# ITERATIVE HEURISTICS FOR CMOL HYBRID CMOS/NANODEVICES CELLS MAPPING

ABDALRAHMAN M. ARAFEH

## Computer Engineering

May 2012

# ITERATIVE HEURISTICS FOR CMOL HYBRID CMOS/NANODEVICES CELLS MAPPING

by

**ABDALRAHMAN M. ARAFEH**

A Thesis Presented to the
DEANSHIP OF GRADUATE STUDIES

In Partial Fulfillment of the Requirements
for the Degree of

**MASTER OF SCIENCE**

IN

**COMPUTER ENGINEERING**

KING FAHD UNIVERSITY OF PETROLEUM & MINERALS

Dhahran, Saudi Arabia

May 2012

KING FAHD UNIVERSITY OF PETROLEUM & MINERALS

DHAHRAN 31261, SAUDI ARABIA

DEANSHIP OF GRADUATE STUDIES

This thesis, written by

ABDALRAHMAN M. ARAFEH

under the direction of his Thesis Advisor and approved by his Thesis Committee, has been presented to and accepted by the Dean of Graduate Studies, in partial fulfillment of the requirements for the degree of

**MASTER OF SCIENCE IN COMPUTER ENGINEERING**

Thesis Committee

Dr. Sadiq M. Sait (Chairman)

Alaa Amin

Dr. Alaaeldin Amin (Member)

Dr. Zubair Baig (Member)

Dr. Basem Almadani
Department Chairman

Dr. Salam Zummo
Dean of Graduate Studies

30/12/12
Date

*Dedicated to*

*My Parents, beloved Brothers and Sister*

*&*

*The undaunted Syrian People*

# Acknowledgements

All sincere praises and thanks are due to Allah (SWT), for His limitless blessings on us. May Allah bestow his peace and blessings upon our leader Prophet Muhammad (P.B.U.H), his family, his companions, and those that follow his guidance until the last day. Acknowledgements are due to King Fahd University of Petroleum & Minerals for providing the computing resources for this research.

I would like to express my profound gratitude and appreciation to my thesis advisor Dr. Sadiq M. Sait for his guidance and patience throughout this thesis. His continuous support, advise and encouragement can never be forgotten. I would also like to express my appreciation to my thesis committee members, Dr. Alaaeldin Amin and Dr. Zubair Baig for their constructive comments. Also, I would like to express my deepest thanks to faculty and staff members of Computer Engineering Department for their cooperation. Thanks are due also to my fellow graduate students, and brothers who supported me with help and encouragement during the work. Especially, Feras Chikh Oughali, Abdulrahman Idlbi, Mouheddin Alhaffar and Abdulnaser Alsharaa.

I also thank my beloved parents, my brothers and sister for their moral support throughout my academic career. Their assistant and sacrifices are truly appreciated and will be remembered. Finally, thanks to everybody who contributed to this achievement in a direct or an indirect way.

# Contents

# List of Tables

# List of Figures

# THESIS ABSTRACT

| | |
|---|---|
| Name: | ABDALRAHMAN M. ARAFEH |
| Title: | ITERATIVE HEURISTICS FOR CMOL HYBRID CMOS/NANODEVICES CELLS MAPPING |
| Major Field: | COMPUTER ENGINEERING |
| Date of Degree: | May 2012 |

*Recently, many CMOS/nanodevices hybrid architectures have been proposed; the new architectures combine the flexibility and high fabrication yield advantages of CMOS technology with nanometer scale latching devices. CMOL (CMOS/Molecular hybrid) is a novel architecture that consists of an overlay of a nanofabric over a CMOS stack. Combinational logic in CMOL is implemented from a netlist of NOR gates and Inverters, by programming nanodevices placed between overlapping nanowires. The length of the nanowires is restricted, and therefore connectivity of the circuit elements is constrained to only cells that are located within proximity square-like connectivity domain. The confined connectivity reduces the flexibility of VLSI design automation and further complicates cells mapping. Furthermore, misassembly of the two-terminals bistable nanodevices will lead to non-programmable crosspoints (i.e., stuck-at defects). The defect rate in nanofabric architectures is expected to be higher than that of conventional CMOS technology. In this work, we solve the problems of cell placement and reconfiguration in CMOL circuits. Simulated Evolution (SimE) and Tabu Search (TS) are employed to find an arrangement of cells that adhere to connectivity constraints and rely on non defective nanodevices. Circuits of various sizes from ISCAS'89 benchmarks are used to evaluate the designed heuristics. Results show that SimE and TS are able to find placement solutions that are better than previously published ones, and in less computation time. Moreover, they yield successful reconfigurations when the defect rate is as high as 50%.*

**Keywords:** *CMOL, Nanofabrics, Placement, Reconfiguration, Simulated Evolution, Tabu Search, Evolutionary Tabu Search, Search Heuristics, Defects, VLSI.*

# ملخص الرســالة

**الإســم:** عبد الرحمن محمد عرفان عرفة

**العنـوان:** توزيع العناصر الإلكترونية ضمن الدارات الهجينة بإستخدام الخوارزميات التكرارية غير الحتمية

**الإختصاص:** هندسة الحاسب الآلي

**تاريخ المنح:** أيار ٢٠١٢

شهدت الآونة الأخيرة عدة إقتراحات لدارات هجينة تجمع بين عناصر الدارات المتكاملة التقليدية CMOS وعناصر نانوية ذات خواص كهربائية. تجمع هذه الدارات الهجينة بين سهولة التصميم ومردود التصنيع العالي لتكنولوجيا CMOS، وبين مواسك نانوية ثنائية الإستقرار ذات كثافة عالية وقدرة على العمل عند ترددات مرتفعة. إحدى هذه البنى الجديدة هي بنية CMOL، وهي بنية مكونة من طبقة أسلاك نانوية متقاطعة تتوضع فوق طبقات CMOS التقليدية. يتم تحقيق المنطق التركيبي في بنية CMOL بإستخدام الأسلاك والتجهيزات النانوية على شكل شبكة من بوابات NOR وعواكس. يتم تحقيق الدارة من خلال برمجة العناصر النانوية المتوضعة بين الأسلاك النانوية المتداخلة. تسبب الصعوبات التصنيعية محدودية في أطوال الأسلاك النانوية، مما يجعل التوصيل بين عناصر الدارة مقيداً بالخلايا الموجودة ضمن مجال محدد للتوصيل. يسبب هذا التقييد صعوبة في تصميم الدارات المتكاملة، وخصيصاً في عملية توزيع العناصر الإلكترونية على الخلايا المتوفرة في الدارة. كما تعاني العناصر النانوية ثنائية الإستقرار الموجودة عند تقاطعات الأسلاك النانوية من عيوب ضمنية، أي أنها تكون غير قابلة للبرمجة أو الإستخدام، ويتوقع بأن تكون نسبة العيوب في هذه التجهيزات النانوية مرتفعة مقارنة بنسبة العيوب الموجودة في دارات CMOS.

في هذه الرسالة سنقدم حلاً لمشكلة توزيع عناصر الدارات الإلكترونية على الخلايا المتاحة في بنية CMOL، كما سنقوم بإعادة توزيع هذه العناصر لتلافي إستخدام أي من التجهيزات النانوية المعيبة. سيتم ذلك بإستخدام خوارزميتين غير حتميتين هما Simulated Evolution و Tabu Search واللتان تقومان بإيجاد توزيع معين للعناصر، بحيث يتم احترام قيود التوصيل واستخدام عناصر نانوية غير معيبة. استخدمنا في عملية تقييم فعالية الخوارزميات المقترحة مجموعة من الدارات القياسية ذات الأحجام المختلفة والمعروفة بإسم ISCAS'89. لقد أظهرت النتائج أن كلاً من الخوارزميتين قادرتان على إعطاء حلول أفضل من مثيلاتهما وبزمن حساب أقل. علاوة على ذلك، فإنهما تسفران عن عملية إعادة توزيع ناجحة عند نسب عالية من العيوب تصل إلى خمسين في المئة.

# Chapter 1

# Introduction

The recent advances in Very Large Scale Integration (VLSI) have led to the fabrication of circuits with millions of transistors. Conventionally, VLSI design process is divided into several intermediate levels of abstraction. More details about the new design are introduced as the design progresses from highest to lowest levels of abstraction. The design is taken from specification to fabrication step by step with the help of various Computer Aided Design (CAD) tools that automate the design flow and manage design information at all levels of VLSI design process. Typical levels of abstraction, together with their corresponding design steps, are illustrated in Figure 1.1.

Feature size scaling in CMOS technology has led to difficulties in manufacturing due to short channel effects, doping fluctuations and expensive lithography process. Meanwhile, advances in nanoelectronics are expected to achieve high density of

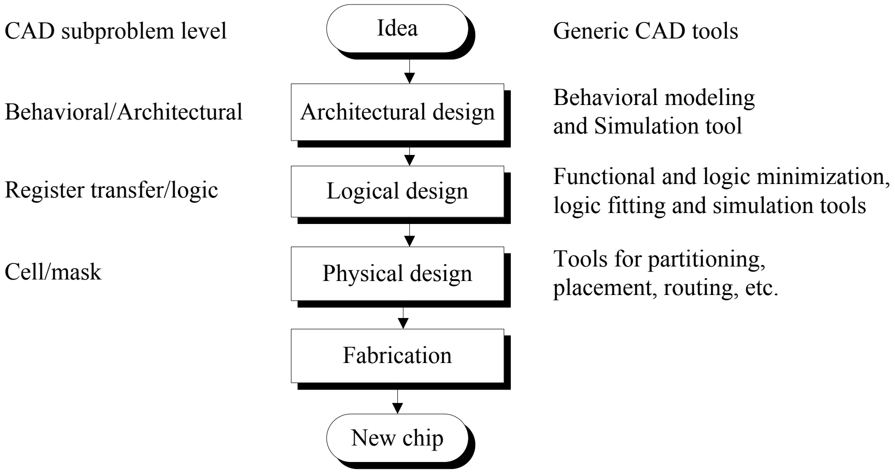


Figure 1.1: Levels of abstraction and corresponding design steps.

devices and to operate at THz frequencies [5]. Many effective applications have been proposed that use molecular nanodevices, nanowires, and nano-crossbar fabrics [3, 1, 6, 7]. A new trend is emerging for combining the flexibility and high fabrication yield advantages of CMOS technology with nanometer-scale molecular devices. A self-assembly of two-terminal nanodevices, with nanowire crossbar fabrics, enables high functional density and sustains acceptable fabrication costs.

Assigning cells to slots is an important step in the process of electronic design automation. The assignment problem has been proven to be NP-hard problem for which iterative heuristics have been employed successfully to reach acceptable solutions. Overtime, the objective of placement has changed from reducing the overall wirelength to reducing the area, to improving timing performance, and then to reducing the overall power dissipation. With new advances in technology come new issues; the CMOS/nanodevices hybrid architectures require combinational logic cells to be placed in slots that are connected by programmable nanodevices placed between overlapping nano-wires. The length of the nanowires is restricted, and therefore connectivity of the circuit elements is constrained.

## 1.1 Iterative Heuristics

Many of the significant optimization problems are NP-Hard. For relatively large instances of such problems, it is not possible to resort to optimal enumerative techniques; instead, we must resort to *approximation algorithms*. Approximation algorithms, also known as heuristic methods, do not guarantee finding an optimal solution, yet they exploit domain specific heuristic knowledge to bias the search toward "good" solution subspace to quickly find an "acceptable" solution which satisfies design constraints. Therefore, the time requirement of a heuristic is small compared to that of full enumerative algorithms.

A number of heuristics have been developed for various problems. Examples of approximation algorithms are the modern general iterative algorithms such as *Simulated Annealing*, *Tabu Search*, and *Simulated Evolution*. All mentioned iterative heuristics constitute very general (i.e., can be applied to solve any combinatorial optimization problem) and effective optimization techniques. Most iterative heuristics are easy to implement; all that is required is to have suitable solution representation, a cost function, and a mechanism to traverse the search space. Although they asymptotically converge to an optimal solution, the rate of convergence is heavily dependent on the adequate choice of several parameters and the utilization of "hill climbing" property.

## 1.2 CMOS/nanodevices Hybrid

Semiconductors have been largely dominated by CMOS (Complementary Metal-Oxide-Silicon), however, the current VLSI paradigm, based on a combination of lithographic patterning, CMOS circuits, and Boolean logic; can hardly be extended into a-few-nm region [8, 9]. The main reason is that at gate length below 10 nm, the sensitivity of parameters (most importantly, the gate threshold voltage) of silicon field-effect transistors (MOSFETs) grows exponentially. As a result, the gate length should be controlled with a few-angstrom accuracy, far beyond even the long-term projections of the semiconductor industry [10]. Even if such accuracy can be technically implemented using sophisticated patterning technologies, this will send the fabrication facilities costs to unprecedented high values, and will lead to annulment of Moore's Law some time during the next decade.

There is a growing consensus that the impending crisis of the microelectronics progress may be resolved only by a radical paradigm shift from the *lithography-based* fabrication to the so-called *bottom-up* approach [11]. In this approach, the smallest active devices of integrated circuits are not defined lithographically but assembled from parts with fundamentally reproducible size and structure, (e.g., few-nm-scale molecules). The most straightforward example of such devices is a specially designed two-terminal single-electron nanodevice [9, 12].

Unfortunately, integrated circuits consisting of molecular devices alone are hardly viable because of limited device functionality. For example, the voltage gain of a 1-nm-scale transistor, based on any known physical effect (e.g., the field effect, quantum interference, or single-electron charging), can hardly exceed one, i.e., the level necessary for sustaining the operation of virtually any active analog or digital circuit [13]. This is why the only plausible way toward high-performance nanoelectronic circuits is to integrate molecular devices, and the connecting nanowires, with CMOS circuits whose (relatively large) field-effect transistors would provide the necessary additional functionality, in particular high voltage gain. Thus, most efforts in the development of high-performance nanoelectronic circuits are focused on hybrid CMOS/nanodevices [3, 14, 15, 16, 17, 18]. Recent reviews of CMOS/nanodevices circuits can be found in [19, 20, 21, 22]. CMOS/nanodevices circuits with feature size below 10 nm have the potential to provide huge density improvement over the current CMOS technology [10, 23]. However, at such a small scale, fabricated chips will exhibit a high percentage of defects, probably as much as 20%-50%.

It is important to note that even though the recent demonstrations of CMOS/nanodevices hybrid architectures are promising, building a practical hybrid circuit is still challenging. One of the major challenges is the interfacing with CMOS environment (necessary for I/O functions). If a crossbar is small (much smaller than the chip it is fabricated on), each nanowire may be gradually widened to eventually fit a broader CMOS wire. Moreover, the requirement of special pins with different heights to connect to the top or bottom crossbar nanowires may render nanowires unreachable, causing circuits to become defective.

## 1.3 Thesis Objectives

The main objective of this work is to investigate the new constraints imposed on cells mapping in CMOL hybrid CMOS/nanodevices architecture. The mapped circuits should adhere to the architecture's constrained connectivity and should avoid to use any defective component. The mapping process will be divided to two main steps; *placement* on defect-free layout, and defect-aware *reconfiguration*.

Simulated Evolution (SimE), and Tabu Search (TS) are the search heuristics to be employed for search space exploration. The work will focus on the design of the search heuristics and their various operators and parameters given the new connectivity and defect constraints.

## 1.4 Thesis Contributions

This thesis presents the results of the investigations related to the objectives discussed in the previous section. The main contributions can be summarized as follows:

- The work illustrates the design of iterative heuristics to address the new constraint related to cells placement in the emerging CMOS/nanodevices circuits.
- The work demonstrates the use of iterative heuristics for fault tolerance in CMOL nanofabric architecture through reconfiguration.
- Implement Simulated Evolution (SimE) and Tabu Search (TS) heuristics for CMOL placement and reconfiguration problems.
- Propose new goodness, allocation and neighborhood generation functions for better exploration of search space and evolutionary enhancement of cells assignments.
- Generate defect maps with three types of defects part of which are based on stuck-at-open model.
- Successfully place circuits without requiring any additional buffers and preserving the circuits timing delay.
- Tolerate up to 50% of Stuck-at-open defects and broken nanowires rate up to 70% by reconfiguring circuits using SimE and TS Heuristics.

## 1.5 Thesis Organization

This thesis is organized as follows: In chapter 2, theoretical aspects of CMOL CMOS/nanodevices hybrid FPGA-like architecture are discussed along with a review of related literature on nanofabric design, iterative heuristics and proposed techniques for cell placement in CMOL. Problem formulation is dealt with in chapter 3. Chapter 4 discusses the parameters and operators of non-deterministic iterative heuristics employed for CMOL placement and reconfiguration problems. Heuristics evaluation and final results are reported in chapter 5, including comparison with previous techniques. The thesis concludes with conclusion and future work in chapter 6.

# Chapter 2

# Literature Review

## 2.1 Nanofabric Crossbars

A considerable amount of research has been done on developing nanoscale devices and devising nanofabric architectures to replace conventional lithography-based CMOS technology. Recently, many nanofabric logic designs have been proposed based on nanoscale componenets such as carbon nanotubes (CNTs) [24, 25, 26, 27], silicon nanowires (SiNWs) [28, 29], single electron devices [30, 31], and quantum dot cells [32]. Crossbar-based architecture is a promising computational nanotechnology, a 2D array formed by the intersection of two orthogonal sets of parallel and uniformly-spaced nanometer-sized wires. Nanoscale wires can be aligned to construct an array with nanometer-scale spacing using a form of directed self-assembly, the formed crosspoints of nanoscale wires can be used as programmable diodes. The nanoscale crossbar systems offers ultra-high density, however, the nanodevics are likely to have many imperfections and defect rates as high as 20% to 50%.

Nanoscale crossbar structures are very regular and can be implemented in a similar manner to the conventional filed programable gate arrays (FPGAs). Goldstein et al [7] proposed chemically assembled electronic nanotechnology FPGA-like architecture called *NanoFabric*. The architecture consists of an array of connected logic blocks, called Nanoblocks. A 2D molecular array inside each Nanoblock provides reprogrammable resistor-diode logic. DeHone et al. [6] presented another nanofabric architecture where the main building block of the design, called the nano programmable logic array *nanoPLA*, is based on self-assembled crossbar arrays of nanowires with non-volatile diode-based switches at the intersections. The individual nanowires can be addressed by a lithographic scale address decoder. Most importantly, Likharev el al [3] proposed *CMOL*, the CMOS/nanodevices circuits. CMOL uses diode-based nano crossbar arrays on top of CMOS cells. The main difference of CMOL compared to previous proposed architectures is how the CMOS/nanodevices are interfaced. Pins are distributed over the circuit on top of the CMOS stack to connect to either lower or upper nanowire levels. Nanowires in CMOL do not need to be precisely aligned with each other and the underlying CMOS layer in order to be able to uniquely access a nanodevice.

Generally, defects can be divided into two classes: permanent defects caused by inherent physics uncertainties in the manufacturing process, and transient faults due to lower noise tolerance. The methods used to cope with the aforementioned defects can be classified into two categories. The first one is based on redundancy, e.g. R-fold Module Redundancy (RMR). Such approach can handle both permanent defects and transient faults, however, it suffers from low reliability. The second category is based on reconfiguration techniques during post-manufacturing design to avoid the defects. It is reported that reconfiguration is the most effective technique, however, it does not effectively handle transient faults.

Defects are a major issue for devices with few atoms in diameter. The small cross-section and contact areas can render nanodevices fragile and defect prone. The order of defects in nanofabric architectures surpass the conventional CMOS devices since the inherent non-determinism in bottom-up self-assembly chemical processes at molecular scale, result in more defects compared to highly controlled lithography-based manufacturing process. Thus, an effective fault tolerance schemes are required, along with test and diagnostic techniques to identify and locate the defects and then reconfigure the circuit to bypass defective elements.

### 2.1.1 Fault Diagnosis in Nanofabric Crossbars

Reconfigurable devices are fault tolerant such that faults can be detected, and their locations can be stored in a defect map. The defect map is a database that stores defect information that can be used during reconfiguration. The faulty devices can be avoided with the help of a defect map which can be constructed by testing and diagnosis techniques. High resolution diagnosis is required to identify defective resources such as programmable switches, wires, or logic cells. A survey of different approaches for fault detection and diagnosis in molecular computing can be found in [33].

The Teramac project at HP-labs [34] applies thorough testing and diagnosis to identify defective unusable resources and maps an entire design to the usable resources. Build-in self-test (BIST) techniques make use of the reconfigurability of nanofabric FPGA-like architectures to provide a complete test and diagnosis of defects. In the built-in self-test approache, the fabric is divided into mainly three groups; a test pattern generator (TPG), blocks under test (BUTs), and output response analyzers (ORAs). The TPG applies test patterns to the BUTs, which send output responses to the ORA, and ORAs compare the responses to determine if there is a defect. Different variations and enhancement to BIST have been proposed in literature, among those the designs reported in [35, 36, 37].

The size of defect map for the entire fabric can be prohibitively large with almost $10^{12}$ nanodevices per chip. The authors in [38] show that Bloom filters can be used as a data structure for defect maps. They develop a defect tolerant nanoscale memory architecture that allow manufacturers to embed defect information within the delivered nanosystem.

### 2.1.2 Reconfiguration/Repair of Nanofabric Crossbars

In the presence of defects, it is still possible to utilize the non defective nanodevices, by reconfiguring circuits around defective ones. Huang et al [39] presented a solution for defect tolerance in two-dimensional crossbars by utilizing defective architecture and determining the expected size of functional (defect-free) crossbar, based on defect density information obtained from the fabrication process. Another attempt to reconfigure nanowire crossbar systems was reported by Yellambalase et al [40]. They presented three different logic mapping algorithms to circumvent defective crossbars. The algorithms namely; Row-wise matching, Column-matching-first, and Redundant column-matching first, are based on matching of two bipartite graphs, one of them represents the defective crossbar and the other represents the circuit to be mapped.

Tahoori has presented a defect tolerant design flow that includes a greedy mapping algorithm [41]. The algorithm finds and locates the maximum defect free $k \times k$ crossbar within the defective crossbar by finding the maximum biclique in a bipartite graph that represents the nanofabric crossbar. Further, a variation tolerant logic mapping for molecular (diode-based) crossbar using heuristic algorithms has been presented by Tahoori [42, 43]. His approach is mainly based on swapping rows (columns) of a crossbar to reduce the output dependency and delay variation.

Although, many heuristic algorithms have been presented in the literature for the reconfiguration or logic mapping in defective nanofabric crossbars, many are only applicable for small size nanofabrics. Moreover, The devised greedy algorithms are expected to have degraded results and to consume considerable computation time in case of high defects rate.

