types	Representation
Single-stuck-at-open: 	 or 
Double-stuck-at-open: 	
Single-stuck-at-short: 	 or 

Fig. 4. The fabric representations of the three types of failures [Huang et al. 2015].

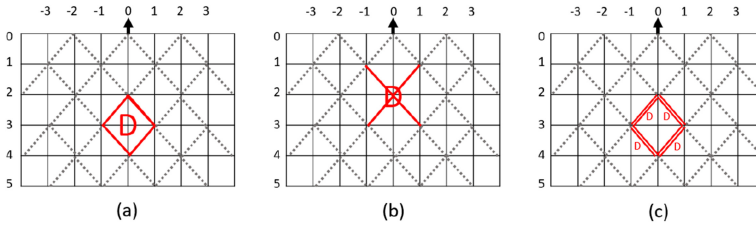


Fig. 5. (a) The first distribution of clustering-defect. (b) The second distribution of clustering-defect. (c) The first distribution of clustering-defect with four D-short edges.

defective, at least one of the edges of the node cannot be changed by a configuration. For example, a double-stuck-at-open node is always open and blocks the current from passing through it regardless of the configuration and the input value. The connections between the SET array and the current sources could be also defective.

In addition to these three types of defects, we also consider a new condition for the defective nodes in an SET array. When a particle, which is larger than an SET node, falls on the SET array, it could cause multiple adjacent nodes defective simultaneously. We name such defective nodes a *clustering-defect*.

The detailed defect model of a clustering-defect is as follows: (1) a clustering-defect consists of four adjacent defective edges, which are distributed as shown in either Figure 5(a) or (b); (2) each edge of a clustering-defect can be either stuck-at-open (D-open) or stuck-at-short (D-short); (3) the defective edges of a clustering-defect do not have to be the same defect-type. Figure 5(c) is an example of a clustering-defect with four D-short under the distribution of Figure 5(a). However, since this clustering-defect in Figure 5(c) will cause a multiple-path conduction, which is not allowed in an SET array due to the property of low transconductance, this defect is considered as a D-open in the analysis.

### 2.4 Assumption of Defect Distribution

Since an SET array is more vulnerable than the CMOS circuits, its defect rate could be as high as 2% = 20,000ppm. The previous works [Huang et al. 2016; Li et al. 2017] considered that the distribution of defects could be quite sparse even with such a high defect rate. Therefore, they adopted two constraints about defect distributions. First, D-open and D-short do not occur on a node device simultaneously. Second, once a node is defective, its six adjacent nodes are defect-free. Figure 6(a) shows an example of defective SET array with the above constraints. The diagnosis algorithm can be easily designed with such constraints. However, the adjacent defective nodes could actually exist in the SET array as mentioned. Once adjacent defects occur on an SET array, the previous diagnosis approaches in [Huang et al. 2016; Li et al. 2017] cannot deal with them. As a result, to

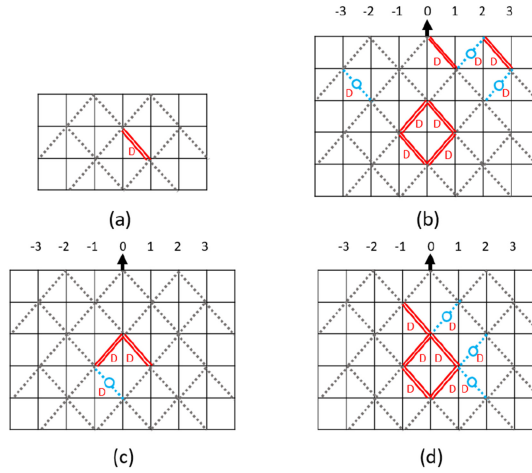


Fig. 6. (a) Defect distribution constraint in Huang et al. [2016] and Li et al. [2017]. (b) Defective SET array without any constraint. (c) Three adjacent defective edges. (d) A “larger” defect involving four adjacent defective edges and a clustering-defect.

enhance the SET diagnosis approach comprehensively, we *remove* these defect distribution constraints in this article. An example of defective SET array in this work is as shown in Figure 6(b), where a clustering-defect and adjacent defective nodes are presented.

In Section 2.3, we assume that a clustering-defect affects exactly four SET edges with two distributions. However, the number of SET edges affected by a particle could be more or fewer than four edges in practice. Since we do not have any constraint about defect distribution in this work, for the case that a defect involves with fewer than four defective edges, this defect can be represented by two or three adjacent defective edges. On the other hand, if there exist more than four adjacent defective edges for a “larger” defect, it can be formed by adjacent multiple defective edges or adjacent clustering-defects. Figure 6(c) and (d) illustrate the examples of three adjacent defective edges and eight adjacent defective edges, respectively.

### 3 DIAGNOSIS APPROACH

#### 3.1 Overview

Since the functionality of a reconfigurable SET array is represented by both configurations and input patterns, the diagnosis approaches for traditional Boolean circuits in Huang and Cheng [1999], Kautz [1968], Liaw et al. [1990], and Veneris and Hajj [1999] are not applicable for the SET array diagnosis. In this article, a node consisting of a pair of edges can be configured as one of (high, low), (short, short), or (open, open). The *diagnosis cost* is the amount of the configurations and input patterns [Li et al. 2017] since they are strongly correlated with the time spent for the diagnosis procedure. Therefore, our approach will determine a diagnosis sequence to minimize the cost.

The problem formulation of this work is as follows: Given an  $H$  (height)  $\times$   $W$  (width) defective SET array, our goal is to identify the locations and the types of defects such that the diagnosis cost is minimized.

The proposed diagnosis approach contains two stages:

- (1) *Vertical path diagnosis.*
- (2) *Horizontal path diagnosis.*



We will elaborate the details of them in Sections 3.2 and 3.3, respectively.

Before elaborating the details of these two stages, we summarize three ideas we used in defect identification.

- (1) *For an edge involved in a conducting path with a configuration from the current source to the current detector, we change the configurations of the corresponding node to a new configuration of (open, open). If the path becomes nonconductive, the edge is defect-free since it can be successfully configured as short and open. Otherwise, the edge is marked as a D-short candidate.*
- (2) *When a D-short candidate is not adjacent to any other D-short candidate, the D-short candidate edge is confirmed as a D-short edge since there is no detoured path to bypass the current from this D-short candidate.*
- (3) *An edge can be confirmed as a D-open edge if and only if all possible paths involving the edge are not conductive. To reduce the computational complexity, we label an edge as a D-open edge when it is not involved in any conductive path of our diagnosis procedure.*

We use an example in Figure 7 to explain these ideas in defect identification. Figure 7(a) is a defect map of an SET array, and the locations as well as types of defects are unknown to the diagnosis algorithm. All the SET node devices are set to (open, open) in the beginning. According to our first idea, we heuristically search for a conducting path by both configuring SET nodes and applying an input pattern. Once a conducting path is found with the corresponding configuration and input pattern as shown in Figure 7(b), we *reconfigure* one node of this path as (open, open) individually as shown in Figures 7(c)–(g). This reconfiguration of (open, open) to an edge in the conducting path is to examine whether this edge is actually involved in the conducting path or not. If the path becomes nonconductive after this reconfiguration, the edge must be involved in the original conducting path such that it is not a D-open edge. Furthermore, the edge is not a D-short edge either since it can be reconfigured to cause the path to become nonconductive. As a result, the edge is defect-free due to the absence of D-open and D-short defects. For example, Figure 7(b) is a conducting path we found. The path becomes nonconductive (output is 0) after reconfiguring the connection at the bottom of SET array as open as shown in Figure 7(c). Therefore, this connection is defect-free (represented in green). Next, the left edge of the node at (1, 5) is identified as defect-free in the same way as shown in Figure 7(d). In Figure 7(e), the right edge of the node at (0, 4) is identified as D-short candidate (represented in yellow) since the path is still conductive after reconfiguring as (open, open). The result of this conducting path after diagnosing all the edges is shown in Figure 7(g).

However, if there exist defects blocking the current of a path, we cannot diagnose the path. Therefore, the main concept of our approach is to configure a conducting path including an edge that needs to be diagnosed. If the configured path is nonconductive, we use the identified defect-free edges or D-short candidates to configure another conducting path for diagnosing the part of the nonconducting path. Figure 7(h) is the result of the vertical path diagnosis in the first stage. All the edges in this SET array have been identified as defect-free, D-short, or D-open candidates. Then, we exploit this information to configure other horizontal paths and diagnose more edges in the horizontal path diagnosis. The final result of the first stage is shown in Figure 7(i).

