A Survey of Fault Tolerance Algorithms for Reconfigurable Nano-Crossbar Arrays

ONUR TUNALI, Istanbul Technical University
MUSTAFA ALTUN, Istanbul Technical University

Nano-crossbar arrays have emerged as a promising and viable technology to improve computing performance of electronic circuits beyond the limits of current CMOS. Arrays offer both structural efficiency with reconfiguration and prospective capability of integration with different technologies. However, certain problems need to be addressed and the most important one is the prevailing occurrence of faults. Considering fault rate projections as high as 20% that is much higher than those of CMOS, it is fair to expect sophisticated fault tolerance methods. The focus of this survey paper is the assessment and evaluation of these methods and related algorithms applied in logic mapping and configuration processes. As a start, we concisely explain reconfigurable nano-crossbar arrays with their fault characteristics and models. Following that, we demonstrate configuration techniques of the arrays in the presence of permanent faults and elaborate on two main fault tolerance methodologies, namely defect-unaware and defect-aware approaches, with a short review on advantages and disadvantages. For both methodologies, we present detailed experimental results of related algorithms regarding their strengths and weaknesses with a comprehensive yield, success rate, and runtime analysis. Next, we overview fault tolerance approaches for transient faults. As a conclusion, we overview the proposed algorithms with future directions and upcoming challenges.

CCS Concepts: •**General and reference** → **Surveys and overviews;**•**Hardware** → **Emerging architectures; Fault Tolerance;**

General Terms: Nano-crossbar, Realiability, Fault Tolerance

Additional Key Words and Phrases: Fault Tolerance, Nano-crossbar

ACM Reference Format:
Onur Tunali and Mustafa Altun, 2017. A Survey of Fault Tolerance Algorithms for Reconfigurable Nano-Crossbar Arrays *ACM Comput. Surv.* V, N, Article A (January YYYY), 35 pages.
DOI: 0000001.0000001

1. INTRODUCTION

Since the first digital computer was developed in 1930's, several breakthroughs have been made both in technological and computational levels to improve the performance of computers. Researchers have aimed at finding the best way to realize the fundamental building element of computers that is a two-terminal switch. First electromechanical systems, then respectively vacuum tubes, *p-n* junction based diodes & transistors, and CMOS transistors are used as switches. Considering these different eras, without a question CMOS era is the longest prevailing and the most fruitful one. For more than 50 years, CMOS computing performance has increased almost in a regular manner, that is often called the Moore's law [Schaller 1997]. However, this trend has started to slow down, and it is widely accepted that another transition to a new era is soon to be occurred [Dubash 2005] and [Conte and Gargini 2015].

This work is supported by the EU-H2020-RISE project NANOxCOMP #691178 and the TUBITAK-Career project #113E760.
Author's addresses: O. Tunali, Nanoscience an Nanoengineering Department, Istanbul Technical University; M. Altun, Electronics and Communication Engineering Department, Istanbul Technical University; Istanbul 36064; E-mails: onur.tunali, altunmus@itu.edu.tr.
Permission to make digital or hard copies of part or all 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 third-party components of this work must be honored. For all other uses, contact the owner/author(s).
© YYYY Copyright held by the owner/author(s). 0360-0300/YYYY/01-ARTA $15.00
DOI: 0000001.0000001

At this point, a significant amount of research has been dedicated to nanoscale technologies with new materials including nanotubes, nanowires, and individual molecules being used to implement a switch [Waser 2012]. Although computing performances of these individual switches are quite satisfactory, generally much better than those of CMOS, there are still problems to be solved before the commercialization: 1) Integration of individual switching elements to generate a fully functional computing architecture is quite costly; and 2) The final forms of architectures are not immune to unusually high fault rates. In this regard, nano-crossbar arrays have been favored by researchers such as nanofabric using molecular switches (programmable diodes) [Goldstein and Budiu 2001], nanoPLA using programmable diode [Dehon 2005], CMOS-like structures using nFET or pFET transistors [Snider et al. 2005], CMOL using programmable diodes [Strukov and Likharev 2005], NASIC using FET transistors or diodes [Wang et al. 2008], [Morgul et al. 2016], and [Alexandrescu et al. 2016]. Another prospective technology is based on memristors or memristive switches which is constructed as crossbar-like structures [Yang et al. 2013] and having the same logic mapping approach [Xie et al. 2015]. Moreover, as a physical realization, three fully operational implementations of nano-crossbar arrays as nanocomputers are shown to be feasible in [Yan et al. 2011], [Shulaker et al. 2013], and [Yao et al. 2014]. Even though these technologies differ in certain levels, basic computing blocks are always crossbars with each crosspoint behaving as a switch, so logic mapping schemes are similar.

Arrays offer both structural efficiency with reconfiguration and prospective capability of integration. In addition, inherent redundancy present in nano-crossbars provides flexibility for fault tolerance that is a much needed help considering that fault tolerance is the main challenge to be resolved. Fault rate projections of nano-crossbars are as high as 20% that is much higher than those of CMOS due mainly to the used bottom-up fabrication techniques with self-assembly that has stochastic nature as opposed to using conventional top-down fabrication techniques with directed-assembly [Wu et al. 2005], [Huang et al. 2001], [Chen et al. 2003], and [Haselman and Hauck 2010]. Therefore, it is fair to expect sophisticated fault tolerance methods for nano-crossbars. The focus of this survey paper is the assessment and evaluation of fault tolerance methods and related algorithms applied in logic mapping and configuration processes of nano-crossbar arrays.

Prior to examining fault tolerance techniques in the literature, a very coarse description of a common crossbar structure and its computing fundamentals can be summarized as follows. Nano-crossbar arrays are formed by placing a group of lines/wires aligned parallel to each other on another group of array lines/wires orthogonally. Vertical and horizontal lines are used as input and outputs, respectively. This is illustrated in Figure 1 (a). Boolean literals are applied to input lines, and each output line corresponds to a product, i.e., AND of literals. Therefore a given function in sum-of-products form can be directly implemented by using each product with an output line by deactivating and activating relevant crosspoints. Note that other forms based on factored Boolean expressions or binary decision diagrams can not be used since these forms require certain wirings/connections between lines and crosspoints that is not applicable for nano arrays [Dehon 2005], [Snider et al. 2005], and [Alexandrescu et al. 2016]. While a deactivated crosspoint behaves as an open circuit between the crossed lines, an activated crosspoint is a two-terminal switch that can behave as a diode [Ziegler and Stan 2003] or a FET [Zhong et al. 2003]. This is shown in Figure 1 (b). If the components are ON (OFF) then their terminals are shorted (open). Note that the distinction of the components is that their ON (OFF) states connect (disconnect) terminals in different lines.

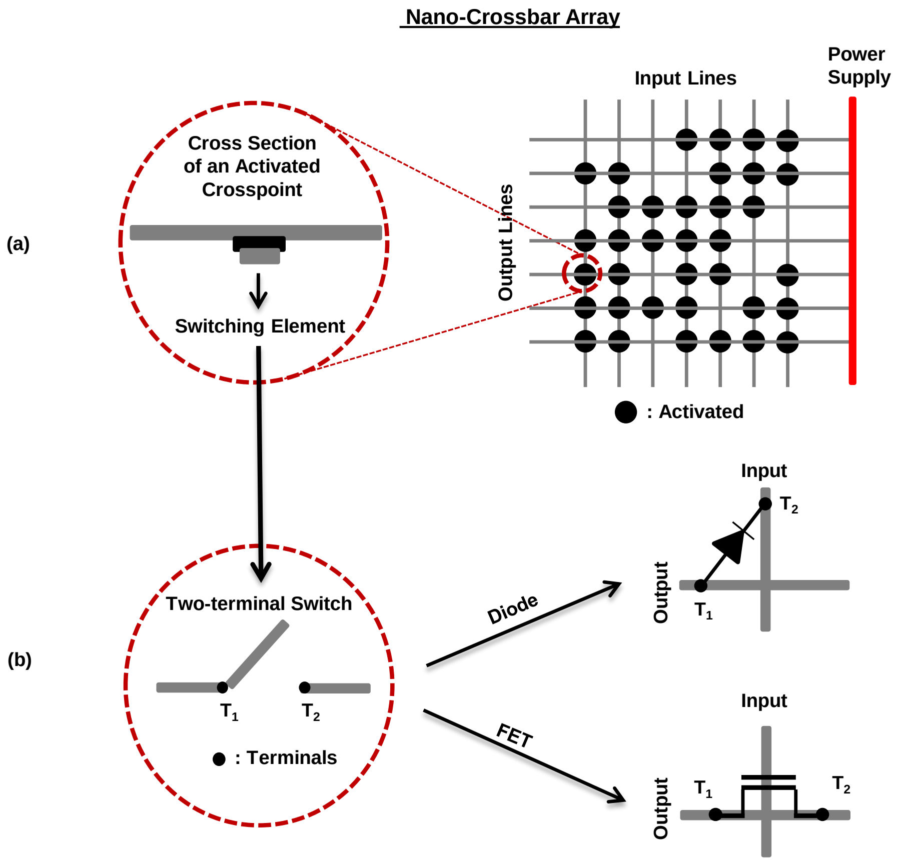


Fig. 1. A nano-crossbar array: (a) activated and deactivated crosspoints, and (b) a switching crosspoint behaving as an electrical component either a diode or a FET.

1.1. Fault Tolerance in Programmable Logic Arrays

In a historical context, nano-crossbar structures are very similar to programmable logic arrays (PLA's) introduced in [Fleisher and Maissel 1975], in terms of circuit structures, programming features, and utilization. Therefore examining the progression of fault tolerance studies regarding PLA's can be insightful. Particular aspects are as follows: 1) Test generation and fault detection [Ostapko and Hong 1979] and [Smith 1979] which basically deals with producing comprehensive test vectors; 2) Yield analysis and redundancy employment [Wey et al. 1987] and [Wey 1988] which aims to maximize yield and allocate redundant elements; and 3) Fault modeling with simulations [Ligthart and Stans 1991] which formalizes the type of faults for stuck-at, bridging, missing, and broken crosspoints that is also adopted in nano-crossbar terminology. Additionally, different from these studies, in [Demjanenko and Upadhyaya 1990] fault tolerance is achieved with reconfiguration of a PLA by using bipartite graph model which can be considered as the archetype of the techniques used in nano-crossbars. In this study, exaggeratedly high fault rates are considered similar to the treatment in nano-crossbars. Indeed, developed with a well established CMOS technology, PLA's do have considerably low fault rates. Therefore simple configuration approaches are adequate for fault tolerance that justifies the lack of related studies in the literature after 1990's.

Table I. Permanent versus Transient Faults.

Permanent Faults (Defects)	**Transient Faults**
• Occurring mostly in fabrication	• Occurring in field
• Tolerated in design phase	• Tolerated in use phase
• Tolerated by configuration (mapping)	• Tolerated by reconfiguration or redundancy

1.2. Fault Tolerance in Nano-Crossbar Arrays

Examining the fault tolerance techniques in the literature, we see a common tendency of considering faults causing only phase shifts between activated and deactivated crosspoints [Tahoori 2006], [Al-Yamani et al. 2007], [Zheng and Huang 2009], [Gören et al. 2011], [Su and Rao 2014], [Yuan and Li 2014], and [Tunali and Altun 2017]. Only a few studies consider faults affecting the functionality of electrical components in crosspoints that causes phase shifts between ON and OFF states of the components [Bhaduri et al. 2004], [Gil et al. 2008], and [Zamani et al. 2013]. Indeed, component based fault modeling is more appropriate for failures seen in field, and this is not quite applicable for emerging technologies, because they have very limited field data including nano-crossbar arrays. Another reason favoring activated/deactivated crosspoint based fault modeling is its capability to deal with faults occurring in input and output lines such as broken and bridging faults. For example, all crosspoints of a broken line can be modeled as deactivated.

Another examination is that distinct approaches are proposed to tolerate permanent and transient faults regarding their exclusive natures as summarized in Table I. While permanent faults called as defects are related to the configuration of nano-crossbars that is performed during post fabrication, transient faults occurring in field are tolerated using either redundancies or detection-reconfiguration cycles.

1.2.1. Permanent Fault Tolerance. In the presence of permanent faults called as defects, tolerance is achieved by mapping Boolean logic functions on a defective crossbar using crossbar row and column permutations. This is an NP-complete problem [Shrestha et al. 2009]. For the worst-case scenario, implementing a given function with an $N \times M$ crossbar requires $N!M!$ permutations; computing time quickly grows to intractable levels with the crossbar size. Additionally, high fault rates complicate the mapping process by constraining the possible valid choices. Nevertheless, seminal Teramac experiment in [Amerson et al. 1995] shows that it is possible to produce reliable computing with using components having excessive faulty parts. As specified in [Heath et al. 1998], as long as adequate connectivity and efficient algorithms for configuring present, it is feasible to use a reconfigurable nano-crossbar to obtain reliable computing structures. In the literature, proposed defect-tolerant logic mapping algorithms of nano-crossbars can be categorized under two main methodologies: defect-unaware and defect-aware. However, it should be noted that both methods use a defect map which shows the location of faults in nano-crossbar, so a more intuitive naming would be “defect-avoiding” and “defect-employing”. Nevertheless, we follow the prevalent terminology present widely in the literature.

Defect-unaware methods determine the size of an $n \times n$ nano-crossbar in order to obtain a $k \times k$ defect-free sub-crossbar, so it is possible to know the required size of a crossbar in advance to implement a given logic function. Using the defect-free sub-crossbar, a straightforward mapping process can be applied. However, the number of studies in this field is limited due to the inefficient area yield. There is a common shortcoming especially for high fault rates – obtained k values are much smaller than n. When $N = 250$ and the fault rate is 15% that is a reasonable value for nano arrays, the proposed algorithms find k values as high as 30 [Yuan and Li 2014]. It means that only 1% of the crossbar can be used. Proposed algorithms use graph based models and heuristics to solve the maximum independent set problem in a complement graph [Tahoori 2006], [Al-Yamani et al. 2007], [Yuan and Li 2011], and [Yuan and Li 2014].

Defect-aware logic mapping methods employ defective elements in the mapping process that results in much better area yields. However, the mapping process is more complicated. The number of studies in this field is abundant due to flexible nature of formalizing the problem. Earlier works utilize graph based models including [Naeimi and DeHon 2004] solving the bipartite matching problem with a greedy approach and [Rao et al. 2009] solving the graph embedding problem with a recursive approach. Additionally, [Yellambalase and Choi 2008] examine the effect of clustered defects with a matrix based algorithm, and [Zheng and Huang 2009] uses satisfiability approach for the mapping process. Another greedy algorithm is proposed in [Simsir et al. 2009] using partial graph constructing. Apart from the graph based models, an ILP model is used in [Yang and Datta 2011] and [Zamani et al. 2013] by introducing constraints related to nano-crossbar defects. Furthermore a novel approach benefiting from graph canonization with sorting is used in [Gören et al. 2011]. In order to handle scalability more efficiently compared to the above methods, [Naeimi and DeHon 2004] uses a greedy approach; [Yuan et al. 2014] uses a graph based approach with memetic fitness approximation; and [Tunali and Altun 2017] implements matrix sorting supported by greedy backtracking.

1.2.2. Transient Fault Tolerance. Another aspect of fault tolerance in nano-crossbars is the transient faults occurring in field. Similar to conventional technologies targeting transient faults, hardware redundant solutions are proposed. In [Rao et al. 2007], two approaches using an online test with reconfiguration and a fault masking scheme are investigated. Comparing the approaches, fault masking offers smaller hardware overhead at the cost of having very limited capability of tolerating multiple faults. In [Rao et al. 2009], another fault masking method with a focus on missing devices (denoted with stuck-at deactivated faults in this paper) is proposed with utilizing logic tautologies. In [Garcia and Orailoglu 2008], an alternative reconfiguration based fault tolerance scheme is proposed with novel online testing using a text vector compaction. It is possible to accomplish input/output level diagnoses with reduced runtime by means of the proposed test vectors. Even though a fault tolerance mechanism is excluded, a novel error detection method proposed in [Farazmand and Tahoori 2009]. Logic implementation is realized with a dual rail structure having both the function and its negation as outputs. By comparing the outputs, it is possible to detect faults conforming to certain assumptions of fault characteristics. So far mentioned papers are related to the logic synthesis level of fault tolerance. In [He et al. 2005] and [He and Jacome 2007], a high level synthesis paradigm with reconfiguration is proposed with choosing certain mapping units with an optimization among many solutions.