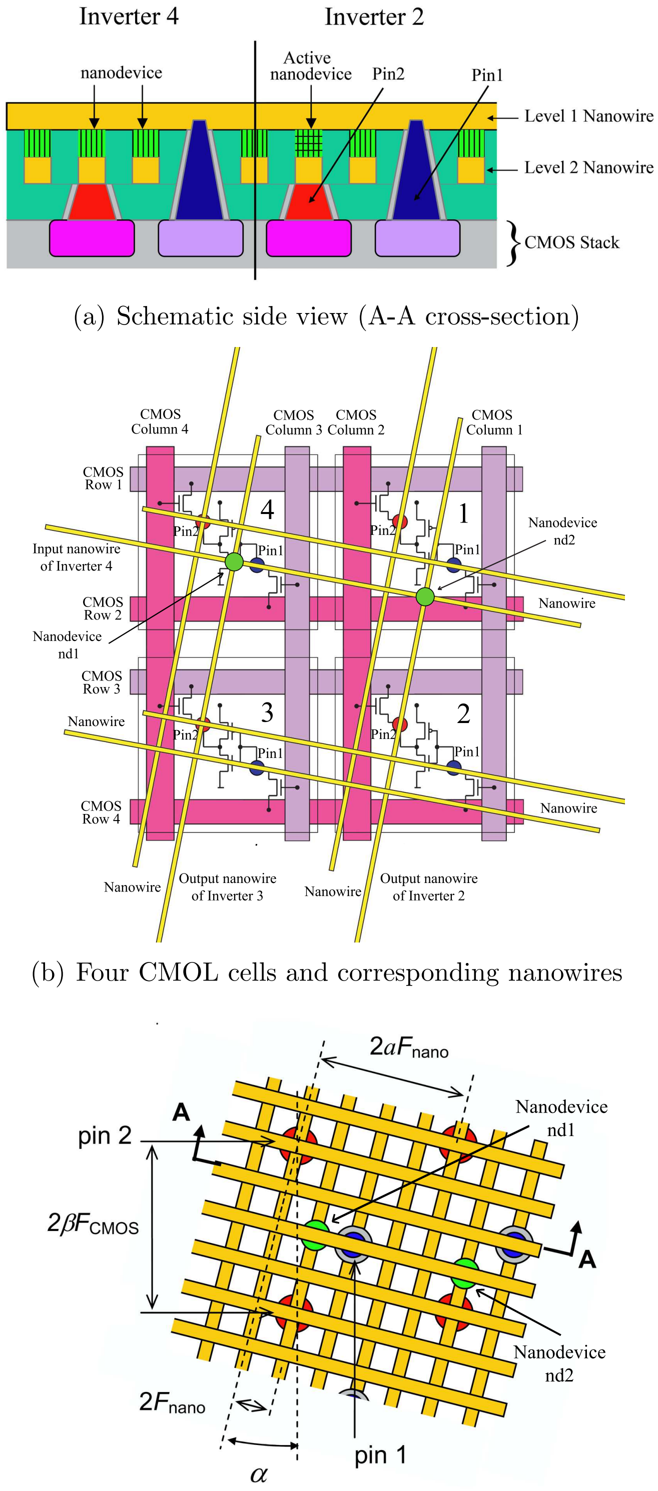
## 2.2 CMOL Hybrid CMOS/Nanodevices Circuits

CMOL (CMOS/nanowire/MOLecular hybrid) is a hybrid circuit architecture which combines a semiconductor MOSFET transistors with uniform reconfigurable nanowires fabric. It was originally developed by Likharev and his colleagues [3], to overcome the CMOS/nanodevices interface problems pertinent to earlier proposals. In CMOL circuits, interfacing between transistors and nanowires is provided by sharp-tip pins that are distributed all over the circuit area on top of the CMOS stack. Two-terminal molecular-scale nanodevices “latching switches”, that have two metastable internal states, are self-assembled at each crosspoint of the nanofabric. Nanodevices work as switches that are programmable to connect the two levels of nanowires. The generic CMOL cell shown in Figure 2.1(a), consists of conventional CMOS stack, two perpendicular nanowires, and two-terminal nanodevices sandwiched between nanowires to form points of contact. The output of inverter 2 (Pin 2) is connected to the input of inverter 4 (Pin 1) by pin-nanowire-nanodevice-nanowire-pin connection. The overlay nanofabric serves as a connection and wiring logic medium with the help of molecular nanodevices.

### 2.2.1 Cell-based FPGA-like CMOL Architecture

CMOL cell-based, field programmable gate array (FPGA)-like architecture is based on a uniform, reconfigurable CMOL fabric, with four transistor CMOS cells and two-terminal nanodevices [3]. Each generic CMOS cell (Four cells are shown in Figure 2.1(b)) consists of an inverter and two pass transistors that serve two pins as the cell input and output, respectively. During the configuration process the inverters are turned off, and the pass transistors are used for setting the binary state of each molecular device. By turning programmable diodes “ON” or “OFF”, the nanowires, nanodevice and CMOS inverters can implement a basic wired NOR with multiple fan-ins. As shown in Figure 2.1(b), Inverter 1 has two pins; pin1 connects the input of the CMOS inverter to one of the nanowires levels, while pin2 connects the CMOS inverter’s output to the second level of nanowires. The lower left cell (Inverter 3) is connected to the upper left cell (Inverter 4) by activating the appropriate nanodevice (nd1) in the crosspoint between the nanowire connected to the output of inverter 3 and nanowire connected to the input of inverter 4. When two or more nanodevices in the input nanowire of inverter 4 are activated (nd1 and nd2) the output of cell 4 will be equivalent to NOR gate whose inputs are cell 2 and cell 3.

The equivalent electrical circuit of the aforementioned configuration is shown



(a) Schematic side view (A-A cross-section)

(b) Four CMOL cells and corresponding nanowires

(c) Nanowires crossbar and pins connectivity

Figure 2.1: Low-level structure of CMOL architecture.

in Figure 2.2. The figure shows five logic stages that the electrical signals should traverse to connect two cells. The first stage is output nanowire; which is equivalent to resistance $R_{wire}$ and the capacitance of the full nanowire fragment $C_{wire}$. Then, comes the nanodeivce which is represented as an open diode with resistance $R_{ON}/D$ in the ON state and as a high resistance $R_{OFF}/D$ in OFF state, where $D$ is the number of parallel molecular-scale devices each with $R_{ON}$ resistance. Then, the connection passes through input nanowire to reach CMOL cell which has a CMOS pass transistor with $R_{pass}$ resistance and a CMOS inverter. Those stages comprise the pin-nanowire-nanodevice-nanowire-pin connection mentioned earlier.

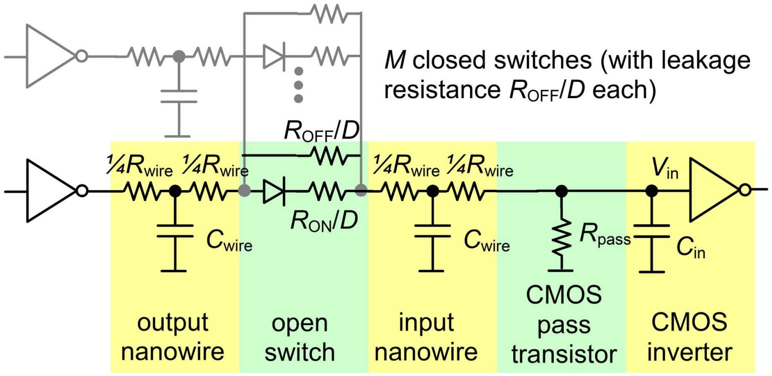


Figure 2.2: The equivalent circuit of a CMOL logic stage.

Figure 2.1(c) shows CMOS pins reaching to the lower and upper nanowire levels. CMOL fabric is arranged into a square array with side $2\beta F_{CMOS}$, where $F_{CMOS}$ is the half-pitch of the CMOS subsystem, while $\beta$ is a dimensionless factor greater than 1 and depends on the CMOS cell complexity [21]. Because nanodevices are non-volatile switches, they can be programmed to route the signals from CMOS to the nanowires and nanodevices, and back to CMOS again. For FPGA applications, the nanowire crossbar is turned by almost $\alpha = 45$ relative to CMOS cells array, though that is not absolutely necessary [3, 21]. More exactly, the requirements for the angle $\alpha$ and the dimensionless factor $\beta$ that determines the CMOS cell area $A = (2\beta F_{CMOS})^2$ is:

$$\alpha = \arcsin\left(\frac{F_{nano}}{\beta F_{CMOS}}\right) \tag{2.1}$$

Where $F_{nano}$ is the nanowiring half-pitch. Also, Figure 2.1(c) shows that any nanodevice may be addressed via the appropriate pin pair (e.g., input pin of Inverter 4 and output pins of Inverters 2 and 3), only two devices are shown but in reality, similar nanodevices are formed at all nanowire crosspoints. Like in the case of most programmable devices, the length of the nanowires is restricted and therefore each CMOL cell can be connected to $M = a^2 - 2$ other cells located within a square-shaped *Connectivity Domain* shown in Figure 2.3. Where $a$ is a positive integer number that constitute CMOL radius. In Case $a = 4$ output pins of cells painted in light-gray may be connected to the input pin of the specified dark-gray cell.

An example of implementing NOR gate using CMOL cells is given in Figure 2.4, if only the two nanodevices shown in Figure 2.4(b) are in the “ON” state, while all other nanodevice connected to the input nanowire of cell ‘$H$’ are in the “OFF”

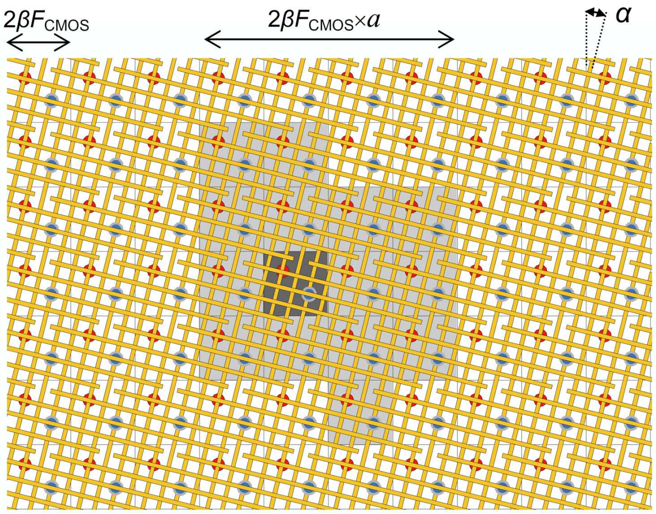


Figure 2.3: CMOL FPGA-like Architecture: Connectivity Domain.

(high resistance) state, then cell '$H$' calculates the NOR function of signals '$A$' and '$B$', and for the sake of clarity only the nanowires used are shown. The advantage of such architecture that gates with high fan-in (Figure 2.5) and fan-out may be readily formed as well by turning "ON" the corresponding latching switches. CMOL architecture is inherently defect-tolerant, since it has $M \approx a^2 >> 1$ nanodevices per CMOS cell, and few of them are required for either logic or routing functionality.

If the nanowires and nanodevices shown in Figure 2.6(b) are all activated, the CMOL circuit will be equivalent to circuit shown in Figure 2.6(a). Shaded cells are connected through combination of nanowires, nanodevices and CMOS pins. The first NOR gate of the circuit can be implemented by connecting inputs '$A$' and '$B$' with inverter '2' to satisfy both connectivity and logic wiring for the desired gate.

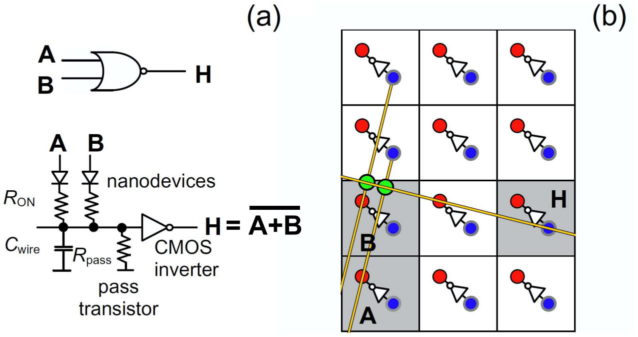


Figure 2.4: Fan-in-two NOR gate: (a) equivalent circuit and (b) physical implementation in CMOL.

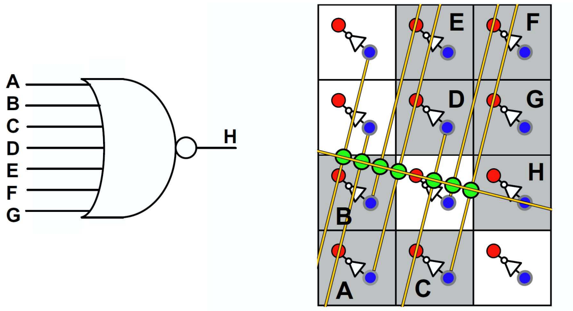


Figure 2.5: Example of CMOL implementation of a 7-input NOR gate.

The abundance of available nanodevices and nanowires provide a variety of different possible configurations for the implementation of one circuitry. Among those there could be only certain configurations that satisfy connectivity domain constraint and do not require additional routing resources.

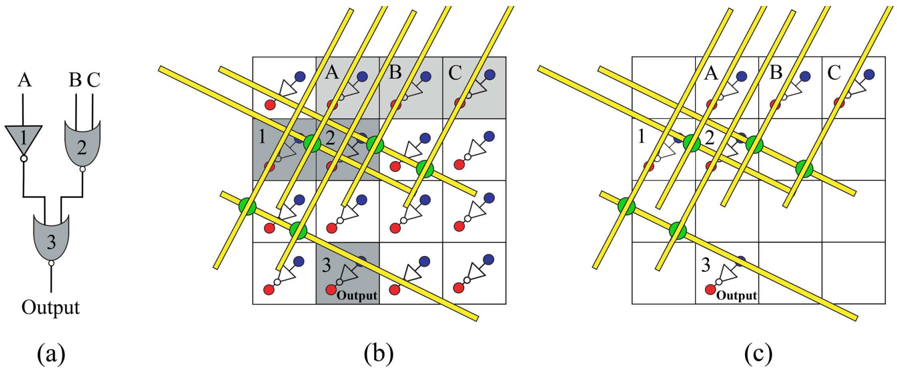
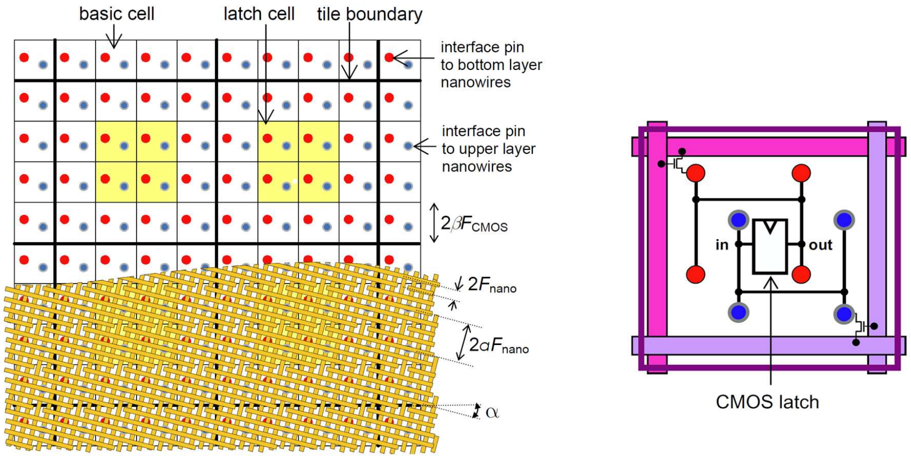


Figure 2.6: Example of CMOL circuit: (a) NOR/INV logical circuit; (b) CMOL implmentaion of (a), (c) showing only used cells.

### 2.2.2 Tile-based FPGA-like CMOL Architecture

Likharev and Strukov extended CMOL architecture into fabric of "tiles" [21, 44]. The fabric is a uniform mesh of square-shaped "tiles" as shown in Figure 2.7(a). Each tile consists of a shell of T basic inverter-based cells surrounding a single "latch" cell shown in Figure 2.7(b). The latter cell is a level-sensitive latch implemented in CMOS subsystem and connected to eight interface pins, plus two pass transistors used for circuit configuration. All four pins of each of the input or the output group are always connected, so nanowires they contact always carry the same signal. The latch cell is assumed to be four times larger than the size of the basic cell. Thus, the total tile area is equal to $T = 12{+}4 = 16 = 4{\times}4$ basic cells. That provide latch/logic resource ratio comparable to conventional FPGAs. For worst case 4-input Boolean function (i.e., 4-input parity function); the function can be implemented using 14 four-input NOR gates, while an average 4-input Boolean function requires much less



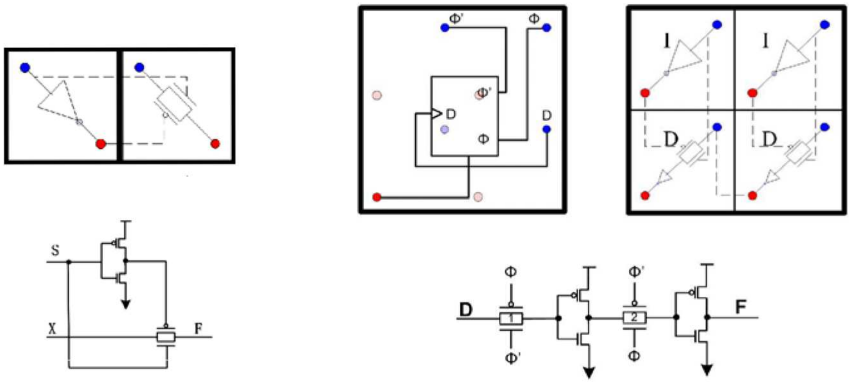
(a) (b)

Figure 2.7: CMOL FPGA: (a) Tiles configuration, (b) Latch cell

gates. Hence, each CMOL tile is crudely similar in functionality to the basic logic element consisting of a four-input LUT and one latch.

### 2.2.3 CMOL Cells Design

Different variations of CMOL cells were developed in the literature; the generic inverter-based cell which was proposed by Likharev [21] is only capable of implementing NOR or NAND based combinational logic. Likharev [21] extended cell types to include latches and later Dong et al [45] presented two CMOL cells called T-Cell and D-Cell for combinational and sequential logic designs. The new cells are capable of implementing functions dependent on transmission or tri-state gates. The proposed T-Cell consists of one transmission gate connected to one generic



(a)

(b)

Figure 2.8: (a) T-Cell: Transmission gate and inverter (b) D-Cell: D flip-flop is formed using two D-Cells and two inverter cells

inverter CMOL cell as shown in Figure 2.8(a). This configuration provide more efficient logic designs for multiplexer (MUX), XOR gate, tri-stat buffers and full adders. The D-Cell consists of one transmission gate and one inverter, two D-Cells and two ordinary inverter cells are connected together to implement a D flip-flop as shown in Figure 2.8(b). The authors claim that those proposed new cells could significantly reduce the number of required CMOL cells in a range of 18% - 43% when implementing circuits that requires transmission or tri-state gates.

Abid et al [46] utilized two types of nanodevices to implement cryptographic algorithms. They develop XOR gate with resistive junctions and XOR/AND gates with diode-like junctions. The proposed design combines two cells each with CMOS inverters and transmission gates to build gates with XOR functionality as shown in Figure 2.9. The input pins of the two cells are the two inputs of the XOR gate, and the output of the right-hand cell is the output of the XOR gate. The output

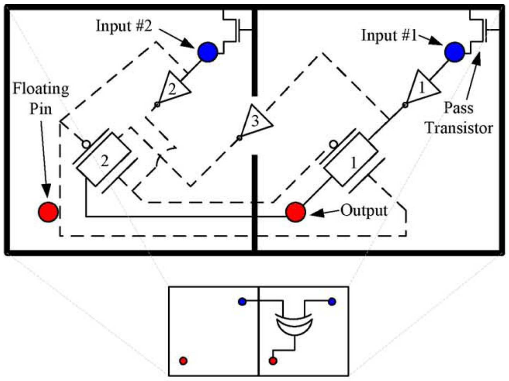


Figure 2.9: XOR cell design based on resistive-based nanodevices.

pin of the left-hand cell is left floating for easier interconnect and routing. The XOR cell is sufficiently larger than conventional inverter cell of CMOL (i.e., three times larger). The AND gate in the XOR/AND cells is implemented using diode-like junction nanodevices, similar to those used to make NOR gates in CMOL. The difference is that the AND gate, Figure 2.10, has diodes in the opposite direction, thus, only when the two inputs are both at logic “1” the diodes are OFF and the output becomes “1”. However, the output of the AND gate is weak and requires a skewed inverter to restore the logic levels, thus, as a limitation each AND gate should be followed with XOR gate, which is sufficient for encryption application but not for general circuitry.

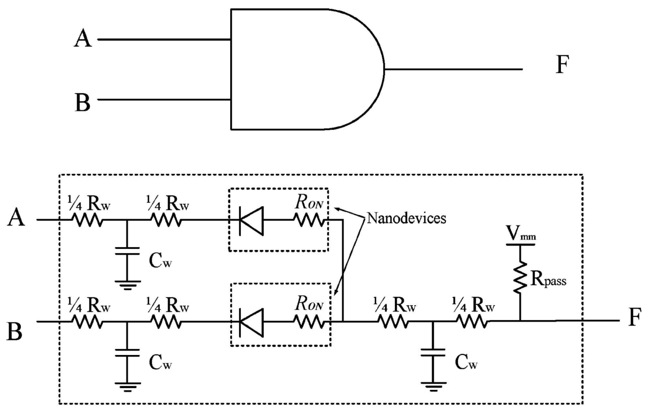


Figure 2.10: AND cell based on diode-like nanodevices and its equivalent circuit. $Rw$ and $Cw$ represent the nanowires.

### 2.2.4 Two-Terminal Latching Nanodevices

The first critical issue in the development of semiconductor/molecular hybrids is making a proper choice in the trade-off between molecule simplicity and functionality. Molecular nanodevices are used to connect perpendicular nanowires at each crosspoint. Nanodevices acting as switches route signals from one nanowire level to the other. Relatively short and rigid molecules (with the number of atoms of the order of one hundred), having two (or a few) metastable internal states, are probably the best choice for the initial development of molecular electronics. A binary "latching switch", i.e., a two-terminal, bistable device with I-V curves, is shown in Figure 2.11(a). Such switch may be readily implemented as a combination of two single-electron devices [9, 12]: a "transistor" and a"trap" (Figure 2.11(b)). If the applied drain-to-source voltage $V = V_d - V_s$ is low, the trap island in equilibrium has no extra electrons, and its net electric charge is zero. Thus, the transistor is in

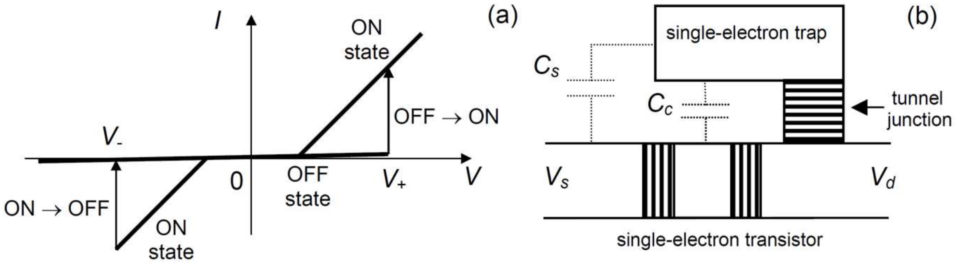


Figure 2.11: Single-Electron two terminal nanodevice: (a) The I - V curves, and (b) A possible implementation of the device.

the virtually closed (OFF) state, and source and drain are essentially disconnected. If $V$ is increased beyond a certain threshold value $V_+$, its electrostatic effect on the trap island potential (via capacitance $C_s$) leads to tunnelling of an additional electron into the trap island. This change of trap charge affects, through the coupling capacitance $C_c$, the potential of the transistor island, and suppresses the Coulomb blockade threshold to a value well below $V_+$. As a result, the transistor, whose tunnel barriers should be thinner than that of the trap, is turned into ON state in which the device connects the source and drain with a finite resistance $R_0$. If the applied voltage stays above $V_+$, this connected state is sustained indefinitely; however, This ON → OFF switching may be forced to happen much faster by making the applied voltage V sufficiently negative, $V = V_-$ [21].

## 2.3 Other CMOS/nanodevices Architectures

Different architectures have been proposed in the literature to address specific design details such as CMOS/nanofabric interfacing, or three dimensional designs. The following sections introduce the Field-Programmable Nanowire Interconnect (FPNI) [1], and 3-D CMOL architecture [2].