In the second diagnosis stage, we configure other paths to confirm the behavior of adjacent D-short candidates. According to our second idea, we can confirm a D-short candidate as a D-short edge when there are no other adjacent D-short candidates. This is because the path is still conductive after the corresponding node of the edge is reconfigured as (open, open), and there is no other possible path detouring the current. For example, the right edge of the node at (0, 4) in Figure 7(i) can be confirmed as a D-short edge. On the other hand, since both edges of the node

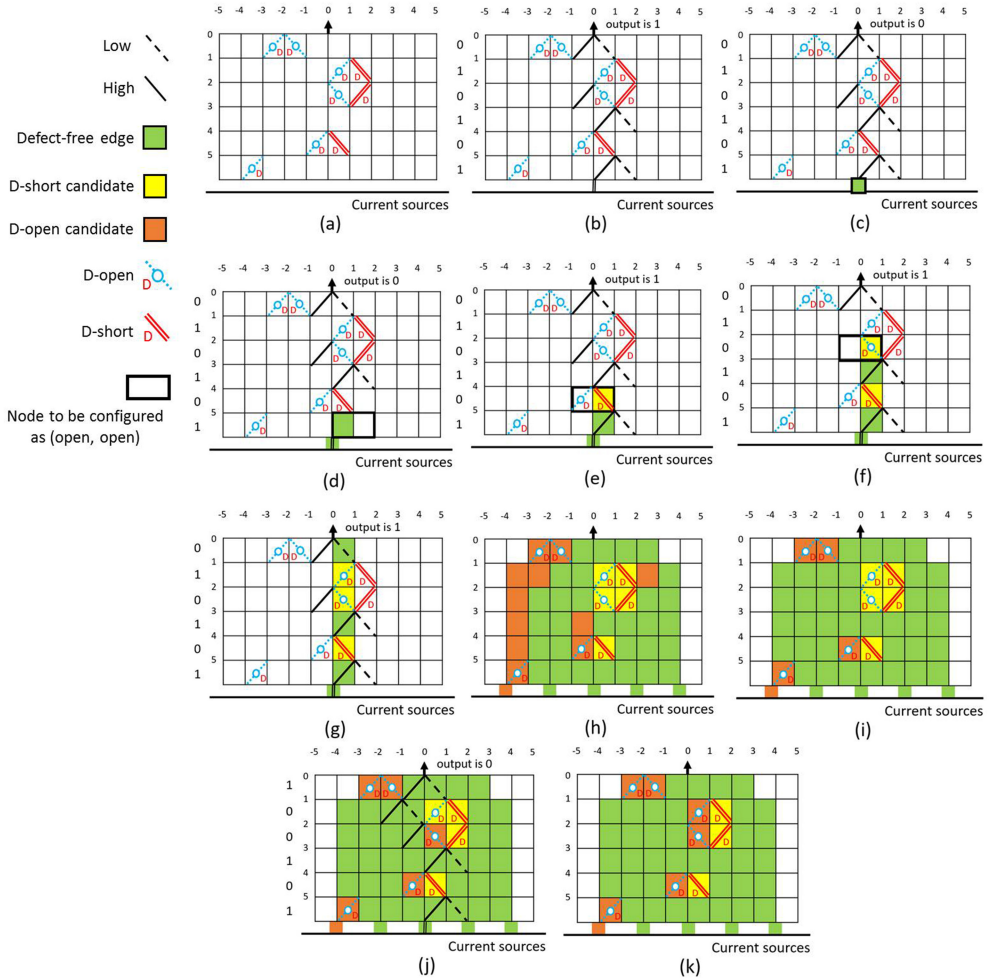


Fig. 7. Example for the three ideas of identifying defects.

at (1, 1), the right edge of the node at (0, 2), and the left edge of the node at (2, 2) are D-short candidates and clustered together, additional paths are needed for diagnosing them. When the right edge of the node at (0, 2), which is actually a D-open, is under diagnosed, the current is blocked from passing through the edge as shown in Figure 7(f). However, the current passes to the current detector through the left edge of the node at (2, 2) and the right edge of the node at (1, 1) since they are D-short edges. Therefore, the right edge of the node at (0, 2) is labeled as a D-short candidate in Figure 7(f). Nevertheless, in Figure 7(j), another path will be configured to diagnose the right edge of the node at (0, 2), and it is nonconductive. We can identify this edge as a D-open edge since other edges in the configured path have been identified as defect-free or D-short edges, and this edge is the only one that can block the current. The left edge of the node at (1, 1) can be identified as a D-open in the same way. The final diagnosis result for this defective SET array is shown in Figure 7(k).

The boundary nodes, which have only one edge, are considered as *useless nodes* since a complete SET node consists of a pair of edges. The useless nodes cannot be used in synthesis and will be



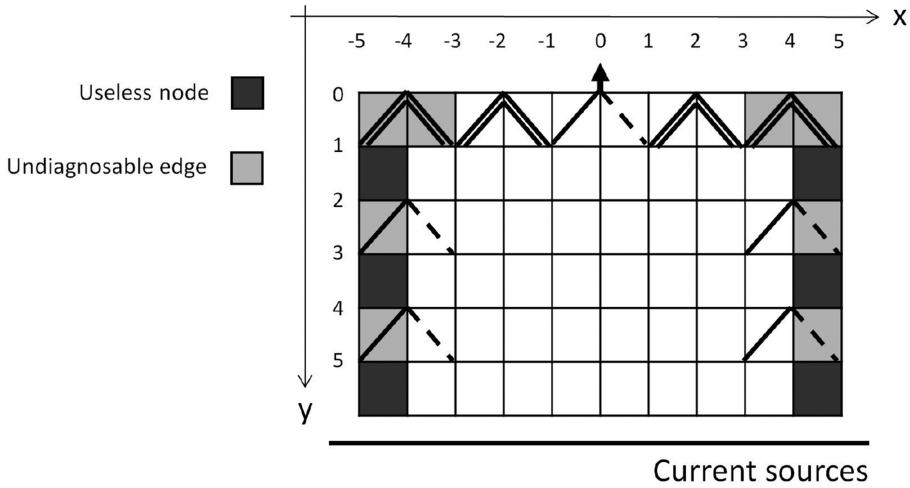


Fig. 8. Example of undiagnosable edges and useless nodes in an SET array.

discarded. Furthermore, there are some edges that cannot be involved in any conducting path since there is no complete node below them to build a conducting path. We name such edges *undiagnosable edges*. Figure 8 illustrates an example for useless nodes and undiagnosable edges. The undiagnosable edges, represented in gray, are unable to be included in any conducting path since they connect to useless nodes, which are represented in black. The undiagnosable edges cannot be used in synthesis as well.

### 3.2 Vertical Path Diagnosis

**3.2.1 Overall Algorithm of Vertical Path Diagnosis.** The overall algorithm of vertical path diagnosis is shown in Algorithm 1. In the vertical path diagnosis, all the edges of an SET array, except the useless nodes and the undiagnosable edges, will be initialized as undiagnosed edges and later included in a path for diagnosis. We build one path at a time from the leftmost column to the rightmost column of the SET array. The edges in the same column are used to build a vertical path, and the nodes at row of  $y = 0$  are configured as (short, short) and (high, low) for connecting the vertical paths to the current detector. When building a new path, some nodes that have been already configured as (high, low) or (short, short) for the last path can be reused in the present path without any additional overhead. If a configured path is conductive, we mark the path as a conducting path and individually reconfigure each node of the path as (open, open) to examine the corresponding edge as shown in lines 4–5 in Algorithm 1; otherwise, we mark the path as a nonconducting path, label each unknown edge of this path as D-open candidates (represented in orange), and adjust the previous path, if it is conductive, to further diagnose the current path as shown in lines 6–8 in Algorithm 1.

**3.2.2 Vertical Path Diagnosis Demonstration.** We use the same defective SET array as shown in Section 3.1 to demonstrate the process of vertical path diagnosis. Figure 9(a) shows that path ① consists of the edges at the first column, and the node at  $(-2, 0)$  and the node at  $(0, 0)$  are configured as (short, short) and (high, low), respectively, to connect path ① to the current detector of the SET array. Since both edges of the node at  $(-2, 0)$  are D-open edges, path ① is nonconductive.