It should be noted that transient fault tolerance of nano-crossbar arrays is in exploratory phase, and only small fraction of the above mentioned papers fully target nano-crossbar arrays. Mostly, conventional PLA based architectures are used. Additionally, as for all emerging technologies, nano-crossbar arrays have very limited field data that is needed for accurate modeling of transient faults.

1.3. Variation Tolerance in Nano-Crossbar Arrays

Considering that similar techniques are used in fault and variance tolerance, it is worth mentioning studies considering variations in crosspoint delay values for performance optimization of nano-crossbar arrays. As an earlier example, in [Gojman and DeHon 2009] authors used fan-out matching to minimize the path delays and extrapolate delay values from [Committee et al. 2008]. In [Ghavami et al. 2010] and [Zamani et al. 2013], the authors focus on minimizing the maximum variation of the overall

Table II. Key Papers for Fault Tolerance in Nano-crossbar Arrays

Contribution of Papers
Fault tolerance of PLA's by reconfiguration [Demjanenko and Upadhyaya 1990]
Fault tolerance with configurable custom computing hardware - Teremac [Heath et al. 1998]
Test methodology to determine defect locations in nano-crossbars [Mishra and Goldstein 2004]
Logic with CMOS-like nano-crossbars [Snider et al. 2004]
Defect-aware logic mapping with graph monomorphism [Hogg and Snider 2004]
Defect-aware logic mapping with greedy matching [Naeimi and DeHon 2004]
Defect-unaware sub-array search methodology [Tahoori 2006]
Built-in nanoscale error correcting for transient faults [Moritz et al. 2007]
Transient fault tolerance for nano-crossbars: possibilities and challenges [Rao et al. 2007]
Nano-crossbar specific fault modeling [Gil et al. 2008]
Defect-aware logic mapping with memetic algorithm [Yuan et al. 2014]
Defect-aware logic mapping with matrix sorting and backtracking [Tunali and Altun 2017]

crossbar. A bipartite graph and integer linear programming models are used respectively. In [Tunc and Tahoori 2010] and [Tahoori 2010], two objectives are employed as minimizing the maximum delay and minimizing the output variations. As an algorithmic aspect, they use a simulated annealing approach. In [Yang et al. 2011], the problem is formulated as a multiojbective optimization problem and an evolutionary algorithm is utilized. In [Zhong et al. 2016], a hybrid evolutionary algorithm is used and the problem is formulated as a bilevel multiobjective optimization.

1.4. Overview and Organization

A selection of key fault-tolerance studies is given in Table II. We determine these papers in terms of their novel contribution to the state-of-the-art. Since configuration is the main power of fault tolerance in nano-crossbars, the list starts with some early efforts exploiting configurability for tolerance (not necessarily for nano-crossbars). Following studies in the list show relatively recent trends and developments specifically for nano-crossbars. Moreover, we add a concise life cycle and fault tolerance steps of a nano-crossbar in Figure 2. This high-level integration picture demonstrates when certain fault tolerance mechanisms come into the picture. The step “function and crossbar models” in the figure correspond to the algorithms' individual problem formalization, so it is approach dependent. The step “connectivity checks between crossbars/planes” is performed after achieving configuration for each nano-crossbar to form a fully functional architecture. Fault tolerance algorithms that constitute the considerable portion of this survey are used in post-fabrication configuration and also employed during reconfiguration to tolerate in-field faults. Another aspect of in-field fault tolerance is hardware redundancy based on fault masking.

The rest of the paper is organized as follows. In Section 2, we present fault characteristics and models. In Section 3, we demonstrate configuration of nano-crossbars for logic mapping. In the following two sections, we focus on permanent faults. In Sections 4 and 5, we explain defect-unaware and defect-aware logic mapping algorithms, respectively and we present experimental results of the algorithms by comparing yield, success rate, and runtime parameters. In Section 6, we examine fault tolerance techniques for transient faults. In Section 7, we discuss future directions of the methods with upcoming challenges.

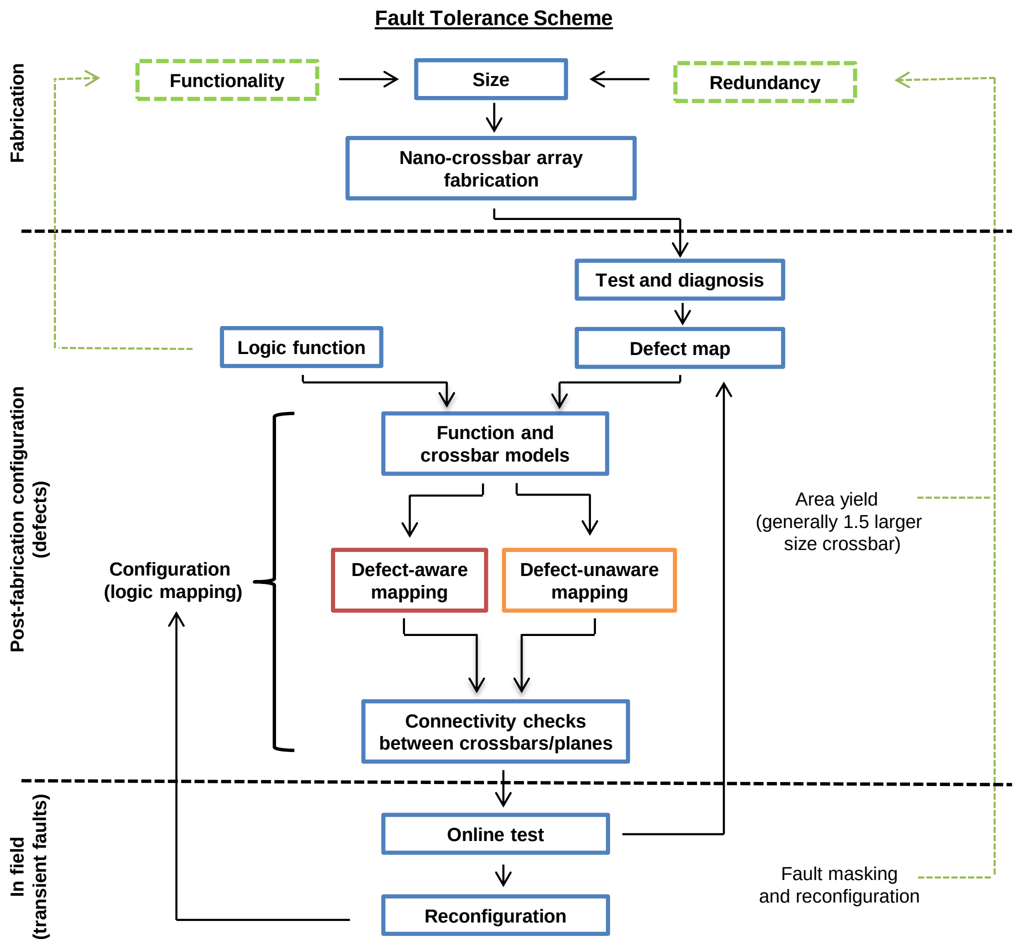

Fault Tolerance Scheme
Fabrication
Functionality
Size
Redundancy
Nano-crossbar array fabrication
Test and diagnosis
Logic function
Defect map
Post-fabrication configuration (defects)
Function and crossbar models
Configuration (logic mapping)
Defect-aware mapping
Defect-unaware mapping
Area yield (generally 1.5 larger size crossbar)
Connectivity checks between crossbars/planes
In field (transient faults)
Online test
Reconfiguration
Fault masking and reconfiguration

Fig. 2. Outline of fault tolerance scheme.

2. FAULT CHARACTERISTICS

In this section, we first define fault models of nano-crossbars in logic level. Then, we elaborate on the distinction between permanent and transient faults with an overview of literature tendencies towards the adoption of different fault models. As a note, we enunciate that there is no consistent modeling preference of faults in the literature. Most of the works only consider certain type of crosspoint faults. In experimental results, we explore the effects of using different fault models in depth.

2.1. Fault Models

We use *fault* as a generic term for problems that might cause en error in computing. Faults in nano-crossbars can be considered under two categories: 1) Faults affecting the configuration of crosspoints that cause phase shifts between activated and deactivated phases; and 2) Faults affecting the functionality of electrical components in crosspoints that causes phase shifts between ON and OFF states of the components. In the first category, *configuring* a crosspoint switch includes *activating* and *deactivating* processes. When a crosspoint is *activated*, it means that there is an electrical component at the crosspoint and its functionality is intact, so phase shifts between ON

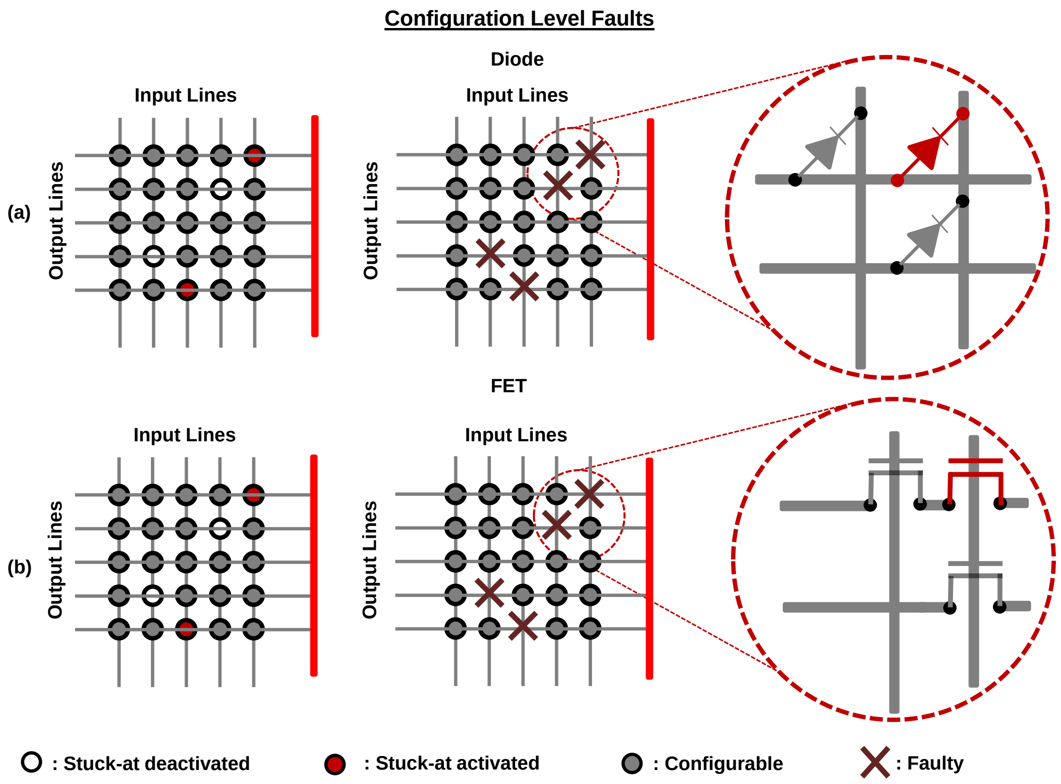


Fig. 3. Configuration level faults: (a) for diode based, and (b) for FET based switching crosspoints.

and OFF states is possible. When a crosspoint is *deactivated*, it means as if no component is present at the crosspoint. Configuration level faults are defined as follows:

— *Stuck-at deactivated* fault makes the corresponding crosspoint switch always deactivated that can not be used as a functional component any more; and
— *Stuck-at activated* fault makes the corresponding crosspoint switch always activated, so there is a functional component.

Representation of configuration level faults and their effects are shown in Figure 3. As can be seen from the figure, this type of faults only affect the corresponding crosspoint itself. In addition to the faults directly affecting crosspoint switches, broken and bridging faults might occur on input and output lines. They can also be modeled using crosspoint faults such that all crosspoints of broken or adjacent lines can be considered stuck-at deactivated.

For the second category of faults, we consider the *functionality* of electrical components in crosspoints. This type of faults are defined as follows:

— *Stuck-at OFF* fault makes the corresponding crosspoint component not capable of conducting current, so the component ideally has an infinite resistance; and
— *Stuck-at ON* fault makes the corresponding crosspoint component constantly conduct current, so the component ideally has a zero resistance.

These types of faults affect the other switches of the crossbar according to the technology preference of either having diode based or FET based crosspoints. For a diode based crosspoint, a stuck-at OFF fault means no connection between the two terminals

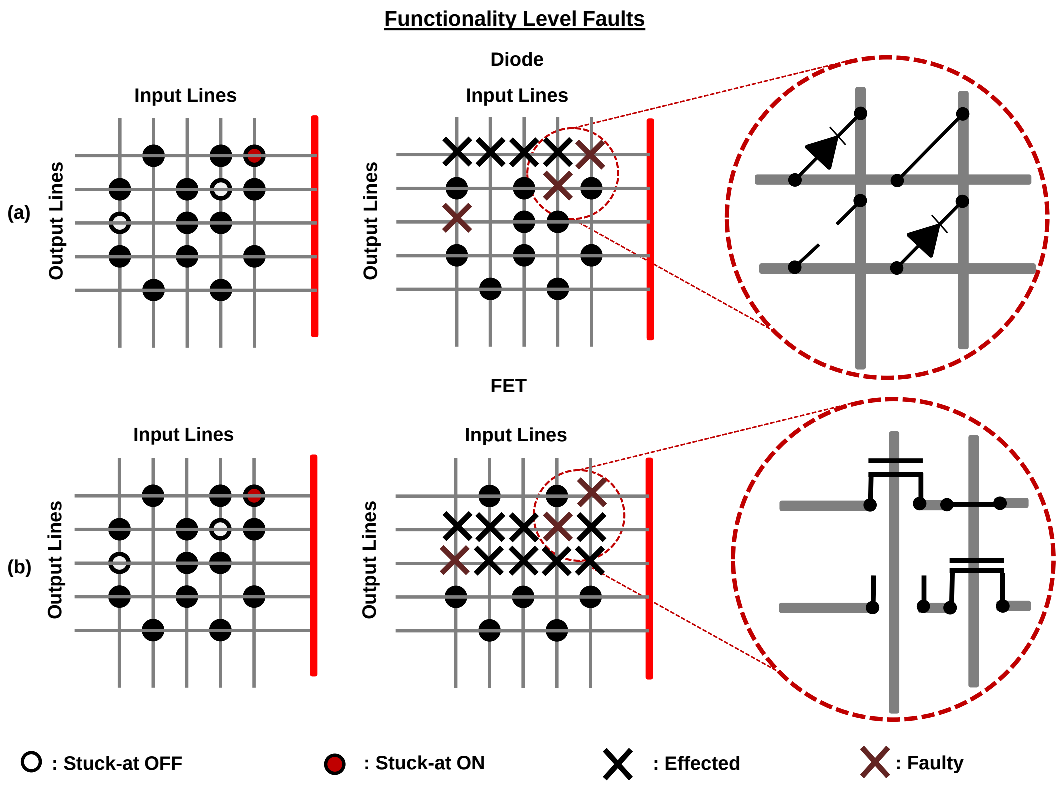


Fig. 4. Functionality level faults: (a) for diode based, and (b) for FET based switching crosspoints.

of the diode, placed in the crossed input and output lines, so it only affects the faulty crosspoint. On the other hand, a stuck-at ON fault means a constant connection between the terminals, so all crosspoints in the corresponding output line are discarded.

For a FET based crosspoint, a stuck-at OFF fault breaks the connection between the two terminals, both placed in the output line, so all crosspoints in the corresponding output line are discarded. On the other hand, a stuck-at ON fault means a constant connection between the terminals, so it only affects the corresponding crosspoint. Representation of faults for diode and FET based arrays are shown in Figure 4 (a) and (b), respectively.

2.2. Permanent and Transient Faults

Permanent and transient fault concepts are more related to the life cycle of a nano-crossbar than the physical characteristics of faults.

— *Permanent faults* or *defects* occur during fabrication process due to physical problems or variations; and
— *Transient faults* occur in field during the operation of a product.

Considering the two categories of faults presented in the previous subsection related to configuration of crosspoints and functionality of components, we can say that both categories are applicable for permanent and transient faults. We can also claim that permanent faults mostly comprise of configuration level faults since it is unlikely to see a case after fabrication that a crosspoint can be perfectly activated/deactivated, but its electrical component is not properly operating. Therefore, faults related to the functionality of components are more likely to be transient that is also supported by the fact that degradation and aging phenomena of components show transient characteristics.