### 2.3.1 Field-Programmable Nanowire Interconnect (FPNI)

Field-programmable nanowire interconnect (FPNI) [1] was introduced as a generalization of CMOL architecture, allowing for simpler fabrication, more conservative process parameters, and greater flexibility in the choice of nanoscale devices. Unlike CMOL, logic in FPNI is done only in CMOS buffer-based cells, while routing is handled by nanowires, which allows for reduction of static power dissipation and the use of linear nanodevices (resistive junctions). Figure 2.12 shows the geometry of nanowires, pins and underlying CMOS stack. FPNI assumes a sea of logic gates, buffers and other components (i.e., NAND gates, D flip-flops) in the CMOS layer. Nanowires are rotated so that each one connects to only one pin. The nanowires crossbar include large “pads” to cover the pins, used to simplify fabrication, and eliminating the need for special pins as in CMOL. However, it needs sparse nanowire crossbar to reserve the space for the contacts of the CMOS pins. Thus, the device density of FPNI circuit is substantially lower than that of the corresponding CMOL

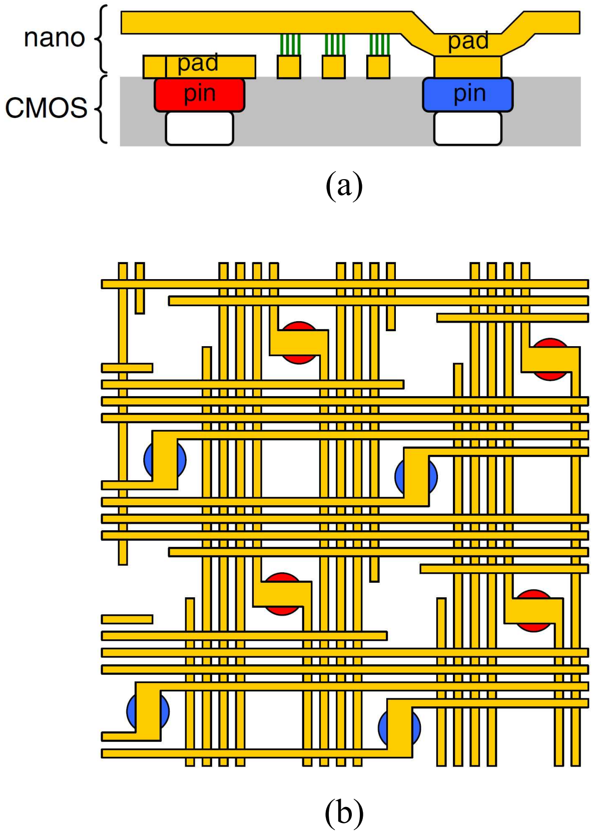


Figure 2.12: FPNI architecture [1]: (a) large pads connecting CMOS and nanowires, (b) nanofabric overlay in FPNI.

circuit (i.e., five folds increase in circuits area [1]).

### 2.3.2 3-D CMOL Architecture

A 3D CMOL FPGA [2] implements circuits in three dimensions, so that it can increase the density of the nanodevices and achieve higher performance compared to 2D CMOL and field programmable nanowire interconnect (FPNI). 3D CMOL can be built by assembling two CMOS layers in a face-to-face manner with the nanowire crossbar layer in between as shown in Figure 2.13. Each CMOS layer reaches to nanowires spanning in one direction. The CMOS layers and nanowire layers are prepared separately, allowing different fabrication technologies for CMOS and nanodevices. The top and bottom nanowires of the crossbar are connected to the top and bottom CMOS dies, respectively, using separate interface pins or vias with the same heights. Thus relaxing the requirement of special pins with different heights in CMOL. This arrangement provides improved density, but with complex inter-cell connectivity as each cell is only restricted to access one level of the nanowire crossbar.

## 2.4 Cell Placement in CMOL Architecture

As CMOL field programmable gate array seems as a promising nanotechnology design that has the potential to be accepted and adopted for future circuits industry,

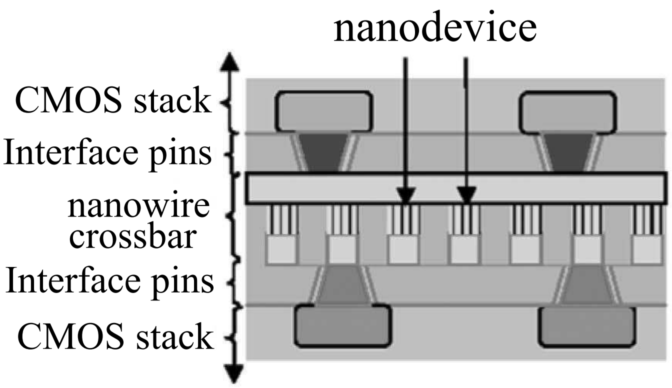


Figure 2.13: 3D CMOL architecture [2].

it is highly important to develop computer aided design (CAD) tools for automated cell placement/assignment. In contrast to traditional placement problem, CMOL cell assignment has the constraint that each gate can only be wired to a limited number of gates in its neighbors *Connectivity Domain.* Thus, investigations should be conducted to devise new solutions to overcome the nanowires connectivity limitation.

### 2.4.1 Theoretical Principles of CMOL Cell Placement

Under the restriction of connectivity domain, an investigation was conducted by Chen et al [47] to outline the principles of CMOL cells placement and to theoretically prove that combinational circuits are placeable in CMOL FPGA generic architecture [3]. The study concludes that any combinational circuit can be transformed to an equivalent circuit which is placeable given a reasonable connection domain size, while, an unlimited fan-out size, could possibly results in unplaceable circuits. For example, given a gate $g$ with fan-out value larger than available cells in the connectivity domain of $g$, then, no connection domain has enough cells to be assigned to all nodes connected to $g$. The problem has to be solved by converting the circuit to an equivalent one with maximum fan-out value of all nodes less than or equal to two [47].

Chen et al, provides certain lemmas and definitions to convert circuit into placeable one in CMOL, mainly by adding even number of inverters (to maintain signal polarity) between cells that could not be placed in each other's connectivity domain. However, adding inverters will further complicate circuit's placement and opens up for tight integration of placement and routing, in which buffer insertion is an indispensable step for complex circuits.

### 2.4.2 Cell Placement in Cell-based CMOL Architecture

Original cell placement into apparently perfect (defect-free) CMOL cell-based architecture (discussed in Section 2.2.1) was done by Likharev et al [3], where an additional reconfiguration step is performed to route the circuit around defective components. The initial assignment is done for artificially confined connectivity domain that have $M' < M$ cells and radius $a' < a$, as analysis showed that most reconfiguration failures come from longest initial connections.

The authors have also developed an automatic procedure (i.e., linear-time algorithm) for reconfiguration assuming only one type of defects (stuck-at-open). The algorithm is based on sequential attempts to move each gate from a cell with bad input or/and output connections to a new cell, while keeping its input and output gates in fixed position. The gate may be swapped with another one, provided that all connections of the swapped gates can be realized with the CMOL fabric and are not defective.

Figure 2.14 shows an example of a circuit fragment reconfiguration, a cell "repair region", Figure 2.14(b), is identified as the overlap of the connectivity domains of all its input and output cells. The cell can be moved to any place in its repair region when all connections are satisfied. In Figure 2.14(c) repair region for two gates intersect, then those two gates can be swapped, keeping the circuit functional. Likharev [3] provided Monte Carlo simulations of two simple circuits ( a 32-bit integer adder and a 64-bit full crossbar switch) which have shown that the reconfiguration allows one to increase the circuit yield above 99% at the fraction of bad nanodevices above 20%.

A novel solution to CMOL cell assignment problem was reported using satisfiability [48], and it was extended as a reconfiguration tool for various CMOL defects. First, the authors transform logically synthesized circuits based on AND/OR/NOT gates to a NOR gate circuits and then, they encode the CMOL cell assignment problem as Boolean conditions. These Boolean constraints are satisfied if and only if there exists a solution to map all the NOR gates to the CMOL cells. Further, they

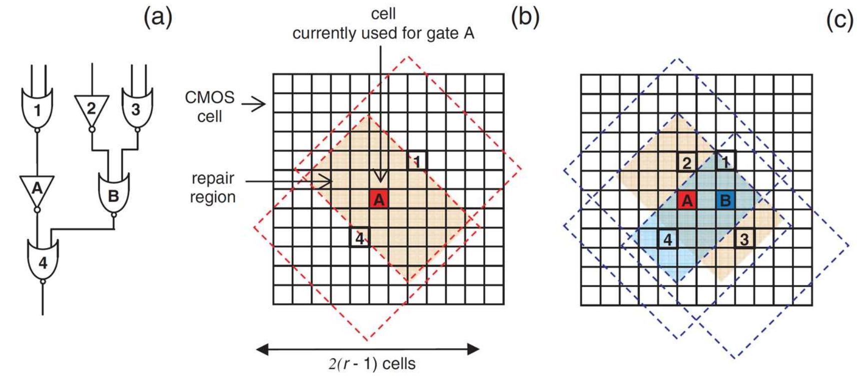


Figure 2.14: Example of a circuit fragment reconfiguration. (a) Circuit whose gate A is to be relocated. (b) The repair region of gate A. (c) intersection of the repair region of cells [3].

introduce additional Boolean conditions to satisfy reconfiguration around bad components. However, satisfiability in general works for small to medium sized problems, and when circuits sizes increased the computation time became exhibitant.

Previous attempts to use sub-optimal search heuristics are reported in [49, 50, 51]. Genetic Algorithm (GA) [49] were formulated using two dimensional block partially mapped crossover operator (PMX). The PMX is applied to solve the duplication problem by position based pair-wise exchanges. Mutation is applied to produce spontaneous random changes in various individuals, by pairwise exchange. Fitness function used to evaluate the adaptive ability of the solution is based on wirelenghts of connected NOR gates, and penalizing infeasible solutions when connection length is longer than "connectivity domain" radius. Nonetheless, memory requirements, choices of data structure for chromosomes representation, and computation time are significant disadvantages of GA.

A more elaborate work was reported in [50]; where Memetic computing approach (MA) was used for cell mapping task in CMOL. Memetic Algorithm is an efficient heuristic for solving complex optimization problems [52]; it implements a hybrid of Genetic Algorithm that use genetic operators to explore the mating pool and Simulated Annealing [4] (SA) local-based search heuristic to improve the quality of solutions. MA have the same genotype structure, crossover, and mutation the one used in GA [49]. Simulated Annealing algorithm was used in each generation to enhance offsprings resulted from PMX crossovers and pairwise interchange mutations in GA.

Hung et al extend their work on Memetic approach by integrating self-learning operators using Lagrangian Multipliers into so called LRMA approach [51]. LRMA uses same local search, based on Simulated Annealing (SA) along with penalty updating mechanism using Lagrangian multipliers. At the end of each loop of the MA search process, the cell assignment solution is examined for any pair of violating gates, then Lagrangian multipliers representing penalty values in fitness function are then updated. Results reported using LRMA approach are promising, however, more computations are needed for penalty updating mechanism. GA, MA and LRMA approaches does not account for fault or defective components, and only assign cells on the assumption of faulty-free fabric.

### 2.4.3 Cell Placement in Tile-based CMOL Architecture

As CMOL has different variations, the cell assignment in CMOL “tile-based” architecture [44] (discussed in Section 2.2.2) is conducted by a custom set of tools for CMOL FPGA design automation [53]. The CMOL tile design was original developed to utilize existing cluster-based FPGA CAD tools, and the netlist of NOR gates is partitioned into logic clusters, each with N gate and one latch, using T-VPack program [54]. The logic clusters are then mapped on the CMOL tile fabric using VPR tool [55, 54] and Simulated Annealing (SA). The cost function try to place gates into locations such that their interconnect is local or within tile connectivity domain of each other.

CMOL tile design, Figure 2.7(a), with size equal to $T$ basic cells, consists of $N$ cells reserved for logic operations, one latch, and $T - N$ cells for routing purposes. Global routing, which is needed to connect gates across different tiles is performed with the help of a routing algorithm which consider a set of “nets”. Each net will be routed by configuring an even number of routing inverters from the $T - N$ logic-free basic cells in each tile. If the net has more than one “output” cluster, the algorithm tries to minimize the total number of routing cells by sharing them among different connections, such problem is equal to finding the shortest-path Steiner tree [56].

At this stage routing congestion could happen as routing resources (i.e., routing cells in each tile) could not be sufficient to route all possible nets. In subsequent step the algorithm identifies the nets which are routed using tiles with the maximum number of routing cells and tries to reroute them by applying the algorithm's steps again. Starting with the longest nets and using slack analysis to keep critical paths the same, rerouting is performed around congestion. Figure 2.15 shows an example of routing procedure, where tile size is assumed to be 5 × 5 basic cells, and cell I needs to be connected to cells O1, O2, O3. Five inverters (routing cells) are used to suffice the connection.

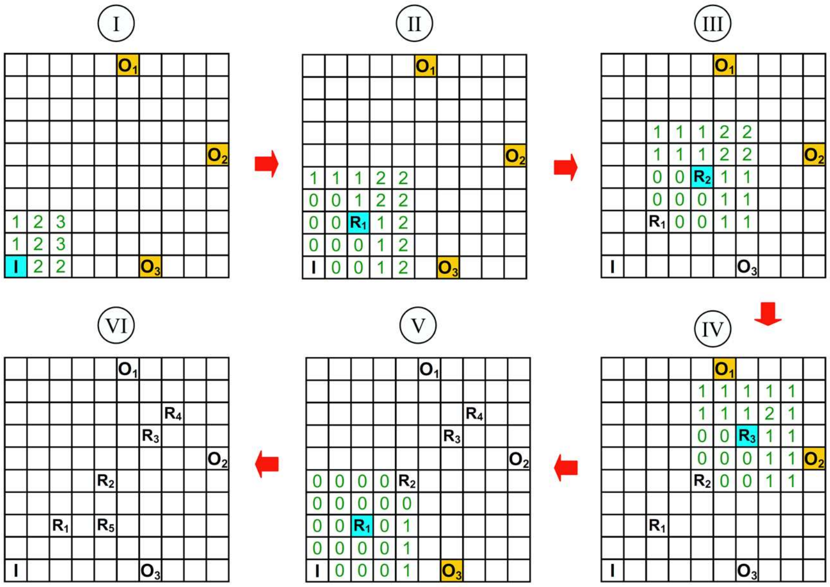


Figure 2.15: Example of global routing for a single net.

The CMOL design flow used in [21, 44] is shown in Figure 2.16. In case no feasible solution can be reached due to congestion, the whole design flow is repeated with a reduced number of logic gates (N) is each tile, leaving more space to routing cells.

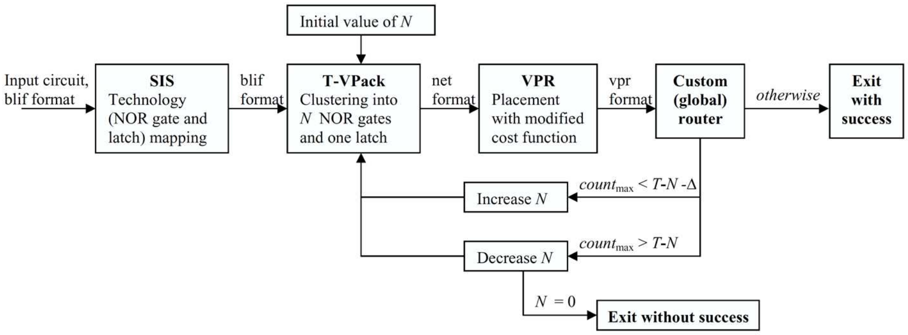

Initial value of N
Input circuit, blif format
SIS Technology (NOR gate and latch) mapping
blif format
T-VPack Clustering into N NOR gates and one latch
net format
VPR Placement with modified cost function
vpr format
Custom (global) router
otherwise
Exit with success
$count_{\max} < T\text{-}N\text{ -}\Delta$
Increase N
$count_{\max} > T\text{-}N$
Decrease N
N = 0
Exit without success

Figure 2.16: CMOL FPGA CAD 1.0 design flow.

When N equals to 0 the process is terminated and the circuit can't be implemented.

Hossein et al [57] presented a modification to the routing flow of CMOL CAD tool. They rank placement solution of each iteration of the placement algorithm according to the placement cost, (i.e., placement solution which has lower cost, has higher priority for being the platform for routing). When circuit is routed without congestion, this placement should be accepted and the algorithm is terminated. Otherwise, a new placement solution from the ranked list is provided for routing. The procedure carries on until the algorithm routes the circuit without congestion. Although, results provided by Hossein et al claims to be able to route circuits that CMOL CAD failed to, but it doesn't account for the consequences (i.e., delay, area) of using inferior placements.

## 2.5 Non-deterministic Iterative Heuristics

Several iterative heuristics has been used to solve combinatorial optimization problems, in this section we discuss two known heuristics; Simulated Evolution (SimE) proposed by Kling and Banerjee [58] and Tabu Search (TS) which was introduced by Fred Glover [59, 60, 61, 62].

### 2.5.1 Simulated Evolution

The SimE heuristic is similar to Simulated Annealing except that the elements that are movable have a goodness value (a number between 0 and 1). Those with goodness value close to 1 have a smaller possibility to leaving their locations, while those with smaller goodness have otherwise.

The structure of the SimE algorithm is shown in Figure 2.17. SimE assumes that there exists a solution $\phi$ of a set M of n (movable) elements or modules. The algorithm starts from an initial assignment $\phi_{initial}$, and then, following an evolution-based approach, it seeks to reach better assignments from one generation to the next by perturbing some ill-suited components and retaining the near-optimal ones. A cost function *Cost* associates with each assignment of movable element $m_i$ a cost $C_i$. The cost $C_i$ is used to compute the goodness (fitness) $goodness_i$ of an element $m_i$, for each $m_i \in$ M. The algorithm has one main loop consisting of three basic steps, Evaluation, Selection, and Allocation. The three steps are executed in sequence until the solution average goodness reaches a maximum value, or no noticeable improvement to the solution cost is observed after a number of iterations. The Evaluation step consists of evaluating the goodness $goodness_i$ of each element $m_i$ of the solution $\phi$. The goodness measure must be a single number expressible in the range [0, 1], and can be defined as

$$goodness_i = \frac{O_i}{C_i} \tag{2.2}$$

Where $O_i$ is an estimate of the optimal cost of element $m_i$, and $C_i$ the actual cost of $m_i$ in its current location. Or simply goodness can be defined as the fraction of two values related to the problem cost.

The second step of the SimE algorithm is Selection. Selection takes as input a bias value B, the solution $\phi$ together with the estimated goodness of each element. It partitions $\phi$ into two disjoint sets; a selection set S and a partial solution $\phi_p$ of the remaining elements of the solution $\phi$. Each element in the solution is considered separately from all other elements. The decision whether to assign an element $m_i$ to the set $S$ is based solely on its goodness $goodness_i$. The selection operator shown in Figure **??**, has a non-deterministic nature, i.e., an individual with a high goodness (close to one) still has a non-zero probability of being assigned to the selection set S. It is this element of non-determinism that gives SimE the capability of escaping local minima. Allocation is the SimE operator that has the most important impact on the quality of solution. Allocation takes as input the set $S$ and the partial solution $\phi_p$ and generates a new complete solution $\phi'$ with the elements of set $S$ mutated according to an allocation function. The goal of Allocation is to favor improvements over the previous generation, without being too greedy [4].

### 2.5.2 Tabu Search

Tabu Search is a general iterative metaheuristic for solving combinatorial optimization problems. TS is an elegant heuristic that proceeds by making iterative perturbations while preventing cycling to certain number of recently visited points in search space. The TS procedure starts from an initial feasible solution $S$ (current solution) in the search space $\Omega$. A neighborhood $\aleph(S)$ is defined for each $S$. A sample of neighbor solutions $\mathbf{V}^* \subset \aleph(S)$ is generated called *trial* solutions ($n = |\mathbf{V}^*| \ll |\aleph(S)|$), and comprises what is known as the candidate list. From this generated set of trial solutions, the best solution, say $S^* \in \mathbf{V}^*$ is chosen for consideration as the next solution. A solution $S^* \in \aleph(S)$ can be reached from $S$ by an operation called a move to $S^*$. The move to $S^*$ is considered even if $S^*$ is worse than $S$, that is, $Cost(S^*) > Cost(S)$. Selecting the best move in $\mathbf{V}^*$ is based on the supposition that good moves are more likely to reach the optimal or near-optimal solutions. The best candidate solution $S^* \in \mathbf{V}^*$ *may* or *may not* improve the current solution, but is still considered. It is this feature that enables *escaping* from local optima. However, with this strategy, it is possible to reach the local optimum, since

```
ALGORITHM $Simulated\_Evolution(B, \Phi_{initial}, StoppingCondition)$
NOTATION
$B$= Bias Value.    $\Phi$= Complete solution.
$m_i$= Module $i$.    $g_i$= Goodness of $m_i$.
$ALLOCATE(m_i, \Phi_i)$=Function to allocate $m_i$ in partial solution $\Phi_i$
Begin
Repeat
  EVALUATION:
        ForEach $m_i \in \Phi$ evaluate $g_i$;
  SELECTION:
        ForEach $m_i \in \Phi$ DO
             begin
                IF $Random \leq 1 - g_m + B$
                THEN
                   begin
                      $S = S \cup m_i$; Remove $m_i$ from $\Phi$
                   end
             end
      Sort the elements of S
  ALLOCATION:
        ForEach $m_i \in S$ DO
             begin
                $ALLOCATE(m_i, \Phi_i)$
             end
Until    Stopping Condition is satisfied
Return Best solution.
End ($Simulated\_Evolution$)
```

Figure 2.17: Structure of the Simulated Evolution algorithm [4].

moves with $Cost(S^*) > Cost(S)$ are accepted, and then in a later iteration return back to local optimum.

In order to prevent returning to previously visited solutions a memory or list $\mathbf{T}$, known as *tabu list*, is maintained. This list contains information that to some extent forbids the search from returning to a previously visited solution. Whenever a move is accepted, its attributes are introduced into the tabu list $\mathbf{T}$. Move reversal are prevented for next $k = |\mathbf{T}|$ iterations because they *might* lead back to a previously visited solution. The tabu list can be visualized as a window on accepted moves as shown in Figure 2.18. The moves which tend to undo previous moves within this window are forbidden.

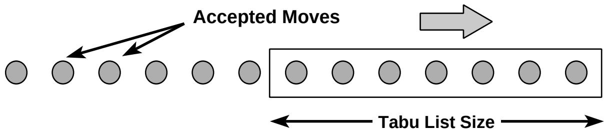


Figure 2.18: Tabu List visualized as window over accepted moves [4].

In some cases, it is necessary to overrule the tabu status since only move attributes (not complete solutions) are stored in tabu lists. These tabu moves may also prevent the consideration of some solutions which were not visited earlier. This is done with help of the notion of *aspiration criterion*. Aspiration criterion is a device used to override the tabu status of moves whenever appropriate. It temporarily overrides the tabu status if the move is sufficiently good. Aspiration criterion must make sure that the reverse of a recently made move leads the search to an unvisited solution, generally a better one. A flow chart illustrating the basic short-term memory tabu search algorithm is given in Figure 2.19. Intermediate-term and long term memory processes are used to intensify and diversify the search respectively [4].

One of the Tabu search algorithm parameters is the size of the tabu list. A small tabu list size is preferred for exploring the solution near a local optimum, and a larger tabu list size is preferable for breaking free of the vicinity of local minimum. The list size varying between 5 and 12 have been used in many applications. Any aspect (feature or component of a solution) that changes as a result of a move from $S$ to $S^{trial}$ can be an attribute of that move, where a single move can have several attributes. The duration for which a move containing the particular tabu attribute is forbidden (the size of tabu list) is called Tabu tenure. An algorithmic description of a simple implementation of tabu search is given in Figure 2.20.