Therefore, path ① is marked as a nonconducting path (path index ① in orange), and the edges of this path are labeled as D-open candidates. Figure 9(b) shows path ②, and we observed that some

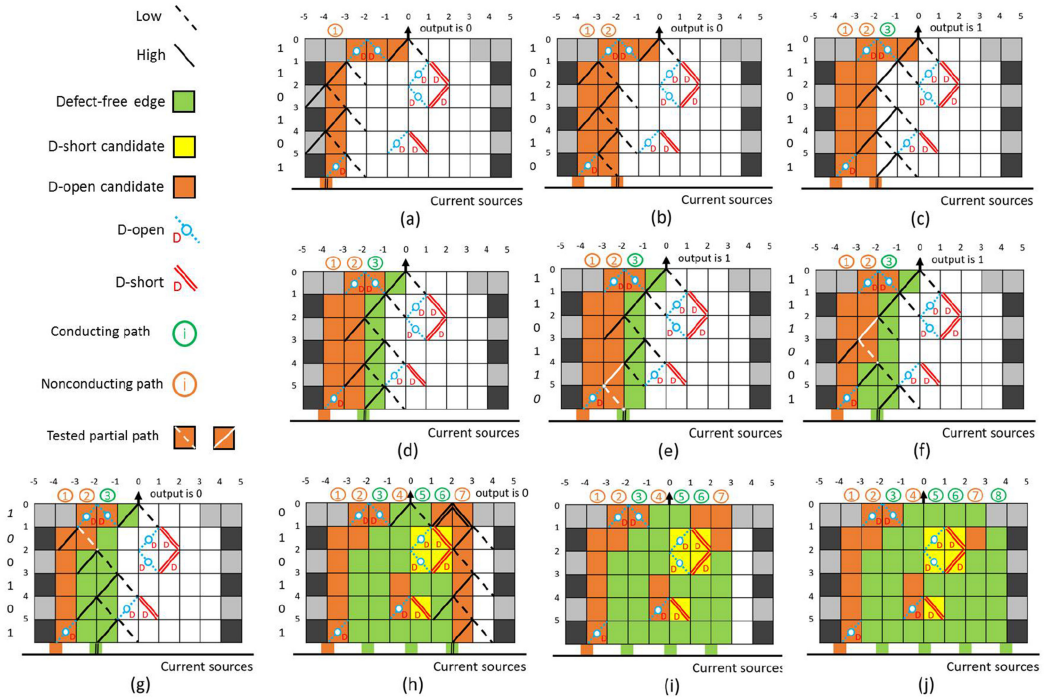


Fig. 9. Example for demonstrating the ideas of vertical path diagnosis.

nodes included in path ② have been configured as (high, low) for path ① such that we can reuse them without any reconfiguration. The remaining nodes of path ① that are not included in path ② are reconfigured as (open, open). We then assign the corresponding input patterns to stimulate path ②, which is also a nonconducting path. The third path is built in the same way, but it is a conducting path. Thus, we mark this path a conducting path (path index ③ in green) as shown in Figure 9(c). Then, we diagnose all the edges of this path and label them as shown in Figure 9(d).

We observed that some defect-free or D-short edges of nonconducting paths are labeled as D-open candidates, which is incorrect. The edges with these incorrect labels could be further re-labeled when they are included in a conducting path. As a result, during the vertical path diagnosis, we will deliberately adjust the conducting paths for diagnosing a portion of a nonconducting path. Specifically, when the present path (previous path) is marked as a nonconducting path, we will check if the previous path (present path) is a conducting path. For the nonconducting paths that are not adjacent to any conducting path, we will handle them in the stage of horizontal path diagnosis, which will be discussed in Section 3.3. A conducting path will serve as a baseline to build a new path containing a part of the nonconducting path, called *tested partial path*. If the new path is still conductive, we can diagnose the edges in the original nonconducting path and update the labels on them. However, since the locations of the defects in the nonconducting path are unknown, we use the concept of Binary Search to shrink the range of defect locations.

For example, when all the edges of path ③ have been diagnosed, we serve path ③ as the baseline to diagnose path ② since path ② is a nonconducting path. Since the top D-open candidate and the bottom D-open candidate of path ② are at (-2, 0) and (-3, 5), respectively, we set the row of  $y = 0$  as the top row and the row of  $y = 5$  as the bottom row. Furthermore, we set the row of  $y = 3$  as the middle row dividing path ② into two halves. However, if the edge of path ② at  $y = 3$

**ALGORITHM 1:** Vertical path diagnosis

---

```

Input: an  $n \times m$  SET array. // height  $n$ , width  $m$ 
Output: a vertical-diagnosed SET array map.
1 Initialize all the SET edges as undiagnosed edges.
  for  $i = 1: m$  //start from the leftmost vertical path to the rightmost one
2   | Configure a vertical path connecting current sources to the current detector.
3   | Record the configuration of path  $i$ .
   | if path  $i$  is conductive
4   | | Label path  $i$  as a conducting path.
5   | | Diagnose each edge by defect identification idea (1). // as mentioned in Section 3.1
   | end
   | else
6   | | Label path  $i$  as a nonconducting path.
7   | | Label un-diagnosed edges of path  $i$  as D-open candidates.
   | | if  $i \neq 1$  and path  $i-1$  is conductive
8   | | | adjust path  $i-1$  to test D-open candidates of path  $i$ .
   | | end
   | end
  endfor

```

---

cannot connect to path ③, we change the middle row to the lower one, i.e.,  $y = 4$ . We consider the second half of path ②, i.e., the left edge of the node at  $(-2, 4)$  and the right edge of the node at  $(-3, 5)$ , as the tested partial path (represented by the white line) currently. Next, we configure a new path consisting of this tested partial path and the edges of path ③ that are above the middle row. Since this new path is conductive (output is 1) as shown in Figure 9(e), the edges in the tested partial path can be identified as defect-free edges or D-short candidates after reconfiguring the corresponding nodes as (open, open). As we discussed in Section 3.1, if the new path becomes nonconductive after the reconfiguration of (open, open) for an edge in the tested partial path, this edge is re-labeled as a defect-free edge; otherwise, this edge is re-labeled as a D-short candidate.

Next, the bottom row is set to the row of  $y = 4$ , and the middle row is set to the row of  $y = 2$ . A new path is built in the same way as shown in Figure 9(f). Since the new path is also a conducting path, the tested partial path can be identified and re-labeled. Last, the bottom row is set to the row of  $y = 2$ , and the middle row is set to the row of  $y = 1$ . Since the edge in the tested partial path of path ② is not capable of building a complete path with path ③, we include the edges between the top row ( $y = 0$ ) and the bottom row ( $y = 2$ ) of path ② into the tested partial path and build a new path as shown in Figure 9(g). However, this path is a nonconducting path, and we cannot diagnose the D-open candidates in the tested partial path. Furthermore, since the remaining D-open candidates in path ② have been included in the tested partial path, we stop diagnosing path ② and proceed to the next vertical path, path ④. Figure 9(h) shows a special case of a nonconducting path, path ⑦, consisting of defect-free edges only. However, both edges of the node at  $(2, 0)$  and the left edge of the node at  $(3, 1)$  of the configured path cause a multiple-path conduction with a D-short edge of the node at  $(1, 1)$ . As mentioned in Section 2.3, if a path is involved in a multiple-path conduction, it is a nonconducting path.

As a result, path ⑦ is marked as a nonconducting path. However, path ⑦ can be diagnosed with the assistance of path ⑥, which is a conducting path as shown in Figure 9(i). The final result of the vertical path diagnosis is shown in Figure 9(j). The green grids represent defect-free edges, yellow grids represent D-short candidates, and orange grids represent D-open candidates.

### 3.3 Horizontal Path Diagnosis

**3.3.1 Overall Algorithm of Horizontal Path Diagnosis.** When the vertical path diagnosis is finished, some defect-free edges might be still diagnosed as D-open candidates. This is because the current passing through these defect-free edges was unexpectedly blocked by other defects. Therefore, in the horizontal path diagnosis, we will build the conducting paths horizontally to configure new paths containing mislabeled edges. The newly configured path is composed of *edges* from the conducting paths and the *defect candidates* in the nonconducting paths. Once a newly configured path is conductive, we can further correct the labels on D-open candidates in this path. In the following paragraphs, we will introduce the details of the horizontal path diagnosis.

The overall algorithm of horizontal path diagnosis is shown in Algorithm 2. First, we build up a *present diagnosis column*, which contains adjacent nonconducting paths we are dealing with. For example, in Figure 10(a), path ① and path ② will be included in a diagnosis column. If the present diagnosis column contains all the paths in the SET array, this SET array is a useless array. If there exists conducting paths adjacent to the present diagnosis column, the conducting path(s) will serve as baseline(s) for diagnosing the present diagnosis column. We divide the horizontal path diagnosis into two cases with respect to the number of baselines adjacent to the present diagnosis column: Boundary case is with one baseline, and normal case is with two baselines. The algorithms dealing with the boundary case and normal case are shown in lines 4–12 and lines 13–15 of Algorithm 2, and will be explained in Section 3.3.2 and Section 3.3.3, respectively.

**3.3.2 Boundary Case.** For the boundary case, the present diagnosis column has only one baseline adjacent to it. The baseline will be used to build a new path containing a tested partial path, represented in white. The tested partial path consists of the present diagnosis column and two edges called *bridges*. The edges in the bridges are configured as (short, short).

