Crosstalk Minimization in Channel Routing for VLSI Circuit Synthesis

Thesis Submitted

by

Achira Pal (*nee* Biswas)

Doctor of Philosophy (Engineering)

Department of Computer Science and Engineering
Faculty Council of Engineering and Technology
Jadavpur University
Kolkata 700 032, India

2017

Dedicated to

The fond memory of my parents
Who would have been the happiest
To see this work in this complete form

JADAVPUR UNIVERSITY

KOLKATA – 700 032, INDIA

INDEX NO.: 57/10/E

1. **Title of the Thesis:** Crosstalk Minimization in Channel Routing for VLSI Circuit Synthesis

2. **Name, Designation, and Institution of the Supervisors:**

   Dr. Atal Chaudhuri
   Professor
   Department of Computer Science and Engineering
   Jadavpur University, Kolkata – 700 032

   Dr. Alak Kumar Datta
   Professor
   Department of Computer and System Sciences
   Visva-Bharati, Santiniketan, Bolpur, West Bengal – 731 235

3. **List of Publications:**

   Refereed Journals

J[1] **Achira Pal**, D. Kundu, A. K. Datta, T. N. Mandal, and R. K. Pal, Algorithms for Reducing Crosstalk in Two-Layer Channel Routing, ***Journal of Physical Sciences (ISSN: 0972-8791)***, vol. 10, pp. 167-177, Dec. 2006.

J[2] **Achira Pal**, A. K. Datta, and R. K. Pal, Parallel Crosstalk Minimization Algorithms for Two-Layer Channel Routing. ***The Icfai Journal of Computer Sciences (Reference # 56J-2007-10-02-01)***, vol. I, no. 2, pp. 31-44, Oct. 2007.

J[3] **Achira Pal**, A. K. Datta, and R. K. Pal, Weighted Hamiltonian Path Problem is also NP-Hard, ***The Icfai Journal of Computer Sciences (Reference # 56J-2007-10-02-01)***, vol. II, no. 2, pp. 80-82, Apr. 2008.

J[4] **Achira Pal**, T. N. Mandal, D. Kundu, A. K. Datta, and R. K. Pal, Algorithms for Generating Random Channel Instances for Channel Routing Problem, ***International Journal of Applied Research on Information***

Technology and Computing (IJARITAC) (ISSN: 0975-8070), vol. 1, no. 1, pp. 106-129, Jan-Apr 2010.

J[5] **Achira Pal**, T. N. Mandal, A. Khan, R. K. Pal, A. K. Datta, and A. Chaudhuri, Two Algorithms for Minimizing Crosstalk in Two-Layer Channel Routing, ***International Journal of Emerging Trends and Technology in Computer Science (IJETTCS) (ISSN: 2278-6856)***, vol. 3, no. 6, pp. 194-204, 2014.

J[6] **Achira Pal**, T. N. Mandal, A. Khan, R. K. Pal, A. K. Datta, and A. Chaudhuri, A Review on Crosstalk Avoidance and Minimization in VLSI Systems, ***International Journal of Emerging Technology and Advanced Engineering (IJETAE) (ISSN: 2250-2459 (Online))***, vol. 5, no. 3, pp. 144-150, 2015.

J[7] **Achira Pal (*nee* Biswas)**, A. Chaudhuri, R. K. Pal, and A. K. Datta, Hardness of Crosstalk Minimization in Two-Layer Channel Routing, ***INTEGRATION, the VLSI Journal (Elsevier) (ISSN: 0167-9260)***, vol. 56, pp. 139-147, 2017.

Refereed Conference Proceedings

C[1] **Achira Pal**, A. Singha, S. Ghosh, and R. K. Pal, High Performance Routing for VLSI Circuit Synthesis, ***Proceedings of the Sixth IEEE VLSI Design and Test Workshops 2002 (IEEE VDAT 2002)***, Bangalore, India, pp. 348-351, Aug. 29-31, 2002.

C[2] **Achira Pal**, A. Singha, S. Ghosh, and R. K. Pal, Crosstalk Minimization in Two-Layer Channel Routing, ***Proceedings of the 17th IEEE Region 10 International Conference on Computers, Communications, Control and Power Engineering (IEEE TENCON 2002)***, Beijing, China, vol. 1, pp. 408-411, Oct. 28-31, 2002.

C[3] **Achira Pal**, B. Dam, S. Sadhu, and R. K. Pal, Performance Driven Physical Synthesis, ***Proceedings of the International Conference on Communications, Devices and Intelligent Systems (CODIS 2004)***, Kolkata, India, pp. 194-197, Jan. 9-10, 2004.

C[4] **Achira Pal**, A. K. Datta, D. Kundu, T. N. Mandal, and R. K. Pal, Algorithms for High Performance Two-Layer Channel Routing, ***Proceedings of the 22nd IEEE Region 10 International Conference on Intelligent Information Communication Technologies for Better Human Life (IEEE TENCON 2007)***, CD: Session: WeSC-O1.4 (Electronic Design Automation (EDA) of System-on-Chip) (Four pages), Taipei, Taiwan, Oct. 30 – Nov. 02, 2007.