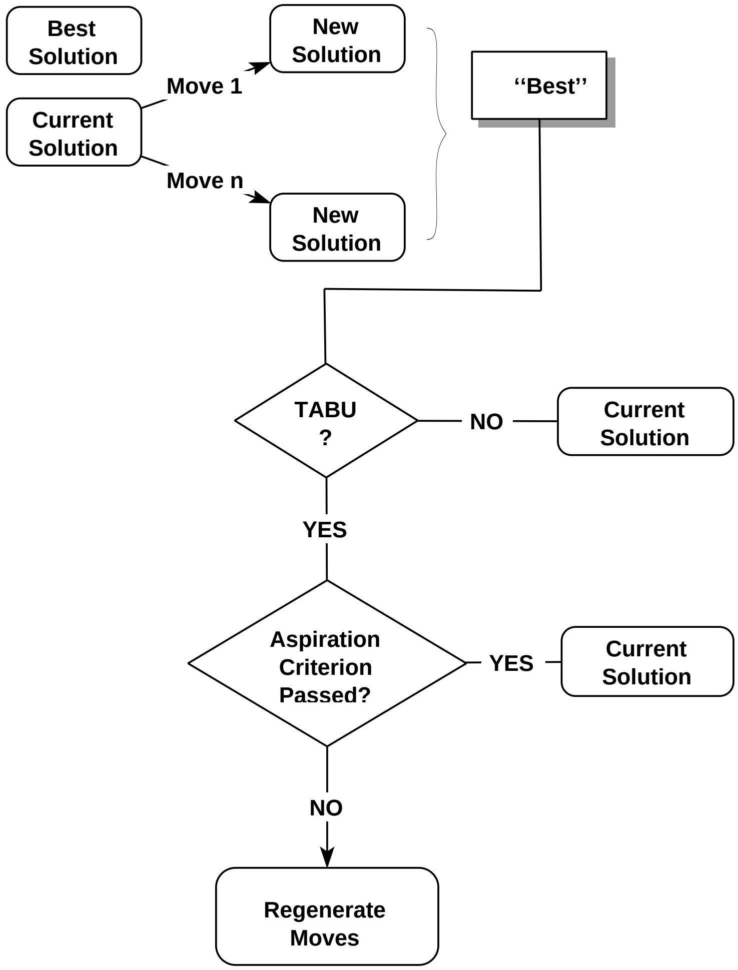


Figure 2.19: Flow-Chart of Tabu Search algorithm [4].

| | | |
|---|---|---|
| $\Omega$ | : | Set of feasible solutions. |
| $S$ | : | Current solution. |
| $S^*$ | : | Best admissible solution. |
| $Cost$ | : | Objective function. |
| $\aleph(S)$ | : | Neighborhood of $S \in \Omega$. |
| $\mathbf{V}^*$ | : | Sample of neighborhood solutions. |
| **T** | : | Tabu list. |
| **AL** | : | Aspiration Level. |

```
Begin
Start with an initial feasible solution $S \in \Omega$.
Initialize tabu lists and aspiration level.
For fixed number of iterations Do
    Generate neighbor solutions $\mathbf{V}^* \subset \aleph(S)$.
    Find best $S^* \in \mathbf{V}^*$.
    If move $S$ to $S^*$ is not in T Then
        Accept move & update best solution.
        Update tabu list.
        Update aspiration level.
        Increment iteration number.
    Else
        If $Cost(S^*) <$ AL Then
            Accept move & update best solution.
            Update tabu list & aspiration level.
            Increment iteration number.
        EndIf
    EndIf
EndFor
End.
```

Figure 2.20: Algorithmic description of short-term Tabu Search (TS) [4].

# Chapter 3

# Problem Description and Design Automation

## 3.1 Problem Statement

Cell placement is a design automation step that involves a collection of cells or modules with input/output ports, and a collection of nets (which are sets of ports that are to be wired together). *Placement* consists of finding suitable physical locations for each cell on the entire layout. By suitable we mean those locations that minimize a given objective function, subject to certain constraints imposed by the designer. Like in the case of most programmable nanofabrics, the length of the nanowires in CMOL is restricted, and therefore connectivity of the circuit's elements (i.e., cells) is constrained to be within a certain radius. In addition, nanofabric architectures have high defect rates. Nanodevices connecting circuit's modules can be defective (i.e., not programmable). A *Reconfiguration* of the circuit's elements is required to avoid using any defective devices. The solution sought for CMOL cells mapping problem comprises of employing iterative search heuristics to find an assignment that respects the connectivity constraint and does not use defective resources.

## 3.2 Problem Formulation

### 3.2.1 Placement

The placement or assignment of cells in order to minimize a cost function has been proven to be an NP-hard problem [4]. Even one dimensional placement, the simplest possible, is hard to solve. In a 2-D array of $n$ locations there are as many as:

$$S = n(n-1)(n-2)...(n-m+1) \tag{3.1}$$

arrangements for placing $m$ cells, where $m$ could be in thousands. Overtime, heuristic techniques have been developed to solve the placement problem.

Implementation of combinational logic using CMOL involves assignment of logic gates (i.e., NORs or Inverters) to slots that are connected by programmable nanodevices placed between overlapping nanowires. The length of the nanowires is restricted, and therefore connectivity of the circuit's elements is constrained. Each CMOL cell is connectable to its proximity cells, those cells comprise its input and output *Connectivity Domains*, each has radius equal to $a$. Any violation of this constraint would impose further processing (i.e., buffer insertion) to satisfy connectivity.

Formally, CMOL placement problem can be stated as follows: for a set of gates $G = g_1, g_2, g_3, ..., g_m$ and a set of nets $\Gamma = \gamma_1, \gamma_2, \gamma_3, ..., \gamma_m$ where $\gamma_i = \{fan - in_i \ \& \ fan - out_i\}$ of $g_i$ and given a set of slots or locations $L = L_1, L_2, L_3, ..., L_n$ where $m \leq n$, the placement problem is to assign each $g_i \in G$ to a unique location $L_j$ such that the objective is optimized. Positions are defined by the coordinate values $(x_j, y_j)$ where the subset of G that represent inputs/outputs may be pre-assigned fixed locations or constrained to certain positions. Mathematically, CMOL placement constraints can be defined as follows; given a gate and its net $(g_i, \gamma i)$ placed in location $L_i$, for any gate $g_k \subseteq G$ and $g_k$ in the net $\gamma i$ the following equations should be satisfied.

$$\forall g_i, g_k \in G : (g_i \neq g_k) \Rightarrow (L_i \neq L_k) \tag{3.2a}$$

$$\forall g_i \in G, \exists g_k \in \gamma_i : dist(L_i, L_k) \leq a \tag{3.2b}$$

Where $L_k$ is the location of $g_k$, $dist$ is Manhattan distance, and $a$ is CMOL connectivity radius. Inequality 3.2(b) defines a domain that is an approximation to the one shown in Figure 2.3, this definition has been used in previous works related to CMOL cell placement [49, 50, 51], nonetheless, adhering to the original shape of the connectivity domain [21] is preferable. The objective of CMOL cell placement is to satisfy the constraints in Equation 3.2, and to minimize distance between connected gates in circuit $G$. Failing to comply with CMOL constraint will result in an implementation that has more delay and area requirements.

The difficulty of CMOL placement arises from the overlap in connectivity domain of adjacent cells; each cell is considered as part of the connectivity domain of other $M = a^2 - 2$ cells. The mapping of a particular gate into a given cell will limit the connectivity options of those other cells that are located in its proximate neighborhood. The value of the connectivity radius $a$ and the size of NOR gates has substantial effect on the realization of CMOL designs; if gates with high fan-in are allowed, the placement problem becomes substantially harder.

### 3.2.2 Reconfiguration

There are tens of thousands of nanodevices (i.e., possible connections) in a CMOL circuit, however, those connections will not be used simultaneously and a small subset of them is sufficient to map a particular circuit. The extra nanodevices are mainly intended for better reconfigurability especially in the presence of defects. Misassembly of the two-terminals bistable nanodevices will lead to non-programmable crosspoints (i.e., stuck-at defects). Defects including broken wires and bridging of adjacent nanowires have also been reported by Dehon [63]. In this work we will consider three of the widely used defect models:

1. Stuck-at-Open: The nanodevice connecting two perpendicular nanowires is stuck-at-open (i.e., not programmable). In this case, the connection between two cells through this nanodevice is not feasible. However, those two CMOL cells can still be used. The connectivity domain of the two cells should be modified.

2. Broken Nanowire: An input or output nanowire of a CMOL cell is broken into two segments. Thus, CMOL cell may not be able to connect to all other CMOL cells within its input/output connectivity domains. The connectivity domains of the cells will be significantly reduced.

3. CMOS Cell Defect: In this defect, CMOS cell could be unusable, because the input/output pins connecting the CMOS stack to the input/output nanowires are broken, or the CMOS inverter is defective. Any cell with this type of defect cannot be used.

Defects of type 1 and type 2 are shown in Figure 3.1; in stuck-at-open defects, the input of cell A can not be connected with the output of cells B and C, because the nanodevices between them are not programable. In broken nanowire defect; the number of unreachable cells (e.g., cells B, C and D) from a given cell (e.g., cell A) could be larger as more connections are affected by the cut in the nanowire.

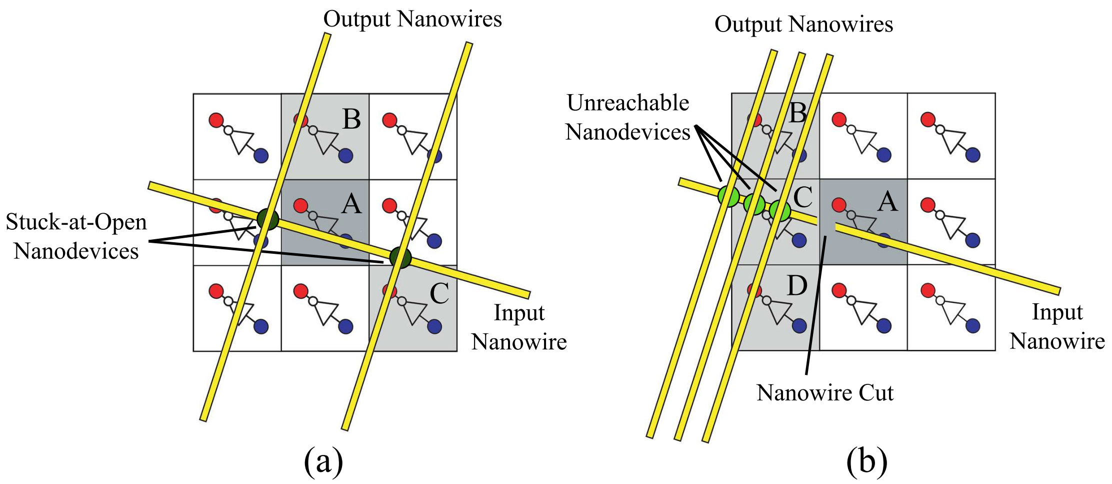


Figure 3.1: Defects in CMOL circuits: (a) Stuck-at-Open defect (b) Broken nanowire defect. The cells shown in light gray are not reachable by the cell in dark gray.

Based on the aforementioned defect models; *Reconfiguration* involves rearranging CMOL cells as to avoid the use of any defective nanodevices. According to the CMOL FPGA topology shown in Figure 2.3, if a particular cell is moved to another location it will use different set of nanodevices to connect with its fan-in and fan-out cells. Reconfiguration should not relocate two cells in which their connectivity is violated (i.e., invalidate the assignment set by *Placement* step), but rather to only avoid using any defective components. For a given gate and its net $(g_i, \gamma i)$, any gate $g_k$ in the net $\gamma i$ should satisfy the following.

$$\forall g_i \in G : L_i \neq 0 \tag{3.3a}$$

$$\forall g_i \in G, \exists g_k \in \gamma_i : N(L_i, L_k) \neq 0 \tag{3.3b}$$

Where $N(L_i, L_k)$ is the nanodevice connecting gate $g_i$ in location $L_i$ and gate $g_k$ in location $L_k$. $N = 0$ means the nanodevice is defective and $L = 0$ means the location (i.e., CMOL cell) has a defect of type 3. CMOL reconfiguration is intended to rearrange cells to honor the constraints in Equation 3.3, and meanwhile not violating the constraints in Equation 3.2. Reconfiguration is highly dependent on the defect rate and connectivity radius $a$. For small connectivity radius, high defect rate may lead to reconfiguration failure.

## 3.3 Cost Functions

The main objective of placement is to find a feasible assignment of cells in which all connections are satisfied. One way to accomplish this is to place strongly connected cells close to each other. A commonly used objective function is the total weighted wirelength over all signal nets and is expressed as:

$$L(P) = \sum_{n \in N} w_n \cdot d_n \tag{3.4}$$

Where, $d_n$ is the estimated wirelength of net $n$ and $w_n$ is weight of net $n$. Since, in CMOL all cells are connected via pre-assembled nanowires, the problem we are trying to optimize is to place connected cells within each others connectivity domain as to avoid insertions of additional buffers. Therefore, we should have a measure which can quantify the overall quality of the solution. A conventional approach is to calculate the number of connections that violate connectivity domain constraint (Equation 3.2). The overall cost of a solution is the total number of connectivity domain violating connections (i.e., the number of additional buffers that are needed to satisfy all connections). The cost of each gate $g \in G$ is expressed in Equation 3.5, where the overall circuit's cost is the sum of individual gates cost.

$$C_i = \sum_{j \in \gamma(i)} u_{i,j} \tag{3.5a}$$

$$u_{i,j} = \begin{cases} 1 & \text{if } dist(L_i, L_j) > a \\ 0 & \text{otherwise} \end{cases} \tag{3.5b}$$

Similarly, the cost of Reconfiguration step is the total number of used defective nanodevices (i.e., the number of connections that violate Equation 3.3(b)). Equation 3.6 shows the cost of each gate $g \in G$ as to be the number of defective components it uses to connect with its fan-in and fan-out cells. The overall circuit's cost is the sum of individual gates cost.

$$C_i = \sum_{j \in \gamma(i)} u_{i,j} \tag{3.6a}$$

$$u_{i,j} = \begin{cases} 1 & \text{if } N(L_i, L_j) = 0 \\ 0 & \text{otherwise} \end{cases} \tag{3.6b}$$

## 3.4 Defect Maps

In nanowire crossbars, imprecision and nondeterminism of the nanoscale fabrication process may cause the programmable nanodevices to be defective. Different methods for simulating faults distribution has been reported in the literature [64]. In this work, two methods are used for stuck-at-open faults simulation. In the first approach, a uniform random distribution is used. For any given nanodevice, a random number $p$ is generated, the nanodevice could be defective if $p$ is less than a pre-defined defect rate $q_{nano}$. In the second approach, clustered faults are injected around multiple defect sources. Each cluster is generated as follow; first a random location $(x_0, y_0)$ is chosen, and then a probability mass function $pmf(x, y)$ is computed for each location using the Gaussian distribution:

$$pmf(x, y) = Ce^{-\frac{(x-x_0)^2+(y-y_0)^2}{2\sigma^2}} \tag{3.7}$$

This probability mass function controls the injection of faults; where $C$ is a constant that sets the density of the simulated faults, and $\sigma$ is the standard deviation that controls the diameter of the defects cluster. For each nanodevice we generate a random number $p$ between 0 and 1. A fault is injected if $p \leq pmf(x, y)$.

For broken nanowires defects; a nanowire is cut if a randomly generated number $p$ is less than wires cut rate $q_{wire}$, and the cut point is randomly specified. All unreachable nanodevices on the cut nanowire are then encoded as if they are stuck-at-open. In similar manner, CMOS cells are assumed to be defective based on a defect rate $q_{cell}$.

Figure 3.2 shows defect maps for stuck-at-open defect rate $q_{nano} = 30\%$ and wires cut rate $q_{wire} = 10\%$. The first map shows randomly distributed defects and the second shows clustered defects when $C = 0.7$ and $\sigma = \frac{a}{3}$. White dots represent non programmable nanodevices (i.e., stuck-at-open), while black dots represent programable ones.

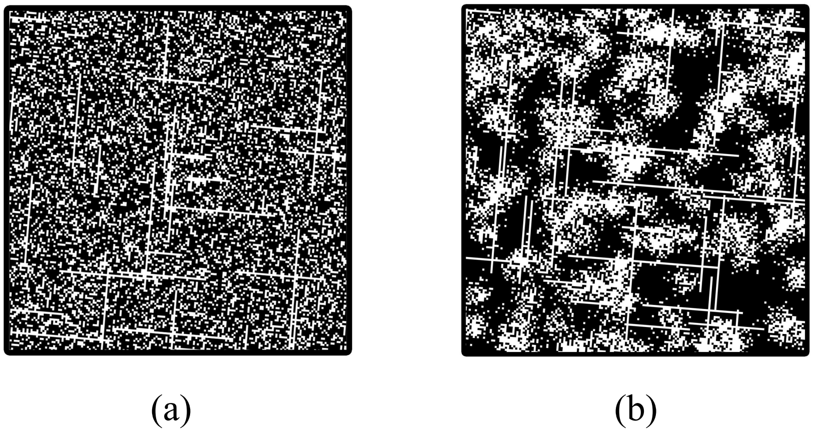


Figure 3.2: Defect maps: (a) Random defects (b) Clustered defects.

## 3.5 Design Flow

Using Technology Mapping and known synthesis tools (e.g., SIS), circuits can be mapped into a network of NOR gates (with a certain maximum fan-in). Figure 3.3 shows the design flow for mapping logic circuits into CMOL grid; three main steps are required to implement a given logical design in CMOL:

1. Defects unaware circuit *Placement.*
2. Circuit *Reconfiguration* around defective components.
3. Complementary *Routing* of connections with violations.

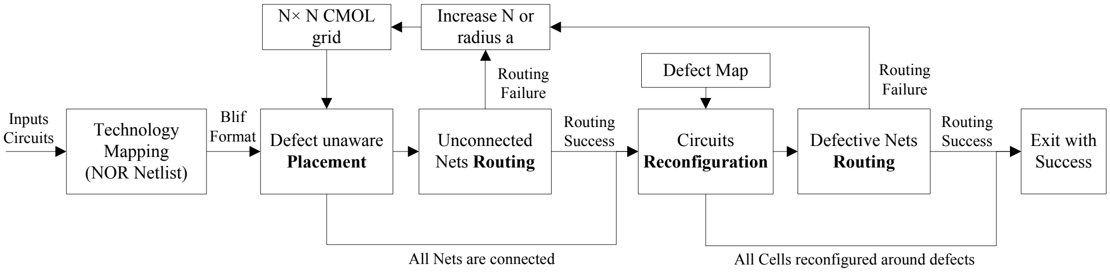

N× N CMOL grid
Increase N or radius a
Routing Failure
Defect Map
Routing Failure
Inputs Circuits
Technology Mapping (NOR Netlist)
Blif Format
Defect unaware **Placement**
Unconnected Nets **Routing**
Routing Success
Circuits **Reconfiguration**
Defective Nets **Routing**
Routing Success
Exit with Success
All Nets are connected
All Cells reconfigured around defects

Figure 3.3: Design flow of CMOL cells mapping.

In the first step, the circuit is placed in defect-free $N \times N$ grid. *Placement* follows the formulation outlined in Section 3.2.1 and constraints in Equation 3.2. Nondeterministic heuristics iteratively rearrange gates locations as to minimize the cost function. The only defect that placement phase is aware of is type 3 defect (i.e., defective CMOL cell). When placement is finished, some connections may still be unresolved (i.e., beyond the connectivity domain). In this case, routing step is needed to insert extra buffers (i.e., pair of inverters to maintain signal polarity) as intermediate cells between unconnected gates. Each pair of inverters can make a cell connect to another cell whose distance is within $3a$. The routing step can only be performed after all NOR gates are placed. Routing may not be successful in case the grid do not have empty slots or some cells are unreachable (i.e., all cells in its connectivity domain are occupied). In that case, placement should be repeated using bigger CMOL grid or longer connectivity radius.

The second step, involves reconfiguring the circuit around defective components. Defects information will be stored in a defect map. *Reconfiguration* iteratively improves circuit's reliability by rearranging gates locations so they do not use defective nanodevices or defective CMOS cells. Constraints in Equation 3.2 and Equation 3.3 should apply for the reconfigured cells. If circuit's reconfiguration is not successful (i.e., some connections still use defective components), a routing step, similar to the one used after placement, can be used to connect those connections with stuck-at-open defective nanodevices.

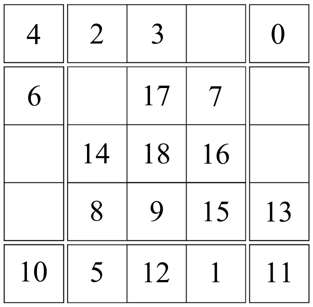
# Chapter 4

# Non-deterministic Evolutionary Heuristics for CMOL Cell Mapping

In this chapter, we discuss the non-deterministic evolutionary heuristics that are used to solve CMOL cells mapping problem. Simulated Evolution (SimE) and Tabu Search (TS) were used to find an arrangement of cells that satisfy the objectives outlined in Chapter 3. In the following sections we will focus on the design of various operators and parameters to better explore the search space.

## 4.1 Solution Representation

A placement solution is an arrangement of logic cells in two dimensional layout surface. The solution representation used in this work is in the form of a 2-D grid with $N \times N$ location. The layout is constructed by computing the number of required CMOL cells to fit each benchmark circuit. The outer cells of the grid are reserved for I/O pins, where placement of the circuit's I/Os is restricted to these reserved locations. Each logic gate is assigned a positive integer value that distinguishes it from the rest. The encoded logic gates are assigned in the 2-D layout as shown in Figure 4.1.



| | | | | |
|---|---|---|---|---|
| 4 | 2 | 3 | | 0 |
| 6 | | 17 | 7 | |
| | 14 | 18 | 16 | |
| | 8 | 9 | 15 | 13 |
| 10 | 5 | 12 | 1 | 11 |

Figure 4.1: 2-D grid layout of CMOL placement for $s27.blif$. 19 cells; 8 gates, 7 inputs and 4 outputs.

## 4.2 Simulated Evolution

The Simulated Evolution algorithm (see Section 2.5.1) is a general search strategy for solving a variety of combinatorial optimization problems. It is stochastic because the selection of which elements to be reallocated is done according to a stochastic rule. Already well located components have a high probability to remain where they are. The following sections describe the design of the main steps of the algorithm.

### 4.2.1 Initialization

Initialization is the step that comes before the iterative evolutionary phase of the algorithm. In this step, the various parameters of the algorithm are set to their desired values; such as the maximum number of iterations, and the selection bias $B$. SimE construct an initial solution by randomly assigning gates to locations in the 2-D grid. It has been proven that the quality of the initial solution has little impact on the convergence aspects of the heuristic, nonethess, strating from a good soultion could reduce the number of iterations required to converge to a near-optimal solution [4].

The quality of the SimE solution improves over iterations; the improvement is significant in early iterations and gets less steeper in later iterations. One significant aspect of SimE iterations that early ones require more time than later iterations. The reason is that as more and more iterations get executed, less and less cells get selected for reallocation. Usually, the number of iterations is fine tuned based on experiments and problem size. The value of the selection bias $B$ should be much less than 1. A positive value will increase the number of elements selected to allow the algorithm to search harder. Although, this may lead to better solutions, but at the expense of runtime. On the other hand, a negative value of $B$ will speed-up the heuristic as less elements are selected for reallocation, but that may results in early convergence.