3. LOGIC MAPPING PROBLEM

Nano-crossbar based architectures are generally composed of specific crossbars/planes each of which implement a 2-level logic such as AND-OR as illustrated in [Dehon 2005], [Strukov and Likharev 2005], [Strukov and Likharev 2007], [Moritz et al. 2007], and [Wang et al. 2007]. Therefore realization of a target logic function whether 2-level or multi-level is closely related to the architecture. However, if we focus on a single plane and use an abstraction (independent of plane character), logic mapping process (whether 2-level or multi-level) gradually can be applied to every connecting/succeeding planes to accomplish the desired result. This is a common practice in the literature with using AND planes for benchmark simulations. The reason of using AND planes is that they are generally much larger than OR planes; using a reconfigurability feature, a single line/wire as an OR plane is even sufficient to have every output at a time. Note that with defect-free OR planes, one can make connections between planes without any constraint for the orderings of input and output lines. For this purpose defect-unaware logic mapping techniques can be preferred. Logic mapping and connectivity checks of planes are previously illustrated in Figure 2 with an integrated high-level view.

Logic mapping is the configuration of crosspoint switches of a nano-crossbar in order to implement a given Boolean logic function given in sum-of-products form such as $f = P_1 +, ..., +P_k$. The main goal is finding a valid mapping, namely a correct assignment of literals and products of the function to inputs and outputs of a given crossbar. Input and output assignments can be represented with an input array $I = [I_1, ..., I_n]$ **IA** and and output array $O = [O_1, ..., O_m]$ **OA**, respectively. Having an $m \times n$ crossbar, **IA[j]** shows the variable assigned to the j$^{\text{th}}$ input where $1 \leq \text{j} \leq n$, and **OA[i]** shows the product assigned to the i$^{\text{th}}$ output where $1 \leq \text{i} \leq m$. To find a valid mapping, configuration process of a crosspoint switch in the i$^{\text{th}}$ row and the j$^{\text{th}}$ column should be as follows:

$$Conf(i,j) = \begin{cases} \text{activate}, & \text{if the variable } \textbf{IA[j]} \text{ is present in the product } \textbf{OA[i]}, \\ \text{deactivate}, & \text{otherwise.} \end{cases}$$

In case of having a fault-free crossbar, every assignment produces a valid mapping, so the configuration process is simple and straightforward. Defect-unaware approaches benefit from this feature by finding a defect-free sub-crossbar, so physical design is not troubled with the locations of defects. Figure 5 (a) shows an example.

In case of having faults, it is not guaranteed that an assignment produces a valid mapping. Figure 5 (b) shows an example. Configuration with the same input and output arrays as used for a fault-free crossbar, produces a different logic function since certain switches cannot be activated or deactivated. However, with using a defect-aware method one can implement the given function with a valid assignment. Figure 5 (c) shows an example.

In the following two sections, we consider defect-unaware and defect-aware algorithms targeting permanent faults or defects. In short, defect-unaware approaches aim to find a defect-free sub-crossbar, so the follow-up assignment procedure is straightforward and trouble-free. Defect-aware approaches aim to find valid input and output assignments using the full size crossbar by considering every defect in a crossbar.

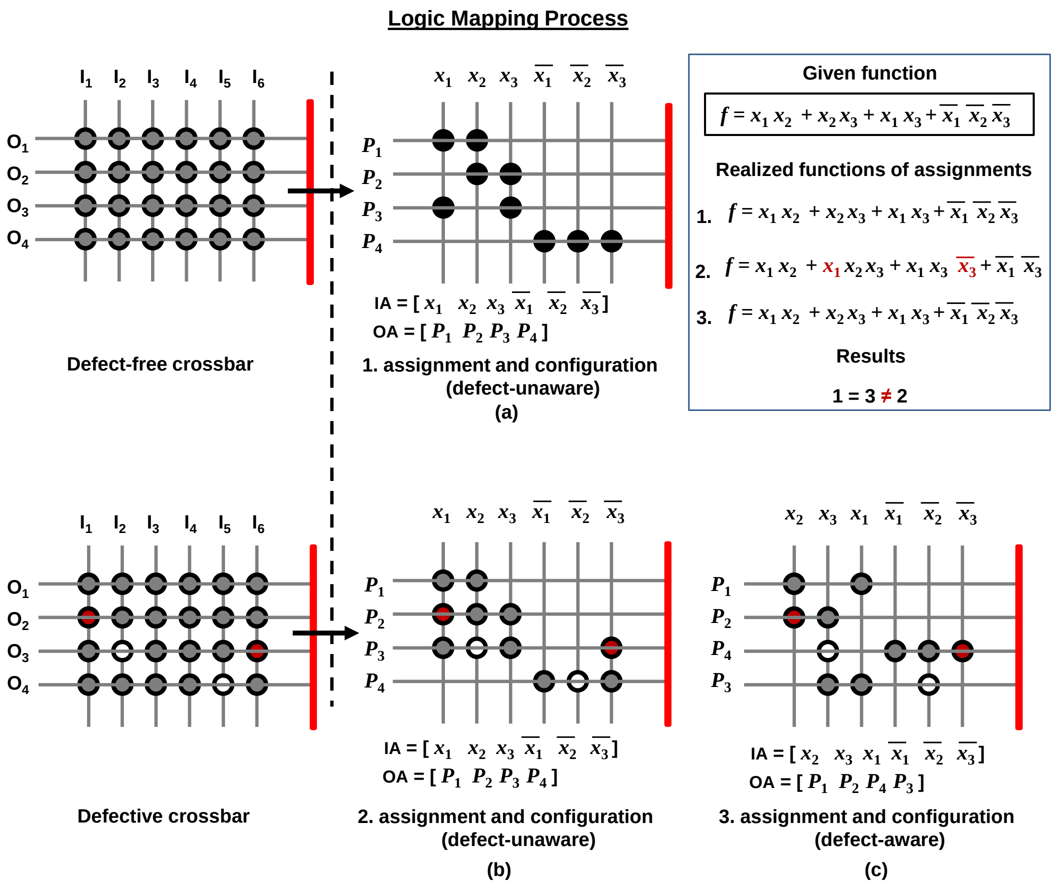

Logic Mapping Process
I1 I2 I3 I4 I5 I6
O1 O2 O3 O4
Defect-free crossbar
P1 P2 P3 P4
IA = [x1 x2 x3 x̄1 x̄2 x̄3]
OA = [P1 P2 P3 P4]
1. assignment and configuration (defect-unaware)
(a)
Given function
f = x1x2 + x2x3 + x1x3 + x̄1x̄2x̄3
Realized functions of assignments
1. f = x1x2 + x2x3 + x1x3 + x̄1x̄2x̄3
2. f = x1x2 + x1x2x3 + x1x3 x̄3 + x̄1x̄3
3. f = x1x2 + x2x3 + x1x3 + x̄1x̄2x̄3
Results
1 = 3 ≠ 2
Defective crossbar
IA = [x1 x2 x3 x̄1 x̄2 x̄3]
OA = [P1 P2 P3 P4]
2. assignment and configuration (defect-unaware)
(b)
IA = [x2 x3 x1 x̄1 x̄2 x̄3]
OA = [P1 P2 P4 P3]
3. assignment and configuration (defect-aware)
(c)

Fig. 5. Logic mapping process for (a) a defect-free crossbar corresponding to defect-unaware mapping, (b) a defective crossbar with direct mapping, and (c) a defective crossbar with defect-aware mapping.

4. DEFECT-UNAWARE LOGIC MAPPING

Defect-unaware logic mapping methods search for a defect-free $k \times k$ sub-crossbar in an $n \times n$ nano-crossbar using graph based algorithms. The main goal is to maximize yield or $(k/n)^2$. Since the obtained $k \times k$ sub-crossbar is defect-free, logic mapping process is straightforward afterwards. Let's first give the common concepts employed in the algorithms.

4.1. Definitions

(1) *Nano-crossbar* has vertical lines as inputs and horizontal lines as outputs. There is a configurable switch in every functional crosspoint. An example is shown in Figure 6 (a).

(2) *Bipartite graph representation* has two disjoint node sets; no edge exists between nodes in the same set. Elements of the node sets are represented by V and U showing output and input lines, respectively. A configurable switch in a crosspoint is shown with an edge connecting nodes from V and U. Connected nodes via edges are called *adjacent* and a *degree* of a node is the number of edges connected to the node. A stuck-at activated defect results in an erasure of the corresponding nodes from V and U as well as all edges connected to these nodes. A stuck-at deacti-

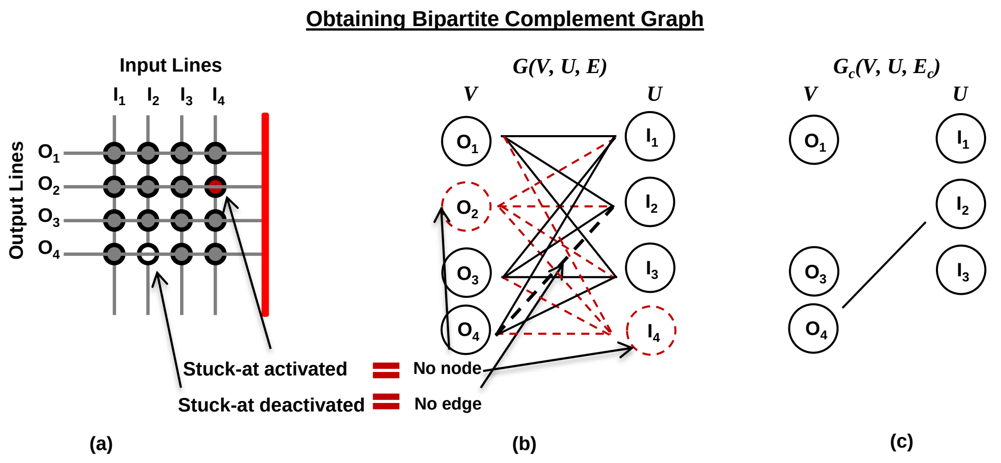


Fig. 6. Steps of obtaining a bipartite complement graph: (a) defective nano-crossbar, (b) bipartite graph representation and modification according to defects, and (c) bipartite complement graph.

vated defect results in an erasure of the corresponding edge. Figure 6 (b) shows a modified graph model in case of defects.

(3) *Bipartite complement graph* is the complement of a graph with keeping all of the nodes with an addition of all edges that do not exist in the original graph. An example is shown in Figure 6 (c).

(4) *Independent set* consists of nodes such that no node pair in the set has an edge connecting the nodes.

(5) *Biclique* is a subgraph of a bipartite graph such that every node has the maximum possible degree. Note that a defect-free sub-crossbar can be denoted with a biclique.

4.2. Algorithms

Finding a $k \times k$ sub-crossbar is equivalent to determining a balanced maximum biclique in a bipartite graph. Condition of being fully balanced ensures that dimensions (k inputs and k outputs) are equal to each other. The problem of obtaining a maximum biclique in a bipartite graph is shown to be an NP-hard in [Garey and Johnson 2002]. For this reason, heuristic algorithms are proposed to find sub-optimal solutions under reasonable time constraints.

The proposed algorithms formulate the problem as finding the maximum independent set in the complementary graph. Since only stuck-at deactivated defects in crosspoints are denoted with edges, degree of a node in the complementary bipartite graph is equal to the number of defects present in the corresponding line. If a node has a degree of zero, no edges with other nodes, it can be included to the independent node set immediately. Proposed algorithms focus on deciding which node to remove for efficiently obtaining zero-degree nodes. Outline of the algorithms in a modular composition is given in Algorithm 1.

Formulation of the problem as finding the maximum independent set in a complementary bipartite graph is first proposed by Tahoori in [Tahoori 2006]. It is observed that by removing nodes having maximum degrees, it is possible to discard lines having maximum number of defects. After the removal, degrees of the remaining nodes are updated and searching process is initialized again. Iterations are performed until either V or U becomes empty. A pseudocode of the algorithm is given in Heuristic 1. As can be seen from the code, the algorithm flips between V and U in order to assure a balanced condition by using the flag value. This way, a sub-crossbar is guaranteed

Algorithm 1 Defect-unaware Outline

```
1: Obtain $G_c(V, U, E_c)|E_c| = n^2 - |E|$                 ▷ Bipartite complement graph
2: $U^b \leftarrow \phi, V^b \leftarrow \phi, flag \leftarrow$ TRUE
3: repeat
4:     $U^b \leftarrow U^b \cup \{u|u \in U, d(u) = 0\}, U \leftarrow U - U^b$
5:     $V^b \leftarrow V^b \cup \{v|u \in U, d(v) = 0\}, V \leftarrow V - V^b$
6:
7:     HEURISTIC(t)                                             ▷ Preferred heuristic is called.
8:
9: until $U = \phi$ or $V = \phi$
10: return $U^b x V^b$ as the maximum biclique
```

Heuristic 1 [Tahoori, 2005]

```
1: if $flag$ then
2:     $u \leftarrow$ node in $U$ with maximum degree
3:     $U \leftarrow U - \{u\}$
4:     for each $v^{'} \in V$ such $(u, v^{'}) \in E_c$ do
5:         $d(v^{'}) \leftarrow d(v^{'}) - 1$
6:     end for
7:     Re-sort V accordingly
8: else
9:     $v \leftarrow$ node in $V$ with maximum degree
10:    $V \leftarrow V - \{v\}$
11:    for each $u^{'} \in U$ such $(u^{'}, v) \in E_c$ do
12:        $d(u^{'}) \leftarrow d(u^{'}) - 1$
13:    end for
14:    Re-sort $U$ accordingly
15: end if
16: $flag \leftarrow \neg flag$
```

Heuristic 2 [Yamani, 2007]

```
1: if flag then
2:     $v \leftarrow$ node in $V$ with minimum degree
3:     $\forall u \in U, w(u) \leftarrow 0$
4:     for each $v' \in V$ such $d(v) = d(v')$ that do
5:         for each $u \in U$ such $(u, v') \in E_c$ that do
6:             $w(u) \leftarrow w(u) + 1$
7:         end for
8:     end for
9:     $u' \leftarrow$ node in U with maximum $w$
10:    $U \leftarrow U - \{u\}$
11:    for each $v^{'} \in V$ such $(u', v^{''}) \in E_c$ that do
12:        $d(v^{''}) \leftarrow d(v^{''}) - 1$
13:    end for
14: else
15:    $u \leftarrow$ node in $U$ with minimum degree
16:    $\forall u \in U, w(u) \leftarrow 0$
17:    for each $u' \in U$ such $d(u) = d(u')$ that do
18:        for each $v \in V$ such $(u', v) \in E_c$ that do
19:            $w(u) \leftarrow w(u) + 1$
20:        end for
21:    end for
22:    $v' \leftarrow$ node in $V$ with maximum $w$
23:    $V \leftarrow V - \{v'\}$
24:    for each $u^{''} \in U$ such $(u^{''}, v') \in E_c$ that do
25:        $d(u^{''}) \leftarrow d(u^{''}) - 1$
26:    end for
27: end if
28: $flag \leftarrow \neg flag$
```

to be balanced by achieving a difference of 1 between its dimensions: $|U^b| - |V^b| = \pm 1$. Nevertheless, regarding the number of variables and products of given logic functions to be realized, restrictions can be relaxed.

The same problem formalization is also used in [Al-Yamani et al. 2007]. As a first step, the algorithm checks the nodes having minimum degrees. In the second step, the adjacent nodes to the checked ones are determined as candidates. As a final step, the candidates adjacent to most checked nodes are removed. The removal process increases the probability of obtaining zero-degree nodes. The pseudocode is given in Heuristic 2. This algorithm produces better results in terms of yield compared to the first algorithm in Heuristic 1. However, finding adjacent nodes increases the computational load of the algorithm.