For example, the present diagnosis column composed of paths ① and ② is a boundary case as shown in Figure 10(a). Path ③ serves as the baseline for this present diagnosis column. In the beginning, we randomly select two rows at  $y = 2$  and  $y = \text{current sources}$  as the bridges. The row at  $y = 2$  is connected to the nonconducting path, path ①, with the baseline, path ③, and the other bridge directly connected to the current sources. Then, the edges over the upper bridge in path ③ connected to the current detector. Next, we configure the edges between the two bridges in path ①. Finally, the edges below the lower bridge in path ③ are configured to connect with the current sources. However, since the lower bridge is at the current sources, there is no edge needed to be configured. Since the new path is nonconductive as shown in Figure 10(b), we randomly select another two rows as the bridges to build a new path in the same way. In our algorithm, we set a parameter  $p$  to limit the number of trails to build the first conducting path for a boundary case. If the number of configured paths for building the first conducting path is larger than  $p$ , we terminate the diagnosis procedure for the present diagnosis column. In this example, we configure the new bridge at the rows of  $y = 1$  and  $y = 3$ , and the new path is built in the same way as shown in Figure 10(c). Since this path is a conducting path, we can successfully identify the edges in the tested partial path to correct their labels by reconfiguring the corresponding nodes as (open, open). However, if both edges of a node are included in the tested partial path and labeled as D-open candidates, it needs additional configurations and input patterns to identify its edges and correct the labels. For example, in Figure 10(c), both edges of the node at  $(-3, 1)$  are D-open candidates and involved in the tested partial path; we cannot identify whether each of these two edges is a defect-free edge or D-short candidate by just reconfiguring the node as (open, open). Therefore, we reconfigure this node as (high, low) and assign an input 0 to identify the left edge of this node as shown in Figure 10(d). Since the path becomes nonconductive after the reconfiguration



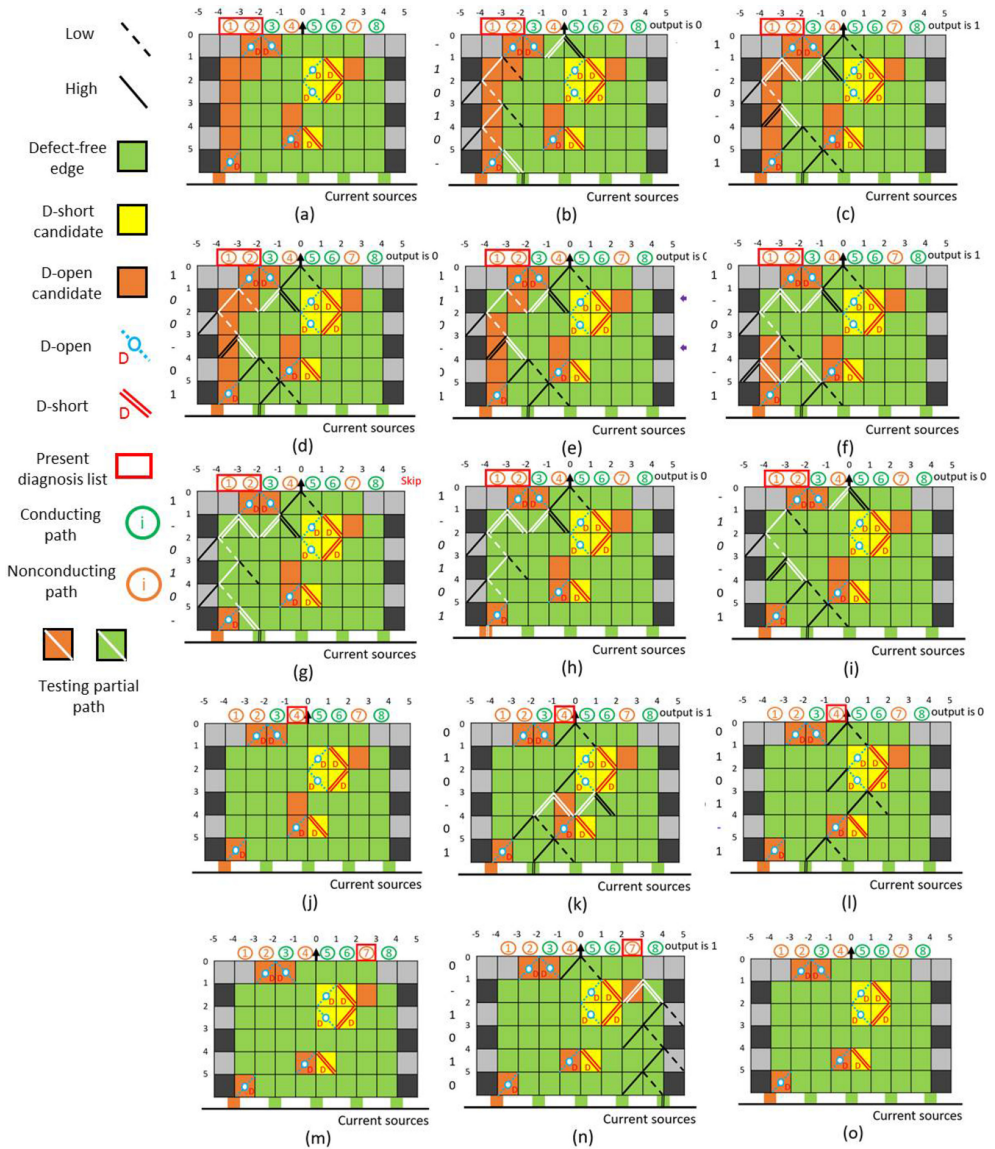


Fig. 10. Example for demonstrating the ideas of horizontal path diagnosis.

with the input 0, the left edge is a defect-free edge.<sup>1</sup> Next, we flip the input to identify the right edge, and this edge is a defect-free edge since the path also becomes nonconductive as shown in Figure 10(e).

After having the first conducting path for a boundary case, we shift one of the two bridges to another row that contains defect candidates in the present diagnosis column to build a new path.

<sup>1</sup>The input 0 (1) allows the current passing through the right (left) edge and blocks the current from passing through the left (right) edge of a node configured as (high, low). Therefore, if both edges are involved in a conducting path, we can reconfigure this node as (high, low) and assign an input 0 (1) to identify the left (right) edge without changing the behavior of the other edge.

**ALGORITHM 2:** Horizontal path diagnosis

---

**Input:** an  $n \times m$  vertical-diagnosed SET array map and a trial count parameter  $p$ // a vertical-diagnosed array map and a user-defined parameter  $p$  for trial count constraint.

**Output:** a diagnosed SET array map.

- 1 Build up diagnosis columns. // diagnosis column consists of adjacent nonconducting paths
- for** each diagnosis column  $i$ 
  - 2 Find the nearest left conducting path  $a$  to the diagnosis column  $i$ .
  - 3 Find the nearest right conducting path  $b$  to the diagnosis column  $i$ .
  - if** only one of path  $a$  and path  $b$  exists // boundary case
    - 4 Initialize trail count  $t = 0$ .
    - 5 Set the conducting path  $a$  (or path  $b$ ) as the baseline.
    - while**  $t < p$ 
      - 6 Randomly select two rows as short bridges.
      - 7 Configure a new path consisting of the two short bridges, the baseline, and the testing partial path.
      - if** configured path is nonconductive
        - 8  $t = t + 1$ .
      - end**
      - else**
        - 9 Diagnose defect candidates of the configured path.
        - while** there exist untested defect candidates of the diagnosis column  $i$ 
          - 10 Shift one short bridge to cover new defect candidates.
          - if** the newly configured path is conductive
            - 11 Diagnose defect candidate of the configured path.
          - end**
        - end**
      - 12 Finish the diagnosis procedure of the diagnosis column  $i$ . // Leave while loop
      - end**
    - end**
  - end**
  - else if** both path  $a$  and path  $b$  exist // normal case
    - 13 **for** each D-open candidate  $j$  in the diagnosis column  $i$ 
      - 14 Select the row of D-open candidate  $j$  as a short bridge.
      - 14 Configure a new path with partial path  $a$ , path  $b$ , and the short bridge.
      - if** the configured path is conductive
        - 15 Diagnose D-open candidate  $j$  by defect identification idea (1).
      - end**
    - endfor**
  - end**
- 16 Set all the D-short candidates as D-short defects.
- 17 Set all the D-open candidates as D-open defects.