C[5] **Achira Pal**, A. K. Datta, and R. K. Pal, On Weighted Hamiltonian Path Problem, ***Proceedings of the International Conference on Electronics, Computer and Communication (ICECC 2008)*** *(ISBN 984-300-002131-3)*, University of Rajshahi, Bangladesh**,** pp. 97-100, Jun. 27-29, 2008.

C[6] **Achira Pal**, T. N. Mandal, A. K. Datta, D. Kundu, and R. K. Pal, Generation of Random Channel Specifications for Channel Routing Problem, ***Proceedings of the 11th IEEE International Conference on Computer and Information Technology (IEEE ICCIT 2008) and Workshops***, Khulna, Bangladesh, pp. 19-24, Dec. 24-27, 2008.

C[7] **Achira Pal**, T. N. Mandal, A. K. Datta, R. K. Pal, and A. Chaudhuri, Approximate and Bottleneck High Performance Routing for Self-healing VLSI Circuits, ***Presented in the Second IEEE International Workshop on Reliability Aware System Design and Test (IEEE RASDAT 2011)*** (In conjunction with the *IEEE 24th International Conference on VLSI Design (IEEE VLSID 2011)* and the *IEEE 10th International Conference on Embedded Systems (IEEE ES 2011)*), Chennai, India, Jan. 6-7, 2011.

4. **List of Patents:** Nil

5. **List of Presentations in International Conferences/Workshops:**

C[1] **Achira Pal**, T. N. Mandal, A. K. Datta, R. K. Pal, and A. Chaudhuri, Approximate and Bottleneck High Performance Routing for Self-healing VLSI Circuits, ***Presented in the Second IEEE International Workshop on Reliability Aware System Design and Test (IEEE RASDAT 2011)*** (In conjunction with the *IEEE 24th International Conference on VLSI Design*

(IEEE VLSID 2011) and the *IEEE 10th International Conference on Embedded Systems (IEEE ES 2011)*), Chennai, India, Jan. 6-7, 2011.

CERTIFICATE FROM THE SUPERVISORS

This is to certify that the thesis entitled “Crosstalk Minimization in Channel Routing for VLSI Circuit Synthesis” submitted by Smt Achira Pal (*nee* Biswas), who got her name registered on 30.08.2010 for the award of Ph.D. (Engineering) degree of Jadavpur University is absolutely based upon her own work under the supervision of Prof. Atal Chaudhuri and Prof. Alak Kumar Datta, and that neither her thesis nor any part of the thesis has been submitted for any degree/diploma or any other academic award anywhere before.

Dr. Atal Chaudhuri	Dr. Alak Kumar Datta
Professor	Professor
Department of Computer Science and Engineering	Department of Computer and System Sciences
Jadavpur University, Kolkata	Visva-Bharati, Santiniketan

Acknowledgement

Foremost, even though only my name appears on the cover of this dissertation, a great many people have contributed towards its production. I owe my gratitude to all those people who have made this dissertation possible and because of whom my research experience has been one that I will cherish forever.

I have great pleasure to express my deepest feelings of gratitude to Dr. Atal Chaudhuri, Professor, Department of Computer Science and Engineering, Jadavpur University and Dr. Alak Kumar Datta, Professor, Department of Computer and System Sciences, Visva-Bharati, Santiniketan, who as my respective supervisor and co-supervisor have extensively helped in completion of this work with their valuable suggestions, guidance, cooperation, patience, and love. I do consider myself very lucky for working under such knowledgeable, helpful, and gentle persons.

I would like to thank all the faculty members and non-teaching staff members of the Department of Computer Science and Engineering along with all my co-researchers for their valuable help, suggestions, and encouragements.

I am also grateful to my co-researchers, Dr. Debasis Dhal, Mr. Tarak Nath Mandal, Mr. Debojit Kundu, and Mr. Abhinandan Khan, to name a few, for their untiring support, sharing thoughts, long discussions, continuous encouragement, and helping me understand and enrich my ideas.

I would also like to express my sincere thanks to my colleagues at my place of work, Harinavi Subhasini Balika Sikshalaya, in South 24 Parganas. They have provided all the support and encouragement needed to complete my dissertation, without which I may not have been able to pursue my research.

Last but not the least, I owe a huge debt to my beloved parents, my husband and our son and daughter whose inspiration, patience, understanding, love and unfailing support have enabled me to complete this dissertation.

(**Achira Pal (*nee* Biswas)**)

Abstract

The *channel routing problem* (*CRP*) is the problem of computing a feasible routing solution for the nets present in a channel so that the number of tracks required to route the channel is minimized. A *channel* is a rectangular routing region that has two open ends, the left and right sides of the region, and the other two sides of the channel contains two rows of fixed terminals. The major cost factors that, in isolation or in combination, are normally minimized in CRP are the area, net wire length, via, and layer of interconnection. Besides, there are several other high performance factors like signal delay, power consumption, heat generation, hot spot formation, electrical hazards, and so on and so forth that are all research issues nowadays and these objectives are needed to be considered in channel routing to maximize the chip performance, even after a feasible routing solution is there.