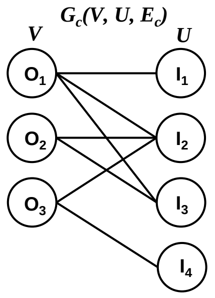
Fundamentally using the Yamani's approach, Yuan proposes two algorithms. In the first one, the node with the minimum degree is checked and the adjacent nodes are determined as candidates. At the end, the one with the maximum degree is chosen

Heuristic 3 [Yuan, 2011]

```
1:  if flag then
2:      $v \leftarrow$ node in $V$ with minimum degree
3:      for each $u \in U$ such $(u, v) \in E_c$ that do
4:          $U \leftarrow U - \{u\}$
5:          for each $v' \in V$ such $(u, v') \in E_c$ that do
6:              $d(v') \leftarrow d(v') - 1$
7:          end for
8:      end for
9:  else
10:     $u \leftarrow$ node in $U$ with minimum degree
11:     for each $v \in V$ such $(u, v) \in E_c$ that do
12:         $V \leftarrow V - \{u\}$
13:         for each $u' \in V$ such $(u', v) \in E_c$ that do
14:             $d(u') \leftarrow d(u') - 1$
15:         end for
16:     end for
17: end if
18: $flag \leftarrow \neg flag$
```

Heuristic 4 [Yuan, 2014]

```
1:  if flag then
2:      $v \leftarrow$ node in $V$ with minimum degree
3:      $U' \leftarrow \phi$
4:      for each $u \in U$ such $(u, v) \in E_c$ that do
5:          $U' \leftarrow U' \cup u$
6:      end for
7:      $u' \leftarrow$ node in $U'$ with maximum degree
8:      $U' \leftarrow U' - u$
9:      for each $v' \in V$ such $(u', v') \in E_c$ that do
10:         $d(v') \leftarrow d(v') - 1$
11:     end for
12: else
13:     $u \leftarrow$ node in $U$ with minimum degree
14:     $V' \leftarrow \phi$
15:     for each $v \in V$ such $(u, v) \in E_c$ that do
16:         $V' \leftarrow V' \cup v$
17:     end for
18:     $v' \leftarrow$ node in $V'$ with maximum degree
19:     $V' \leftarrow V' - v'$
20:     for each $u' \in U$ such $(u', v') \in E_c$ that do
21:         $d(u') \leftarrow d(u') - 1$
22:     end for
23: end if
24: $flag \leftarrow \neg flag$
```



(a)

Heuristics Node Removal Table

	1. Heuristic		2. Heuristic		3. Heuristic		4. Heuristic	
	Condition	Example	Condition	Example	Condition	Example	Condition	Example
Checked node	Max. degree node	O_1	Min. degree nodes	O_2, O_3	Min. degree node	O_2	Min. degree node	O_2
Candidates	The same node	O_1	Adjacent nodes	I_2, I_3, I_4	Adjacent nodes	I_2, I_3	Adjacent nodes	I_2, I_3
Removed node	The same node	O_1	Node adjacent to most checked nodes	I_2	Max. degree node	I_2	All adjacent nodes	I_2, I_3

(b)

Fig. 7. Node removal processes for 4 different heuristic algorithms: an example of a bipartite complement graph in (a) followed up by the removal process in (b).

to be removed [Yuan and Li 2011]. The pseudocode is given in Heuristic 3. In the second study, analyzing the major loop iterations, it is concluded that removing all of the adjacent nodes determined as candidates cuts down considerable number of iterations, decreases the runtime, and improves the yield marginally [Yuan and Li 2014]. The pseudocode is given in Heuristic 4. The only disadvantage of the approach is that balance condition is not guaranteed for the resulted biclique. In simulation results, they show that $|U^b| - |V^b| \leq \pm 3$. Summary of all four heuristics and their node removal preferences are shown in Figure 7.

4.3. Evaluation of Algorithms

The presented algorithms directly use a graph representation of a defective nano-crossbar as an input. They do not deal with defect types and their representations in graphs. Indeed, one can directly find a graph representation of a crossbar by erasing edges for stuck-at deactivated defects and nodes for stuck-at activated defects as previously explained in Section 4.1.

A real concern of defect-unaware methods is their considerably low yield or $(k/n)^2$, especially for high defect rates. When $n = 250$ and the fault rate is 15% that is a reasonable value for nano arrays [Chen et al. 2003], the best algorithm finds k values as high as 30. It means that only 1% of the crossbar can be used. This phenomena undermines the basic attraction of nano-crossbars offering superior density features.

Another concern is that all of the mentioned papers study only stuck-at deactivated defects except for [Tahoori 2006]. Tahoori shows that in case of having stuck-at activated defects that results in removal of all corresponding nodes, yield is impractically low.

All of the examined heuristic algorithms have polynomial runtime complexities. As long as crossbar size is under 1000×1000 and defect rates smaller than 15%, they run in a micro second domain that is quite satisfactory. Another interesting observation is that, the algorithms are immune to the differentiation of fault rates. In the next subsection, we conduct detailed simulations regarding their runtime and yield.

As a final note, defect-unaware methods in principle try to solve the problem of finding the maximum biclique in a bipartite graph. For this reason, similar problem formulations used in different fields such as those proposed in [Mubayi and Turán 2010] and [Yuan et al. 2015] can be directly applied to the defect-unaware methods especially for the improvement of yield.

4.4. Simulation Results of Algorithms

In this section, we present experimental results of the examined defect-unaware algorithms. We generate defective nano-crossbars with assigning an independent defect probability/rate to each crosspoint that shows a uniform distribution. All defects are modeled as configuration level defects either stuck-at activated or stuck-at deactivated that is the common tendency in the literature. Monte Carlo simulations are performed for assessment with a sample size of 200. We observe that fluctuating of parameter values stabilize nearly after this threshold value. All algorithms are implemented in MATLAB$^{\text{TM}}$. All experiments run on a 3.30GHz Intel Core i5 CPU (only single core used) with 4GB memory.

In order to evaluate the performance of the algorithms, two different parameters are used: *runtime* and *area yield*. Area yield is the ratio of the defect-free sub-crossbar size to the initial defective crossbar size. In simulations, two defect settings are used to evaluate the four proposed heuristics. For the first setting, we only consider stuck-at deactivated types with a corresponding defect rate of P_D having values of 5%, 10%, and 15%. In the second setting, stuck-at activated defects are also included with a corresponding defect rate $P_A = 1\%$ with $P_D = 10\%$. As a reminder, we define yield as $(k/n)^2$ with respect to k^2 being the size of a defect-free sub-crossbar and n^2 being the size of an initial defective crossbar.

Results of the first setting are given in Table III. Here, we select crossbar sizes up to 200×200 after which yield values become impratically low. The table shows that area yield values differ at most 6% and Heuristic 4 is the most efficient one in terms of runtime. However, yield values are still inadequate that kills the main advantage of using nano-crossbars having high area density. In the best case scenario, only 33% of a nano-crossbar can be utilized and it significantly decreases for larger crossbar sizes.

Table III. Runtime and Area Yield of Defect-unaware Algorithms with Stuck-at Deactivated Defects.

Initial Size	Defect Rate	Tahoori, 2006		Yamani, 2007		Yuan, 2011		Yuan, 2014	
		Heuristic 1		Heuristic 2		Heuristic 3		Heuristic 4	
		Yield	Run.(ms)	Yield	Run.(ms)	Yield	Run.(ms)	Yield	Run.(ms)
50 x 50	5%	33%	2	27%	2	27%	2	27%	2
	10%	12%	2	14%	3	16%	2	14%	1
	15%	9%	7	7%	3	7 %	2	10%	1
100 x 100	5%	16%	3	16%	6	16%	5	17%	3
	10%	6%	4	8%	7	8%	5	7%	1
	15%	3%	4	4%	9	4 %	6	4%	2
150 x 150	5%	9%	6	11%	10	10%	5	11%	8
	10%	4%	6	4%	10	4%	8	4%	6
	15%	1%	20	2%	10	3 %	8	2%	5
200 x 200	5%	6%	6	8%	10	7%	10	8%	10
	10%	3%	9	3%	10	3%	10	3%	9
	15%	1%	9	1%	10	1 %	10	1%	8

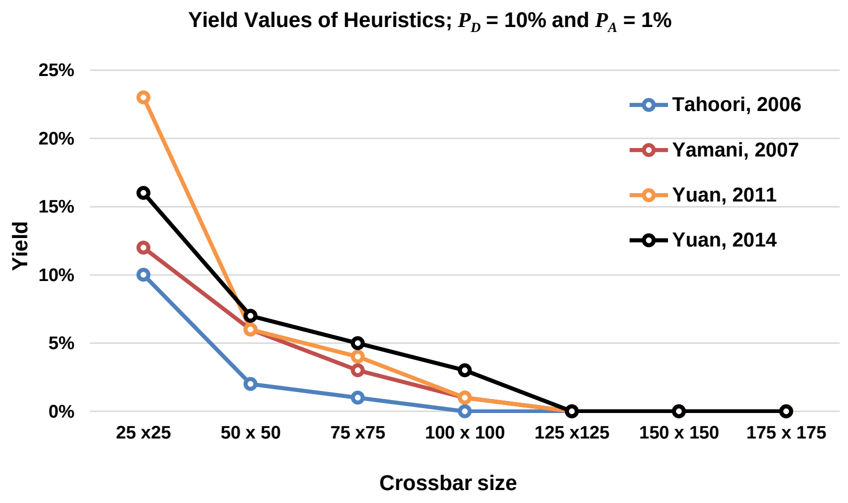


Fig. 8. Area yield results of defect-unaware algorithms; $P_D = 10\%$ and $P_A = 1\%$.

Results of the second setting are given in Figure 8. We see that defect-unaware methods is very vulnerable to stuck-at activated defects that cause eliminating both input and output lines as we previously explain in Section 4. Although we select a relatively low value of 1% for P_A, area yield values are not satisfactorily. Due to the very low yield values, we pursue no other experiments regarding stuck-at activated defects higher than 1%.

Table IV summarizes the general features of the algorithms by using four levels: poor, moderate, good, and excellent. Defect-unaware algorithms produce poor area yield results. Also, stuck-at activated defects severely decrease the already very low yield values. Their runtime results are satisfactory, but it is not not generally acceptable to discard all other features for a straightforward mapping process. Obviously, this inference is obtained by considering the existing methods; further increase of the yield would clearly make defect-unaware approaches much more attractive.

Table IV. Cheatsheet of Defect-unaware Algorithms

Author	**Algorithm Features**			
	Multiple-type Defects	**Area Yield**	**Runtime**	**Area Scalability**
Tahoori, 2006	Poor	Poor	Good	Good
Yamani, 2007	Poor	Poor	Moderate	Good
Yuan, 2011	Poor	Poor	Moderate	Good
Yuan, 2014	Poor	Moderate	Excellent	Excellent

Table V. Timeline of Defect-aware Logic Mapping Methodologies

Date	**Author**	**Contribution**
2004	Naemi	Graph based greedy matching [Naeimi and DeHon 2004]
2004	Hogg	Graph monomorphism [Hogg and Snider 2004]
2006	Rao	Graph embedding [Rao et al. 2006]
2008	Yellabalase	Clustered Defects [Yellambalase and Choi 2008]
2008	Polia	Hardening techniques [Polian and Rao 2008]
2009	Su	Runtime analysis of logic mapping [Su and Rao 2009]
2009	Simsir	Partial graph constructing [Simsir et al. 2009]
2009	Zheng	Logic mapping with satisfiability [Zheng and Huang 2009]
2011	Goren	Graph canonization and radix sorting [Gören et al. 2011]
2011	Yang	Integer linear programming [Yang and Datta 2011]
2011	Su	Logic morphing [Su and Rao 2011]
2014	Yuan	Graph based fitness approximation [Yuan et al. 2014]
2014	Su	Logic morphing and hardening [Su and Rao 2014]
2017	Tunali	Matrix based greedy backtracking [Tunali and Altun 2017]

5. DEFECT-AWARE LOGIC MAPPING

Defect-aware approaches employ defects present in a nano-crossbar during the configuration process to map a given logic function. It is indicated that mapping a logic function on a defective nano-crossbar array is an NP-complete problem [Shrestha et al. 2009]. The problem is directly equivalent to finding a subgraph isomorphism between graph representations of a logic function and a nano-crossbar. However, in formalization of the problem, a variety of different approaches are adopted to fasten the process, especially for high defect rates. As a common practice, maximum bipartite matching and graph based heuristics are used to lighten the computational load [Naeimi and DeHon 2004], [Rao et al. 2009], [Simsir et al. 2009], and [Yuan et al. 2014]. A different approach using matrix representations of a given logic function and a defective nano-crossbar is used with row by row matching [Gören et al. 2011] and [Tunali and Altun 2017]. Another method is integer linear programming with transforming defects as constraints [Yang and Datta 2011] and [Zamani et al. 2013]. A comprehensive and chronological list of the main papers and their contributions to the defect tolerant logic mapping problem is shown in Table V.

Concerning the different types of defects, we see an overwhelming tendency in the literature with only considering stuck-at deactivated types (different names used in the literature are non-programmable, stuck-at 0, stuck-open) [Naeimi and DeHon 2004], [DeHon and Naeimi 2005], [Rao et al. 2006], [Yellambalase and Choi 2008], [Simsir et al. 2009], [Zamani et al. 2013], [Yuan et al. 2014], and [Tunali and Altun 2017]. The only exceptions are [Zheng and Huang 2009], [Gören et al. 2011], and [Su and Rao 2014] that analyze the occurrence of multiple-type defects. As follows, we separately explain the main approaches in subsections followed by their evaluations.

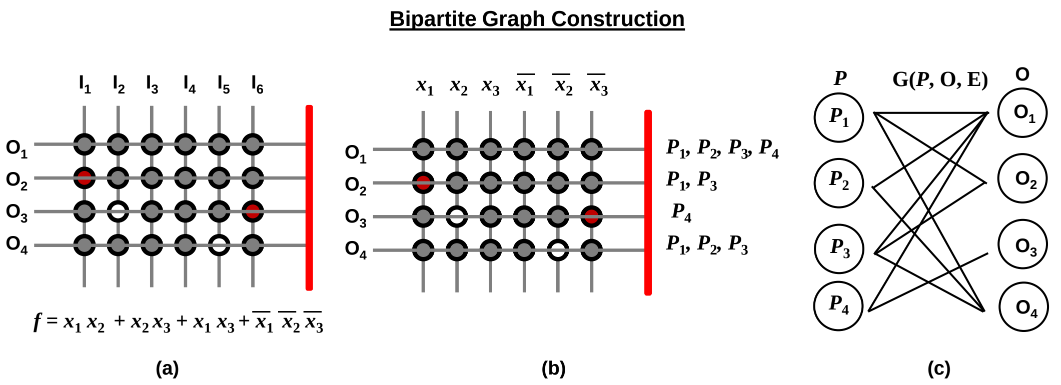

Bipartite Graph Construction
I_1 I_2 I_3 I_4 I_5 I_6
O_1 O_2 O_3 O_4
$f = x_1 x_2 + x_2 x_3 + x_1 x_3 + \overline{x_1}\,\overline{x_2}\,\overline{x_3}$
(a)
x_1 x_2 x_3 $\overline{x_1}$ $\overline{x_2}$ $\overline{x_3}$
P_1, P_2, P_3, P_4
P_1, P_3
P_4
P_1, P_2, P_3
(b)
P
G(P, O, E)
O
P_1 P_2 P_3 P_4
O_1 O_2 O_3 O_4
(c)

Fig. 9. Steps of bipartite graph construction: (a) a given logic function and a crossbar, (b) an input assignment and finding which product can be mapped to which output, and (c) constructing the graph.

Algorithm 2 Naeimi, 2004

```
1:  Sort $P_i$'s in decreasing order regarding fan-in.      ▷ Fan-in is # of variables in a product
2:  while $P \neq \emptyset$ do
3:      Choose the first $P_i$
4:      while $P_i$ is not matched and $O$ has non-visited $O_i$'s do
5:          Choose a random $O_i \in O$
6:          if valid mapping exist then
7:              mark $(P_i, O_i)$ matched
8:              remove $P_i$ from $P$ and $O_i$ from $O$
9:          else
10:             set $O_i$ visited by $P_i$
11:         end if
12:     end while
13: end while
```

5.1. Maximum Bipartite Matching

In a defective nano-crossbar, arbitrary assignment of products of the given function to the crossbar outputs might result in an error as previously visualized in Figure 5. To determine the erroneous cases, one can use a bipartite graph having product (P) and output (O) nodes. If there is an edge between a product and an output, then the product can be mapped to the output. Thus, all possible mapping configurations can be represented by edges. An example is given in Figure 9. After the graph construction, finding a maximum or perfect matching that corresponds to a set of edges such that every node is incident to exactly one edge, would yield a valid mapping. For this purpose, exact algorithms can be used including Ford-Fulkerson maximum flow network [Ford Jr and Fulkerson 1955] and Hopcroft-Karp algorithm [Hopcroft and Karp 1973]. However, constructing a bipartite graph is a costly process especially for high defect rates seen in nano-crossbars, so certain heuristics are proposed.