### 4.2.2 Goodness Function

In Simulated Evolution, goodness function is used to evaluate individual elements in each generation, where unfit elements are selected and reassigned to other locations. The goodness measure must be strongly related to the objective of the problem, in that sense the goodness function of each individual element (i.e., gate) in the *placement phase* is defined as following:

$$goodness_i = \frac{inside_i}{|\gamma_i|} \tag{4.1}$$

Where $inside_i$ represents the number of those gates in set $\gamma_i$ (the net of gate $i$), that satisfy the connectivity constraint (i.e., inside the connectivity domain of element $i$) and $|\gamma_i|$ is the number of fanin and fanout gates of gate $i$. The above equation assumes a minimization problem (or a maximization of goodness). Figure 4.2 shows an example on how goodness value is calculated, where two gates in $\gamma_i$ are outside the connectivity domain of gate $i$ and three otherwise. The proposed goodness function results in probabilistic selection of those elements that violate constraint expressed in Equation 3.2(b), and therefore, directs the heuristic into enhancing the overall cost of the problem. Figure 4.2 assumes that the connectivity

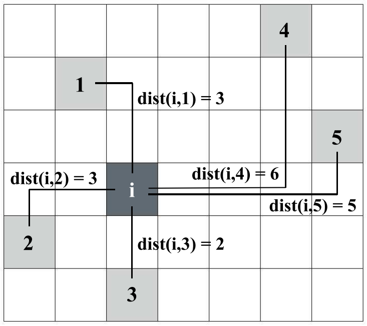


Figure 4.2: Evaluation of gate $i$'s goodness; for $r = 3$ cells 1, 2 and 3 are inside $i$'s connectivity domain (i.e., $dist \leq r$), while cells 4 and 5 are out of it (i.e., $dist > r$), $goodness_i = 3/5 = 0.6$.

domain is defined based on Manhattan distance and Equation 3.2(b). The same can apply if connectivity domain is defined as in Figure 2.3. *Reconfiguration phase* uses similar goodness evaluation, where each element's goodness is defined as follows:

$$goodness_i = \frac{connect_i}{|\gamma_i|} \tag{4.2}$$

Where $connect_i$ represents the number of connections in set $\gamma_i$ that do not use defective nanodevices (i.e., the connections that are defect free). According to the aforementioned definition, if cell $i's$ connections violate the constraint in Equation 3.3(b), the cell will have low goodness value. An example of such cell is shown in Figure 4.3, where two defective nanodevices are used to connect gates 4 and 5 with gate $i$. According to the given definitions of the goodness function, the value of $|\gamma_i|$ do not change from generation to generation, but it is only computed once based on the original circuit description.

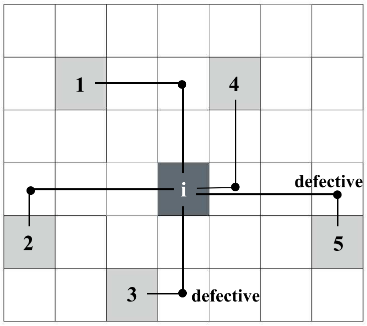


Figure 4.3: Evaluation of gate $i$'s goodness; connection between cell $i$ and cells 4 and 5 use defective nanodevices, $goodness_i = 3/5 = 0.6$. Nanodevices are shown as black dots.

### 4.2.3 Selection Function

The Selection phase uses original SimE selection function [4]; an element (i.e., gate) is selected for reallocation if its goodness score is less than a randomly generated number between 0 and 1. The higher is the goodness value of the element, the higher is its chance of retaining its current location. While, the lower is the goodness value, the more likely the element will be perturbed and reallocated in the next generation. SimE selection function has a nondeterministic nature; an individual with a high goodness (i.e., close to one) still has a non zero probability of being selected. This stochastic role gives SimE the hill climbing property. Reallocating the selected elements can be done in a deterministic order that is correlated with the objective function being optimized. Hence, prior to the Allocation step, the elements in the selection set are sorted in an ascending order based on their net size, where elements with higher cardinality of $\gamma_i$ are processed first.

### 4.2.4 Allocation Function

Allocation function has the most impact on the quality of the solution; it's intended to generate a new solution that is inherently better than the old one. The design of the allocation function is related to the problem specifications. The allocation function is a complex form of genetic mutation, it alters the locations of all elements in the selection set one after the other. In our case the alteration consists of swapping the location of one module with the location of another one. Allocation function seeks to swap an element with any other element in the solution. The trial that leads to the best configuration with the respect to the objective being optimized is accepted and made permeant. This constitute a global allocation policy, which could prove to be very useful specially in the early iterations.

The allocation function in *placement phase* chooses to swap a selected element (i.e., gate) with another one, given that this swap is the best in terms of the cost function (i.e., number of buffers). If two swaps has the same cost, the one that results in smaller Manhattan distance will be chosen. In this phase, allocation procedure is unaware of defects (i.e., swaps or moves can result in using defective nanodevices). The only aspect of defects that this phase is aware of is the one related to Equation 3.3(a), where a particular gate should not be placed in a defective CMOS cell. On the other hand, allocation function in *reconfiguration phase* is fully aware of the presence of defects. It actually, swap cells based on the cost defined by the number of used defective nanodevices. For each selected element, the allocation function evaluates the cost of swapping the element with another one in the grid based on the cost function in Equation 3.6. Then, the best swap is chosen. An additional constraint also applies for gates movements in reconfiguration phase; the reallocation of cells is constrained to the region defined by the intersection of the connectivity domains of the two cells under investigation and their fanin and fanout cells (see Figure 2.14). This insure that reconfiguration do not invalidate the assignment made by placement phase (i.e., do not move cells in which some of the connections would again require buffers to be resolved).

### 4.2.5 Routing

Since CMOL grid may be highly congested, using a greedy algorithm to exhaustively route connections (i.e., insert buffers) may not be successful. Instead, the problem can be solved by the same iterative methods used for placement and reconfiguration; for that we recall the SimE heuristic given a number of modifications. All blank cells in the grid are considered for buffers insertion. Inverters are placed in empty locations as to maximize their goodness. Initially, the *routing phase* starts with one buffer (pair of inverters) for each unconnected or defective connections. The inverters are randomly assigned. The allocation functions (similar to those of placement and reconfiguration) only permit the interchange of two routing inverters or an inverter and blank cell. The algorithm continuously improve the locations of the inverters. If the algorithm terminates and still some connections violate the connectivity domain, additional buffers are added for those connections and the routing algorithm is repeated. This process repeats until all connected gates are within each others connectivity domain, or when no blank cells are left. A reasonable way to avoid worsening the circuit's timing delay by inserting too many buffers, is to limit the number of buffers that can be inserted for each gate pair.

## 4.3 Tabu Search

Tabu Search algorithm (see Section 2.5.2) is a non-deterministic iterative heuristic that has been applied to solve many combinatorial optimization problems, it is considered as a generalization of local search algorithms. At each step, Tabu Search explore the local neighborhood of the current solution and the best solution in that neighborhood is selected as the new current solution. In the following sections we will discuses the various parameters used in solving the objectives of this work.

### 4.3.1 Initialization

Based on the solution representations outlined in Section 4.1, TS begin by randomly assigning the circuit's gates to the cells (i.e., locations) in CMOL grid.

### 4.3.2 Neighborhood Solutions Generation

A move in Tabu Search consists of a small perturbation in the current solution to explore solutions that are in the proximate neighborhood of the current one. An efficient move should help to quickly explore the neighborhood search space. In *placement phase*, one perturbation is performed as follows: two cells (two I/O pins or two logic cells) are selected randomly, then their locations are interchanged. In each iteration we generate a number of neighbor solutions (i.e., a candidate list), each one of these solution is generated by performing one move (i.e., perturbation). Each solution in the candidate list is evaluated based on the change in number of buffers before and after the swap. If two or more neighborhood solutions have equal swap cost, which also happens to be the best cost in the candidate list, the solution with lesser Manhattan distance is chosen. Similarly, *reconfiguration phase* uses the same tabu move; the cost in Equation 3.3 is used to evaluated each swap, and it is based on the number of defective nanodevices that are being used. Furthermore, the swaps should not violate the constraints set by Equation 3.2 (see Figure 2.14). TS will favor the swap (from those in the candidate list) that results in least cost (i.e., no. of buffers, or no. of defective nanodevices).

The size of the candidate list constitute the amount of search space exploration in the local neighborhood of a given solution. A small neighborhood size ensures quicker iterations, but may require more iterations to reach a good solution. On the contrary, a bigger candidate list provides more exploration of the search space at the cost of increased computation overhead. Nonetheless, it may ensure arrival to a good solution in lesser number of iterations. It is thus concluded that the same solution can be reached in equivalent amount of time by either using smaller candidate list and higher number of iterations or bigger candidate list and smaller number of iterations [4]. The actual size of the candidate list is empirically set based on the performance of the heuristic and the problem behavior.

### 4.3.3 Tabu List and Move Attributes

Tabu search algorithm avoids returning to previously visited solutions by using a Tabu List (i.e., memory element that stores information about previous moves). The search therefore is forced away from recently visited solutions. The tabu list contains attributes of some $k$ most recent moves. The size of the tabu list is the number of iterations for which a move containing that attribute is forbidden after it has been made. Early experiments on practical problems reported good performance with list sizes varying between 5 and 12. The size of the tabu list is also related to the strictness of the tabu restriction; the more stringent the restrictions the smaller should be the size of the tabu list. The actual size of the list can be determined by experimental runs, watching for occurrence of cycling when the size is too small, and the deterioration of solution quality when the size is too large. A short term memory element is used in our implementation, where Tabu list is implemented as a queue (FIFO) data structure.

Move attributes are used to impose tabu restrictions to prevent reversal of changes represented by these attributes. Many attributes of a move can be stored, for example, when two cells $i$ and $j$ are swapped, one attribute is to forbid moves related to cell $i$; which means any move that includes $i$ (even swapping $i$ with $j$) is restricted. Another attribute considers both $i$ and $j$, forbidding any perturbations that include either of them. The Tabu attribute of a move that is used in our implementation is swap reversal. That means if cell $i$ and $j$ are swaped, the reversal swap is forbidden for the next $k$ iteration. Tabu restrictions play the role of diversifying the search by perturbing unperturbed elements.

### 4.3.4 Aspiration Criterion

The aspiration criterion consists of overriding the tabu status when plausible solutions is reached. In our implementation aspiration criterion is based on the following: if the current solution is the best seen so far (i.e., better than the global best solution) then tabu restriction is overridden and the current solution is accepted as new best solution and tabu list is updated. This criterion is know as Global Aspiration by Objective and is widely used.

### 4.3.5 Routing

The *routing phases* use the same parameters and operators explained earlier; such as tabu move, move attribute, candidate list and tabu list sizes. The only difference is that swaps are allowed only between added inverters or an inverter and blank cell. The other cells locations are assumed to be fixed and no perturbation can involve any one of them. The routing phase iteratively try to insert buffers (pair of inverters) between unconnected or defective connections. The insertion follows the same constraints and objectives as in placement and reconfiguration. The locations of the inverters are perturbed as to place them within the connectivity domains of their fanin and fanout gates. The routing procedure follows the same steps as explained under Section 4.2.4 for SimE implementation.

# Chapter 5

# Experimental Results and Comparison

This chapter presents results obtained by implementing Simulated Evolution and Tabu Search for CMOL cell placement and reconfiguration. The following sections describe simulation environment, benchmarks, and defect maps. Next, an evaluation of the employed heuristics is given. This is followed by comparison with previously published techniques about CMOL cell placement (see Section 2.4.2). Finally, we show how the solutions obtained by SimE and TS are verified.

## 5.1 Simulation Environment

Simulated Evolution and Tabu Search are implemented using Java programming language and run on a machine comparable to the one used by other simulations published in literature. The machine has 1.5 GHz Intel Pentium M processor with 512MB memory. Technology mapping is done using SIS logic synthesis tool [65]. Verification and defect maps modelling programs are also written in Java programming language. Comparisons between CMOL solutions and original benchmarks are done by HOPE simulator [66], which is run on a LINUX machine.

Simulated Evolution and Tabu Search heuristics stop when solution cost (either number of buffers or number of defective nanodevices) becomes zero or when reaching a predefined number of iterations, in our case the number of iterations in SimE for all phases of the design flow is equal to 4000, on the other hand, a significantly larger number of iterations is used in Tabu Search. The median value of the results of 20 successful runs for each benchmark is reported where each run uses different random numbers seed.

### 5.1.1 Benchmarks

Evaluation of the employed search heuristics is conducted using 19 circuits of different sizes from ISCAS'89 benchmarks suite [67]. Further consideration should be given to ISCAS'89 circuits as they include structural faults that should be eliminated. Furthermore, the benchmarks contain sequential elements (i.e., flip-flop); CMOL generic architecture can only implement combinational logic, thus, the sequential elements' inputs and outputs are substituted with POs and PIs respectively. The circuits are then mapped by SIS synthesis tool [65] to a NOR netlist with maximum of five inputs. Details of the circuits are shown in Table 5.1; the numbers of *Cells* to be placed including *Gates*, *Inputs* and *Outputs* are given.

Table 5.1: ISCAS'89 Benchmarks.

| Circuits | Cells | Gates | Inputs | Outputs |
|---|---|---|---|---|
| s27 | 19 | 8 | 7 | 4 |
| s208 | 136 | 109 | 18 | 9 |
| s298 | 122 | 85 | 17 | 20 |
| s344 | 180 | 130 | 24 | 26 |
| s349 | 184 | 134 | 24 | 26 |
| s382 | 175 | 124 | 24 | 27 |
| s386 | 164 | 138 | 13 | 13 |
| s400 | 188 | 137 | 24 | 27 |
| s420 | 299 | 248 | 34 | 17 |
| s444 | 187 | 136 | 24 | 27 |
| s510 | 304 | 266 | 25 | 13 |
| s526 | 273 | 222 | 24 | 27 |
| s641 | 302 | 206 | 54 | 42 |
| s713 | 321 | 225 | 54 | 42 |
| s820 | 447 | 400 | 23 | 24 |
| s832 | 454 | 407 | 23 | 24 |
| s838 | 606 | 507 | 66 | 33 |
| s1196 | 675 | 613 | 31 | 31 |
| s1238 | 724 | 662 | 31 | 31 |

### 5.1.2 Defect Maps and CMOL Grids

Given the random and clustered defect maps discussed in Section 3.4, we have evaluated the heuristics performance using a randomly generated map ($R1$) and two clustered maps ($C1$ and $C2$). Those maps have $C = 0.8$ and standard deviation $\sigma = \frac{2a}{3}$ and $\sigma = \frac{4a}{3}$ for $C1$ and $C2$ respectively. The experiments involving defects follow one of the scenarios shown in Table 5.2.

Table 5.2: Defect Scenarios.

| Scenario | $q_{nano}$ | $q_{wire}$ | $q_{cell}$ |
|---|---|---|---|
| (i) | 10% - 50% | 20% | 0% |
| (ii) | 20% | 10% - 70% | 0% |
| (iii) | 20% | 20% | 10% - 20% |

Where $q_{nano}$ is the probability a nanodevice is stuck-at-open (type 1 defect), $q_{wire}$ is the probability a nanowire is broken (type 2 defect), and $q_{cell}$ is the probability a CMOS cell is defective (type 3 defect). For example, scenario (i) include five experiments when $q_{nano}$ ranges between 10% and 50%, where $q_{wire} = 20\%$ and $q_{cell} = 0\%$.

Table 5.3 shows CMOL grids used to place ISCAS'89 benchmarks; Area ($Tiles$) is the area used by CMOL FPGA CAD 1.0 tool [44], it uses $4 \times 4$ cells in each tile. Area ($Row \times Column$) is the area used in GA [49], MA [50], LRMA [51] and in our implementation of SimE and TS for defect free placement and for reconfiguration of defect scenarios (i) and (ii). The table also shows the grids for scenario (iii), when

$q_{cell}$ equals to 10% and 20%. Table 5.4 shows the area utilization for CMOL FPGA CAD 1.0, GA, MA, LRMA and our implementation using SimE and TS. The $AU\%$ represents the ratio of those cells that are used to implement the benchmarks in a given CMOL grid. The table further gives the ratio of utilized cells when CMOS defect probability $q_{cell}$ equals to 10% or 20%.

Table 5.3: CMOL 2-D grid sizes.

| Circuit | Area ($Tiles$) | Area<br>($Row \times Column$) | Area<br>($Row \times Column$)<br>$q_{cell} = 10\%$ | Area<br>($Row \times Column$)<br>$q_{cell} = 20\%$ |
|---|---|---|---|---|
| s27 | $64(2 \times 2)$ | $25(5 \times 5)$ | - | - |
| s208 | $256(4 \times 4)$ | $169(13 \times 13)$ | - | - |
| s298 | $256(4 \times 4)$ | $144(12 \times 12)$ | - | - |
| s344 | $400(5 \times 5)$ | $196(14 \times 14)$ | - | - |
| s349 | $400(5 \times 5)$ | $196(14 \times 14)$ | - | - |
| s382 | $400(5 \times 5)$ | $196(14 \times 14)$ | - | - |
| s386 | $400(5 \times 5)$ | $196(14 \times 14)$ | - | - |
| s400 | $400(5 \times 5)$ | $196(14 \times 14)$ | - | - |
| s420 | $400(5 \times 5)$ | $361(19 \times 19)$ | - | - |
| s444 | $400(5 \times 5)$ | $196(14 \times 14)$ | - | - |
| s510 | - | $361(19 \times 19)$ | - | - |
| s526 | $576(6 \times 6)$ | $324(18 \times 18)$ | - | - |
| s641 | $576(6 \times 6)$ | $676(26 \times 26)$ | $676(26 \times 26)$ | $676(26 \times 26)$ |
| s713 | - | $676(26 \times 26)$ | $676(26 \times 26)$ | $676(26 \times 26)$ |
| s820 | - | $529(23 \times 23)$ | $576(24 \times 24)$ | $625(25 \times 25)$ |
| s832 | - | $529(23 \times 23)$ | $576(24 \times 24)$ | $625(25 \times 25)$ |
| s838 | - | $676(26 \times 26)$ | $729(27 \times 27)$ | $784(28 \times 28)$ |
| s1196 | - | $729(27 \times 27)$ | $841(29 \times 29)$ | $900(30 \times 30)$ |
| s1238 | - | $784(28 \times 28)$ | $900(30 \times 30)$ | $961(31 \times 31)$ |

Table 5.4: CMOL area utilization.

| Circuit | AU% ($Tiles$) | AU% | AU% $q_{cell} = 10\%$ | AU% $q_{cell} = 20\%$ |
|---|---|---|---|---|
| s27 | 18.75 | 76.00 | - | - |
| s208 | 48.05 | 80.47 | - | - |
| s298 | 48.83 | 84.72 | - | - |
| s344 | 43.50 | 91.84 | - | - |
| s349 | 26.50 | 93.88 | - | - |
| s382 | 43.25 | 89.29 | - | - |
| s386 | 54.75 | 83.67 | - | - |
| s400 | 47.25 | 95.92 | - | - |
| s420 | 75.00 | 82.83 | - | - |
| s444 | 52.50 | 95.41 | - | - |
| s510 | - | 84.21 | - | - |
| s526 | 57.12 | 84.26 | - | - |
| s641 | 50.17 | 44.67 | 44.67 | 44.67 |
| s713 | - | 47.49 | 47.49 | 47.49 |
| s820 | - | 84.50 | 77.60 | 71.52 |
| s832 | - | 85.82 | 78.82 | 72.64 |
| s838 | - | 89.64 | 83.13 | 77.30 |
| s1196 | - | 92.59 | 80.26 | 75.00 |
| s1238 | - | 92.35 | 80.44 | 75.34 |

## 5.2 Placement

As stated in Section 3.5; placement in CMOL assumes a defect free grid, in which gates are placed following the constraints and cost function given in Chapter 3. The following sections evaluate the employed heuristics; first we start with simulated evolution, then Tabu Search parameters are discussed. The comparison with previous techniques (e.g., Genetic Algorithm, and Memtic Algorithm) is presented afterward.

### 5.2.1 Simulated Evolution

A verification of Simulated Evolution heuristic is shown in Figure 5.1; at the beginning of iterations many elements are selected for perturbation, after that less and less elements are selected until a final cells arrangement is decided. The feasibility of CMOL cell placement is heavily dependent on the number of circuit's connections and the size of the connectivity domain. For example, a circuit with high fan-in NOR gates will result in bigger nets that could not possibly be fit inside a particular connectivity domain. On the other hand, limiting the size of NOR gates will simplify CMOL placement, but on the expense of using more cells per circuit.

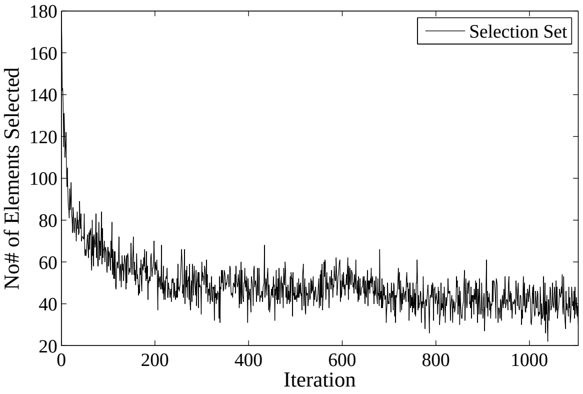


Figure 5.1: Evaluation of SimE performance: selected elements Vs. iterations ($a =$ 12 - s1238.blif).

Three options were available to find a placement solution that satisfies the CMOL connectivity constraint; one, was to minimize Manhattan distance, another was to minimize number of inserted buffers by using the cost function discussed in Section 3.5, and the third was to minimize both distance and buffers. Results obtained

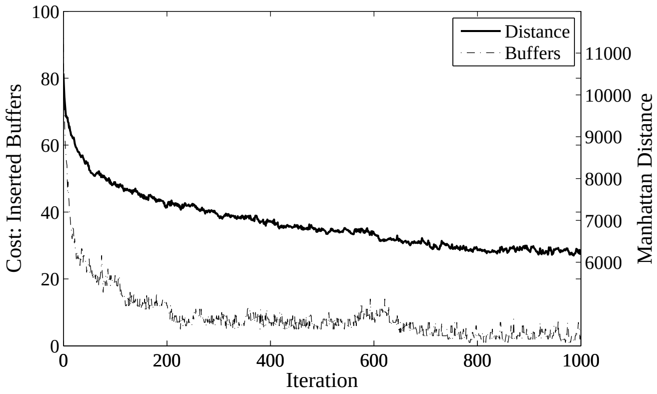


Figure 5.2: Correlation between distance minimization and buffers insertion in SimE iterations ($a = 12$ - s1238.blif).

for the first option showed that the number of inserted buffers are more than that of second option. Moreover, when minimizing buffers, Manhattan distance was reduced to similar levels as if distance was being optimized. Minimizing both distance and buffers required more processing and didn't have significant advantage over minimizing buffers only.

Figure 5.2 shows the cost (i.e., number of buffers inserted) per iteration. SimE progressively converges, in reasonable number of iterations, toward an optimal configuration where each gate is optimally located, it also performs hill-climbing to avoid local optimum. Simulated Evolution heuristic is directed into selecting and reallocating only those gates that are violating the connectivity domain constraint. Reducing the number of buffers will inevitably reduce the overall Manhattan distance.

### 5.2.2 Tabu Search

A short term memory element is used throughout the implementation where experiments of tabu list size ranging from 5 to 12 were conducted. It was concluded that change in tabu list size in this range has little impact on the quality of the solutions, thus the size of tabu list is taken as a fixed value equal to 5.

We have experimented with different sizes of the candidate list; Figure 5.3 shows the final cost yielded by TS in four benchmark circuits when candidate list size is changed, given that all other parameters are constant. It is clearly seen that for this problem, TS had better results when more neighbor solutions are considered. Figure 5.4 shows the per iteration cost of one circuitry given different sizes of candidate list. Candidate list size of 50 is reaching the optimal solution of zero buffers (when $a = 12$) in less iterations, thus this size has been used throughout our implementation.