In this thesis, we have considered the problem of crosstalk minimization as a kind of electrical hazard that need to be reduced to enhance circuit performance. As fabrication technology advances and feature size reduces, devices are placed in closer to each other and interconnecting wire segments are assigned with narrower pitch, whereas the circuits' operations are realized at higher frequencies. As a result, electrical hazards, viz., *crosstalk* between wire segments are evolved. More crosstalk means more noise and more signal delay that reduce the circuit performance. Therefore, it is desirable to develop channel routing algorithms that not only reduce the channel area but also crosstalk. Work on routing channels with reduced crosstalk is very important from high performance requirement for VLSI circuit synthesis.

There are several theoretical problems on crosstalk minimization in two-layer channel routing, some of which are posed and proved as NP-complete in this thesis. Subsequently, this thesis includes algorithms that have been designed for reducing crosstalk in two-layer channel routing, devises algorithms for generation of a large number of random channel specifications, parallel algorithms for minimizing crosstalk, heuristics for lessening crosstalk for reduced area routing solutions in order to optimize cost and maximize circuit performance that are some of the prime contributions in brief to mention. In this work, we have studied the computational complexity issues of the crosstalk minimization problem for simple and general instances of channel specification with a partition of nets so that the nets in a class of the given partition are to be assigned to the same track, the simple as well as general instances of channel specifications with only two-terminal nets but without any imposed partition of (non-overlapping) nets to tracks, the bottleneck crosstalk minimization problem, and so on and so forth in the reserved no-dogleg two-layer VH channel routing model.

In all these cases, the problems are considered when doglegging is allowed as well. We further investigate the existence of exact or heuristic algorithms and approximation algorithms for the abovementioned problems of crosstalk minimization in two-layer channel routing. This thesis also identifies that the crosstalk minimization problem in the three-layer VHV and HVH channel routing models are yet open for future researchers.

Table of Contents

List of Figures

List of Tables

List of Abbreviations

3D	Three Dimensional
BER	Bit Error Rate
CAC	Circular Arc Colouring
CAC	Crosstalk Avoidance Coding
CAD	Computer Aided Design
CE	Circuit Extraction
CMOS	Complementary Metal-Oxide Semiconductor
CR	Concurrent Read
CRCW	Concurrent Read Concurrent Write
CREW	Concurrent Read Exclusive Write
CRP	Channel Routing Problem
CW	Concurrent Write
DAG	Directed Acyclic Graph
DDE	Deutsch’s Difficult Example
DFS	Depth First Search
DRC	Design Rule Checking
DSM	Deep Submicron
DTM	Deterministic Turing Machine
DSP	Digital Signal Processing
EA	Exhaustive Algorithm
ER	Exclusive Read
EREW	Exclusive Read Exclusive Write
FEN	Far-End Noise
FRR	Flit Reordering / Rotation
FWM	Four-Wave Mixing
GA	Genetic Algorithm
GA	Greedy Algorithm
HCG	Horizontal Constrained Graph
HNCG	Horizontal Non-Constrained Graph
HP	Hamiltonian Path
HP*	Weighted Hamiltonian Path
IC	Integrated Circuit

ICP	Interval Containment Problem
ILP	Integer Linear Programming
LEA	Left Edge Algorithm
LSI	Large Scale Integration
MBFF	Multi-Bit Flip-Flop
MC	Matrix Combination
MCC1	Minimum Clique Cover 1
MCC2	Minimum Clique Cover 2
MRR	Microring Resonator
MS	Matrix Selection
MSI	Medium Scale Integration
MOSFET	Metal-Oxide-Semiconductor Field-Effect-Transistor
NC	Net Change
NDTM	Non-Deterministic Turing Machine
NEN	Near-End Noise
NoC	Network-on-Chip
NP	Non-deterministic Polynomial Time Computable Problem
OD	Opposite Direction (transitions)
OSF	Optical Switching Fabric
P	Deterministic Polynomial Time Computable Problem
PON	Passive Optical Network
PRAM	Parallel Random Access Machine
QAP	Quadratic Assignment Problem
RVCG	Reduced Vertical Constrained Graph
SA	Simulated Annealing
SEC–DED	Single-Error-Correction–Double-Error-Detection
SEQ	Sequencing to Minimize Weighted Completion Time
SPM	Self-Phase Modulation
SoC	System-on-Chip
SSI	Small Scale Integration
TAH	Track Assignment Heuristic
TC	Track Change
TFTEC	Thin-Film Thermo-Electric Cooler
TSP	Travelling Salesman Problem
TTL	Transistor-Transistor Logic

ULSI	Ultra Large Scale Integration
VCG	Vertical Constrained Graph
VDSM	Very Deep Submicron
VHDL	VHSIC Hardware Description Language
VLSI	Very Large Scale Integration
WDM	Wavelength-Division Multiplexing
WRM	Wavelength-Routing-Matrix
WSI	Wide Scale Integration
XPM	Cross-Phase Modulation

Chapter: 1

Introduction to Crosstalk Minimization Problem

1.1 Overview

The eventual endeavour in the present age of Information Technology is information generation and dissemination of the same by anybody, anytime, and anywhere. Very Large Scale Integration (VLSI) technology has revolutionized the electronics industry and established the twentieth century as the computer age, but even now in the second decade of the twenty-first century, it is approaching its fundamental limits in the submicron process of miniaturization. These are needed due to the increasing demands of high speed, throughput, and overall performance in modern computing applications plus the explosive proliferation of data volume to be stored and processed, that necessitate a revolutionary supercomputing technology.