---

In this example, we shift the row of  $y = 3$  to the row  $y = 4$  as a new bridge, and the other bridge is still at the row of  $y = 1$ . Then, we build a new path with these two bridges in the same way as shown in Figure 10(f). The tested partial path of this new path contains D-open candidates. Since this path is also conductive, the D-open candidates in the tested partial path can be identified. Next, we shift the bridge in the row of  $y = 4$  to  $y = 5$  as a new path and build the next path as shown in



Figure 10(g). However, since the tested partial path contains no defect candidate, we skip this path without any configuration. In Figure 10(h), we build a new path with the new bridge at current sources. Since this path is nonconductive, the edge in the tested partial path cannot be identified. Finally, we select a new combination of rows at  $y = 0$  and  $y = 3$  to construct the new bridges. The new path built with these two bridges is also a nonconducting path as shown in Figure 10(i). Since all the rows containing defect candidates in the present diagnosis column have been selected for constructing the bridges, we terminate the diagnosis procedure for the present diagnosis column and move to the next diagnosis column.

**3.3.3 Normal Case.** For the normal case, the present diagnosis column has two baselines. Therefore, we can configure a bridge for connecting with these two baselines to build a new path. The bridge needs to contain at least a defect candidate in the present diagnosis column.

For example, the present diagnosis column in Figure 10(j) consists of path ④ with two baselines, paths ③ and ⑤. In the beginning, we select the row of  $y = 3$  to configure a bridge since this row contains a D-open candidate in the present diagnosis column. The nodes in the bridge are configured as (short, short). Next, since path ⑤ is closer to the current detector than path ③, we configure the nodes above the bridge in path ⑤ to connect the current detector with the bridge. Then, we configure the nodes below the bridge in path ③ to connect the bridge with the current sources. The new path is a conducting path, and the involved edges in the bridge are considered as the tested partial path, represented in white, as shown in Figure 10(k). Therefore, the D-open candidate of the node at  $(-1, 3)$  can be identified as a defect-free edge, and its label can be corrected. Next, we reconstruct a new bridge at the row of  $y = 4$  that also contains a D-open candidate and build another path in the same way. However, this path is nonconductive as shown in Figure 10(j), and the D-open candidate cannot be identified. Finally, we diagnose the last diagnosis column composed of path ⑦ with two baselines, path ⑥ and path ⑧, as shown in Figure 10(k). We build a new path with a bridge at the row of  $y = 1$  in the same way as shown in Figure 10(n). Since this path is a conducting path, the D-open candidate in the tested partial path can be identified as a defect-free edge.

The diagnosis procedure of horizontal path diagnosis is finished since all the diagnosis columns in this example have been diagnosed. The final result of the horizontal path diagnosis is shown in Figure 10(o). The green grids represent defect-free edges, yellow grids represent D-short candidates, and orange grids represent D-open candidates.

### 3.4 Time Complexity Analysis

The complexity analyses of vertical path diagnosis and horizontal path diagnosis are as follows:

Given an  $n \times m$  SET array

- (1) Time complexity analysis of vertical path diagnosis:
  - (a) Configuring a vertical path  $i$  and testing if it is conductive need  $(n)$  configurations and (1) pattern.
  - (b) If the path  $i$  is conductive:
    - i. We need  $(2n)$  configurations and  $(n)$  patterns to test each edge.
  - (c) If  $i \neq 1$  and the path  $i$  is nonconductive:
    - i. If  $i \neq 1$  and the path  $i-1$  is conductive, we can use the concept of binary search and adjust path  $i-1$  to test path  $i$ . Therefore, we need  $(n)$  configurations to reconfigure for testing edges of path  $i$ , and  $(\frac{1}{2}n)$  patterns to test if these paths are conductive.
    - ii. Once a newly configured path is conductive, we need (2) configurations and (1) pattern to test if an edge is defect-free.

- iii. The worst case is that every newly configured path is conductive. Therefore, we need  $(2n)$  configurations and  $(n)$  patterns in total.

According to the analysis, we need  $(3n)$  configurations and  $(\frac{3}{2}n)$  patterns to test a vertical path in the worst case. Therefore, we need at most  $(m) \times (3n)$  configurations and  $(m) \times (\frac{3}{2}n)$  patterns to finish the vertical path diagnosis.

- (2) Time complexity analysis of horizontal path diagnosis:
  - (a) If the diagnosis column  $i$  is the boundary case:
    - i. If the baseline path is a boundary path, we need  $(n-3)$  configurations to configure the three partial paths and  $(2m)$  configurations to configure two bridges.
    - ii. The worst case is that we conduct  $t$  trials to find the conducting paths. Therefore, we need  $(t) \times (2m+n-3)$  configurations and  $(t)$  patterns.
    - iii. Furthermore, if there exists a D-open candidate in each row of column  $i$ , we also need  $(n) \times (2m+2m)$  configurations and  $(n) \times (m)$  patterns to shift the bridges and test each D-open candidate of column  $i$ .
    - iv. We need  $[(t) \times (2m+n-3) + (n) \times (4m)]$  configurations and  $[(t) + (n) \times (m)]$  patterns in total in the worst case.
  - (b) If the diagnosis column  $i$  is the normal case:
    - i. The worst case is that the nearest left conducting path  $a$  is a left boundary path and the right conducting path  $b$  is a right boundary path. We need  $(n-1)$  configurations to configure two partial paths,  $(m)$  configurations to configure the short bridge, and  $(1)$  pattern to test if the path is conductive.
    - ii. If the newly configured path is conductive, we need at most  $(2m)$  configurations and  $(m)$  patterns to test the D-open candidates on the bridges.
    - iii. The worst case is that there exists a D-open candidate in each row of column  $i$ . We need  $(n) \times (2m+2m)$  configurations and  $(n) \times (m)$  patterns to shift the bridges and test each D-open candidate of column  $i$ .
    - iv. We need  $[m + (n-1) + n \times (4m)]$  configurations and  $[(1) + (n) \times (m)]$  patterns in total in the worst case.

According to the analysis, the boundary case needs more configurations and patterns to test a diagnosis column. There will be at most  $(\frac{1}{2}m)$  diagnosis columns; therefore, we need at most  $(\frac{1}{2}m) \times [(t) \times (2m+n-3) + (n) \times (4m)]$  configurations and  $(\frac{1}{2}m) \times [(t) + (n) \times (m)]$  patterns to finish the horizontal path diagnosis.

In summary, we need at most  $(m) \times (3n) + (\frac{1}{2}m) \times [(t) \times (2m+n-3) + (n) \times (4m)]$  configurations and  $(m) \times (\frac{3}{2}n) + (\frac{1}{2}m) \times [(t) + (n) \times (m)]$  patterns, i.e.,  $O(m^2t + mtn + m^2n)$  configurations and  $O(mt + nm^2)$  patterns in the worst case for the proposed diagnosis algorithm. If we assume that  $m \approx n$  and  $t = C_2^n \approx n^2$ , we will need  $O(n^4)$  configurations and  $O(n^3)$  patterns for the worst case.

However, we know that the time complexity is for the worst cases. From the experimental results, the required CPU time for running our algorithm is not much. On the other hand, from the physical implementation suggestion in the previous work [Ho et al. 2016], the SET array size is limited to  $10 \times 50$ . Thus, the high time complexity of the proposed diagnosis algorithm is not a concern in practice.

## 4 EXPERIMENTAL RESULTS

### 4.1 Environmental Setting of Experiments

The proposed algorithm was implemented in C++ language and the experiments were conducted on an Intel Xeon® E5-2643 CentOS 5.11 platform with 128-GBytes memory. We conducted two

Table 1. The Model Comparison between the Proposed Diagnosis Approach and the State-of-the-Art [Li et al. 2017]

	Our model	[Li et al. 2017]
Defect model	Allow adjacent defects	Not allow adjacent defects
Defect types	D-open/D-short/Clustering-defect	D-open/D-short

experiments in this article with different sizes of defective SET arrays. We individually applied the algorithm in Li et al. [2017] and ours for each defective SET array we generated.

Note that, based on the original model in Li et al. [2017], without adjacent defects, its algorithm could always find the new conducting path based on a baseline. However, the program in Li et al. [2017] would be trapped into an infinite loop when diagnosing some defective SET arrays with adjacent defects. Therefore, we have to re-implement the program based on the algorithm in Li et al. [2017], and terminate the diagnosis process when it cannot find a new conducting path.

To show the effectiveness of the proposed algorithm, we inject defects into the SET arrays for the experiments. The experimental environment is software-based with computer simulation. Hence, the experiments can be considered as a logical model at an algorithmic level. From the related study [Ho et al. 2016], the number of inputs in an SET array is limited to the height constraint of SET arrays. The height constraint was suggested as 10. Therefore, we conducted the experiments for SET arrays with different sizes from  $10 \times 10$  to  $50 \times 50$ .