Naeimi proposes a greedy heuristic algorithm without constructing a bipartite graph; instead he uses expected values of node degrees [Naeimi and DeHon 2004]. Since a product with a high number of variables (fan-in) are harder to map, its node degree would be smaller. The algorithm starts matching products in a decreasing order of fan-in's with choosing random output nodes. Naemi's approach is fairly competent in terms of scalability. A pseudocode of the algorithm is given in Algorithm 2.

Simsir also uses a heuristic algorithm with partially constructing the bipartite graph [Simsir et al. 2009]. Firstly, variables from most common to least common (fan-out values) are assigned to least defective to most defective input lines, respectively. This process is named as pin assignment. Secondly, a distinct edge is found for every prod-

Algorithm 3 Simsir, 2009

```
1: Input Assignment :   ▷ Fan-out is # of products a variable is present
2: Inputs to variables decreasing order of defects and increasing order of fan-outs
3: while $P \neq \emptyset$ do
4:     choose a $P_i$
5:     sort $O$ nodes with increasing number of edges
6:     check mapping until one found
7:     if no mapping is available then
8:         return no matching
9:     elseinsert an edge between the $P_i$ and $O_t$ node and remove $P_i$ from $P$
10:    end if
11:    if $O_t$ node has other edges then
12:        add all neighbor $P_j$'s of $O_t$ to $P$
13:    end if
14: end while
15: repeat
16:    check for a maximum bipartite matching
17:    if matching exist then
18:        return matching information
19:    end if
20:    choose a $P_i$ node and check a mapping in $O$
21:    if no new mappings are available then
22:        goto until
23:    else
24:        add all neighbor $P_j$'s of $O_t$ to $P$
25:    end if
26: until all possible mappings are checked or $P = \emptyset$
27: restart the search with another input assignment
```

Algorithm 4 Yuan, 2014

```
1: $I_i$ = random input permutation
2: $f(I_i)$ = EXACT_MBM_EVALUATION($I_i$)   ▷ $f$ gives fitness value
3: t = 0
4: repeat
5:     t = t + 1
6:     for i = 1 to N do
7:         select two parents $I_j, I_k$ from $I$ randomly
8:         $B_i$ = CROSSOVER($I_k, I_k, P_{cross}$)   ▷ $P_{cross}$ Probability of crossover
9:     end for
10:    for i = 1 to N do
11:        $B_i$ = MUTATION($B_i, P_{mut}$)   ▷ $P_{mut}$ Probability of mutation
12:        $B_i$ = GREEDY REASSIGNMENT($B_i, P_{ls}, \lambda$)   ▷ $P_{ls}$ Probability of local search
13:    end for
14:    for i = 1 to N do
15:        if $t\%\Delta == 0$ then ▷ $\Delta$ exact evaluation gap
16:            $f(B_i)$ = EXACT_MBM_EVALUATION($B_i$)
17:        else
18:            $f(B_i)$ = APPROXIMATED_MBM_
19: EVALUATION($B_i$)
20:        end if
21:    end for
22:    $I$ = SELECTION_FOR_SURVIVAL($I, B$)
23: until runtime reached or a valid mapping founded
```

uct node and then an exact algorithm is performed to find a maximum matching. In case no matching is found, an extra edge is searched for product nodes of bipartite graph and the exact algorithm is performed at the end. This process continues until a valid matching is found or another pin assignment is tried in case of no matching. A pseudocode of the algorithm is given in Algorithm 3.

Yuan proposes a memetic algorithm with fitness approximation [Yuan et al. 2014]. Unlike Simsir's approach, an initial random input assignment is made and fitness of the assignment is evaluated with an objective function f. Ford-Fulkerson's maximum flow method is mainly used for finding a maximum bipartite matching. Furthermore while searching for a matching, a greedy reassignment is performed by changing the input assignment for better fitness. Greediness factor (λ) of the method determines the number of input assignments to be changed. In addition, for every 10 trials (chosen as an exact evaluation gap Δ in the paper), an approximate matching algorithm is used for the first 9 and an exact matching algorithm is used just for the last one that is very similar to Naeimi's greedy method. A pseudocode of the algorithm is given in Algorithm 4.

5.2. Matrix Matching

A logic function and a defective nano-crossbar can be both denoted with matrices similar to incident matrices of graphs. Figure 10 (a) and (b) respectively show a function

Matrix Representation

Function Matrix (FM)

$f = x_1 x_2 + x_2 x_3 + x_1 x_3 + \overline{x_1}\,\overline{x_2}\,\overline{x_3}$

$$\begin{array}{c|cccccc} & x_1 & x_2 & x_3 & \overline{x_1} & \overline{x_2} & \overline{x_3} \\ \hline P_1 & +1 & +1 & -1 & -1 & -1 & -1 \\ P_2 & -1 & +1 & +1 & -1 & -1 & -1 \\ P_3 & +1 & -1 & +1 & -1 & -1 & -1 \\ P_4 & -1 & -1 & -1 & +1 & +1 & +1 \end{array}$$

+1 : Variable present
- 1 : Otherwise

(a)

Crossbar Matrix (CM)

$$\begin{array}{c|cccccc} & \mathrm{I}_1 & \mathrm{I}_2 & \mathrm{I}_3 & \mathrm{I}_4 & \mathrm{I}_5 & \mathrm{I}_6 \\ \hline \mathrm{O}_1 & 0 & 0 & 0 & 0 & 0 & 0 \\ \mathrm{O}_2 & +1 & 0 & 0 & 0 & 0 & 0 \\ \mathrm{O}_3 & 0 & -1 & 0 & 0 & 0 & +1 \\ \mathrm{O}_4 & 0 & 0 & 0 & 0 & -1 & 0 \end{array}$$

+1 : Stuck-at activated defect
0 : Configurable switch
- 1 : Stuck-at deactivated defect

(b)

Compatibility Table

FM	CM	Matching
+1	+1	Yes
+1	0	Yes
-1	0	Yes
-1	-1	Yes
+1	-1	No
-1	+1	No

(c)

Fig. 10. Matrix representations of (a) logic function and (b) defective nano-crossbar, and (c) compatibility table of elements.

matrix (FM) and a crossbar matrix (CM) of the logic function and the defective nano-crossbar previously used in Figure 9. If we define which elements of logic and crossbar matrices can be matched, then it is possible to decide a valid mapping between a product and an output by checking corresponding rows. Figure 10 (c) demonstrates a compatibility table for matching. The key idea behind matrix matching is to make two matrices easily matchable by assigning proper elements for defects and variables. As follows, we examine two studies.

In the first study, Goren appoints k-neighbor values to all rows and columns individually [Gören et al. 2011]. After determining the values, rows and columns are sorted in ascending order according to the k-neighbor values. A 1-neighbor value of a row or a column of an FM (CM) is the number of +1's (0's and +1's) in the corresponding row or column. A 2-neighbor value of a row of an FM (CM) is the sum of 1-neighbor values of the columns having intersections of +1's (0's and +1's) with the row. Following the same logic, in order to find a k-neighbor value of a row of an FM (CM), we add (k-1)-neighbor values of the columns if they are +1 (0 or +1). The same procedure is applied for columns. An example for finding 2-neighbor values is shown in Figure 11 (a).

After the initial operations, a two-dimensional sorting is applied to rows and columns of matrices. Every row/column is regarded as a k-ary number (radix or base equals to k) with most to least significant bits (MSB and LSB) being arranged from left to right and from top to bottom. In Figure 11 (b) and (c), manipulation of the FM and CM is shown according to the radix values. In order to sort rows and columns, radix sort algorithm is used in ascending order. Starting with the rows, radix sort begins with the LSB and sorts the rows and moves to the next bit (next column in our context) until it reaches the MSB. The same process is applied to the columns as well. This interleaving sorting continues till a stable sort is obtained. Important point is that, since an FM and a CM has two and three different elements, respectively, radix being 2 and 3 is applied.

After all of these sorting processes are finalized, the matrices are matched row by row using the element compatibility as previously given in Figure 10 (c). The pseudocode is given in Algorithm 5.

In the second study, Tunali first sorts matrix columns according to the number of compatible elements and rows with placing most defective rows at the top of the crossbar matrix [Tunali and Altun 2017]. Then a greedy row by row matching using

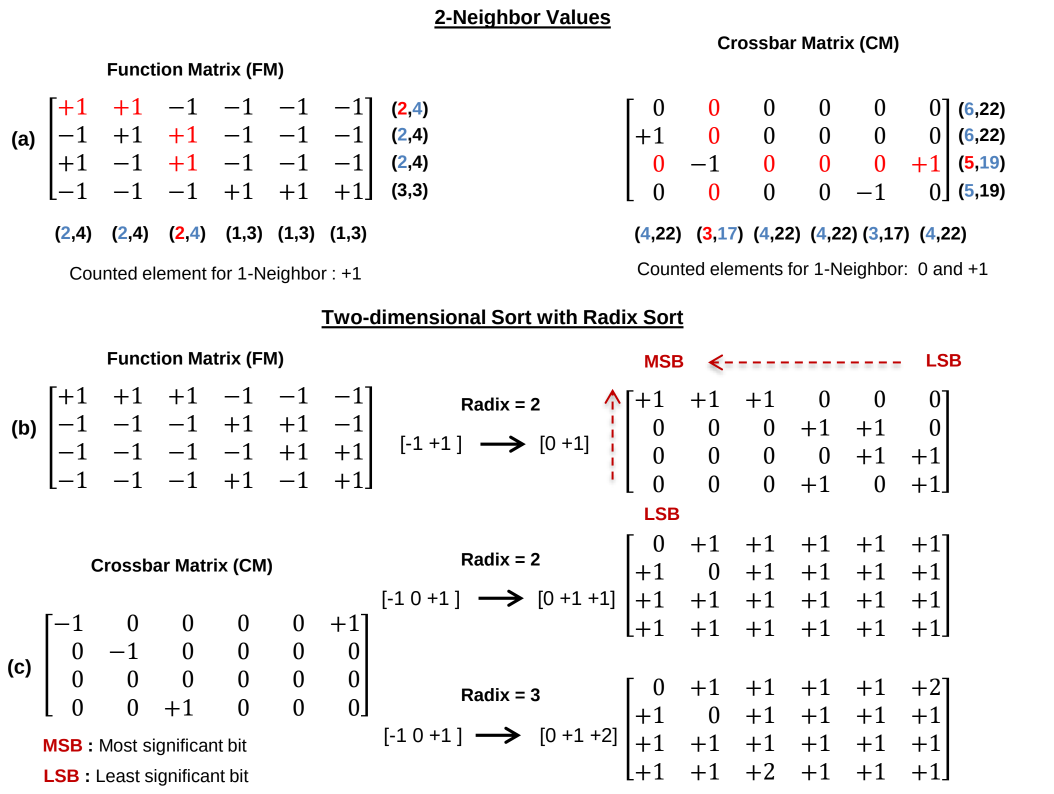


Fig. 11. Two dimensional sort with radix sort: (a) k-neighbor values of function and crossbar matrices, (b) function matrix with radix = 2 values, and (c) crossbar matrix with radix = 2 and radix = 3 values.

Algorithm 5 Goren, 2011

```
1: FM = KNS(FM)                                      ▷ K-neighbor sort
2: CM = KNS(CM)
3: $FM_{canon1} = 2DS_{radix2}(FM)$                  ▷ Two dimensional sort with radix sort
4: $CM_{canon2} = 2DS_{radix2}(CM)$
5: $CM_{canon3} = 2DS_{radix3}(CM)$                  ▷ Radix = 3 since CM has 3 different elements
6: $CM_{canon2.5} = 2DS_{radix2.5}(CM)$              ▷ First radix =2 and then radix = 3 sort is applied
7: Return Best()
8:         $RowMatch(FM_{canon1}, CM_{canon2})$
9:         $RowMatch(FM_{canon1}, CM_{canon3})$
10:        $RowMatch(FM_{canon1}, CM_{canon2.5})$
```

Hadamard multiplication is applied with backtracking excluding previously matched rows. Since most defective rows are at the top, harder matchings are eliminated in the beginning of the process. If no matching is found for a row even with backtracking, column permutation is altered and row by row matching is initialized again. It should be noted that, greedy row by row matching discard the overhead of the bipartite graph construction. The pseudocode is given in Algorithm 6.

5.3. Graph Embedding

Rao proposes a recursive algorithm with heuristics to prune impossible mappings by denoting both a given logic function and a crossbar with bipartite graphs [Rao et al. 2009]. The algorithm, called as Embed, explores solution space with using fanout,

Algorithm 6 Tunali, 2017

```
1: FM = INDEX_SORT(FM)    ▷ row and column sorting
2: CM = INDEX_SORT(CM)
3: repeat
4:     matchedRows ={}
5:     for i = 1 to N do
6:         flag = false;
7:         F_i ← i^th row of FM
8:         for t = 1 to N and C_t ∉ matchedRows do
9:             Try matching F_i with C_t
10:            if a matching exist then
11:                flag = true;
12:                break
13:            end if
14:        end for
15:        if ¬matching then
16:            flag = BACKTRACKING(Row_i)
17:        end if
18:        if ¬flag then
19:            change input permutation
20:            break
21:        end if
22:    end for
23: until valid mapping found or permutation limit is reached
```

Algorithm 7 Rao,2009

```
1: if $N_1 = \emptyset$ then    ▷ No more unmapped nodes in $B_1$
2:     return success
3: end if
4: if $N_2 = \emptyset$ then    ▷ No more unmapped nodes in $B_1$ to map $N_1$, thus fail
5:     return fail
6: end if
7: Select one unmapped node $n_1 \in N_1$
8: if $n_1 \in N_{var}$ then
9:     repeat step (a) through (d) for every node
10: else$n_1 \in N_{prod}$    ▷ Variable → unmapped vertical wires
11:     repeat step (a) through (d) for every node
12:         (a) justify $n_1 \rightarrow n_1$ if $n_1$ can be mapped to $n_2$ mark $n_1 \rightarrow n_2$ else repeat with a different $n_2$
13:         (b) Embed $(N_1 - n_1, N_2 - n_2)$
14:         (c) If (success) return success
15:         (d) Else(fail)
16:     Return fail
17: end if
```

fanout-fanout, and fanout-chain heuristics. Since a graph model is used, fanout of the logic function is shown with node degrees; graphs $G_1(N_{var}, N_{prod}, E_1)$ for the function and $G_2(N_{vert}, N_{hor}, E_2)$ for the crossbar are defined. The Embed algorithm finds a matching between these graphs. Fanout heuristic compares the degrees of the nodes to eliminate impossible mappings. Fanout-fanout heuristic checks the degrees of the separate and connected nodes to discard impossible mappings. Fanout-chain heuristic constructs all possible node-pair connections to conclude if there is a matching. Using backtracking, nodes with insufficient degrees are eliminated. All these heuristics are implemented in the 12. step of the pseudocode shown in Algorithm 7.

5.4. ILP Based Algorithms

Yang and Zamani propose to transform the logic mapping problem into a constrained integer linear programming (ILP) problem in [Yang and Datta 2011] and [Zamani et al. 2013]. To obtain a valid mapping, constraint equations are derived. Parameters X and Y respectively show the matching status of the rows corresponding to function product and crossbar output, and columns corresponding to function variable and crossbar input. If a product P_1 can be matched with an output O_1, then $X_{P_1\_O_1}$ becomes 1; otherwise $X_{P_1\_O_1} = 0$. The same condition is defined for Y for column matching.