From the beginning of 1960, *Integrated Circuit* (*IC*) fabrication technology has evolved from being able to integrate a few transistors in *Small Scale Integration* (*SSI*) to today’s integration of well over millions of transistors in *Very Large Scale Integration* (*VLSI*). Gradually from the very beginning to the present era, some generation of ICs with the number of transistors on a single chip growing from 4 to more than 40 million has been realized. A tentative time dependent numbers of transistors that have relatively been accumulated to realize scaled integrated circuits are shown in Table 1.1 as a growth of VLSI technology in the last five decades [1, 4, 23, 24, 43, 69].

Table 1.1: Generation-wise Integrated Circuits.

Generation of Integrated Circuits (ICs)	**Number of Transistors**	**Number of Gates**
Small Scale Integration (SSI)	4 to 400	1 to 100
Medium Scale Integration (MSI)	> 400 to 4,000	> 100 to 1,000
Large Scale Integration (LSI)	> 4,000 to 40,000	> 1,000 to 10,000
Very Large Scale Integration (VLSI)	> 40,000 to 4,00,000	> 10,000 to 1,00,000
Ultra Large Scale / Wide Scale Integration (ULSI / WSI)	4,00,001 and above	1,00,001 and above

Integrated circuits consist of a number of electronic components realized by a family of transistors, built by layering several different materials in a well-defined fashion on a silicon base called a *wafer*. For smaller ICs, *transistor-transistor logic* (*TTL*) is a well-accepted logic family, but nowadays, in general, metal oxide semiconductor field effect transistor (MOSFET), or only MOS, either in the form of an n- or a p-type, or in the complementary form has considerably been used in realizing most scaled and compacted VLSI circuits. By the way, at some point in time, *Ultra Large Scale Integration* (*ULSI*) or *Wide Scale Integration* (*WSI*) were the terms to convey the sense of level of integration beyond VLSI, but ultimately, worldwide academicians and industry people have accepted only VLSI as a generalized term to cover all these (ULSI / WSI) beyond a number of transistors that are accumulated in realizing a chip and ULSI / WSI as a generation of integrated circuits no longer exists.

In practice, the designer of an IC transforms a circuit description into a geometric description, which is known as a *layout*. A layout consists of a set of planar geometric shapes in several layers. In some other words, a geometric description that is obtained from a circuit description while designing an IC is known as a *layout*. A layout consists of a set of planar geometric shapes in several layers, and then it is checked to ensure that it meets all the design requirements. Through some intermediate steps, the design files are then converted into pattern generator files, which are used to create patterns called *masks* by an optical pattern generator. During fabrication, these masks are used to pattern a silicon wafer using a series of photolithographic steps.

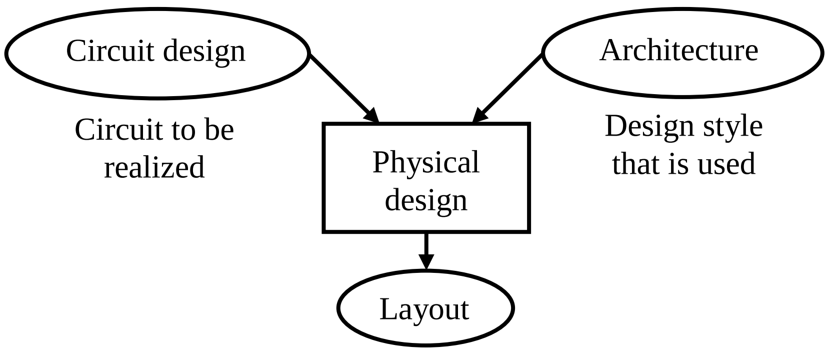


Figure 1.1: Input and output to the physical design step. The circuit to be realized along with the design style is inputted while the layout is the output of this step.

The process of converting the specifications of an electrical circuit into a layout is called the *physical design*. In this step, the circuit a designer likes to design is obtained as the output of the circuit design step that is inputted to the physical design along with the design style as architecture (see Figure 1.1). This design style could either be full or semi-custom design, or standard cell design, or gate array design, or some of their combinations or variations. Besides, the output to the physical design step is the layout, which is supposed to be realized through fabrication in a laboratory.

The physical design step is an extremely error prone and tedious process because of the minuteness of the individual components and the tight tolerance requirements. For about last two decades, the VLSI design is realized at the submicron level, where 1 micron (or micrometre) = 1.0×10^{-6} metres. At present, the smallest geometric feature of a component can be as small as 14 nanometres, where 1 nanometre = 1.0×10^{-9} metres [14, 88]. However, it is expected that the feature size can be reduced even further within a couple of years. This small feature size may allow fabrication of as many as 40 million (or more) transistors on a 25 mm × 25 mm chip (or even less).

Due to the requirement of exacting details for each component and interconnecting wire segment in the fabrication process and the accumulation of a very large number of components on a single semiconductor chip, the physical design process is not practical without the help of computers. As a result, almost all the phases of physical design comprehensively use different computer aided design (CAD) tools, and many phases have already been partially or fully automated. This automation of the physical design process has increased the level of integration, enhanced chip performance, and reduced the turn-around time.