The first experiment shows the number of useless defective SET arrays our algorithm identified, and that of SET arrays having current detectors blocked by D-open edges or a multiple-path conduction. The second experiment demonstrates the effectiveness and the efficiency of our algorithm. In the experiments, we set the defect rate as 2%, 3%, and 4% for showing the impact of defect rates on the diagnosis results. The defect rate is calculated as  $|\text{defective edge}| / |\text{total edge}|$ . First, we calculated the total number of defective edges,  $|\text{defective edge}|$ , of a defective SET array according to the defect rate and its size. Then, we randomly selected SET nodes to be defective nodes and the number of these nodes were equal to the  $|\text{defective edge}|$ . The defective nodes were randomly determined as *single-stuck-at-open*, *double-stuck-at-open*, *single-stuck-at-short*, or *clustering-defect*. If a node was determined as clustering-defect, two possible combinations of its two neighboring edges will be selected as defective edges. For example, assume that in Figure 5, the node at (0, 2) is set as a clustering-defect. The first combination is to select the left edge of the lower left neighboring node and the right edge of the lower right neighboring node to form a clustering-defect as shown in Figure 5(a). The other combination is to select the right edge of the upper left neighboring node and the left edge of the upper right neighboring node to form a clustering-defect as shown in Figure 5(b). Each one of the four defective edges in this clustering-defect will be randomly determined as D-open or D-short.

Before we explain the experimental results, we compare the defect model and the defect type with the state-of-the-art [Li et al. 2017]. The comparison results are summarized in Table 1. Locations of defects in both works were selected randomly. Adjacent defects were not allowed in the defective SET array in Li et al. [2017] due to its assumption. However, the adjacent defects were allowed in our model. It means that our model can diagnose a defective SET array despite the defect distribution. Furthermore, we added the clustering-defect as one of the defect types. Once an SET node is selected as having a clustering-defect in an SET array, it will have four adjacent defective edges as we discussed in Section 2.3.

## 4.2 Experimental Results of Identifying Useless SET Arrays

In the first experiment, we randomly generated a defective SET array and performed our algorithm. We terminated the algorithm after the vertical path diagnosis stage, and reported if the SET array

Table 2. The Experimental Results of Identifying Useless SET Arrays

H	W	Def.(%)	Useless	Blocked	H	W	Def.(%)	Useless	Blocked			
10	10	2	0	0	20	40	2	2	2			
		3	0	0			3	2	2			
		4	0	0			4	2	2			
	15	2	0	0		50	2	0	0			
		3	0	0			3	0	0			
		4	0	0			4	0	0			
	20	2	0	0		25	25	2	0	0		
		3	0	0				3	0	0		
		4	0	0				4	1	1		
	30	2	0	0		30	2	0	0			
		3	0	0			3	0	0			
		4	0	0			4	0	0			
	40	2	0	0		40	2	0	0			
		3	1	1			3	0	0			
		4	1	1			4	0	0			
	50	2	0	0		50	2	0	0			
		3	0	0			3	0	0			
		4	1	1			4	0	0			
	15	15	2	0		0	30	30	2	1	1	
			3	0		0			3	2	2	
			4	0		0			4	2	2	
		20	2	0		0		40	2	1	1	
			3	0		0			3	1	1	
			4	0		0			4	1	1	
30		2	1	1	50	2		0	0			
		3	1	1		3		0	0			
		4	1	1		4		0	0			
40		2	0	0	40	40		2	2	2		
		3	0	0				3	3	3		
		4	0	0				4	3	3		
50		2	0	0	50	2		0	0			
		3	2	2		3		0	0			
		4	3	3		4		0	0			
20		20	2	2	2	50		50	2	0	0	
			3	2	2				3	0	0	
			4	2	2				4	3	3	
		30	2	1	1							
			3	1	1							
			4	2	2							

is useless. Then, we examined each defect map to see if its current detector was blocked. We conducted 100 runs of experiments and the results are summarized in Table 2. In Table 2, Columns 1 and 2 list the dimension of the defective SET arrays. Column 3 lists the different defect (Def.) rates of the SET arrays. Column 4 lists the number of useless SET arrays reported by our algorithm.

Table 3. Experimental Results of the Proposed Diagnosis Approach and the State-of-the-Art [Li et al. 2017] with Defect Rate 2%

H	W	Re. Def.	Ours					[Li et al. 2017]				
			F-neg.(%)	Misjud.(%)	[Config.]	[Pat.]	[F-pos. edg]	F-neg.(%)	Misjud.(%)	[Config.]	[Pat.]	[F-pos. edg]
10	10	3.0	0.6	0.3	290.7	91.3	0	5.0	0.5	440.0	161.5	3
	15	2.2	0.5	0.2	440.9	134.6	0	5.3	0.3	822.2	280.0	4
	20	3.2	0.9	0.3	638.4	205.4	0	12.0	0.5	1,100.7	370.5	6
	30	3.0	0.9	0.2	975.8	316.2	0	17.5	0.4	1,592.3	489.7	20
	40	2.9	1.7	0.2	1,344.7	429.5	0	23.2	0.4	2,098.8	606.5	17
	50	2.8	1.3	0.2	1,703.8	542.0	0	26.5	0.4	2,562.9	703.9	15
20	20	3.2	0.8	0.3	1,364.8	415.2	0	10.6	0.6	3,891.6	786.9	7
	30	3.1	0.9	0.3	2,261.7	642.1	0	13.0	0.5	6,097.5	1,153.7	13
	40	3.1	1.0	0.3	3,044.1	863.2	0	18.2	0.5	8,022.2	1,500.7	18
	50	2.9	1.3	0.3	3,883.6	1,089.7	0	25.4	0.5	9,067.0	1,652.2	13
30	30	3.1	1.5	0.4	3,675.3	964.7	0	16.7	0.6	13,745.1	1,773.0	20
	40	3.0	1.3	0.3	5,008.7	1,313.4	0	20.6	0.5	19,561.0	2,371.9	22
	50	2.9	1.3	0.3	6,286.6	1,652.3	0	20.4	0.4	23,273.3	2,909.0	25
40	40	3.1	1.3	0.3	7,410.9	1,816.8	0	23.2	0.5	35,192.7	3,209.7	25
	50	3.0	1.3	0.3	8,875.9	2,240.7	0	24.2	0.5	43,725.3	3,983.8	36
50	50	3.1	1.8	0.3	11,871.2	2,854.2	0	27.9	0.6	71,307.5	4,990.4	43
Average	-	2.98	1.15	0.28	3,692.3	973.3	0	18.11	0.48	15,156.3	1,684	17.9
Ratios	-	-	0.06	0.58	0.24	0.58	-	1	1	1	1	-

Column 5 lists the number of SET arrays where the current detector is blocked. Columns 6–10 are the same results as Columns 1–5.

For example, in the SET array of  $40 \times 40$  with defect rate of 2%, there are 2 out of 100 SET arrays reported as useless. There are exactly two SET arrays having blocked the current detector. In fact, we have verified that the blocked SET arrays are exactly the reported useless SET arrays, i.e., we accurately reported the blocked SET arrays, which cannot be used for synthesizing any function as useless SET arrays. In the first experiment, the runtime of verifying useless SET arrays for 100 defective SET arrays of each size was less than one second.

### 4.3 Experimental Results of Proposed Diagnosis Approach and the State-of-the-Art

In the second experiment, the defective SET arrays with different defect rates were generated for each size of SET arrays. We excluded the mentioned SET arrays with blocked current detectors in the experiments. We conducted 100 runs of experiments to obtain the average results and compared the experimental results with the state-of-the-art [Li et al. 2017].