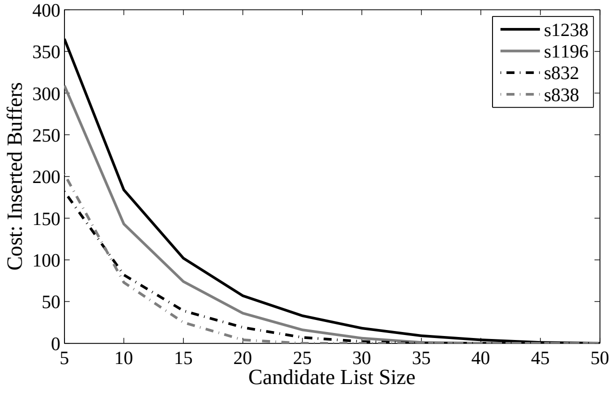


Figure 5.3: Final cost yielded by TS in four circuits vs. candidate list size ($a = 12$).

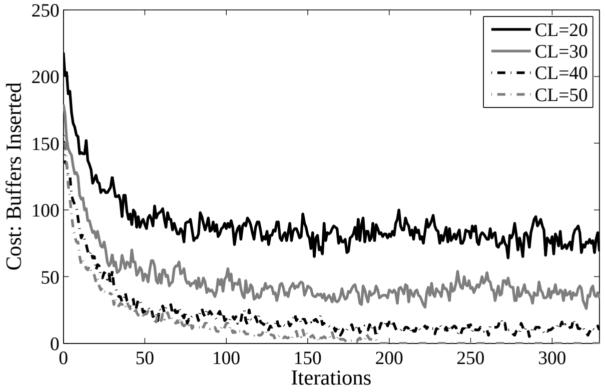


Figure 5.4: Change in cost per iteration of s1238.blif for different candidate list sizes ($a = 12$).

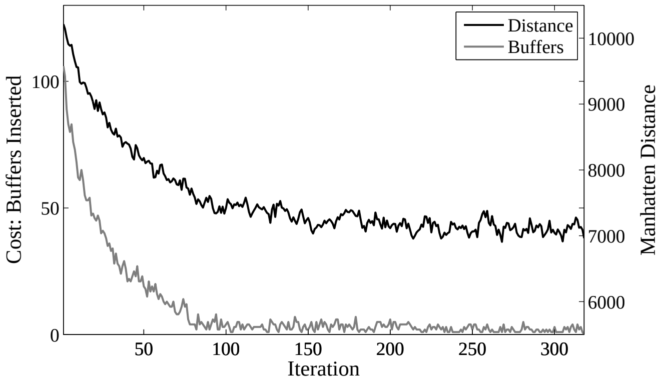


Figure 5.5: Change of problem cost and Manhattan distance in TS iterations ($a = 12$ - s1238.blif).

Figure 5.5 shows the correlation between the number of inserted buffers and Manhattan distance. It's clear that TS is accepting bad moves in order to reach better solutions in terms of inserted buffers, which are also better in terms of Manhattan distance.

### 5.2.3 Results Comparison

In this section we compare the results of Simulated Evolution and Tabu Search with CMOL FPGA CAD 1.0 [44], GA [49], MA [50] and LRMA [51] (see Section 2.4.2), when connectivity domain radius $a = 12$ and $a = 9$. Table 5.5 and Table 5.6 show the final results obtained for ISCAS'89 benchmarks; ($D$) is the circuit's logical levels reported by SIS tool after inserting the buffers, computation time ($T$) in seconds, ($B$) shows the number of inserted buffers to satisfy CMOL connectivity domain, and ($T\%$) is the percentage of time improvement compared with computation time reported for LRMA. GA, MA and LRMA use population size equals to 24 and stopping criteria when fitness score is not updated for 50 times. The crossover rate in MA and LRMA is $RC = 0.33$ and mutation rate $RM = 0.01$. Simulated Annealing used in each of GA iterations has initial temperature $T = 0.2$ and terminating temperature 0.01.

Simulated Evolution and Tabu Search solutions are more effective than those of CMOL CAD 1.0 in terms of computation time, delay, and even they are able to place large circuits that CMOL CAD failed to. Results obtained from implementation of SimE for $a = 12$ are better than those obtained in GA, MA and LRMA in both computation time and buffers count. SimE required shorter CPU processing time compared to time required by genetic crossover, mutation and Lagrangian multipliers calculation in LRMA. Table 5.5 shows that Simulated Evolution found the optimal solutions with zero buffers for all benchmarks, with up to 82% average computation time saving. For example, SimE required only 23.50 seconds to find the optimal solution of zero buffers and delay equals to 23 for benchmark s1196, while LRMA required 179.47 seconds and needed 9 buffers to satisfy connectivity raising timing delay to 24.

Results of Tabu Search for $a = 12$ are better than those obtained in GA, MA and LRMA in both computation time and buffers count and similar to those of Simulated Evolution. TS required shorter CPU processing time due to its simplified operations compared to genetic crossover and Lagrangian multipliers calculation in LRMA. Table 5.6 shows that Tabu Search found the optimal solutions with zero buffers for all benchmarks, with 92% average computation time saving. For example, s1238 benchmark needed only 12.87 seconds in TS, comprising only a 3.6% of time needed by LRMA, and 23.93% of time needed by SimE.

Table 5.7 shows SimE, and TS results when $a = 9$; solutions found by TS are better than those of MA for all benchmark circuits. TS falls behind LRMA in only two circuits (s820 and s1238) while sustaining equal averaged results. TS found solutions in lesser time with 73% saving. Simulated Evolution is better than published techniques; it finds placement solutions that have less cost and in efficient computation time. As shown in Table 5.7, SimE required almost quarter the time to find a solution with one third of the number of buffers reported by LRMA for s1238 benchmark. SimE has worse results than LRMA only in one circuit (s713).

Table 5.5: SimE Comparison With CMOL CAD, GA, MA and LRMA - ($a = 12$).

| Circ. | CAD | | GA [49] | | | MA [50] | | | LRMA [51] | | | SimE | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | D | T | D | T | B | D | T | B | D | T | B | D | T | B | T% |
| s27 | 9 | 1 | 7 | 0.01 | 0 | 7 | 0.01 | 0 | 7 | 0.01 | 0 | 7 | 0.01 | 0 | 0.00 |
| s208 | 18 | 3 | 16 | 1.12 | 0 | 16 | 0.12 | 0 | 16 | 0.10 | 0 | 16 | 0.01 | 0 | 90.00 |
| s298 | 13 | 7 | 11 | 0.17 | 0 | 11 | 0.11 | 0 | 11 | 0.09 | 0 | 11 | 0.01 | 0 | 88.89 |
| s344 | 20 | 8 | 18 | 0.57 | 0 | 1 | 0.29 | 0 | 18 | 0.16 | 0 | 18 | 0.01 | 0 | 93.75 |
| s349 | 20 | 7 | 18 | 0.49 | 0 | 18 | 0.28 | 0 | 18 | 0.18 | 0 | 18 | 0.02 | 0 | 88.89 |
| s382 | 13 | 7 | 11 | 1.60 | 0 | 11 | 0.38 | 0 | 11 | 0.32 | 0 | 11 | 0.04 | 0 | 87.50 |
| s386 | 16 | 11 | 10 | 1.05 | 0 | 10 | 0.33 | 0 | 10 | 0.34 | 0 | 10 | 0.02 | 0 | 94.12 |
| s400 | 15 | 8 | 11 | 2.12 | 1 | 11 | 0.40 | 0 | 11 | 0.34 | 0 | 11 | 0.03 | 0 | 91.18 |
| s420 | 20 | 8 | 16 | 8.50 | 1 | 16 | 3.41 | 0 | 16 | 1.57 | 0 | 16 | 0.09 | 0 | 94.27 |
| s444 | 17 | 9 | 11 | 1.86 | 2 | 11 | 0.40 | 0 | 11 | 0.34 | 0 | 11 | 0.03 | 0 | 91.18 |
| s510 | - | - | 18 | 16.56 | 2 | 18 | 7.56 | 0 | 18 | 3.42 | 0 | 18 | 0.16 | 0 | 95.32 |
| s526 | 16 | 13 | 11 | 9.75 | 5 | 11 | 4.36 | 0 | 11 | 1.59 | 0 | 11 | 0.25 | 0 | 84.28 |
| s641 | 25 | 8 | 23 | 82.66 | 15 | 19 | 39.40 | 4 | 16 | 22.02 | 0 | 16 | 2.92 | 0 | 86.74 |
| s713 | - | - | 24 | 52.84 | 34 | 19 | 30.11 | 3 | 19 | 41.77 | 2 | 19 | 3.40 | 0 | 91.86 |
| s820 | - | - | 15 | 77.52 | 41 | 12 | 61.71 | 10 | 12 | 54.09 | 6 | 12 | 27.72 | 0 | 48.75 |
| s832 | - | - | 16 | 69.27 | 54 | 12 | 60.17 | 11 | 12 | 63.77 | 4 | 12 | 31.00 | 0 | 51.39 |
| s838 | - | - | 28 | 201.37 | 50 | 24 | 85.62 | 7 | 24 | 100.40 | 4 | 24 | 2.42 | 0 | 97.59 |
| s1196 | - | - | 30 | 234.88 | 84 | 23 | 208.15 | 19 | 24 | 179.47 | 9 | 23 | 23.50 | 0 | 86.91 |
| s1238 | - | - | 37 | 268.92 | 121 | 28 | 267.34 | 31 | 26 | 353.00 | 9 | 26 | 53.76 | 0 | 84.77 |
| Avg. | - | - | 17 | 54.28 | 22 | 15 | 40.53 | 4 | 15 | 43.31 | 2 | 15 | 7.65 | 0 | 82.33 |

D: Delay (i.e., Logic Levels).
T: Computation Time in Seconds.
B: Buffers Inserted.
T%: Percentage time improvement.

Table 5.6: TS Comparison With CMOL CAD, GA, MA and LRMA - ($a = 12$).

| Circ. | CAD | | GA [49] | | | MA [50] | | | LRMA [51] | | | TS | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | D | T | D | T | B | D | T | B | D | T | B | D | T | B | T% |
| s27 | 9 | 1 | 7 | 0.01 | 0 | 7 | 0.01 | 0 | 7 | 0.01 | 0 | 7 | 0.01 | 0 | 0.00 |
| s208 | 18 | 3 | 16 | 1.12 | 0 | 16 | 0.12 | 0 | 16 | 0.10 | 0 | 16 | 0.01 | 0 | 90.00 |
| s298 | 13 | 7 | 11 | 0.17 | 0 | 11 | 0.11 | 0 | 11 | 0.09 | 0 | 11 | 0.01 | 0 | 88.89 |
| s344 | 20 | 8 | 18 | 0.57 | 0 | 1 | 0.29 | 0 | 18 | 0.16 | 0 | 18 | 0.01 | 0 | 93.75 |
| s349 | 20 | 7 | 18 | 0.49 | 0 | 18 | 0.28 | 0 | 18 | 0.18 | 0 | 18 | 0.01 | 0 | 94.44 |
| s382 | 13 | 7 | 11 | 1.60 | 0 | 11 | 0.38 | 0 | 11 | 0.32 | 0 | 11 | 0.03 | 0 | 90.63 |
| s386 | 16 | 11 | 10 | 1.05 | 0 | 10 | 0.33 | 0 | 10 | 0.34 | 0 | 10 | 0.03 | 0 | 91.18 |
| s400 | 15 | 8 | 11 | 2.12 | 1 | 11 | 0.40 | 0 | 11 | 0.34 | 0 | 11 | 0.02 | 0 | 94.12 |
| s420 | 20 | 8 | 16 | 8.50 | 1 | 16 | 3.41 | 0 | 16 | 1.57 | 0 | 16 | 0.07 | 0 | 95.54 |
| s444 | 17 | 9 | 11 | 1.86 | 2 | 11 | 0.40 | 0 | 11 | 0.34 | 0 | 11 | 0.03 | 0 | 91.18 |
| s510 | - | - | 18 | 16.56 | 2 | 18 | 7.56 | 0 | 18 | 3.42 | 0 | 18 | 0.18 | 0 | 94.74 |
| s526 | 16 | 13 | 11 | 9.75 | 5 | 11 | 4.36 | 0 | 11 | 1.59 | 0 | 11 | 0.48 | 0 | 96.81 |
| s641 | 25 | 8 | 23 | 82.66 | 15 | 19 | 39.40 | 4 | 16 | 22.02 | 0 | 16 | 6.27 | 0 | 71.53 |
| s713 | - | - | 24 | 52.84 | 34 | 19 | 30.11 | 3 | 19 | 41.77 | 2 | 19 | 8.69 | 0 | 79.20 |
| s820 | - | - | 15 | 77.52 | 41 | 12 | 61.71 | 10 | 12 | 54.09 | 6 | 12 | 11.77 | 0 | 78.24 |
| s832 | - | - | 16 | 69.27 | 54 | 12 | 60.17 | 11 | 12 | 63.77 | 4 | 12 | 10.55 | 0 | 83.46 |
| s838 | - | - | 28 | 201.37 | 50 | 24 | 85.62 | 7 | 24 | 100.40 | 4 | 24 | 4.48 | 0 | 95.54 |
| s1196 | - | - | 30 | 234.88 | 84 | 23 | 208.15 | 19 | 24 | 179.47 | 9 | 23 | 6.87 | 0 | 96.17 |
| s1238 | - | - | 37 | 268.92 | 121 | 28 | 267.34 | 31 | 26 | 353.00 | 9 | 26 | 12.87 | 0 | 96.35 |
| Avg. | - | - | 17 | 54.28 | 22 | 15 | 40.53 | 4 | 15 | 43.31 | 2 | 15 | 3.28 | 0 | 92.42 |

D: Delay (i.e., Logic Levels).
T: Computation Time in Seconds.
B: Buffers Inserted.
T%: Percentage time improvement.

Compared with TS, SimE has better results in all benchmarks except for two circuits (s641 and s713), it requires more computation time, still it produces results in significantly lesser time compared to MA, and LRMA.

Placement solutions reported for $a = 9$ have more violating connections compared to those reported for $a = 12$; that is justified as the placement problem become harder when the connectivity domain size get decreased. Some of the results reported in Table 5.7 have not been successfully routed, either because there is no enough blank cells or simply because some gates are not reachable given that the connectivity radius $a$ is small. Based on this observation we conclude that the value of the connectivity domain radius is very important to successful place circuits in CMOL. In our test cases, $a = 12$ seemed appropriate.

Table 5.7: SimE and TS Comparison With CMOL CAD, MA and LRMA - ($a = 9$).

| Circuit | CAD | MA [50] | | LRMA [51] | | SimE | | | TS | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | T | T | C | T | C | T | C | T% | T | C | T% |
| s27 | 0.07 | 0.01 | 0 | 0.01 | 0 | 0.01 | 0 | 0.00 | 0.01 | 0 | 0.00 |
| s208 | 509.84 | 0.22 | 0 | 0.20 | 0 | 0.01 | 0 | 95.00 | 0.01 | 0 | 95.00 |
| s298 | 370.3 | 0.27 | 0 | 0.37 | 0 | 0.02 | 0 | 94.59 | 0.05 | 0 | 86.94 |
| s344 | 6.18 | 0.85 | 0 | 0.65 | 0 | 0.02 | 0 | 96.92 | 0.04 | 0 | 93.85 |
| s349 | 7.6 | 0.57 | 0 | 0.72 | 0 | 0.03 | 0 | 95.83 | 0.04 | 0 | 94.44 |
| s382 | 12.88 | 5.70 | 0 | 1.43 | 0 | 0.38 | 0 | 73.43 | 0.67 | 0 | 53.15 |
| s386 | 10.3 | 1.89 | 0 | 1.62 | 0 | 0.09 | 0 | 94.44 | 0.20 | 0 | 87.65 |
| s400 | 7.52 | 4.48 | 0 | 1.82 | 0 | 0.29 | 0 | 84.07 | 0.61 | 0 | 66.48 |
| s420 | - | 13.83 | 0 | 7.73 | 0 | 0.36 | 0 | 95.34 | 1.24 | 0 | 83.96 |
| s444 | 7.59 | 5.74 | 0 | 2.05 | 0 | 0.38 | 0 | 81.46 | 0.97 | 0 | 52.68 |
| s510 | 213.27 | 22.71 | 7 | 25.49 | 5 | 23.09 | 1 | 9.42 | 64.57 | 1 | - |
| s526 | - | 21.72 | 5 | 23.13 | 2 | 4.30 | 0 | 81.41 | 39.44 | 0 | - |
| s641 | - | 48.26 | 11 | 106.64 | 6 | 38.57 | 5 | 63.83 | 51.73 | 1 | 51.49 |
| s713 | - | 79.63 | 12 | 97.38 | 3 | 44.13 | 5 | 54.68 | 51.88 | 2 | 46.72 |
| s820 | - | 202.60 | 42 | 153.20 | 31 | 127.62 | 18 | 16.70 | 75.91 | 32 | 50.45 |
| s832 | - | 118.83 | 45 | 164.06 | 39 | 152.75 | 26 | 6.89 | 77.75 | 37 | 52.61 |
| s838 | - | 22.60 | 15 | 189.12 | 10 | 73.90 | 2 | 60.92 | 63.13 | 1 | 66.62 |
| s1196 | - | 502.22 | 49 | 565.41 | 36 | 158.09 | 8 | 72.04 | 72.35 | 35 | 87.20 |
| s1238 | - | 404.11 | 55 | 856.69 | 39 | 210.90 | 13 | 75.38 | 73.00 | 54 | 91.48 |
| Avg. | - | 76.64 | 13 | 115.67 | 9 | 43.94 | 4 | 62.01 | 30.19 | 9 | 73.90 |

T: Computation Time in Seconds.

C: Connectivity Violating Connection.

T%: Percentage time improvement.

## 5.3 Reconfiguration

The defect tolerance of the circuits directly after placement phase is poor, the damage in any of the nanodvices used to connect the circuits elements leads to circuits failure. Reconfiguration phase rearrange the circuits cells to ensure that they implement the desired functionality. Simulated Evolution and Tabu Search are used to explore the search space looking for an arrangement of cells that avoid defects.

### 5.3.1 Simulated Evolution

Simulated Evolution is employed according to the formulation given in Chapter 3. A particular gate goodness is evaluated following the number of defective nanodevices it's associated with (Equation 4.2). Figure 5.6 shows the number of elements selected for perturbation in each iteration, the size of the selection set decresses with time, until a final solution with the least cost is reached. In addition, Figure 5.7 shows the change of the problem cost (Equation 3.6) per iteration, it shows how the heuristic is evolving to better solutions without being too greed.

### 5.3.2 Tabu Search

In a similar manner to the verification given for Tabu Search in Section 5.2.2; Figure 5.8 shows the quality of the solutions produced by TS over iterations. The heuristic steadily converges to near optimal solution.

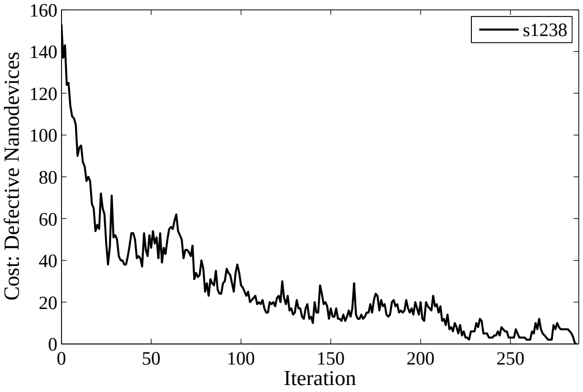


Figure 5.6: SimE reconfiguration heuristic: selection set size Vs. iterations - s1238.blif.

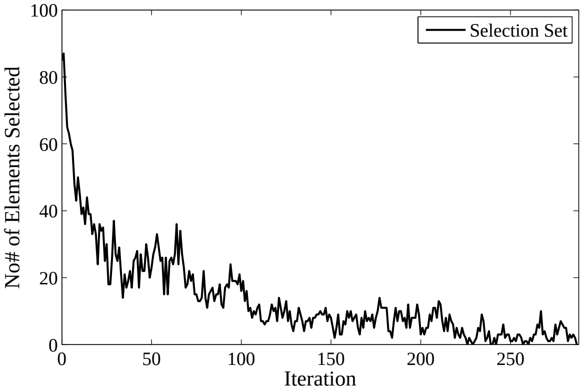


Figure 5.7: Change of reconfiguration cost per iteration in Simulated Evolution - s1238.blif.

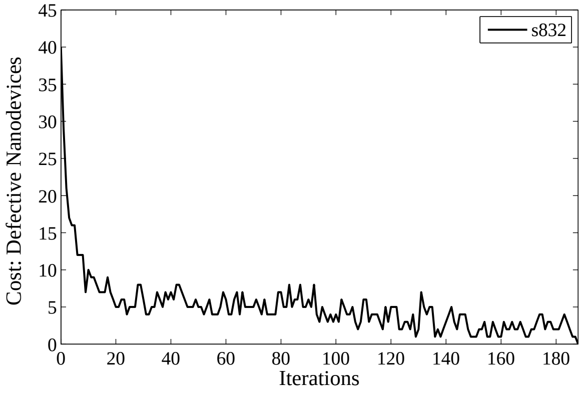


Figure 5.8: Change of reconfiguration cost per iteration in Tabu Search - s832.blif.

Figure 5.9 shows the change in the problem cost in respect to candidate list size and given different defect rates. Each rate constitutes a distinct instance of the reconfiguration problem. In high defect rates, a small candidate list results in bad solutions, whereas for low rates, a small list is sufficient. The problem becomes more constrained when many nanodevices are defective, thus, TS requires more choices to effectivly explore the search space. Throughout our implementation we have used different sizes for different defect rates; a maximum value of 60 is used as an upper bound limit on the list size. Further, we have investigated the impact of tabu list size on solutions quality; it was concluded that a size less than 5 is suitable for our problem; therefore we have chosen a fixed value equals to 2 for all of the circuits.

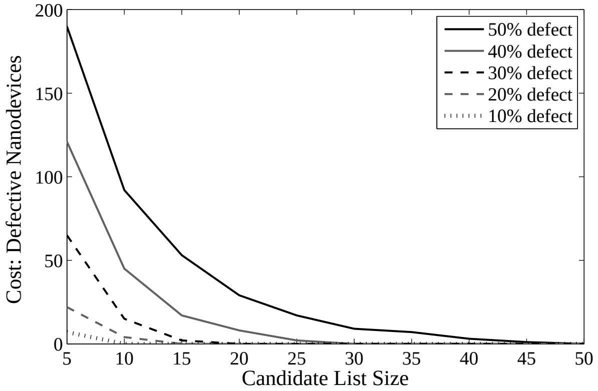


Figure 5.9: Cost yielded by TS for $q_{nano}$ between 10% and 50% vs. candidate list size - s1196.blif.

### 5.3.3 Results

This section presents the final results for CMOL reconfiguration problem using Simulated Evolution and Tabu Search iterative heuristics. In reconfiguration phase we have adhered to the original description of the connectivity domain shown in Figure 2.3, instead of depending on the definition outlined in Equation 3.2(b) using Manhattan distance. Given a particular connectivity radius $a$, the later definition has bigger connectivity domains than the former one. For that reason, we have used a connectivity radius $a = 18$ to compensate the difference between the two definitions. We have used a bias $B$ between $[-0.06, 0.05]$, where small bias values are used for high defect rates. Negative bias was required to reduce the number of selected elements (especially in early iterations) to prevent the heuristic from performing conflicting moves, which results in poor exploration of the search space.

The following tables show results for the given defect scenarios in Section 5.1.2; Tabel 5.8 report the results for random defect map $R1$ and defect scenario (i). *Time* is the computation time in seconds, while *Buffers* is the number of buffers the routing phase has inserted to reroute those nets that still use stuck-at-open defective nanodevices. The results show that for high defect rates SimE required more computation time. Similarly, Table 5.9 and Table 5.10 present the results for defect scenario (i) when clustered defect maps $C1$ and $C2$ are used. The results show that finding a successful reconfiguration is harder and requires more time when defects are clustered. In SimE results, only circuits $s820$ and $s832$ required additional buffers to reroute the connections that could not be reconfigured.