The VLSI physical design automation is essentially the study of different problems and associated algorithms, and the data structures related to the physical design process. There are problems in physical design that are polynomial time computable, but most of the problems in physical design are beyond polynomial time (or exponential time) computable. The objective of this phase is to study the optimal arrangement of devices on a plane (or sometimes in a three-dimensional space in the case of 3D design) and efficient interconnection schemes between these devices to obtain the desired functionality. Since the use of space on a wafer is quite expensive, and the level of purity is never 100%, algorithms that designers develop must use the space very efficiently and competently so that the cost is lowered while the yield of the products is improved.

Besides, a well-fashioned collection of components and devices (in different levels of design) along with their assignment over the chip floor assumes an important role in determining the performance of a chip. Algorithms intended for physical design are also expected to make sure that all the rules required for the fabrication are observed, and the layout is within the limits of tolerance of the fabrication process. To end with, the algorithms ought to be efficient and should be able to grasp and handle very large designs. Each algorithm not only leads to rapid turn-round time but also allows designers to iteratively improve the layouts.

The overall fast growth in integration the chunky has been realized due to automation of various steps involved in design, verification, and fabrications of chips. In this thesis, we have primarily emphasized on developing techniques for crosstalk minimization in two-layer channel routing as one of the most important high performance factors in realizing the desired VLSI circuit. Now we briefly discuss the steps present in the VLSI chip design process in the following section.

1.2 The VLSI Chip Design Process

The VLSI design process starts with a given plan and specification of a VLSI circuit, follows a sequence of steps, and ultimately creates a packaged chip. A typical design process may be represented by a series of steps in the form of a flowchart as shown in Figure 1.2. This process starts with a system specification of the circuit we like to design up to the packaged marketable chip. To achieve a broad viewpoint, a sketch of all the steps of VLSI design process is briefed below [81].

1. **System Specification:** For the necessity of a high level representation of a system, at first we set down the specifications of the system to be designed. The factors that are considered in this process include: performance, functionality, and physical dimensions; the selection of fabrication technology, design style(s), and design methods are also taken care of. Stipulation for size, speed, power, and functionality of the VLSI system to be designed are expected as the end results.

2. **Functional Design:** Behavioural aspects of the system are considered and measured in this step. The outcome is usually a timing diagram or other relationships among the subunits. This information is used to get better the general design process and to lessen the complexity of the subsequent phases.

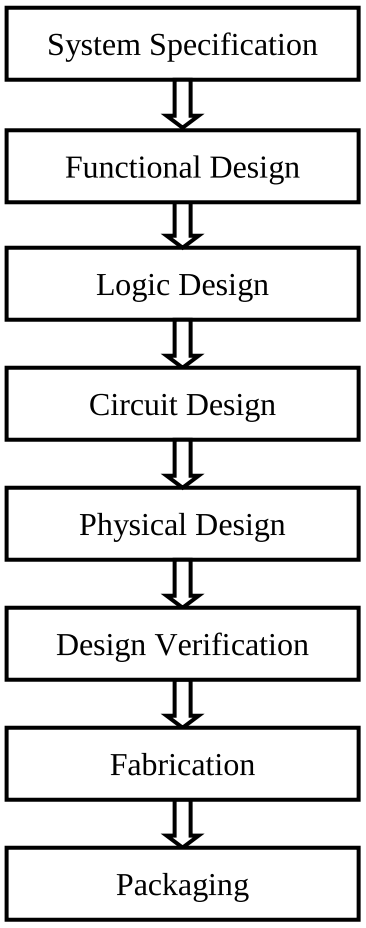


Figure 1.2: The VLSI chip design process.

3. **Logic Design:** In this step, the logic configuration (or structure) that stands for the functional design is derived and tested. The realized design is represented in the form of a textual, schematic, or graphic description. Usually, the logic design is represented by Boolean expressions. Then these expressions are reduced to attain the smallest and often simplest logic design that matches to the above functional design. These are also simulated and tested to confirm their correctness, or customized accordingly.

4. **Circuit Design:** The aim of circuit design is to develop a circuit representation derived from the logic design. The Boolean expressions are transformed into a

circuit depiction by taking into consideration the speed, power requirements, and the electrical performance of different components of the original design. A designed circuit is usually obtained in the form of an entire circuit diagram.

5. **Physical Design:** This is the most essential and central step in the whole process of developing a VLSI chip. In this step, the circuit representation of each component is changed into a geometric representation. This illustration is, in fact, a set of geometric patterns that perform the projected logic function of the related component. Connections among different components are also expressed as geometric patterns (or sketch). This geometric representation of a circuit is called a *layout*. The precise aspects of a layout also depend on design rules that are guidelines based on the inadequacies of the manufacturing process and the electrical properties of the production stuff. Physical design is a very complex process, and in order to handle the complexity of the problem, it is usually broken down into various sub-steps like partitioning, placement, routing, etc. Perhaps, the physical design consumes the maximum time among all steps in the VLSI design process.

6. **Design Verification:** To ensure that the layout meets the system specification and the fabrication requirements, the created layout is verified in this step. Design verification consists of *design rule checking* (*DRC*) and *circuit extraction* (*CE*). DRC is a process that verifies that each individual geometric model must meet the design rules inflicted by the fabrication process. After checking the design rule violation(s), if any, the functionality of the layout is established by CE. This is a reverse engineering process that generates the circuit representation from the layout.