We break down the experimental results of the second experiment into Tables 3, 4, and 5 with 2%, 3%, and 4% defect rates, respectively. Columns 1 and 2 list the dimension of defective SET arrays, and Columns 3 and the real defect (Re. Def.) rate are in the SET arrays. The real defect rate in the SET arrays is higher than the defect rate we set since the injected defects include clustering-defects, which affect three SET nodes (i.e., four edges) in the SET array. Columns 4–8 (9–13) show the diagnosis results of our (the state-of-the-art [Li et al. 2017]) diagnosis approach. The results listed in Columns 4–7 (9–12) are averaged over 100 runs of experiments for each dimension and defect rate. Column 4 (9) lists the false-negative rate (F-neg.) representing the rate of the defect-free edges

Table 4. Experimental Results of the Proposed Diagnosis Approach and the State-of-the-Art [Li et al. 2017] with Defect Rate 3%

H	W	Re. Def.	Ours					[Li et al. 2017]				
			F-neg.(%)	Misjud.(%)	Config.	Pat.	F-pos. edg	F-neg.(%)	Misjud.(%)	Config.	Pat.	F-pos. edg
10	10	3.1	0.3	0.3	290.9	92.1	0	6.4	0.5	434.8	158.6	2
	15	4.3	1.1	0.5	494.2	153.3	0	10.7	0.7	805.4	268.0	6
	20	4.5	1.3	0.4	690.0	213.1	0	15.9	0.6	1,110.5	360.1	9
	30	4.0	1.2	0.3	1,025.7	323.4	0	22.1	0.6	1,580.8	470.0	24
	40	4.4	2.6	0.4	1,331.1	440.5	0	33.0	0.7	2,034.0	545.4	15
20	50	4.1	2.3	0.4	1,788.8	549.3	0	33.6	0.6	2,456.5	648.3	19
	20	4.7	1.5	0.5	1,620.0	435.7	0	15.5	0.9	4,036.0	776.5	13
	30	4.6	2.0	0.5	2,517.2	668.9	0	20.2	0.8	6,210.4	1,112.2	19
30	40	4.5	1.8	0.4	3,339.6	898.1	0	25.6	0.7	7,968.7	1,313.4	23
	50	4.4	3.0	0.4	4,251.5	1,113.4	0	33.9	0.7	9,023.9	1,525.9	23
	30	4.8	2.6	0.6	4,219.1	1,024.0	0	22.7	0.8	13,246.4	1,735.7	24
40	40	4.5	2.6	0.5	5,528.5	1,363.7	0	27.0	0.8	19,440.6	2,287.5	41
	50	4.5	3.6	0.5	6,906.9	1,693.8	0	28.2	0.7	23,983.3	2,787.7	55
	40	4.9	3.6	0.5	8,482.8	1,940.9	0	32.2	0.9	36,244.0	3,118.5	54
50	50	4.6	3.4	0.5	9,928.9	2,333.3	0	34.3	0.8	45,667.1	3,796.2	53
	50	4.7	4.6	0.6	13,040.5	2,939.6	0	38.2	0.9	73,856.5	4,768.7	64
Average	-	4.4	2.3	0.5	4,095.5	1,011.4	0	25.0	0.7	15,506.2	1,604.5	27.8
Ratios	-	-	0.92	0.71	0.26	0.63	-	1	1	1	1	-

Table 5. Experimental Results of the Proposed Diagnosis Approach and the State-of-the-Art [Li et al. 2017] with Defect Rate 4%

H	W	Re. Def.	Ours					[Li et al. 2017]				
			F-neg.(%)	Misjud.(%)	Config.	Pat.	F-pos. edg	F-neg.(%)	Misjud.(%)	Config.	Pat.	F-pos. edg
10	10	6.1	1.4	0.6	333.9	98.4	0	9.1	0.9	457.0	159.2	4
	15	6.1	1.8	0.6	534.8	159.5	0	16.0	0.9	807.3	257.5	7
	20	6.1	2.1	0.7	738.1	219.1	0	20.6	0.9	1,084.7	342.1	7
	30	5.9	2.0	0.5	1,105.6	335.1	0	31.1	0.9	1,386.8	423.0	30
	40	5.9	3.1	0.5	1,535.1	452.8	0	38.6	1.0	1,975.6	505.7	19
20	50	5.9	4.6	0.6	1,960.1	562.3	0	41.1	1.0	2,357.6	586.5	30
	20	6.3	2.3	0.8	1,767.7	456.1	0	20.9	1.2	4,175.1	757.0	15
	30	6.1	2.7	0.6	2,706.0	687.6	0	24.7	1.0	6,197.4	1,074.6	30
30	40	6.1	3.2	0.6	3,637.2	923.8	0	33.1	1.0	7,561.0	1,288.1	25
	50	5.8	4.2	0.6	4,573.3	1,139.3	0	41.3	1.0	8,902.4	1,309.6	23
	30	6.5	4.9	0.8	4,494.2	1,043.8	0	31.0	1.2	15,002.7	1,682.2	39
40	40	6.2	4.5	0.7	6,058.4	1,306.9	0	35.3	1.1	19,525.5	2,173.6	54
	50	6.1	5.8	0.7	7,508.0	1,740.2	0	35.8	1.1	23,280.3	2,630.1	66
	40	6.4	6.3	0.7	8,987.8	1,961.3	0	38.0	1.2	36,641.5	3,009.3	65
50	50	6.1	5.8	0.8	10,519.3	2,348.4	0	40.5	1.2	44,619.3	3,640.0	77
	50	6.2	6.2	0.7	13,079.7	3,024.9	0	45.4	1.1	75,772.6	4,639.4	80
Average	-	6.1	3.8	0.7	4,346.2	1,028.7	0	31.4	1.0	15,609.2	1,529.9	35.7
Ratios	-	-	0.12	0.70	0.28	0.67	-	1	1	1	1	-



that were misjudged as defective ones in the SET array, and it is calculated as  $|\text{misjudged defect-free edge}| / |\text{total edge}|$ . Column 5 (10) lists the misjudged-category (Misjud.) rate of defective edges that were identified as defective but labeled as incorrect defect-type, i.e., the D-open (D-short) edges are diagnosed as D-short (D-open) edges. The misjudged-category rate is calculated as  $|\text{misjudged defective edge}| / |\text{total edge}|$ . Columns 6 (11) and 7 (12) list the number of SET node configurations and input patterns that were used to diagnose the SET arrays. Column 8 (13) lists the total number of false-positive edges (F-pos. edge) of 100 runs of each dimension and defect rate. The false-positive edges represent the defective edges that were misjudged as defect-free edges. The last two rows of Tables 3, 4, and 5 are the averaged results and ratios as compared with the state-of-the-art [Li et al. 2017].

For example, in the SET array of  $20 \times 50$  with a defect rate of 4%, the real defect rate was 5.8%, the false-negative rate of our approach was 4.2%, the misjudged-category rate of our approach was 0.6%, and our approach required 4,573.3 node configurations and 1,139.3 input patterns to diagnose the defects. On the other hand, in the work of Li et al. [2017], the false-negative rate was 41.3%, the misjudged-category rate was 1%, and Li et al. required 8,902.4 node configurations and 1,409.6 input patterns to diagnose the defects. Furthermore, our results had no false-positive edge while there were 23 false-positive edges totally from Li et al. [2017].

According to the last two rows of Tables 3, 4, and 5, our false-negative rate, misjudged-category rate, node configuration count, and input pattern count are all much smaller than Li et al. [2017]. Furthermore, our approach does not have any false-positive edge while Li et al. [2017] has 27.1 false-positive edges on average. The false-negative rate in Li et al. [2017] was about 10 times higher than ours. This is because when diagnosing a defective SET array with clustering-defects, the method in Li et al. [2017] cannot maintain a conducting path as a baseline for neighboring paths. Furthermore, Li et al. [2017] also has a much larger node configuration count. This is because in the diagnosis procedure in Li et al. [2017], if the newly configured path is not conductive, the algorithm will recover the path to the last conducting path. Therefore, when the diagnosis procedure cannot maintain a conducting path due to clustering-defects, it will take a lot of node re-configurations to recover the path. The averaged run time of both the proposed algorithms and Li et al. [2017] for diagnosing 100 defective SET arrays of each size was about one second.

#### 4.4 Undiagnosable Defect-free Edges

For most defective SET arrays, our approach can have a small false-negative rate. However, this rate was large for some SET arrays. This is because some defects clustered together in an area and blocked the current of the paths that were configured for diagnosing the defect-free edges in the area. Therefore, these defect-free edges were considered as D-open edges since we failed to configure any conducting path to diagnose them. These misjudged defect-free edges could be diagnosed with complicated and tortuous paths that detour all the defects surrounding them, which is a rare case. Furthermore, some of these misjudged defect-free edges will not be involved in any conducting path due to blocking. For example, in Figure 11(a), the two edges labeled as *misjudged defect-free edges*<sup>Ⓞ</sup> were diagnosed as D-open candidates and will be considered as D-open edges in the end. These edges are actually undiagnosable edges due to blocking accidentally. In our experiments, there are a lot of defect maps having such misjudged defect-free edges, which rising the false-negative rate. Especially, for the SET arrays with a higher defect rate, this situation occurs more frequently. On the other hand, a clustering-defect can also create misjudged defect-free edges. For example, for a clustering-defect with four D-short edges, it forms a multiple-path conduction as shown in Figure 11(b). Once we configure a path including any one of its neighboring edges, which is labeled as misjudged defect-free edges, the path will connect to the clustering-defect and result in a multiple-path conduction. These edges cannot be used for mapping functions. There-