In order to map a k_1 x k_2 function matrix to n_1 x n_2 crossbar matrix following constraints need to be met. If the i^{th} row (product P) of a function matrix can be matched with the j^{th} row (output O) of a crossbar matrix, then $X_{i\_j} = 1$; otherwise $X_{i\_j} = 0$. For a valid row mapping two conditions are imposed: 1) Each row in the function matrix should be matched with only a single row in the crossbar matrix; and 2) Each row in the crossbar matrix should be matched at most one row in the function matrix. The following constraint (1) formalize these conditions. Same procedures are applied for column matching as given in a constraint (2).

$$\sum_{j=1}^{n_1} X_{i\_j} = 1 \text{ for each } i = 1, ..., k_1 \qquad \sum_{i=1}^{k_1} X_{i\_j} \leq 1 \text{ for each } j = 1, ..., n_1 \tag{1}$$

$$\sum_{j'=1}^{n_2} Y_{i'\_j'} = 1 \text{ for each } i^{'} = 1, ..., k_2 \qquad \sum_{i'=1}^{k_2} Y_{i'\_j'} \leq 1 \text{ for each } j^{'} = 1, ..., n_2 \tag{2}$$

One more parameter is introduced $Z_{i\_j,i'\_j'}$ such that it takes the value of 1 if $X_{i\_j} = Y_{i'\_j'} = 1$; otherwise $Z = 0$. In order to find a valid mapping, every element of a function matrix requires a valid mapping. The below constraint (3) ensures this condition.

$$\sum_{i=1}^{k_1} \sum_{j=1}^{n_1} \sum_{i'=1}^{k_2} \sum_{j'=1}^{n_2} Z_{i\_j,i'\_j'} = i \times j \tag{3}$$

By using these 3 constraints, a valid mapping can be found. An important point is that the proposed algorithm helps to prune solution space and decrease the computation time by finding impossible matchings corresponding to zero valued X's and Y's.

5.5. Evaluation of Algorithms

It is reasonable to assume that defect rates of nano-crossbars is high up to 20% that dramatically elevate the computational load of the algorithms. Because of that, runtime parameter of the algorithms can be considered as the major factor in evaluations. Generally, ILP and graph embedding approaches yield poor runtime results. Especially, recursive nature of the embedding algorithm causes impractical runtimes for larger crossbar sizes. On the other hand, heuristic nature of maximum bipartite matching approaches provides an upper hand in terms of runtime. Nevertheless constructing the bipartite graph is very time consuming. Also, if the graph is rather sparse meaning that a few edges are present for possible matchings, exact algorithms need to be applied that is also time consuming. As for matrix matching approaches, runtime results are fairly satisfactory for even larger size logic functions. However, since matching is advancing through one dimension (rows), increase of the column size drastically worsens the success rate of the algorithms.

Considering the yield, maximum bipartite matching and matrix matching based algorithms produce the best results. However, a prevalent trend in the literature is realizing a logic function with a larger size crossbar, generally 1.5 times larger, than the optimal size. Therefore, yield analysis is not conducted extensively for most of the studies.

Another important point is the algorithms' capability of handling multiple-type defects. As we previously mention, only [Gören et al. 2011] fine-tunes the algorithm according to defect types. The rest of the papers only consider stuck-at deactivated types, so their proposed heuristics perform under this restriction. If multiple-type defects were to be considered, bipartite graphs would be sparser (lower possibility of matching between products and outputs) that makes harder to find a maximum bipartite matching. This problem is also applicable for graph embedding and ILP based methods.

As a final note, defect-aware methods use a wide range of different problem formalizations not necessarily proposed for nano-crossbar arrays. Therefore studies related to subgraph isomorphism [Cordella et al. 2004] and assignment problems [Chu and Beasley 1997] can be directly applied to the defect-aware methods.

5.6. Simulation Results of Algorithms

The same system and simulation settings of defect-unaware methods are also used in this section. Unlike defect-unaware algorithms for which the algorithms' performance are independent of the implemented functions, defect-aware algorithms give different results for different functions. Therefore, we use standard benchmark circuits presented in [McElvain 1993].

In order to evaluate the performance of the algorithms, four different parameters are used: *success rate*, *runtime* and its *standard deviation*, and *area yield*. Area yield is defined as the ratio of the optimal size defect-free array to the defective crossbar size adequate to realize a given benchmark function. For example, a logic function with 4 variables and 10 products would require an optimal crossbar size of 10×8 (10 for outputs and 8 for inputs with 4 variables and 4 negated forms).

We also introduce a new parameter *logic inclusion ration* (IR) to help us to form a more intuitive understanding of the mapping problem. The number of switching crosspoints adequate to realize a logic function is denoted by IR in the form of percentage. For example, if the optimal crossbar size to implement a logic function is 10×10 and IR = 40%, then it means that a logic function has a literal count of 40 and we need 40 switching crosspoints to implement the function. Since we can use stuck-at activated defects as switching crosspoints, higher IR values ease the tolerance of stuck-at activated defects. Oppositely, lower IR values are preferred for the tolerance of stuck-at deactivated defects. Therefore, it is possible to categorize the easiness of the tolerance or mapping problem by considering the values of IR$/P_A$ and (1-IR)$/P_D$ for stuck-at activated and deactivated defects, respectively. As an empirical measure, if these values below three, then the sample space shrinks significantly that makes the problem very hard. We state this threshold phenomena as an experimental tendency based on our observations during the simulations of mapping trials rather than a strict compliance. Different defect distributions such as clustered, or a different set of benchmark functions might certainly produce a different threshold. Therefore, when we say easier or harder to solve, reader should understand the categorization context in terms of our experimental settings.

We use three defect settings. For the first and the second ones, benchmark functions are respectively mapped to optimal ($n/n = 100\%$ yield) and 1.5 larger size ($(n/(1.5 \times n))^2 \approx 44\%$ yield) crossbars with $P_D = 15\%$. For the third setting, 1.5 larger size crossbars are used with $P_D = 10\%$ and $P_A = 5\%$. We choose those three settings to evaluate the algorithms' response to stricter area conditions and multiple-type defects.

On the selection of the approaches previously given in the subsections of Section 5, we have excluded satisfiability, graph embedding and integer linear programming based ones from simulations. The reason behind that, SAT approach is already shown to be inferior to matrix model in [Gören et al. 2011]. In addition, graph embedding algorithm adopts a recursive characteristic and is also demonstrated to be inferior to maximum bipartite matching in [Yuan et al. 2014]. Finally, ILP needs to cope with very large number of constraint equations that requires a drastic computational operations.

5.6.1. Graph Based Approaches. We start our experiments with optimal size crossbars that result in $\%100$ area yield, and $P_D = 15\%$. In terms of average runtime and its deviation, Naemi's algorithm always produces the best results. The other two methods sometimes show runtime deviations higher than average runtimes. Considering the success rate, Yuan's and Simsir's algorithms are clearly superior. Random nature of Naeimi's approach causes an issue when the constraints prune the solution space severely. However, Yuan's and Simsir's algorithms are not able find a valid mapping for larger size examples, so scalability is an issue. Results are given in Table VI.

Table VI. Runtime and Success Rate Comparison of Graph based Algorithms using Optimal Size Crossbars; $P_D = 15\%$.

Benchmarks			**Naemi, 2004**			**Simsir, 2009**			**Yuan, 2014**		
Name	Size	IR	Psucc	Run.(ms)	Std	Psucc	Run.(ms)	Std	Psucc	Run.(ms)	Std
5xp1	75×14	28%	42%	3	1	100%	269	21	100%	17	2
inc	34×14	40%	0%	3	1	56%	3262	1871	100%	109	166
clip	167×18	29%	10%	12	3	100%	5934	618	100%	130	4
misex2	40×29	12%	43%	2	0	100%	45	10	100%	9	3
9sym	87×18	33%	0%	11	2	100%	7024	19236	100%	38	21
bw	65×10	35%	5%	4	1	92%	11657	13159	100%	107	172
rd53	32×10	42%	3%	2	1	84%	962	1612	100%	7	2
t481	481×32	30%	0%	517	86	-	-	-	-	-	-
alu4	1028×28	27%	2%	235	34	-	-	-	-	-	-
misex3	1848×28	34%	0%	6758	760	-	-	-	-	-	-
table3	645×28	40%	0%	2348	187	-	-	-	-	-	-
apex4	1732×18	47%	0%	3674	415	-	-	-	-	-	-
rd84	411×16	50%	0%	673	60	-	-	-	-	-	-

Table VII. Runtime and Success Rate Comparison of Graph based Algorithms using 1.5 Larger Size Crossbars; $P_D = 15\%$.

Benchmarks			**Naemi, 2004**			**Simsir, 2009**			**Yuan, 2014**		
Name	Size	IR	Psucc	Run.(ms)	Std	Psucc	Run.(ms)	Std	Psucc	Run.(ms)	Std
5xp1	75×14	28%	100%	3	1	100%	459	21	100%	27	1
inc	34×14	40%	99%	1	0	100%	1191	1227	100%	8	4
clip	167×18	29%	100%	12	1	100%	7934	440	100%	210	2
misex2	40×29	12%	93%	2	0	100%	58	4	100%	7	0
9sym	87×18	33%	100%	4	0	100%	656	30	100%	42	0
bw	65×10	35%	100%	2	0	100%	3610	5511	100%	16	0
rd53	32×10	42%	98%	1	0	100%	27	1	100%	7	3
t481	481×32	30%	100%	58	5	-	-	-	-	-	-
alu4	1028×28	27%	100%	102	10	-	-	-	-	-	-
misex3	1848×28	34%	100%	304	21	-	-	-	-	-	-
table3	645×28	40%	96%	138	18	-	-	-	-	-	-
apex4	1732×18	47%	100%	25	21	-	-	-	-	-	-
rd84	411×16	50%	95%	46	6	-	-	-	-	-	-

In our second experiment, we loosen the area restrictions and use 1.5 larger size crossbars that results in $\%44$ area yield. Here, Naemi's algorithm is the clear winner for all of the performance parameters. Only exception is that, Yuan's approach produces better success rates for the benchmarks "inc" and “"misex2". In addition, its runtime deviation is relatively stabilized. Results are given in Table VII.

In our last experiment, once more we use 1.5 larger size crossbars with $P_D = 10\%$ and $P_A = 5\%$. Here, Naemi's algorithm is again the clear winner for all of the performance parameters. Only exception is that, Yuan's approach is the only one being able to find a valid mapping for the benchmark "misex2". Results are given in Table VIII.

Table VIII. Runtime and Success Rate Comparison of Graph based Algorithms using 1.5 Larger Size Crossbars; $P_D = 10\%$ and $P_A = 5\%$.

Benchmarks			**Naemi, 2004**			**Simsir, 2009**			**Yuan, 2014**		
Name	Size	IR	Psucc	Run.(ms)	Std	Psucc	Run.(ms)	Std	Psucc	Run.(ms)	Std
5xp1	75×14	28%	99%	6	1	96%	13890	23540	100%	25	3
inc	34×14	40%	77%	2	0	96%	709	1390	100%	4	0
clip	167×18	29%	98%	14	3	-	-	-	100%	190	0
misex2	40×29	12%	0%	18	3	0%	6570	1470	60%	6590	1200
9sym	87×18	33%	98%	9	1	-	-	-	100%	42	2
bw	65×10	35%	99%	3	0	100%	4350	11090	100%	16	0
rd53	32×10	45%	100%	1	0	100%	94	250	100%	3	0
t481	481×32	30%	0%	986	200	-	-	-	-	-	-
alu4	1028×28	27%	6%	550	150	-	-	-	-	-	-
misex3	1848×28	34%	99%	637	40	-	-	-	-	-	-
table3	645×28	40%	0%	1138	28	-	-	-	-	-	-
apex4	1732×18	47%	100%	251	23	-	-	-	-	-	-
rd84	411×16	50%	100%	37	3	-	-	-	-	-	-

Table IX. Runtime and Success Rate Comparison of Matrix based Algorithms using Optimal Size Crossbars; $P_D = 15\%$.

Benchmarks			**Goren, 2011**			**Tunali, 2017**		
Name	Size	IR	Psucc	Run.(ms)	Std	Psucc	Run.(ms)	Std
5xp1	75×14	28%	100%	20	7	100%	0	0
inc	34×14	40%	5%	280	60	98%	57	170
clip	167×18	29%	100%	70	6	100%	1	0
misex2	40×29	12%	4%	400	80	100%	1	1
9sym	87×18	33%	74%	310	489	100%	1	3
bw	65×10	35%	87%	90	210	100%	3	4
rd53	32×10	45%	64%	78	98	100%	1	2
t481	481×32	30%	0%	31240	2450	0%	67430	2570
alu4	1028×28	27%	-	-	-	100%	60	10
misex3	1848×28	34%	-	-	-	-	-	-
table3	645×28	40%	-	-	-	0%	840	20
apex4	1732×18	47%	-	-	-	0%	1470	230
rd84	411×16	50%	-	-	-	0%	350	7

5.6.2. Matrix Based Approaches. Similar to graph based approaches, we start our experiments with optimal size crossbars, and $P_D = 15\%$. Here, Tunali's algorithm is the clear winner in terms of the all parameters. However, it is not able to find a valid mapping for larger size examples with an only exception of the benchmark ”alu4”. Therefore scalability is still an important issue for matrix based algorithms. Results are given in Table IX. In our last two experiments, Tunali's approach also produces superior results. It is clear that Goren's approach is not scalable due mainly to its fairly complicated sorting process. However, it should be noted that, Tunali's success rate for a harder case like ”misex2” is inferior to Yuan's approach when stuck-at activated defects are introduced to the problem. Memetic nature of Yuan's approach fine tunes

Table X. Runtime and Success Rate Comparison of Matrix based Algorithms using 1.5 Larger Size Crossbars; $P_D = 15\%$.

Benchmarks			**Goren, 2011**			**Tunali, 2017**		
Name	Size	IR	Psucc	Run.(ms)	Std	Psucc	Run.(ms)	Std
5xp1	75×14	28%	100%	37	2	100%	1	0
inc	34×14	40%	100%	10	0	100%	0	0
clip	167×18	29%	100%	154	4	100%	1	1
misex2	40×29	12%	96%	50	151	100%	1	0
9sym	87×18	33%	100%	40	2	100%	1	0
bw	65×10	35%	100%	24	1	100%	1	0
rd53	32×10	45%	100%	8	0	100%	1	0
t481	481×32	30%	53%	16270	2910	100%	36	2
alu4	1028×28	27%	-	-	-	100%	19	1
misex3	1848×28	34%	-	-	-	100%	382	18
table3	645×28	40%	-	-	-	100%	107	8
apex4	1732×18	47%	-	-	-	100%	340	18
rd84	411×16	50%	-	-	-	100%	39	3

Table XI. Runtime and Success Rate Comparison of Matrix based Algorithms using 1.5 Larger Size Crossbars; $P_D = 10\%$ and $P_A = 5\%$.

Benchmarks			**Goren, 2011**			**Tunali, 2017**		
Name	Size	IR	Psucc	Run.(ms)	Std	Psucc	Run.(ms)	Std
5xp1	75×14	28%	100%	34	5	100%	1	0
inc	34×14	40%	97%	32	6	100%		0
clip	167×18	29%	100%	126	4	100%	4	1
misex2	40×29	12%	0%	760	3	28%	23	1
9sym	87×18	33%	100%	35	2	100%	1	0
bw	65×10	35%	100%	19	1	100%	1	0
rd53	32×10	45%	100%	7	0	100%	1	0
t481	481×32	30%	60%	12930	1820	100%	60	12
alu4	1028×28	27%	-	-	-	100%	70	4
misex3	1848×28	34%	-	-	-	100%	430	16
table3	645×28	40%	-	-	-	100%	130	17
apex4	1732×18	47%	-	-	-	100%	210	14
rd84	411×16	50%	-	-	-	100%	18	3

mapping process better than Tunali's sorting approach. Results are given in Table X and XI.

5.6.3. Comparisons of all Algorithms . In Table XII, general features of the algorithms are evaluated by using four levels: poor, moderate, good, and excellent. Features of satisfiability, ILP and graph embedding based algorithms are interpreted through the comparison performed in the papers [Gören et al. 2011], [Zamani et al. 2013] and [Yuan et al. 2014], respectively. Examining the results, we see that certain attributes need to be considered before choosing a suitable method. When area yield is an important factor, for easier cases Tunali's approach and for harder cases Yuan's approach is the

Table XII. Cheatsheet of Defect-Aware Algorithms

Author	**Algorithm Features**				
	Success Rate	**Multiple-type Defects**	**Area Yield**	**Runtime**	**Scalability**
Naemi, 2004	Good	Good	Poor	Good	Good
Zheng, 2009*	Good	Good	Poor	Poor	Poor
Simsir, 2009	Good	Moderate	Moderate	Poor	Poor
Rao, 2009*	Poor	Poor	Poor	Poor	Poor
Goren, 2011	Moderate	Moderate	Moderate	Moderate	Poor
Zamani, 2013*	Poor	Poor	Poor	Poor	Poor
Yuan, 2014	Good	Good	Moderate	Good	Moderate
Tunali, 2017	Excellent	Moderate	Good	Excellent	Excellent

* cases are excluded from simulations and results are taken from referenced papers

best choice. When it is not an issue, Tunali's approach produces better results considering the runtime values. When multiple-type defects are introduced, Naemi's, Yuan's, and Tunali's approach are fairly competent in terms of the success rate. Nevertheless Tunali's algorithm are superior considering the runtime. However, it is less equipped for harder cases, so if runtime is not an issue, Yuan's approach is more favorable. Note that ”multiple-type defects” and ”area yield” features are not assigned with excellent grade; the related research fields are open to further investigations with new methods.