7. **Fabrication:** After verification, the layout is ready for fabrication. The fabrication process is composed of quite a few steps: preparation of wafer, deposition, and diffusion of various materials on the wafer according to layout description. ICs consist of a number of electronic components, fabricated by layering several different materials in a well-defined fashion on a silicon base, known as a wafer. A typical wafer may be of 10 cm in diameter and can be used to produce tens or hundreds of chips. Before a chip is mass produced, a prototype (or archetype) is prepared and tested.

8. **Packaging, Testing, and Debugging:** To end with, the wafer is fabricated and diced in a fabrication laboratory where such a facility is available. Each chip is then packaged and tested to ensure that it meets all design specifications and functions properly. Based on the application, either the chips are packaged, or they are kept bare.

The VLSI chip design process is a very big and complex process that involves iterations as and when necessary, both within a step and among different steps. The complete design process may be viewed as conversion and renovation of depiction in a variety of steps. In each step, a new illustration of the system to be designed is created and analysed. The manifestation is iteratively improved to meet system specifications, or often the system specifications are tuned accordingly. If some design violations are detected, particularly in the physical design step, then this step needs to be repetitive to rectify the faults. Interestingly, this is the reason for which the VLSI design process is also often called the VLSI design cycle. The purpose of VLSI CAD tools is to reduce the number of iterations and thus lessen the time-to-market.

1.3 The VLSI Physical Design Process

The physical design is the most time consuming, droning, and error-prone process in the VLSI chip design cycle. The input to the physical design process is a circuit diagram, and the output is the layout of the circuit to be designed. This is carried out in a number of stages such as *partitioning*, *floorplanning*, *placement*, *routing*, and *compaction*. The different stages of the physical design process are shown in Figure 1.3. Below we present a short outline of all the stages to give a broad perception of VLSI physical design [49, 77, 81].

1. **Partitioning:** A VLSI chip may contain several million transistors (or active devices). The layout of the whole circuit cannot be handled due to the limitation of memory space as well as available computation power. Thus, the circuit is usually partitioned by grouping the components into sub-circuits, known as *blocks* or *modules*. In practice, the partitioning process considers the following factors such as the size of the blocks, number of blocks, and number of interconnections among the blocks. The output of the partitioning stage is a set of blocks along with the interconnections required among the blocks at

some level of the hierarchy. The set of interconnections (or routing) required is referred to as a *net-list*. The partitioning process is hierarchical for designing a typical VLSI circuit, and at the topmost level a circuit may have between 5 to 25 modules, and each module is then partitioned recursively into hundreds of smaller blocks. As for example, a VLSI chip hierarchy is shown in Figure 1.4.

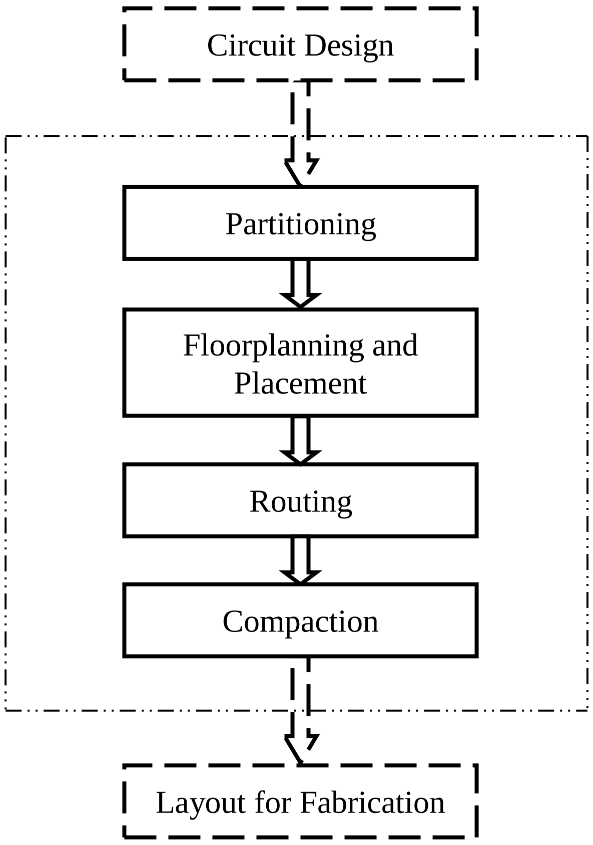


Figure 1.3: The VLSI physical design process.

2. **Floorplanning and Placement:** The selection of *good* layout alternatives for each block as well as the entire chip is the main concern of this step. At some level of the hierarchy, the area of each block can be estimated and computed after partitioning, as it is based approximately on the number and the type of components present in the block. *Floorplanning* is the placement of flexible blocks, i.e. blocks with fixed area but unknown dimensions. It is a much more difficult problem as compared to the placement problem. In floorplanning, several layout options and choices for each block are considered to substitute itself. Usually, the blocks are rectangular in shape, and the lengths and widths

of all the blocks are determined in addition to their locations; however, the lengths and widths of the blocks may vary within a prespecified range.