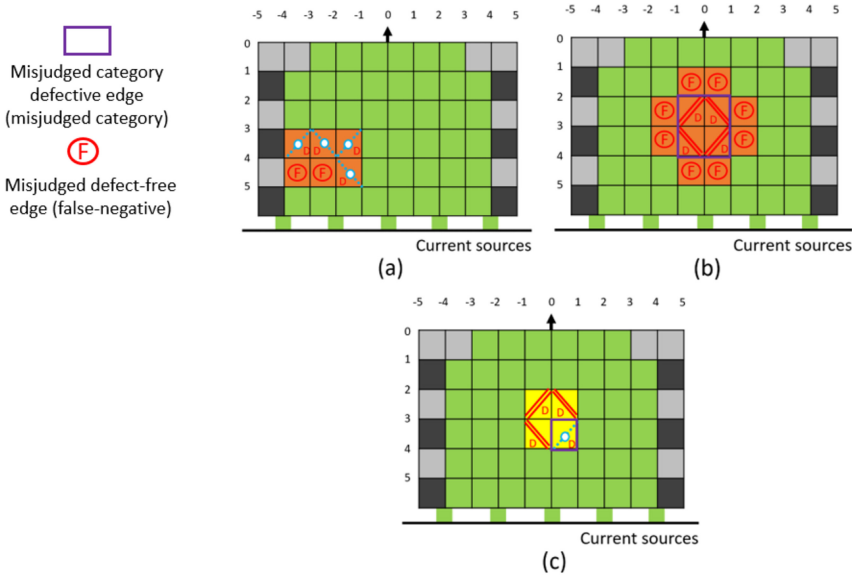


Fig. 11. Example for the false-negative and misjudged category.

fore, these edges have the same defect effect as D-open edges, and we mark them as D-open edges in our approach, which increases the false-negative rate.

According to Tables 3, 4, and 5, our approach categorizes a few edges into an incorrect defect type. Most of the misjudged defective edges were from clustering-defects in our experiments. There are two situations of clustering-defects where our algorithm will misjudge the defective edges. One is that the clustering-defect consists of four D-short edges as mentioned and shown in Figure 11(b). The other is that the clustering-defect consists of three D-short edges and one D-open edge as shown in Figure 11(c). The D-open edge in the purple rectangle is labeled as a misjudged defective edge since it was actually diagnosed as D-short edge. This is because the current from any path involving this D-open edge can pass through the other three D-short edges. Therefore, the D-open edge has the same defect effect as a D-short edge.

## 5 CONCLUSION

We propose the first diagnosis approach for the defective SET arrays without the two defect distribution assumptions that were adapted in the prior arts. This more generalized defect model includes clustering-defect, and is more practical when particle size happened to be greater than that of an SET node. The experimental results show that our approach can achieve low false-negative rates and misjudged-category rates without reporting any false-positive edge. Furthermore, our approach is more efficient since it needs fewer node configurations and input patterns for diagnosis. Therefore, the proposed approach can efficiently diagnose the defective SET arrays with adjacent defects and clustering-defects, and elevate the reliability of SET arrays in the synthesis flow.

## REFERENCES

- N. Asahi, M. Akazawa, and Y. Amemiya. 1997. Single-electron logic device based on the binary decision diagram. *IEEE Transactions on Electron Devices* 44, 7 (1997), 1109–1116.
- Y.-C. Chen, S. Eachempati, C.-Y. Wang, S. Datta, Y. Xie, and V. Narayanan. 2011. Automated mapping for reconfigurable single-electron transistor arrays. In *Proceedings of the Design Automation Conference*. 878–883.

- Y.-C. Chen, S. Eachempati, C.-Y. Wang, S. Datta, Y. Xie, and V. Narayanan. 2013. A synthesis algorithm for reconfigurable single-electron transistor arrays. *ACM Journal on Emerging Technologies in Computing System* 9, 1, Article 5 (2013).
- Y.-H. Chen, J.-Y. Chen, and J.-D. Huang. 2014. Area minimization synthesis for reconfigurable single-electron transistor arrays with fabrication constraints. In *Proceedings of the Design Automation and Test in Europe*. 1–4.
- Y.-H. Chen, Y. Chen, and J.-D. Huang. 2015. Area minimization synthesis for reconfigurable single-electron transistor arrays with fabrication constraints. In *Proceedings of the International Symposium on VLSI Design, Automation and Test*. 1–4.
- C.-E. Chiang, L.-F. Tang, C.-Y. Wang, C.-Y. Huang, Y.-C. Chen, S. Datta, and V. Narayanan. 2013. On reconfigurable single-electron transistor arrays synthesis using reordering techniques. In *Proceedings of the Design, Automation and Test in Europe*. 1807–1812.
- S. Eachempati, V. Saripalli, V. Narayanan, and S. Datta. 2008. Reconfigurable BDD-based quantum circuits. In *Proceedings of the International Symposium on Nanoscale Architectures*. 61–67.
- C.-H. Ho, Y.-C. Chen, C.-Y. Wang, C.-Y. Huang, S. Datta, and V. Narayanan. 2016. Area-aware decomposition for single-electron transistor arrays. *ACM Transactions on Design Automation of Electronic Systems* 21, 4, Article 70 (2016).
- C.-Y. Huang, Y.-J. Li, C.-W. Liu, C.-Y. Wang, Y.-C. Chen, S. Datta, and V. Narayanan. 2016. Diagnosis and synthesis for defective reconfigurable single-electron transistor arrays. *IEEE Transactions on VLSI Systems* 24, 6 (2016), 2321–2334.
- C.-Y. Huang, C.-W. Liu, C.-Y. Wang, Y.-C. Chen, S. Datta, and V. Narayanan. 2015. A defect-aware approach for mapping reconfigurable single-electron transistor arrays. In *Proceedings of the Asia and South Pacific Design Automation Conference*. 118–123.
- S.-Y. Huang and K.-T. Cheng. 1999. ErrorTracer: A fault simulation based approach to design error diagnosis. *IEEE Transactions on Computer-Aided Design* (1999), 1341–1352.
- W. H. Kautz. 1968. Fault testing and diagnosis in combinational digital circuits. *IEEE Transactions on Computers* C-17, 4 (1968), 352–366.
- Y.-J. Li, C.-Y. Huang, C.-C. Wu, Y.-C. Chen, C.-Y. Wang, S. Datta, and V. Narayanan. 2017. Dynamic diagnosis for defective reconfigurable single-electron transistor arrays. *IEEE Transactions on VLSI Systems* 25, 4 (2017), 1477–1489.
- H.-T. Liaw, J.-H. Tsaih, and C.-S. Lin. 1990. Efficient automatic diagnosis of digital circuits. In *Proceedings of the International Conference on Computer-Aided Design*. 464–467.
- C.-W. Liu, C.-E. Chiang, C.-Y. Huang, C.-Y. Wang, Y.-C. Chen, S. Datta, and V. Narayanan. 2014. Width minimization in the single-electron transistor array synthesis. In *Proceedings of the Design, Automation and Test in Europe*.
- C.-W. Liu, C.-E. Chiang, C.-Y. Huang, C.-Y. Wang, Y.-C. Chen, S. Datta, and V. Narayanan. 2015a. Synthesis for width minimization in the single-electron transistor array. *IEEE Transactions on VLSI Systems* 23, 12 (2015), 2862–2875.
- L. Liu, X. Li, V. Narayanan, and S. Datta. 2015b. A reconfigurable low-power BDD logic architecture using ferroelectric single-electron transistors. *IEEE Transactions on Electron Devices* 62, 3 (2015), 1052–1057.
- L. Liu, V. Saripalli, V. Narayanan, and S. Datta. 2011. Device circuit co-design using classical and non-classical III–V multi-gate quantumwell fets (MuQFETs). In *Proceedings of the IEEE International Electron Devices Meeting*. 4.5.1–4.5.4.
- H. W. Ch. Postma, T. Teepen, Z. Yao, M. Grifoni, and C. Dekker. 2001. Carbon nanotube single-electron transistors at room temperature. *Science* 293 (2001), 76–79.
- Y. T. Tan, T. Kamiya, Z. A. K. Durrani, and H. Ahmed. 2003. Room temperature nanocrystalline silicon single-electron transistors. *Journal of Applied Physics* 94 (2003), 633–637.
- A. Veneris and I. N. Hajj. 1999. Design error diagnosis and correction via test vector simulation. *IEEE Transactions on Computer-Aided Design* 18, 12 (1999), 1803–1816.
- Z. Zhao, C.-W. Liu, C.-Y. Wang, and W. Qian. 2014. BDD-based synthesis of reconfigurable single-electron transistor array. In *Proceedings of the International Conference on Computer-Aided Design*. 47–54.
- L. Zhuang, L. Guo, and S. Y. Chou. 1998. Silicon single-electron quantum-dot transistor switch operating at room temperature. *Applied Physics Letters* (1998), 1205–1207.

Received March 2020; revised August 2020; accepted December 2020