Simulated Evolution results for defect scenario (ii) are shown in Table 5.11 and Table 5.12. The reconfiguration of two circuits (i.e., $s820$ and $s1238$) is performed when up to 70% of the nanowires are cut and 20% of the nanodevices are stuck-at-open. SimE has found successful reconfigurations even when the probability of broken wires is high. Table 5.13 gives the results for defect scenario (iii). Those results are for circuits placement when some of CMOL cells are defective. Unlike earlier placement results, here we have used connectivity domain of Figure 2.3 and connectivity radius $a = 18$. The heuristic found solutions with zero buffers for all circuits under test except for circuit $s820$.

Further we have invistigated the roubutness of our heuristic design by testing the recofiguration of a given placement for benchmark circuit s1238 in 20 different clusterd defect maps. The defect maps had $C = 0.8$, $\sigma = \frac{4a}{3}$, defect rate $q_{nano} =$ 50% and cut rate $q_{wire} = 20\%$. The heuristic was run for 40 times for each map; reconfiguration was successful in 19 out of 20 maps, where the overally successful reconfiguration rate (for 20 maps, each runned 40 times) was 60%. For defect maps with rate $q_{nano} < 50\%$ the heuristic successfully reconfigured all of the 20 defect maps, and the overall successful reconfiguration rate was almost equal to 100%.

Table 5.8: Circuits reconfiguration using SimE and random defect map $R1$, ($q_{wire}$ = 20% - $q_{cell}$ = 0%)

| Circ. | $q_{nano}$ = 10% | | $q_{nano}$ = 20% | | $q_{nano}$ = 30% | | $q_{nano}$ = 40% | | $q_{nano}$ = 50% | |
|---|---|---|---|---|---|---|---|---|---|---|
| | Time | Buffers | Time | Buffers | Time | Buffers | Time | Buffers | Time | Buffers |
| s27 | 0.03 | 0 | 0.03 | 0 | 0.03 | 0 | 0.03 | 0 | 0.03 | 0 |
| s208 | 0.03 | 0 | 0.03 | 0 | 0.03 | 0 | 0.03 | 0 | 0.03 | 0 |
| s298 | 0.06 | 0 | 0.06 | 0 | 0.03 | 0 | 0.06 | 0 | 0.06 | 0 |
| s344 | 0.03 | 0 | 0.03 | 0 | 0.03 | 0 | 0.03 | 0 | 0.06 | 0 |
| s349 | 0.03 | 0 | 0.03 | 0 | 0.03 | 0 | 0.06 | 0 | 0.10 | 0 |
| s382 | 0.03 | 0 | 0.03 | 0 | 0.06 | 0 | 0.10 | 0 | 1.22 | 0 |
| s386 | 0.03 | 0 | 0.03 | 0 | 0.10 | 0 | 0.26 | 0 | 3.78 | 0 |
| s400 | 0.03 | 0 | 0.03 | 0 | 0.06 | 0 | 0.13 | 0 | 0.64 | 0 |
| s420 | 0.03 | 0 | 0.03 | 0 | 0.06 | 0 | 0.16 | 0 | 0.32 | 0 |
| s444 | 0.03 | 0 | 0.03 | 0 | 0.10 | 0 | 0.16 | 0 | 0.77 | 0 |
| s510 | 0.03 | 0 | 0.10 | 0 | 0.22 | 0 | 1.02 | 0 | 1.09 | 0 |
| s526 | 0.03 | 0 | 0.10 | 0 | 0.26 | 0 | 1.06 | 0 | 2.21 | 0 |
| s641 | 0.06 | 0 | 0.10 | 0 | 0.13 | 0 | 0.29 | 0 | 0.61 | 0 |
| s713 | 0.06 | 0 | 0.10 | 0 | 0.16 | 0 | 0.38 | 0 | 0.64 | 0 |
| s820 | 0.26 | 0 | 0.61 | 0 | 1.66 | 0 | 4.26 | 0 | 8.06 | 3 |
| s832 | 0.32 | 0 | 0.96 | 0 | 2.46 | 0 | 6.50 | 0 | 9.18 | 3 |
| s838 | 0.16 | 0 | 0.22 | 0 | 0.32 | 0 | 0.90 | 0 | 1.31 | 0 |
| s1196 | 0.26 | 0 | 0.54 | 0 | 1.09 | 0 | 0.99 | 0 | 3.04 | 0 |
| s1238 | 0.51 | 0 | 0.70 | 0 | 0.96 | 0 | 0.99 | 0 | 4.99 | 0 |
| Avg. | 0.11 | 0.00 | 0.20 | 0.00 | 0.41 | 0.00 | 0.92 | 0.00 | 2.01 | 0 |

Time: Computation Time in Seconds.
Buffers: Buffers Inserted.

Table 5.9: Circuits reconfiguration using SimE and clustered defect map $C1$ - $\sigma = \frac{2a}{3}$, ($q_{wire} = 20\%$ - $q_{cell} = 0\%$)

| Circ. | $q_{nano} = 10\%$ | | $q_{nano} = 20\%$ | | $q_{nano} = 30\%$ | | $q_{nano} = 40\%$ | | $q_{nano} = 50\%$ | |
|---|---|---|---|---|---|---|---|---|---|---|
| | Time | Buffers | Time | Buffers | Time | Buffers | Time | Buffers | Time | Buffers |
| s27 | 0.03 | 0 | 0.03 | 0 | 0.03 | 0 | 0.03 | 0 | 0.03 | 0 |
| s208 | 0.03 | 0 | 0.03 | 0 | 0.03 | 0 | 0.03 | 0 | 0.03 | 0 |
| s298 | 0.03 | 0 | 0.03 | 0 | 0.03 | 0 | 0.06 | 0 | 0.19 | 0 |
| s344 | 0.03 | 0 | 0.03 | 0 | 0.03 | 0 | 0.06 | 0 | 0.10 | 0 |
| s349 | 0.03 | 0 | 0.03 | 0 | 0.06 | 0 | 0.10 | 0 | 0.10 | 0 |
| s382 | 0.03 | 0 | 0.03 | 0 | 0.06 | 0 | 0.26 | 0 | 0.32 | 0 |
| s386 | 0.03 | 0 | 0.03 | 0 | 0.19 | 0 | 0.80 | 0 | 3.10 | 0 |
| s400 | 0.03 | 0 | 0.03 | 0 | 0.13 | 0 | 0.32 | 0 | 0.64 | 0 |
| s420 | 0.06 | 0 | 0.06 | 0 | 0.10 | 0 | 0.16 | 0 | 0.26 | 0 |
| s444 | 0.03 | 0 | 0.03 | 0 | 0.10 | 0 | 0.32 | 0 | 0.35 | 0 |
| s510 | 0.06 | 0 | 0.13 | 0 | 0.26 | 0 | 0.90 | 0 | 2.34 | 0 |
| s526 | 0.06 | 0 | 0.06 | 0 | 0.35 | 0 | 0.90 | 0 | 1.06 | 0 |
| s641 | 0.06 | 0 | 0.13 | 0 | 0.16 | 0 | 0.32 | 0 | 1.22 | 0 |
| s713 | 0.06 | 0 | 0.13 | 0 | 0.19 | 0 | 0.38 | 0 | 1.79 | 0 |
| s820 | 0.32 | 0 | 0.61 | 0 | 2.46 | 0 | 5.44 | 0 | 12.13 | 4 |
| s832 | 0.64 | 0 | 1.02 | 0 | 2.72 | 0 | 6.50 | 0 | 18.27 | 6 |
| s838 | 0.16 | 0 | 0.26 | 0 | 0.42 | 0 | 1.02 | 0 | 1.79 | 0 |
| s1196 | 0.51 | 0 | 0.67 | 0 | 0.74 | 0 | 2.62 | 0 | 6.24 | 0 |
| s1238 | 0.42 | 0 | 1.38 | 0 | 1.89 | 0 | 3.68 | 0 | 9.12 | 0 |
| Avg. | 0.14 | 0 | 0.25 | 0 | 0.52 | 0 | 1.26 | 0 | 3.11 | 1 |

Time: Computation Time in Seconds.
Buffers: Buffers Inserted.

Table 5.10: Circuits reconfiguration using SimE and clustered defect map $C2$ - $\sigma = \frac{4a}{3}$, ($q_{wire} = 20\%$ - $q_{cell} = 0\%$)

| Circ. | $q_{nano} = 10\%$ | | $q_{nano} = 20\%$ | | $q_{nano} = 30\%$ | | $q_{nano} = 40\%$ | | $q_{nano} = 50\%$ | |
|---|---|---|---|---|---|---|---|---|---|---|
| | Time | Buffers | Time | Buffers | Time | Buffers | Time | Buffers | Time | Buffers |
| s27 | 0.03 | 0 | 0.03 | 0 | 0.03 | 0 | 0.03 | 0 | 0.03 | 0 |
| s208 | 0.03 | 0 | 0.03 | 0 | 0.03 | 0 | 0.06 | 0 | 0.03 | 0 |
| s298 | 0.03 | 0 | 0.03 | 0 | 0.03 | 0 | 0.03 | 0 | 0.29 | 0 |
| s344 | 0.03 | 0 | 0.03 | 0 | 0.03 | 0 | 0.06 | 0 | 0.10 | 0 |
| s349 | 0.03 | 0 | 0.03 | 0 | 0.03 | 0 | 0.10 | 0 | 0.13 | 0 |
| s382 | 0.03 | 0 | 0.06 | 0 | 0.06 | 0 | 0.19 | 0 | 1.28 | 0 |
| s386 | 0.03 | 0 | 0.03 | 0 | 0.13 | 0 | 0.48 | 0 | 8.54 | 0 |
| s400 | 0.03 | 0 | 0.03 | 0 | 0.10 | 0 | 0.29 | 0 | 0.48 | 0 |
| s420 | 0.06 | 0 | 0.10 | 0 | 0.10 | 0 | 0.19 | 0 | 0.22 | 0 |
| s444 | 0.03 | 0 | 0.03 | 0 | 0.10 | 0 | 0.26 | 0 | 0.74 | 0 |
| s510 | 0.06 | 0 | 0.22 | 0 | 0.35 | 0 | 1.63 | 0 | 2.21 | 0 |
| s526 | 2.02 | 0 | 0.29 | 0 | 0.58 | 0 | 2.18 | 0 | 6.34 | 0 |
| s641 | 0.10 | 0 | 0.10 | 0 | 0.19 | 0 | 0.32 | 0 | 1.06 | 0 |
| s713 | 0.10 | 0 | 0.13 | 0 | 0.26 | 0 | 0.32 | 0 | 0.61 | 0 |
| s820 | 0.51 | 0 | 0.64 | 0 | 1.60 | 0 | 4.19 | 0 | 11.78 | 4 |
| s832 | 0.48 | 0 | 0.99 | 0 | 2.82 | 0 | 7.07 | 0 | 15.84 | 6 |
| s838 | 0.19 | 0 | 0.26 | 0 | 0.48 | 0 | 1.22 | 0 | 1.82 | 0 |
| s1196 | 0.35 | 0 | 0.51 | 0 | 1.02 | 0 | 2.75 | 0 | 7.01 | 0 |
| s1238 | 0.51 | 0 | 1.15 | 0 | 1.57 | 0 | 7.62 | 0 | 5.54 | 0 |
| Avg. | 0.25 | 0 | 0.25 | 0 | 0.50 | 0 | 1.53 | 0 | 3.37 | 1 |

Time: Computation Time in Seconds.
Buffers: Buffers Inserted.

Table 5.11: Circuit s820 reconfiguration around cut wires using SimE, ($q_{nano} = 20\%$ - $q_{cell} = 0\%$).

| $q_{wire}$ | $R1$ | | $C1$ - $\sigma = \frac{2a}{3}$ | | $C2$ - $\sigma = \frac{4a}{3}$ | |
|---|---|---|---|---|---|---|
| | Time | Buffers | Time | Buffers | Time | Buffers |
| 10% | 0.29 | 0 | 0.29 | 0 | 0.45 | 0 |
| 20% | 0.42 | 0 | 0.48 | 0 | 0.77 | 0 |
| 30% | 1.15 | 0 | 1.12 | 0 | 1.98 | 0 |
| 40% | 1.34 | 0 | 2.59 | 0 | 4.13 | 0 |
| 50% | 2.56 | 0 | 10.11 | 0 | 9.79 | 0 |
| 60% | 7.20 | 0 | 13.47 | 0 | 18.30 | 0 |
| 70% | 13.60 | 0 | 26.14 | 0 | 22.85 | 0 |
| Avg. | 3.79 | 0 | 7.74 | 0 | 8.32 | 0 |

Time: Computation Time in Seconds.
Buffers: Buffers Inserted.

Table 5.12: Circuit s1238 reconfiguration around cut wires using SimE, ($q_{nano} = 20\%$ - $q_{cell} = 0\%$).

| $q_{wire}$ | $R1$ | | $C1$ - $\sigma = \frac{2a}{3}$ | | $C2$ - $\sigma = \frac{4a}{3}$ | |
|---|---|---|---|---|---|---|
| | Time | Buffers | Time | Buffers | Time | Buffers |
| 10% | 0.38 | 0 | 0.45 | 0 | 0.61 | 0 |
| 20% | 0.58 | 0 | 0.54 | 0 | 1.22 | 0 |
| 30% | 0.83 | 0 | 0.86 | 0 | 0.90 | 0 |
| 40% | 0.86 | 0 | 1.73 | 0 | 1.76 | 0 |
| 50% | 2.21 | 0 | 1.95 | 0 | 1.70 | 0 |
| 60% | 3.46 | 0 | 3.04 | 0 | 4.99 | 0 |
| 70% | 4.77 | 0 | 5.54 | 0 | 6.14 | 0 |
| Avg. | 1.87 | 0 | 2.02 | 0 | 2.47 | 0 |

Time: Computation Time in Seconds.
Buffers: Buffers Inserted.

Table 5.13: Implementation of SimE for defect scenario (iii), ($q_{nano} = 20\%$ - $q_{wire} = 20\%$).

| Circ. | $q_{cell} = 10\%$ | | $q_{cell} = 20\%$ | |
|---|---|---|---|---|
| | Time | Buffers | Time | Buffers |
| s641 | 17.06 | 0 | 18.50 | 0 |
| s713 | 23.17 | 0 | 28.22 | 0 |
| s820 | 136.96 | 2 | 240.13 | 3 |
| s838 | 60.48 | 0 | 95.04 | 0 |
| s1196 | 295.68 | 0 | 320.90 | 0 |
| s1238 | 390.88 | 0 | 415.30 | 0 |
| Avg. | 154.04 | 0 | 186.35 | 1 |

Time: Computation Time in Seconds.
Buffers: Buffers Inserted.

Table 5.14 shows Tabu Search reconfiguration results for random defect map $R1$. Defect rates below 30% are easily tolerated, whereas, for 40% and 50% defects, additional buffers are needed to reroute defective connections. When more defects are involved TS required more computation time to find the best solution. Table 5.15 and Table 5.16 present the results when defects are clustered around multiple points (i.e., defect maps $C1$ and $C2$). Most of the cells are reconfigured around defects and some of them required additional buffers. The heuristic has failed to find solutions that do not use any defective nanodevices or tolerate defective connections by using additional buffers for circuits $s820$ and $s832$. Those two circuits contain many 5-input NOR gates, which make them harder to reconfigure.

Tabu search has been also employed to reconfigure circuits when nanowires are broken. Table 5.17 shows the reconfiguration cost for circuit $s820$ when wires cut rate $q_{wire}$ is between 10% and 70% and defective nanodevices rate $q_{nano}$ equals 20%. For high cut rates, the heuristic has failed to reconfigure the circuits. Results reported when defective nanodevices are randomly distributed are better than those when nanodevices defects are clustered. Clustered defects are harder to tolerate as some cells may have many of thier connectivity domain cells unreachable. In addition, Table 5.18 presents the reconfiguration results for circuit $s1238$ for different cut rates. All reconfigurations are successful and zero additional buffers are needed.

Comparing the results of Simulated Evolution and Tabu Search indicate that SimE is able to produce better results and in less computation time for both random and clustered defects. That is because SimE is applying an evolutionary technique to choose which elements to be reconfigured. Although, TS is an efficient search heuristic but it fails in circuits with high defect rates or in circuits that have many connections.

Table 5.14: Circuits reconfiguration using TS and random defect map $R1$, ($q_{wire} = 20\%$ - $q_{cell} = 0\%$)

| Circ. | $q_{nano} = 10\%$ | | $q_{nano} = 20\%$ | | $q_{nano} = 30\%$ | | $q_{nano} = 40\%$ | | $q_{nano} = 50\%$ | |
|---|---|---|---|---|---|---|---|---|---|---|
| | Time | Buffers | Time | Buffers | Time | Buffers | Time | Buffers | Time | Buffers |
| s27 | 0.03 | 0 | 0.03 | 0 | 0.03 | 0 | 0.03 | 0 | 0.03 | 0 |
| s208 | 0.03 | 0 | 0.03 | 0 | 0.10 | 0 | 0.10 | 0 | 0.22 | 0 |
| s298 | 0.03 | 0 | 0.06 | 0 | 0.10 | 0 | 0.22 | 0 | 1.28 | 0 |
| s344 | 0.06 | 0 | 0.06 | 0 | 0.10 | 0 | 0.19 | 0 | 1.92 | 0 |
| s349 | 0.06 | 0 | 0.06 | 0 | 0.10 | 0 | 0.38 | 0 | 6.43 | 0 |
| s382 | 0.10 | 0 | 0.10 | 0 | 0.26 | 0 | 0.61 | 0 | 5.60 | 0 |
| s386 | 0.35 | 0 | 0.48 | 0 | 2.72 | 0 | 2.43 | 0 | 69.95 | 1 |
| s400 | 0.16 | 0 | 0.22 | 0 | 0.70 | 0 | 1.50 | 0 | 15.36 | 0 |
| s420 | 0.06 | 0 | 0.13 | 0 | 0.19 | 0 | 0.64 | 0 | 1.50 | 0 |
| s444 | 0.22 | 0 | 0.26 | 0 | 0.96 | 0 | 3.68 | 0 | 9.50 | 0 |
| s510 | 0.19 | 0 | 0.26 | 0 | 1.50 | 0 | 12.64 | 0 | 85.98 | 1 |
| s526 | 0.19 | 0 | 0.32 | 0 | 0.83 | 0 | 4.19 | 0 | 72.67 | 1 |
| s641 | 0.32 | 0 | 0.38 | 0 | 0.86 | 0 | 2.21 | 0 | 37.38 | 1 |
| s713 | 0.32 | 0 | 0.45 | 0 | 0.83 | 0 | 2.05 | 0 | 41.47 | 1 |
| s820 | 0.77 | 0 | 3.01 | 0 | 15.26 | 0 | 97.98 | 2 | 160.16 | 8 |
| s832 | 1.06 | 0 | 3.81 | 0 | 17.76 | 0 | 103.14 | 2 | 171.87 | 7 |
| s838 | 0.64 | 0 | 0.96 | 0 | 1.54 | 0 | 3.04 | 0 | 11.10 | 0 |
| s1196 | 1.15 | 0 | 2.02 | 0 | 4.74 | 0 | 57.38 | 0 | 99.26 | 0 |
| s1238 | 1.70 | 0 | 3.17 | 0 | 8.19 | 0 | 33.22 | 0 | 153.54 | 0 |
| Avg. | 0.39 | 0 | 0.83 | 0 | 2.99 | 0 | 17.14 | 0 | 49.75 | 1 |

Time: Computation Time in Seconds.
Buffers: Buffers Inserted.

Table 5.15: Circuits reconfiguration using TS and clustered defect map $C1$ - $\sigma = \frac{2a}{3}$, ($q_{wire} = 20\%$ - $q_{cell} = 0\%$)

| Circ. | $q_{nano} = 10\%$ | | $q_{nano} = 20\%$ | | $q_{nano} = 30\%$ | | $q_{nano} = 40\%$ | | $q_{nano} = 50\%$ | |
|---|---|---|---|---|---|---|---|---|---|---|
| | Time | Buffers | Time | Buffers | Time | Buffers | Time | Buffers | Time | Buffers |
| s27 | 0.03 | 0 | 0.03 | 0 | 0.03 | 0 | 0.03 | 0 | 0.03 | 0 |
| s208 | 0.03 | 0 | 0.06 | 0 | 0.32 | 0 | 1.60 | 0 | 9.82 | 0 |
| s298 | 0.03 | 0 | 0.03 | 0 | 0.10 | 0 | 0.26 | 0 | 4.32 | 0 |
| s344 | 0.06 | 0 | 0.06 | 0 | 0.16 | 0 | 0.93 | 0 | 3.94 | 0 |
| s349 | 0.06 | 0 | 0.10 | 0 | 0.19 | 0 | 3.30 | 0 | 29.95 | 0 |
| s382 | 0.10 | 0 | 0.10 | 0 | 0.16 | 0 | 1.60 | 0 | 5.41 | 0 |
| s386 | 0.19 | 0 | 0.35 | 0 | 3.01 | 0 | 21.06 | 0 | 69.38 | 1 |
| s400 | 0.16 | 0 | 0.19 | 0 | 0.51 | 0 | 5.09 | 0 | 15.84 | 0 |
| s420 | 0.13 | 0 | 0.13 | 0 | 0.22 | 0 | 0.64 | 0 | 2.46 | 0 |
| s444 | 0.16 | 0 | 0.32 | 0 | 0.70 | 0 | 12.99 | 0 | 7.30 | 0 |
| s510 | 0.42 | 0 | 0.42 | 0 | 1.41 | 0 | 11.90 | 0 | 80.48 | 1 |
| s526 | 0.16 | 0 | 0.26 | 0 | 0.70 | 0 | 4.70 | 0 | 61.22 | 1 |
| s641 | 0.32 | 0 | 0.64 | 0 | 0.83 | 0 | 2.66 | 0 | 37.15 | 1 |
| s713 | 0.35 | 0 | 0.64 | 0 | 1.06 | 0 | 2.30 | 0 | 45.22 | 2 |
| s820 | 0.93 | 0 | 1.92 | 0 | 25.60 | 0 | 90.05 | 1 | - | - |
| s832 | 1.02 | 0 | 2.59 | 0 | 44.96 | 0 | 98.53 | 2 | - | - |
| s838 | 0.67 | 0 | 1.34 | 0 | 2.75 | 0 | 4.38 | 0 | 23.14 | 0 |
| s1196 | 1.25 | 0 | 2.72 | 0 | 7.26 | 0 | 48.26 | 0 | 136.93 | 0 |
| s1238 | 1.28 | 0 | 3.20 | 0 | 10.14 | 0 | 49.50 | 0 | 184.06 | 3 |
| Avg. | 0.39 | 0 | 0.79 | 0 | 5.27 | 0 | 18.94 | 0 | 42.16 | 1 |

Time: Computation Time in Seconds.
Buffers: Buffers Inserted.