The exact rectangular shape of a block is determined by the aspect ratio, i.e. aspect ratio is used to assign the block dimensions. The *aspect ratio* of a block is the ratio of the width (*v*) of the block to its length (*h*); see Figure 1.5. Usually, there is an upper and a lower bound on the aspect ratio that a block can have such that the blocks cannot take shapes that are too long or very thin.

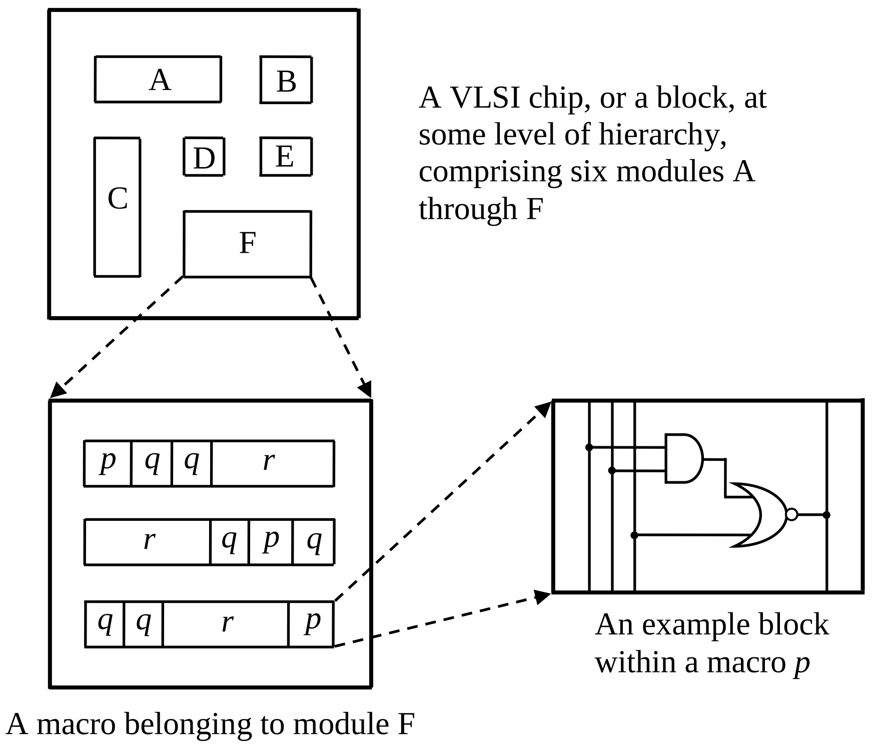

A
B
C
D
E
F
A VLSI chip, or a block, at some level of hierarchy, comprising six modules A through F
p q q r
r q p q
q q r p
An example block within a macro p
A macro belonging to module F

Figure 1.4: An example VLSI chip hierarchy.

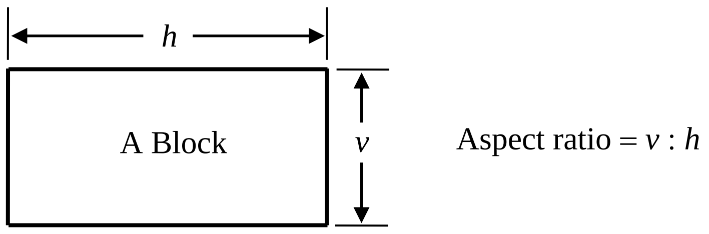


Figure 1.5: Aspect ratio of a block, where *v* is the vertical dimension (or width) of the block and *h* is the horizontal dimension (or length) of the block.

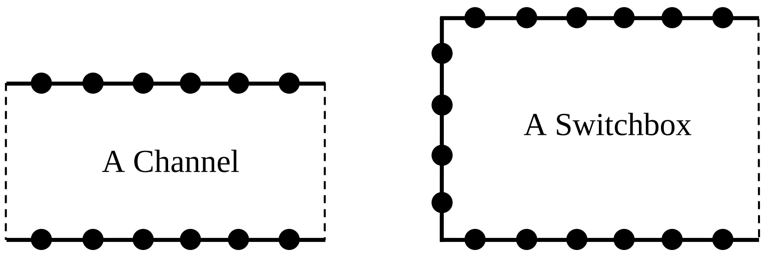
Floorplanning is a very decisive and significant step in the VLSI physical design phase. This step sets up the foundation towards achieving the desired layout, though it is computationally quite hard. Very often the task of the floorplan layout is done by design engineers, rather than automated CAD tools. This is at times obligatory as the key components of an IC are often anticipated for specific locations on the chip.

At the time of placement, the blocks are precisely located on the chip. The objective of placement is to discover a minimum area arrangement for the blocks that aids complete interconnections amongst themselves. Placement is characteristically completed in two phases. In the first phase, a preliminary placement is produced. In the second phase, the early placement is assessed, and iterative upgrading is made in anticipation that the layout has minimum area and obeys the rules of design specifications. Note that some space between neighbouring blocks is left vacant to allocate wires to connect the blocks.

The judgment of the merit of placement is imperfect unless the subsequent routing phase is completed. Often a placement may direct to unroutable design, i.e. routing may not be feasible in the space available. In such a case, the next iteration of placement is essential. To bind the number of iterations of the placement algorithm, an approximation of the required routing space is used during the placement phase. Only a good placement algorithm can endow with a good routing scheme and a desired performance of the circuit. This is due to the fact that once the location of each block is made final, very little can be done to better the routing and on the whole the circuit performance.