6. TRANSIENT FAULT TOLERANCE

In the previous two sections, we have focused on permanent faults (defects) occurring during the fabrication process which are deterministic in nature and known in advance. However, transient faults appear in field and as for all emerging technologies, nano-crossbar arrays have very limited field data needed for accurate modeling of transient faults. Because of that, current literature is limited. Moreover, transient fault tolerance schemes are closely related to the architecture, so certain major assumptions are necessary. Another point is that, even though the architecture in question is assumed to be based on nano-crossbar arrays, investigative assumptions are based on the existing PLA's and their presumable responses to occurrence of faults.

In order to detect and correct errors due to transient faults, hardware redundant solutions are proposed. Main methods are fault masking and reconfiguration with online testing. Fault masking methods realize a logic function using more than one AND-OR plane. In reconfiguration based approaches, first faults are detected with online testing and then crossbar is reconfigured. In regard to diagnostic capabilities, a different degree of integration is favored according to fault occurrence and granularity of access to input and output lines. A generic scheme of transient fault tolerance preference is shown in Figure 12. Hardware overhead is utilized as multiple use of AND and OR planes in fault masking in (a) and online diagnostic with reconfiguration tools in (b).

Considering the two categories of faults previously explained in Section 2 related to configuration of crosspoints and functionality of components, we can say that both categories are applicable for transient faults. However, since faults related to the functionality of components require field data for justification that is very limited, current literature only adopts the configuration level faults classified as stuck-at activated and stuck-at deactivated.

Stuck-at deactivated faults cause a missing device in AND and OR planes, so a variable and a product is erased from the logic function, named as G and D, respectively. While G type faults produce $0 \longrightarrow 1$ error, D type faults produce $1 \longrightarrow 0$ error. Stuck-

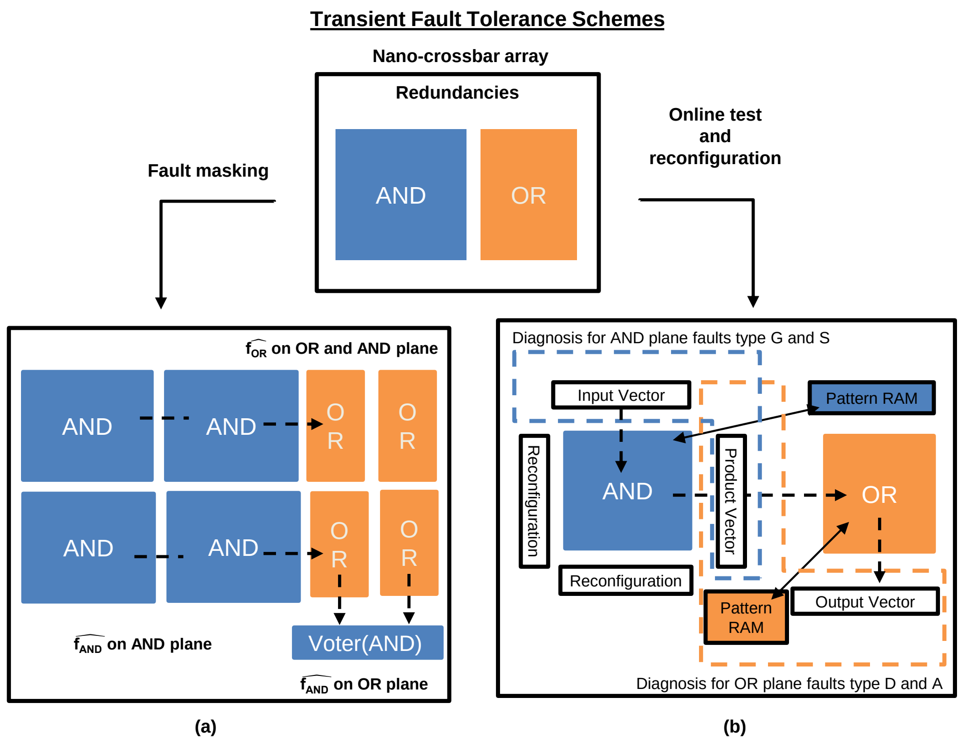

Transient Fault Tolerance Schemes
Nano-crossbar array
Redundancies
AND
OR
Fault masking
Online test and reconfiguration
$\widehat{f_{OR}}$ on OR and AND plane
AND
OR
Voter(AND)
$\widehat{f_{AND}}$ on AND plane
$\widehat{f_{AND}}$ on OR plane
Diagnosis for AND plane faults type G and S
Input Vector
Pattern RAM
Reconfiguration
Product Vector
Output Vector
Diagnosis for OR plane faults type D and A
(a)
(b)

Fig. 12. Transient fault tolerance schemes: (a) fault masking with multiple use of AND and OR planes, and (b) online test with a reconfiguration mechanism.

Table XIII. Transient Fault Model used for Nano-crossbar Arrays

Type	Naming	Cause	Effect	Output	Example
Stuck-at deactivated	G (growth)	missing device in AND plane	missing variable	$0 \longrightarrow 1$	F = A.B + C.D$\longrightarrow$ B + C.D
Stuck-at activated	S (shrink)	extra device in AND plane	extra variable	$1 \longrightarrow 0$	F = A.B + C.D $\longrightarrow$ A.B.E + C.D
Stuck-at deactivated	D (disappear)	missing device in OR plane	missing product	$1 \longrightarrow 0$	F = A.B + C.D$\longrightarrow$ C.D
Stuck-at activated	A (appear)	extra device in OR plane	extra product	$0 \longrightarrow 1$	F = A.B + C.D$\longrightarrow$ A.B + C.D + E

at activated faults cause an extra device in AND and OR planes, so a variable and a product is added to the logic function, named as S and A, respectively. While S type faults produce $1 \longrightarrow 0$ error, A type faults produce $0 \longrightarrow 1$ error. Table XIII shows the complete list of cause and effect of faults with a logic function example.

6.1. Fault Masking

Fault masking approach is first introduced in comparison with conventional methods in [Rao et al. 2007], and detailed examination of the approach considering stuck-at deactivated faults is given in [Rao et al. 2009]. In fault masking, two tautologies form of a Boolean function as follows.

$$\widehat{f_{AND}} = f \cdot f = f \qquad \widehat{f_{OR}} = f + f = f \tag{4}$$

Table XIV. Hardware Overhead for the Fault Tolerant Schemes

Fault Masking Scheme	**Hardware**		**Logic Level**
	Device	**Wire**	
Original	$D_A + D_O$	$I + P + O$	2
A-O	$4D_A + 2D_O$	$2I + 2P + O$	2
A-O-O	$2D_A + 2D_O + 2O$	$2I + P + 3O$	3
A-A-O-O	$2D_A + 2D_O + 2P + 2O$	$I + 3P + 3O$	4
A-O-O-A	$2D_A + 4D_O + 6O$	$I + 2P + 7O$	4

While the first tautology is for masking G and A type faults, S and D type faults correspond to the second one. To determine the area overhead of a PLA structure in terms of the used functional devices and connecting wires, we define the number of inputs/variables as I, the number of products as P, and the number outputs (implemented functions) as O. Table XIV gives the formulations of the used area for the implementations. By analyzing the table and given logic functions, we can fine-tune our area overhead. For example, a logic function with many products but a few outputs, using A-O-O is more reasonable than using A-O. It should be noted that this systematic is only for area optimization. For different performance parameters such as power and delay, a much more detailed analysis and optimization techniques are needed that can be considered as future work.

6.2. Online Testing and Reconfiguration

Constructed mainly on conventional fault detection and correction techniques, fault tolerance schemes using testing and reconfiguration are proposed in [Rao et al. 2007], [Garcia and Orailoglu 2008], and [Farazmand and Tahoori 2009].

In [Rao et al. 2007], a straightforward diagnostics technique is introduced with pattern RAMs that hold the correct output values of the mapped logic function. Therefore it is an easy task to determine erroneous results by checking input, product, and output vectors. One RAM for each AND and OR plane is integrated to a nano-crossbar. After locating the faults, reconfiguration process is assumed to be performed.

In [Garcia and Orailoglu 2008], a checkpoint-based fault tolerance offering reconfigurability is proposed. Online test is performed to a group of PLA blocks with choosing two of them as surrogates. Every block is checked in a round-table manner and if no fault is detected, a safe checkpoint is identified. In order to determine faults, row and column based diagnostic test vectors are designed. Starting with row based diagnostics, S type faults (previously explained in Table XIII) can be found by setting all of the inputs or variables of the corresponding product to 1, and the rest of the inputs to 0 to make all of the products except the corresponding one being 0. In case of having a $1 \longrightarrow 0$ error, we can conclude that an S type fault occurs in the product. Since D type faults show the same $1 \longrightarrow 0$ error characteristics that is also applicable to detect A type faults by using duality, we can say that row based diagnostics cover all types of faults except G types. Column based diagnostics is proposed to locate G type faults. However, it is not possible to locate all G type faults with a single row test vector. For this reason a compaction algorithm is used to optimize the number of test vectors by finding products sharing most variables.

In [Farazmand and Tahoori 2009], a dual rail implementation is proposed as an online test of detection. Both a logic function f and its negation $\neg f$ are realized in an AND plane and outputs are received using an OR plane. Since f and $\neg f$ always have opposite values, it is possible to detect faults by comparing the two values. The area overhead in comparison with 3- modular redundancy, 5- modular redundancy, parity, and duplication methods are favorably given regarding I, O, and P parameters defined in the previous subsection. Additionally, fault coverage ratios are given. Although the results are overwhelmingly better for the proposed dual rail technique, the used assumptions are quite weak covering very certain fault characteristics.

7. DISCUSSIONS

In this study, we survey fault tolerance algorithms of reconfigurable nano-crossbar arrays applicable in logic mapping and configuration processes. Both permanent and transient faults with different fault modeling approaches are covered. We conduct comprehensive simulations to evaluate the proposed algorithms using different fault rates on industrial benchmark functions. In addition, different area yield values and multiple type defect occurrences are considered.

To make concrete discussions with future directions, brief explanations of the historical development of fault tolerance in nano-crossbar arrays are given as follows.Nano-crossbar arrays were first proposed in 1990's to overcome the upcoming challenges of integrated circuit miniaturization. Its configurable/reconfigurable attributes as well as inherent fault tolerance capabilities attracted numerous researchers. However, as expected, this new technology comes with some challenges and fault tolerance is one of the significant ones. Fault rates are much higher for nano-crossbars compared to those of conventional CMOS circuits. Therefore, developing efficient fault tolerance techniques for nano-crossbars is a must.

At first, defect-unaware methods were proposed motivated by the fact that, configuring defective crossbars would be time consuming and impractical. However, area yields of defect-unaware approaches are proven to be less than ideal. In addition, we show that, stuck-at activated defects severely decrease the already low area yield values. As a result, although the number of studies in this field is limited, improving the yield remains to be a strong motivation for future studies with the fact that achieving defect-free sub crossbars enables us to use existing and well studied tools.

The line of research which have the most abundant studies is defect-aware methods. Although research on defect-aware approaches can be considered as mature, there are still important problems waiting to be solved including a need for specific algorithms to fine-tune the mapping problem according to multiple-type defect occurrences and different defect distributions. Additionally, current methods such as fitness approximation and matrix sorting are only able to respond to low defect rates; this issue should be solved. Another important research direction is developing techniques to restore defect mappings during the configuration process. A similar attempt using probabilistic data structure is presented in [Wang et al. 2006] with using bloom filters. Last but not least, in terms of area yield current methods are not fully equipped to produce optimum results for the realization of given logic functions. Inspiring studies considering both fault tolerance and yield analysis through the manipulation of logic function are presented in [Angiolini et al. 2007] and [Hogg and Snider 2006]. New methods producing better results under stricter area yield constraints would be a good line of investigation. Additionally, presented approaches can be applicable for memory structures due to similar problem formalizations [Huang et al. 2004] and [Feng et al. 2013], as well as for variance tolerance of the crossbars [Tunc and Tahoori 2010], [Yuan et al. 2016], [Ghavami 2016], and [Zhong et al. 2016].

Another trend is developing transient fault tolerance techniques. Fault masking and reconfiguration with online testing have been proposed. Even though presented methods are competent, without the field data it is hard to justify the results. In addition, only configuration level faults are considered in the literature; component level faults (or regarding the functionality) are open to further investigation. Also, physical realization of the architectures is still in infancy, so this line of inquiry is more reasonable with robust development and wide fabrication of nano-crossbars.

As a summary, we can list the future directions for fault tolerance techniques for nano-crossbar arrays as follows:

— Fine-tuning for multiple-type faults and different fault distributions;
— Compression and representation of defect maps;
— Improvement of area yield to increase density;
— Decomposition of given logic functions for area optimization;
— Developing variance tolerance techniques;
— Developing fault tolerance techniques for nano-crossbar based memory structures;
— Transient fault tolerance covering component level faults;
— Reliability forecasting for nano-crossbar arrays;
— Developing architectural level transient fault tolerance techniques; and
— Developing fault tolerance techniques for new technologies based on crossbar arrays including resistive/memristive networks.

REFERENCES

Ahmad A Al-Yamani, Sundarkumar Ramsundar, and Dhiraj K Pradhan. 2007. A defect tolerance scheme for nanotechnology circuits. *IEEE Transactions on Circuits and Systems I: Regular Papers* 54, 11 (2007), 2402–2409.

Dan Alexandrescu, Mustafa Altun, Lorena Anghel, Anna Bernasconi, Valentina Ciriani, Luca Frontini, and Mehdi Tahoori. 2016. Synthesis and Performance Optimization of a Switching Nano-Crossbar Computer. In *Digital System Design (DSD), 2016 Euromicro Conference on*. IEEE, 334–341.

Rick Amerson, Richard J Carter, W Bruce Culbertson, Philip Kuekes, and Greg Snider. 1995. Teramac-configurable custom computing. In *FPGAs for Custom Computing Machines, 1995. Proceedings. IEEE Symposium on*. IEEE, 32–38.

Federico Angiolini, M Haykel Ben Jamaa, David Atienza, Luca Benini, and Giovanni De Micheli. 2007. Improving the fault tolerance of nanometric PLA designs. In *Design, Automation & Test in Europe Conference & Exhibition, 2007. DATE'07*. IEEE, 1–6.

Debayan Bhaduri, Sandeep Shukla, Heather Quinn, D Bhaduri, S Shukla, and H Quinn. 2004. Reliability driven probabilistic design paradigm for transient error tolerant architectures on nanofabrics. In *Tech. Rep., Virginia Tech*. Citeseer.

Yong Chen, Gun-Young Jung, Douglas AA Ohlberg, Xuema Li, Duncan R Stewart, Jan O Jeppesen, Kent A Nielsen, J Fraser Stoddart, and R Stanley Williams. 2003. Nanoscale molecular-switch crossbar circuits. *Nanotechnology* 14, 4 (2003), 462.

Paul C Chu and John E Beasley. 1997. A genetic algorithm for the generalised assignment problem. *Computers & Operations Research* 24, 1 (1997), 17–23.

International Roadmap Committee and others. 2008. International technology roadmap for semiconductors. (2008).

Thomas M Conte and Paolo A Gargini. 2015. On The Foundation Of The New Computing Industry Beyond 2020. *International Technology Roadmap for Semiconductors (ITRS)* (2015).

Luigi P Cordella, Pasquale Foggia, Carlo Sansone, and Mario Vento. 2004. A (sub) graph isomorphism algorithm for matching large graphs. *IEEE transactions on pattern analysis and machine intelligence* 26, 10 (2004), 1367–1372.

André Dehon. 2005. Nanowire-based programmable architectures. *ACM Journal on Emerging Technologies in Computing Systems (JETC)* 1, 2 (2005), 109–162.

Andre DeHon and Helia Naeimi. 2005. Seven strategies for tolerating highly defective fabrication. *Design & Test of Computers, IEEE* 22, 4 (2005), 306–315.