Table 5.16: Circuits reconfiguration using TS and clustered defect map $C2$ - $\sigma = \frac{4a}{3}$, ($q_{wire} = 20\%$ - $q_{cell} = 0\%$)

| Circ. | $q_{nano} = 10\%$ | | $q_{nano} = 20\%$ | | $q_{nano} = 30\%$ | | $q_{nano} = 40\%$ | | $q_{nano} = 50\%$ | |
|---|---|---|---|---|---|---|---|---|---|---|
| | Time | Buffers | Time | Buffers | Time | Buffers | Time | Buffers | Time | Buffers |
| s27 | 0.03 | 0 | 0.03 | 0 | 0.03 | 0 | 0.03 | 0 | 0.03 | 0 |
| s208 | 0.06 | 0 | 0.06 | 0 | 0.16 | 0 | 0.96 | 0 | 1.09 | 0 |
| s298 | 0.13 | 0 | 0.06 | 0 | 0.10 | 0 | 0.19 | 0 | 5.25 | 0 |
| s344 | 0.06 | 0 | 0.06 | 0 | 0.13 | 0 | 0.42 | 0 | 4.51 | 0 |
| s349 | 0.13 | 0 | 0.16 | 0 | 0.16 | 0 | 0.93 | 0 | 26.75 | 0 |
| s382 | 0.06 | 0 | 0.10 | 0 | 0.22 | 0 | 1.28 | 0 | 11.94 | 0 |
| s386 | 0.32 | 0 | 0.61 | 0 | 2.88 | 0 | 22.21 | 0 | 65.76 | 2 |
| s400 | 0.19 | 0 | 0.19 | 0 | 0.67 | 0 | 4.80 | 0 | 25.63 | 0 |
| s420 | 0.10 | 0 | 0.22 | 0 | 0.26 | 0 | 0.90 | 0 | 2.46 | 0 |
| s444 | 0.26 | 0 | 0.26 | 0 | 0.96 | 0 | 7.52 | 0 | 16.54 | 0 |
| s510 | 0.29 | 0 | 1.22 | 0 | 2.05 | 0 | 18.62 | 0 | 70.40 | 1 |
| s526 | 0.77 | 0 | 0.90 | 0 | 1.63 | 0 | 8.64 | 0 | 72.70 | 1 |
| s641 | 0.42 | 0 | 0.74 | 0 | 1.63 | 0 | 4.13 | 0 | 46.18 | 2 |
| s713 | 0.45 | 0 | 0.77 | 0 | 1.28 | 0 | 2.88 | 0 | 5.12 | 3 |
| s820 | 1.92 | 0 | 3.94 | 0 | 44.19 | 0 | 88.19 | 1 | - | - |
| s832 | 1.31 | 0 | 4.54 | 0 | 25.15 | 0 | 92.42 | 1 | - | - |
| s838 | 0.70 | 0 | 1.22 | 0 | 2.88 | 0 | 5.50 | 0 | 63.84 | 0 |
| s1196 | 1.38 | 0 | 2.62 | 0 | 11.04 | 0 | 52.29 | 0 | 133.25 | 0 |
| s1238 | 2.08 | 0 | 2.91 | 0 | 17.95 | 0 | 98.24 | 0 | 156.03 | 0 |
| Avg. | 0.56 | 0 | 1.08 | 0 | 5.97 | 0 | 21.59 | 0 | 41.62 | 1 |

Time: Computation Time in Seconds.
Buffers: Buffers Inserted.

Table 5.17: Circuit s820 reconfiguration around cut wires using TS, ($q_{nano} = 20\%$ - $q_{cell} = 0\%$).

| $q_{wire}$ | $R1$ | | $C1$ - $\sigma = \frac{2a}{3}$ | | $C2$ - $\sigma = \frac{4a}{3}$ | |
|---|---|---|---|---|---|---|
| | Time | Buffers | Time | Buffers | Time | Buffers |
| 10% | 1.38 | 0 | 1.25 | 0 | 3.65 | 0 |
| 20% | 4.29 | 0 | 2.88 | 0 | 5.79 | 0 |
| 30% | 12.80 | 0 | 9.22 | 0 | 71.81 | 1 |
| 40% | 25.28 | 0 | 92.48 | 1 | 99.81 | 2 |
| 50% | 119.97 | 1 | 134.50 | 2 | 111.55 | 3 |
| 60% | 150.72 | 2 | 140.67 | 3 | - | - |
| 70% | 190.21 | 2 | - | - | - | - |
| Avg. | 72.09 | 1 | 63.50 | 1 | 58.52 | 1 |

Time: Computation Time in Seconds.
Buffers: Buffers Inserted.

Table 5.18: Circuit s1238 reconfiguration around cut wires using TS, ($q_{nano} = 20\%$ - $q_{cell} = 0\%$).

| $q_{wire}$ | $R1$ | | $C1$ - $\sigma = \frac{2a}{3}$ | | $C2$ - $\sigma = \frac{4a}{3}$ | |
|---|---|---|---|---|---|---|
| | Time | Buffers | Time | Buffers | Time | Buffers |
| 10% | 1.50 | 0 | 2.34 | 0 | 3.52 | 0 |
| 20% | 2.27 | 0 | 3.58 | 0 | 6.05 | 0 |
| 30% | 3.78 | 0 | 5.28 | 0 | 4.93 | 0 |
| 40% | 5.92 | 0 | 9.60 | 0 | 8.38 | 0 |
| 50% | 9.38 | 0 | 14.50 | 0 | 8.74 | 0 |
| 60% | 16.13 | 0 | 26.21 | 0 | 32.96 | 0 |
| 70% | 35.04 | 0 | 47.36 | 0 | 38.18 | 0 |
| Avg. | 10.57 | 0 | 15.55 | 0 | 14.68 | 0 |

Time: Computation Time in Seconds.
Buffers: Buffers Inserted.

## 5.4 Solutions Verification

In this section, we discuss the verification of CMOL circuits produced by aforementioned heuristic implementations. Each mapped circuit uses a number of nanodevices to connect its modules (i.e., gates). The final result of CMOL cell mapping heuristics is the list of nanodevices that should be programmed (i.e., set to "ON" state). Circuits verification follows the steps shown in Figure 5.10.

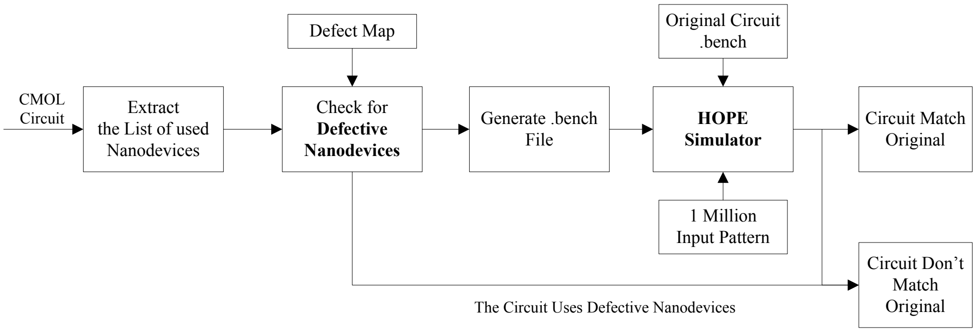

CMOL Circuit
Extract the List of used Nanodevices
Defect Map
Check for Defective Nanodevices
Generate .bench File
Original Circuit .bench
HOPE Simulator
1 Million Input Pattern
Circuit Match Original
Circuit Don't Match Original
The Circuit Uses Defective Nanodevices

Figure 5.10: Verification steps.

The verification procedure starts by checking if any of the nanodevices used to connect CMOL cells is defective, this is done with the help of the same defect map used when the circuit is mapped and reconfigured. Then, the circuit is written in bench formate and forwarded to HOPE simulator [66], along with the original circuit description (i.e., the one before mapping) and a randomly generated input patterns. We use perl script to run the simulator and compare the output of the two circuits to decide if they match. Based on the generated outputs we conclude if the two circuits are equal or not. The verification procedure make sure that our heuristics perturbations do not change the circuits description, and verify that mapped circuits have the same logical functionality as the original ones.

# Chapter 6

# Conclusion & Future Work

## 6.1 Conclusion

In this work, a design automation solution was presented for cell placement and reconfiguration in CMOL nanofabric architecture. CMOL is a novel hybrid architecture that has been proposed by Likharev and Strukov [3]. Two general iterative search heuristics (Simulated Evolution and Tabu Search) were implemented.

The placement of the circuits modules given the confined CMOL connectivity is a hard problem, the problem tend to be more constrained when the fan-in of the NOR gates become bigger. The value of the connectivity radius $a$, which constitute the size of the connectivity domain, is very important for the successful implementation and adaptation of CMOL circuits.

Nanodevices high defect rates is the draw back of the new hybrid architectures;

our implementation has shown that a rate of up to 50% stuck-at-open defects (either clustered or randomly located) can be tolerated. Other types of defects (e.g. nanowires cut, defective CMOS cells) can also be avoided.

The routing of unconnected nets (i.e., those violate connectivity radius or use stuck-at-open devices) is a very important step in CMOL design flow. Insertion of many buffers (i.e., pair of inverters) may worsen the timing delay of the circuit. Successful connections rerouting depends on the available resources (blank cells), connectivity radius $a$, and defect rate. An efficient placement and reconfiguration heuristics reduce the need for invoking routing procedure.

## 6.2 Future Work

For the future work, other search heuristics can be implemented. An investigation on the causes of placement and reconfiguration failures can be done. A new modifications can be incorporated as to increase the probability of having successful circuits mapping in CMOL. Furthermore, the mapping procedure can be extended to cover the different versions of CMOL architecture. As we have only addressed permeant faults, an investigation on transient faults in CMOL could be very timely and needed. The various techniques for testing nanofabrics and extraction/compaction of defect maps can be explored.

# Bibliography

[1] Gregory S. Snider and Stanley R Williams. Nano/CMOS architectures using a field-programmable nanowire interconnect. *Nanotechnology*, 18(3):035204, 2007.

[2] Z. Abid, Ming Liu, and Wei Wang. 3D integration of CMOL structures for FPGA applications. *Computers, IEEE Transactions on*, 60(4):463 –471, april 2011.

[3] Dmitri B Strukov and Konstantin K. Likharev. CMOL FPGA: a reconfigurable architecture for hybrid digital circuits with two-terminal nanodevices. *Nanotechnology*, 16(6):888–900, 2005.

[4] Sadiq M. Sait and Habib Youssef. *Iterative Computer Algorithms with Applications in Engineering: Solving Combinatorial Optimization Problems.* IEEE Computer Society Press, California, December 1999.

[5] Michael Butts and Andre DeHon. Molecular electronics: Devices, systems and tools for Gigagate, Gigabit chips. In *In ICCAD-2002*, pages 433–440, 2002.

[6] A. DeHon. Array-based architecture for FET-based, nanoscale electronics. *Nanotechnology, IEEE Transactions on*, 2(1):23 – 32, mar 2003.

[7] S. Copen Goldstein and M. Budiu. NanoFabrics: spatial computing using molecular electronics. In *Computer Architecture, 2001. Proceedings. 28th Annual International Symposium on*, pages 178 –189, 2001.

[8] D.J. Frank, R.H. Dennard, E. Nowak, P.M. Solomon, Y. Taur, and Hon-Sum Philip Wong. Device scaling limits of Si MOSFETs and their application dependencies. *Proceedings of the IEEE*, 89(3):259 –288, mar 2001.

[9] Konstantin Likharev. Electronics below 10 nm. In *Nano and Giga Challenges in Microelectronics*, pages 27–68, 2003.

[10] International technology roadmap for semiconductors, 2003 Edition.

[11] J. Tour. *Molecular Electronics*. World Scientific, Singapore, 2003.

[12] S. Folling, O. Turel, and K. Likharev. Single-electron latching switches as nanoscale synapses. In *Neural Networks, 2001. Proceedings. IJCNN '01. International Joint Conference on*, volume 1, pages 216 –221 vol.1, 2001.

[13] Konstantin K Likharev. Hybrid CMOS/nanoelectronic circuits: Opportunities and challenges. *Journal of Nanoelectronics and Optoelectronics*, 3(3):203–230, 2008.

[14] Andr DeHon. Design of programmable interconnect for sublithographic programmable logic arrays. *Proceedings of the 2005 ACMSIGDA 13th international symposium on Fieldprogrammable gate arrays FPGA 05*, C-28(9):127, 2005.

[15] S. Copen Goldstein and M. Budiu. NanoFabrics: spatial computing using molecular electronics. In *Computer Architecture, 2001. Proceedings. 28th Annual International Symposium on*, pages 178 –189, 2001.

[16] Philip J. Kuekes, Duncan R. Stewart, and R. Stanley Williams. The crossbar latch: Logic value storage, restoration, and inversion in crossbar circuits. *Journal of Applied Physics*, 97(3):034301 –034301–5, Feb 2005.

[17] O"zgr Trel, Jung Hoon Lee, Xiaolong Ma, and Konstantin K Likharev. Neuromorphic architectures for nanoelectronic circuits. *International Journal of Circuit Theory and Applications*, 32(5):277–302, 2004.

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

[19] A. DeHon and K.K. Likharev. Hybrid CMOS/nanoelectronic digital circuits: devices, architectures, and design automation. In *Computer-Aided Design, 2005. ICCAD-2005. IEEE/ACM International Conference on*, pages 375 – 382, nov. 2005.

[20] G. S. Snider P. J. Kuekes and R. S. Williams. Crossbar nanocomputers. *Scientific American*, 293(5):7276, 2005.

[21] Dmitri B. Strukov and Konstantin K. Likharev. CMOL FPGA circuits. In *In Proc. of Int. Conf. on Computer Design, CDES2006*, pages 213–219, 2006.

[22] M.R. Stan, P.D. Franzon, S.C. Goldstein, J.C. Lach, and M.M. Ziegler. Molecular electronics: from devices and interconnect to circuits and architecture. *Proceedings of the IEEE*, 91(11):1940 – 1957, nov 2003.

[23] Changjian Gao and D. Hammerstrom. Cortical models onto CMOL and CMOS; architectures and performance / price. *Circuits and Systems I: Regular Papers, IEEE Transactions on*, 54(11):2502 –2515, nov. 2007.

[24] Sumio Iijima. Helical microtubules of graphitic carbon. *Nature*, 354(6348):56–58, 1991.

[25] A Bachtold, P Hadley, T Nakanishi, and C Dekker. Logic circuits with carbon nanotube transistors. *Science*, 294(5545):1317–1320, 2001.

[26] Hj Choi, J Ihm, Sg Louie, A Zettl, with Fuhrer Nygard J McEuen, Pl, L Shih, M Forero, Yg Yoon, and Mazzoni. Crossed Nanotube Junctions. *Science*, 288(5465):494–497, 2000.

[27] T Rueckes, K Kim, E Joselevich, Gy Tseng, Cl Cheung, and Cm Lieber. Carbon Nanotube-based nonvolatile random access memory for molecular computing. *Science*, 289(5476):94–97, 2000.

[28] T. I. Kamins, R. Stanley Williams, Y. Chen, Y.-L. Chang, and Y. A. Chang. Chemical vapor deposition of Si nanowires nucleated by TiSi2 islands on Si. *Applied Physics Letters*, 76(5):562–564, 2000.

[29] Y Huang, X Duan, Y Cui, L J Lauhon, K H Kim, and C M Lieber. Logic gates and computation from assembled nanowire building blocks. *Science*, 294(5545):1313–7, 2001.

[30] H Park, J Park, A K L Lim, E H Anderson, A P Alivisatos, and P L McEuen. Nanomechanical oscillations in a single-C60 transistor. *Nature*, 407(6800):57–60, 2000.

[31] Sergey Kubatkin, Andrey Danilov, Mattias Hjort, Jerome Cornil, Jean-Luc Bredas, Nicolai Stuhr-Hansen, Per Hedegard, and Thomas Bjornholm. Single-electron transistor of a single organic molecule with access to several redox states. *Nature*, 425(6959):698–701, 2003.

[32] P. Douglas Tougaw and Craig S. Lent. Logical devices implemented using quantum cellular automata. *Journal of Applied Physics*, 75(3):1818–1825, 1994.

[33] Mehdi B. Tahoori and Subhasish Mitra. Fault detection and diagnosis techniques for molecular computing. In *in NanoTech*, 2004.

[34] J R Heath. A defect-tolerant computer architecture: Opportunities for nanotechnology. *Science*, 280(5370):1716–1721, 1998.

[35] J.G. Brown and R.D. Blanton. CAEN-BIST: testing the nanofabric. In *Test Conference, 2004. Proceedings. ITC 2004. International*, pages 462 – 471, oct. 2004.

[36] M. Tehranipoor and R.M.P. Rad. Built-In Self-Test and recovery procedures for molecular electronics-based nanofabrics. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 26(5):943 –958, may 2007.

[37] M.V. Joshi and W.K. Al-Assadi. A BIST approach for configurable nanofabric arrays. In *Nanotechnology, 2008. NANO '08. 8th IEEE Conference on*, pages 695 –698, aug. 2008.

[38] Gang Wang, Wenrui Gong, and Ryan Kastner. On the use of Bloom filters for defect maps in nanocomputing. In *Proceedings of the 2006 IEEE/ACM international conference on Computer-aided design*, ICCAD '06, pages 743–746, New York, NY, USA, 2006. ACM.

[39] Jing Huang, M.B. Tahoori, and F. Lombardi. 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*, pages 96 – 104, oct. 2004.

[40] Yadunandana Yellambalase and Minsu Choi. Cost-driven repair optimization of reconfigurable nanowire crossbar systems with clustered defects. *J. Syst. Archit.*, 54(8):729–741, August 2008.

[41] M. B. Tahoori. A mapping algorithm for defect-tolerance of reconfigurable nano-architectures. In *Proceedings of the 2005 IEEE/ACM International conference on Computer-aided design*, ICCAD '05, pages 668–672, Washington, DC, USA, 2005. IEEE Computer Society.

[42] C. Tunc and M.B. Tahoori. On-the-fly variation tolerant mapping in crossbar nano-architectures. In *VLSI Test Symposium (VTS), 2010 28th*, pages 105 –110, april 2010.

[43] Masoud Zamani and Mehdi B. Tahoori. Variation-aware logic mapping for crossbar nano-architectures. In *Proceedings of the 16th Asia and South Pacific Design Automation Conference*, ASPDAC '11, pages 317–322, Piscataway, NJ, USA, 2011. IEEE Press.

[44] Dmitri B. Strukov and Konstantin K. Likharev. A reconfigurable architecture for hybrid CMOS/Nanodevice circuits. In *Proceedings of the 2006 ACM/SIGDA 14th international symposium on Field programmable gate arrays*, FPGA '06, pages 131–140, New York, NY, USA, 2006. ACM.

[45] C. Dong, W. Wang, and S. Haruehanroengra. Efficient logic architectures for CMOL nanoelectronic circuits. *Micro Nano Letters, IET*, 1(2):74 –78, december 2006.

[46] Z. Abid, A. Alma'aitah, M. Barua, and W. Wang. Efficient CMOL gate designs for cryptography applications. *Nanotechnology, IEEE Transactions on*, 8(3):315 –321, may 2009.

[47] Gang Chen, Xiaoyu Song, and Ping Hu. A theoretical investigation on CMOL FPGA cell assignment problem. *Nanotechnology, IEEE Transactions on*, 8(3):322 –329, may 2009.

[48] William N.N. Hung, Changjian Gao, Xiaoyu Song, and D. Hammerstrom. Defect-tolerant CMOL cell assignment via satisfiability. *Sensors Journal, IEEE*, 8(6):823 –830, june 2008.

[49] Yinshui Xia, Zhufei Chu, William N.N. Hung, Lunyao Wang, and Xiaoyu Song. CMOL cell assignment by genetic algorithm. In *NEWCAS Conference (NEWCAS), 2010 8th IEEE International*, pages 25 –28, june 2010.

[50] Zhufei Chu, Yinshui Xia, William N.N. Hung, Lunyao Wang, and Xiaoyu Song. A memetic approach for nanoscale hybrid circuit cell mapping. In *Digital System Design: Architectures, Methods and Tools (DSD), 2010 13th Euromicro Conference on*, pages 681 –688, sept. 2010.

[51] Y. Xia, Z. Chu, W. Hung, L. Wang, and X. Song. An integrated optimization approach for nano-hybrid circuit cell mapping. *Nanotechnology, IEEE Transactions on*, PP(99):1, 2011.

[52] Quang Huy Nguyen, Yew-Soon Ong, and Meng Hiot Lim. A probabilistic memetic framework. *Evolutionary Computation, IEEE Transactions on*, 13(3):604 –623, june 2009.

[53] CMOL FPGA CAD 1.0. Technical report, Stony Brook, State University of New York, 2008.

[54] J. Rose V. Betz and A. Marquardt. *Architecture and CAD for deep-submicron FPGAs*, volume 497 of *Kluwer Int. Series in Eng. and Comp. Science.* Kluwer Academic, Boston, London, 1999.

[55] Vaughn Betz and Jonathan Rose. VPR: A new packing, placement and routing tool for FPGA research. pages 213–222, 1997.

[56] Frank K Hwang. *Steiner Tree Problem, The. Annals of Discrete Mathematics*, volume 53. ELSEVIER SCIENCE AND TECHNOLOGY, February 2010.

[57] Hossein Hamidipour, Parviz Keshavarzi, and Ali Naderi. Routing congestion removing of CMOL FPGA circuits by a recursive method. In *Proceedings of the 9th WSEAS international conference on Microelectronics, nanoelectronics, optoelectronics*, MINO'10, pages 75–79, Stevens Point, Wisconsin, USA, 2010. World Scientific and Engineering Academy and Society (WSEAS).

[58] Ralph M. Kling and Prithviraj Banerjee. ESP: A new Standard Cell Placement Package using Simulated Evolution. *Proceedings of 24th Design Automation Conference*, pages 60–66, 1987.

[59] Fred Glover. Tabu SearchPart I. *ORSA Journal on Computing*, 1(3):190–206, Summer 1989.

[60] Fred Glover. Tabu SearchPart II. *ORSA Journal on Computing*, 2(1):4–32, Summer 1990.

[61] Fred Glover, Eric Taillard, and Eric Taillard. A user's guide to Tabu Search. *Annals of Operations Research*, 41:1–28, 1993. 10.1007/BF02078647.

[62] Fred Glover and Manuel Laguna. *Tabu Search.* Kluwer Academic Publishers, Norwell, MA, USA, 1997.

[63] Andre Dehon. Nanowire-based programmable architectures. *ACM Journal on Emerging Technologies in Computing Systems*, 1:109–162, 2005.

[64] C.H. Stapper. Simulation of spatial fault distributions for integrated circuit yield estimations. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 8(12):1314 –1318, dec 1989.

[65] E. M. Sentovich, K. J. Singh, L. Lavagno, C. Moon, R. Murgai, A. Saldanha, H. Savoj, P. R. Stephan, R. K. Brayton, and A. S. Vincentelli. SIS: A System for Sequential Circuit Synthesis. *Electronics Research Laboratory Memorandum*, (UCB/ERL M92/41), May 1992.

[66] H.K. Lee and D.S. Ha. HOPE: an efficient parallel fault simulator. In *Design Automation Conference, 1992. Proceedings., 29th ACM/IEEE*, pages 336 –340, jun 1992.

[67] F. Brglez, D. Bryan, and K. Kozminski. Combinational profiles of sequential benchmark circuits. In *Circuits and Systems, 1989., IEEE International Symposium on*, pages 1929 –1934 vol.3, may 1989.

# Vitae

- Abdalrahman M. Arafeh
- Born in Damascus, Syria, April 5$^{th}$ 1987.
- Received B.S. degree in Computer Engineering from Damascus University, Damascus, Syria in September 2009.
- Joined Computer Engineering, King Fahd University of Petroleum & Minerals, as Research Assistant in February 2010.
- Awarded M.S degree in Computer Engineering from KFUPM, Saudi Arabia in May 2012.
- Co-author on the following publications:
  - Sadiq M. Sait, Feras Chikh Oughali, Abdalrahman Arafeh, "FSM State-Encoding for Area and Power Minimization Using Simulated Evolution Algorithm", Journal of Applied Research and Technology, December 2012.
  - Sadiq M. Sait, Abdalrahman M. Arafeh, "Efficient CMOL Nanoscale Hybrid Circuit Cell Assignment Using Simulated Evolution Heuristic" in

The 22nd Great Lakes Symposium on VLSI, GLSVLSI2012, Salt Lake City, Utah, 2012.

- Abdalrahman Arafeh, Feras Chikh Oughali, Tarek Sheltami, Ashraf Mahmoud, “A Contention Free Multi-Channel MAC Protocol With Improved Negotiation Efficiency for Wireless Adhoc Networks”, Proceedings of the International Conference on Ambient Systems, Networks and Technologies, Paris, France, 2010.