Michael Demjanenko and Shambhu J Upadhyaya. 1990. Yield enhancement of field programmable logic arrays by inherent component redundancy. *IEEE transactions on computer-aided design of integrated circuits and systems* 9, 8 (1990), 876–884.

Manek Dubash. 2005. Moores Law is dead, says Gordon Moore. *Techworld (April 2005)* (2005).

Navid Farazmand and Mehdi B Tahoori. 2009. Online multiple error detection in crossbar nano-architectures. In *Computer Design, 2009. ICCD 2009. IEEE International Conference on*. IEEE, 335–342.

Wenyi Feng, Floriana Lombardi, Haider AF Almurib, and T Nandha Kumar. 2013. Testing a Nanocrossbar for Multiple Fault Detection. *Nanotechnology, IEEE Transactions on* 12, 4 (2013), 477–485.

Harold Fleisher and Leon I. Maissel. 1975. An introduction to array logic. *IBM Journal of Research and Development* 19, 2 (1975), 98–109.

Lester R Ford Jr and Delbert R Fulkerson. 1955. *A simple algorithm for finding maximal network flows and an application to the Hitchcock problem*. Technical Report. DTIC Document.

Saturnino Garcia and Alex Orailoglu. 2008. Online test and fault-tolerance for nanoelectronic programmable logic arrays. In *2008 IEEE International Symposium on Nanoscale Architectures*. IEEE, 8–15.

Michael R Garey and David S Johnson. 2002. *Computers and intractability*. Vol. 29. wh freeman New York.

Behnam Ghavami. 2016. Joint defect-and variation-aware logic mapping of multi-outputs crossbar-based nanoarchitectures. *Journal of Computational Electronics* 15, 3 (2016), 959–967.

Behnam Ghavami, Alireza Tajary, Mohsen Raji, and Hossein Pedram. 2010. Defect and variation issues on design mapping of reconfigurable nanoscale crossbars. In *VLSI (ISVLSI), 2010 IEEE Computer Society Annual Symposium on*. IEEE, 173–178.

Daniel Gil, David De Andrés, Juan-Carlos Ruiz, and Pedro Gil. 2008. Developing fault models for nanowire logic circuits. In *2nd Workshop on Dependable and Secure Nanocomputing at IEEE Int. Conf. on Dependable Systems & Networks. USA, pp. C6-C11*. Citeseer.

Benjamin Gojman and André DeHon. 2009. VMATCH: Using logical variation to counteract physical variation in bottom-up, nanoscale systems. In *Field-Programmable Technology, 2009. FPT 2009. International Conference on*. IEEE, 78–87.

Seth Copen Goldstein and Mihai Budiu. 2001. Nanofabrics: Spatial computing using molecular electronics. *ACM SIGARCH Computer Architecture News* 29, 2 (2001), 178–191.

Sezer Gören, H Fatih Ugurdag, and Okan Palaz. 2011. Defect-aware nanocrossbar logic mapping through matrix canonization using two-dimensional radix sort. *ACM Journal on Emerging Technologies in Computing Systems (JETC)* 7, 3 (2011), 12.

Michael Haselman and Scott Hauck. 2010. The future of integrated circuits: A survey of nanoelectronics. *Proc. IEEE* 98, 1 (2010), 11–38.

Chen He and Margarida F Jacome. 2007. Defect-aware high-level synthesis targeted at reconfigurable nanofabrics. *IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems* 26, 5 (2007), 817.

Chen He, Margarida F Jacome, and Gustavo de Veciana. 2005. A Reconfiguration-Based Defect-Tolerant Design Paradigm for Nanotechnologies. *IEEE DESIGN & TEST* 22, 4 (2005), 0316–326.

James R Heath, Philip J Kuekes, Gregory S Snider, and R Stanley Williams. 1998. A defect-tolerant computer architecture: Opportunities for nanotechnology. *Science* 280, 5370 (1998), 1716–1721.

Tad Hogg and Greg Snider. 2004. Defect-tolerant logic with nanoscale crossbar circuits. In *HP Labs*. Citeseer.

Tad Hogg and Greg S Snider. 2006. Defect-tolerant adder circuits with nanoscale crossbars. *IEEE Transactions on Nanotechnology* 5, 2 (2006), 97–100.

John E Hopcroft and Richard M Karp. 1973. An n^5/2 algorithm for maximum matchings in bipartite graphs. *SIAM Journal on computing* 2, 4 (1973), 225–231.

Jing Huang, Mehdi B Tahoori, and Fabrizio Lombardi. 2004. On the defect tolerance of nano-scale two-dimensional crossbars. In *Defect and Fault Tolerance in VLSI Systems, 2004. DFT 2004. Proceedings. 19th IEEE International Symposium on*. IEEE, 96–104.

Yu Huang, Xiangfeng Duan, Qingqiao Wei, and Charles M Lieber. 2001. Directed assembly of one-dimensional nanostructures into functional networks. *Science* 291, 5504 (2001), 630–633.

Michiel M Ligthart and Rudi J Stans. 1991. A fault model for PLAs. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 10, 2 (1991), 265–270.

Ken McElvain. 1993. IWLS93 benchmark set: Version 4.0. In *Distributed as part of the MCNC International Workshop on Logic Synthesis*, Vol. 93.

Mahim Mishra and Seth C Goldstein. 2004. Defect tolerance at the end of the roadmap. In *Nano, quantum and molecular computing*. Springer, 73–108.

Muhammed Ceylan Morgul, Furkan Peker, and Mustafa Altun. 2016. Power-Delay-Area Performance Modeling and Analysis for Nano-Crossbar Arrays. In *VLSI (ISVLSI), 2016 IEEE Computer Society Annual Symposium on*. IEEE, 437–442.

Csaba Andras Moritz, Teng Wang, Pritish Narayanan, Michael Leuchtenburg, Yao Guo, Catherine Dezan, and Mahmoud Bennaser. 2007. Fault-tolerant nanoscale processors on semiconductor nanowire grids. *IEEE Transactions on Circuits and Systems I: Regular Papers* 54, 11 (2007), 2422–2437.

Dhruv Mubayi and György Turán. 2010. Finding bipartite subgraphs efficiently. *Inform. Process. Lett.* 110, 5 (2010), 174–177.

Helia Naeimi and André DeHon. 2004. A greedy algorithm for tolerating defective crosspoints in NanoPLA design. In *Field-Programmable Technology, 2004. Proceedings. 2004 IEEE International Conference on*. IEEE, 49–56.

Daniel L. Ostapko and Se June Hong. 1979. Fault analysis and test generation for programmable logic arrays (PLA's). *IEEE Trans. Comput.* 28, 9 (1979), 617–627.

Ilia Polian and Wenjing Rao. 2008. Selective hardening of nanopla circuits. In *2008 IEEE International Symposium on Defect and Fault Tolerance of VLSI Systems*. IEEE, 263–271.

Wenjing Rao, Alex Orailoglu, and Ramesh Karri. 2006. Topology aware mapping of logic functions onto nanowire-based crossbar architectures. In *Proceedings of the 43rd annual Design Automation Conference*. ACM, 723–726.

Wenjing Rao, Alex Orailoglu, and Ramesh Karri. 2007. Fault tolerant approaches to nanoelectronic programmable logic arrays. In *37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'07)*. IEEE, 216–224.

Wenjing Rao, Alex Orailoglu, and Ramesh Karri. 2009. Logic mapping in crossbar-based nanoarchitectures. *IEEE Design & Test of Computers* 1 (2009), 68–77.

Robert R Schaller. 1997. Moore's law: past, present and future. *IEEE spectrum* 34, 6 (1997), 52–59.

Anish Man Singh Shrestha, Satoshi Tayu, and Shuichi Ueno. 2009. Orthogonal Ray Graphs and Nano-PLA Design. In *ISCAS*. 2930–2933.

Max M Shulaker, Gage Hills, Nishant Patil, Hai Wei, Hong-Yu Chen, H-S Philip Wong, and Subhasish Mitra. 2013. Carbon nanotube computer. *Nature* 501, 7468 (2013), 526–530.

Muzaffer O Simsir, Srihari Cadambi, Franjo Ivančić, Martin Roetteler, and Niraj K Jha. 2009. A hybrid nano-CMOS architecture for defect and fault tolerance. *ACM Journal on Emerging Technologies in Computing Systems (JETC)* 5, 3 (2009), 14.

James E. Smith. 1979. Detection of faults in programmable logic arrays. *IEEE Trans. Computers* 28, 11 (1979), 845–853.

Greg Snider, P Kuekes, T Hogg, and R Stanley Williams. 2005. Nanoelectronic architectures. *Applied Physics A* 80, 6 (2005), 1183–1195.

Greg Snider, Philip Kuekes, and R Stanley Williams. 2004. CMOS-like logic in defective, nanoscale crossbars. *Nanotechnology* 15, 8 (2004), 881.

Dmitri B Strukov and Konstantin K Likharev. 2005. CMOL FPGA: a reconfigurable architecture for hybrid digital circuits with two-terminal nanodevices. *Nanotechnology* 16, 6 (2005), 888.

Dmitri B Strukov and Konstantin K Likharev. 2007. Defect-tolerant architectures for nanoelectronic crossbar memories. *Journal of Nanoscience and Nanotechnology* 7, 1 (2007), 151–167.

Yehua Su and Wenjing Rao. 2009. Runtime analysis for defect-tolerant logic mapping on nanoscale crossbar architectures. In *Proceedings of the 2009 IEEE/ACM International Symposium on Nanoscale Architectures*. IEEE Computer Society, 75–78.

Yehua Su and Wenjing Rao. 2011. Defect-tolerant logic implementation onto nanocrossbars by exploiting mapping and morphing simultaneously. In *Proceedings of the International Conference on Computer-Aided Design*. IEEE Press, 456–462.

Yehua Su and Wenjing Rao. 2014. An integrated framework toward defect-tolerant logic implementation onto nanocrossbars. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 33, 1 (2014), 64–75.

Mehdi B Tahoori. 2006. Application-independent defect-tolerant crossbar nano-architectures. In *Proceedings of the 2006 IEEE/ACM international conference on Computer-aided design*. ACM, 730–734.

Mehdi Baradaran Tahoori. 2010. Variation and defect tolerance for diode-based nano crossbars. *Nano Communication Networks* 1, 4 (2010), 264–272.

Onur Tunali and Mustafa Altun. 2017. Permanent and Transient Fault Tolerance for Reconfigurable Nano-Crossbar Arrays. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 36, 5 (2017), 747–760.

Cihan Tunc and Mehdi B Tahoori. 2010. Variation tolerant logic mapping for crossbar array nano architectures. In *Proceedings of the 2010 Asia and South Pacific Design Automation Conference*. IEEE Press, 855–860.

Gang Wang, Wenrui Gong, and Ryan Kastner. 2006. On the use of Bloom filters for defect maps in nanocomputing. In *Proceedings of the 2006 IEEE/ACM international conference on Computer-aided design*. ACM, 743–746.

Teng Wang, Pritish Narayanan, Michael Leuchtenburg, and Csaba Andras Moritz. 2008. NASICs: A nanoscale fabric for nanoscale microprocessors. In *Nanoelectronics Conference, 2008. INEC 2008. 2nd IEEE International*. IEEE, 989–994.

Teng Wang, Pritish Narayanan, and Csaba Andras Moritz. 2007. Combining 2-level logic families in grid-based nanoscale fabrics. In *Proceedings of the 2007 IEEE International Symposium on Nanoscale Architectures*. IEEE Computer Society, 101–108.

Rainer Waser. 2012. *Nanoelectronics and information technology*. John Wiley & Sons.

C Wey, M Vai, and Fabrizio Lombardi. 1987. On the design of a redundant programmable logic array (RPLA). *IEEE Journal of Solid-State Circuits* 22, 1 (1987), 114–117.

C-L Wey. 1988. On yield consideration for the design of redundant programmable logic arrays. *IEEE transactions on computer-aided design of integrated circuits and systems* 7, 4 (1988), 528–535.

W Wu, G-Y Jung, DL Olynick, J Straznicky, Z Li, X Li, DAA Ohlberg, Y Chen, S-Y Wang, JA Liddle, and others. 2005. One-kilobit cross-bar molecular memory circuits at 30-nm half-pitch fabricated by nanoimprint lithography. *Applied Physics A* 80, 6 (2005), 1173–1178.

Lei Xie, Hoang Anh Du Nguyen, Mottaqiallah Taouil, Said Hamdioui, and Koen Bertels. 2015. Fast boolean logic mapped on memristor crossbar. In *Computer Design (ICCD), 2015 33rd IEEE International Conference on*. IEEE, 335–342.

Hao Yan, Hwan Sung Choe, SungWoo Nam, Yongjie Hu, Shamik Das, James F Klemic, James C Ellenbogen, and Charles M Lieber. 2011. Programmable nanowire circuits for nanoprocessors. *Nature* 470, 7333 (2011), 240–244.

J Joshua Yang, Dmitri B Strukov, and Duncan R Stewart. 2013. Memristive devices for computing. *Nature nanotechnology* 8, 1 (2013), 13–24.

Joon-Sung Yang and Rudrajit Datta. 2011. Efficient function mapping in nanoscale crossbar architecture. In *2011 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems*. IEEE, 190–196.

Yi Yang, Bo Yuan, and Bin Li. 2011. Defect and variation tolerance logic mapping for crossbar nanoarchitectures as a multi-objective problem. In *Information Science and Technology (ICIST), 2011 International Conference on*. IEEE, 1139–1142.

Jun Yao, Hao Yan, Shamik Das, James F Klemic, James C Ellenbogen, and Charles M Lieber. 2014. Nanowire nanocomputer as a finite-state machine. *Proceedings of the National Academy of Sciences* 111, 7 (2014), 2431–2435.

Yadunandana Yellambalase and Minsu Choi. 2008. Cost-driven repair optimization of reconfigurable nanowire crossbar systems with clustered defects. *Journal of Systems Architecture* 54, 8 (2008), 729–741.

Bo Yuan and Bin Li. 2011. A low time complexity defect-tolerance algorithm for nanoelectronic crossbar. In *International Conference on Information Science and Technology*. IEEE, 143–148.

Bo Yuan and Bin Li. 2014. A fast extraction algorithm for defect-free subcrossbar in nanoelectronic crossbar. *ACM Journal on Emerging Technologies in Computing Systems (JETC)* 10, 3 (2014), 25.

Bo Yuan, Bin Li, Huanhuan Chen, and Xin Yao. 2015. A new evolutionary algorithm with structure mutation for the maximum balanced biclique problem. *IEEE transactions on cybernetics* 45, 5 (2015), 1054–1067.

Bo Yuan, Bin Li, Huanhuan Chen, and Xin Yao. 2016. Defect-and Variation-Tolerant Logic Mapping in Nanocrossbar Using Bipartite Matching and Memetic Algorithm. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 24, 9 (2016), 2813–2826.

Bo Yuan, Bin Li, Thomas Weise, and Xin Yao. 2014. A new memetic algorithm with fitness approximation for the defect-tolerant logic mapping in crossbar-based nanoarchitectures. *Evolutionary Computation, IEEE Transactions on* 18, 6 (2014), 846–859.

Masoud Zamani, Hanieh Mirzaei, and Mehdi B Tahoori. 2013. ILP formulations for variation/defect-tolerant logic mapping on crossbar nano-architectures. *ACM Journal on Emerging Technologies in Computing Systems (JETC)* 9, 3 (2013), 21.

Yexin Zheng and Chao Huang. 2009. Defect-aware logic mapping for nanowire-based programmable logic arrays via satisfiability. In *Proceedings of the Conference on Design, Automation and Test in Europe*. European Design and Automation Association, 1279–1283.

Fugui Zhong, Bo Yuan, and Bin Li. 2016. A hybrid evolutionary algorithm for multiobjective variation tolerant logic mapping on nanoscale crossbar architectures. *Applied Soft Computing* 38 (2016), 955–966.

Zhaohui Zhong, Deli Wang, Yi Cui, Marc W Bockrath, and Charles M Lieber. 2003. Nanowire crossbar arrays as address decoders for integrated nanosystems. *Science* 302, 5649 (2003), 1377–1379.

Matthew M Ziegler and Mircea R Stan. 2003. CMOS/nano co-design for crossbar-based molecular electronic systems. *IEEE Transactions on Nanotechnology* 2, 4 (2003), 217–230.