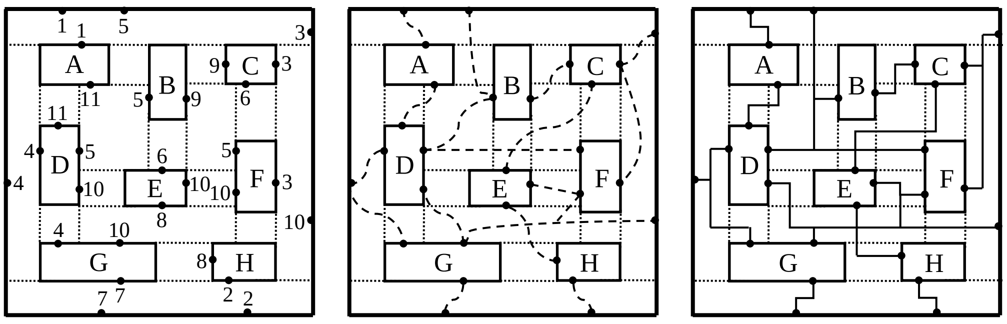
3. **Routing:** At some level of design, the circuit after the placement of modules on the chip floor is completed by producing interconnection amongst the blocks using the vacant region separating the blocks through which a careful interconnection is routed, which is known as *routing*. The routing problem consists of interconnections in the midst of adjacent blocks, according to a precise netlist, that has been assigned positions as a solution of a placement problem. The arrangement of a routing problem consists of the position of

terminals, the *netlist* that indicates which terminals are to be interconnected (in separation) and the routing space available for routing.



(a) **(b)**

Figure 1.6: (a) A *channel* is a rectangular routing region with fixed terminals only on its two opposite sides, and the other two opposite sides are open ends. **(b)** A *switchbox* is a (closed) rectangular routing region with fixed terminals on any three or all four sides of the region.



(a) **(b)** **(c)**

Figure 1.7: (a) Layout of rectangular circuit blocks A through H and pins after placement, at some level of the hierarchy. Terminals are located on the periphery of blocks as well as on the boundary of the chip. Local (rectangular) routing regions (either channels or switchboxes) are separated by dotted lines. **(b)** Interconnection among the blocks after global routing through different local routing regions, as per the netlist. **(c)** Interconnection among the blocks after detailed routing showing each exact geometric assignment.

Hence, the goal of the routing phase is to complete interconnections amongst the blocks according to a given netlist. At first, the space, known as the routing space (which is not covered by blocks) is divided into rectangular regions known as *channels* and *switchboxes*. As for example, a channel and a switchbox are shown in Figures 1.6(a) and 1.6(b), respectively. Then, the routing in each subsequent (routing) region is completed to accomplish the entire desired circuit. The scheme of a routing algorithm is to complete all circuit connections using the shortest possible wire length and using only the channels and switchboxes. This is usually done in two phases, referred to as the *global routing* and *detailed routing* phases.

Connections amongst the appropriate blocks of a circuit are completed in global routing forgetting (temporarily) about the precise geometric details of each wire and pin. For each wire, a global routing algorithm finds a list of channels and switchboxes that are to be used as course of the path for that wire. In other words, global routing specifies the ‘flexible route’ of a wire through different local routing regions in the routing space. Figure 1.7(a) shows a layout of circuit blocks A through H and pins after placement. The remaining space on the chip floor is known as the routing region that has been partitioned into a collection of local rectangular routing regions, distinctive by dotted boundaries, either a channel or a switchbox. Interconnection among the blocks after global routing using loose routes is shown in Figure 1.7(b), where same pins (or terminals), differentiated by distinct numbers, are electrically connected through wires, as per the given netlist.

Detailed routing is the next step after global routing that performs each point-to-point connection between terminals (of the same pin number) on the blocks, as guided by global routing. In this step, the flexible (global) routing is transformed into the preferred exact routing by stating geometric information such as the width of wires along with their track and layer assignment. Channel routing and switchbox routing are the kinds of local routing that help to achieve detailed routing necessary for a chip. Figure 1.7(c) shows detailed routing for the layout of circuit blocks A through H followed by global routing.

Routing is a premeditated and well-studied problem, and a few hundreds of articles have been published about all its usual facets. In view of the fact that most of the problems in routing are computationally hard, the researchers worldwide have paid attention generally on developing heuristic algorithms in the last four or more decades. Consequently, experimental estimation has become an essential part of all such algorithms and some benchmarks have been standardized. Owing to the very nature of the routing problems, entire routing for all the connections can never be guaranteed in many a case.

4. **Compaction:** It is essentially the assignment of condensing the layout in all dimensions, mainly along the orthogonal axes, such that the total area is concentrated as much as possible. By making a chip smaller, wire lengths are reduced which in turn reduces the signal delay amongst the components of the circuit. Along with a smaller area of a chip, it may result in realizing more chips on a given wafer, which in turn reduces the cost of manufacturing. At the same time, the compaction must make sure that no rules regarding the design and fabrication process are violated towards accomplishing a VLSI chip.

The physical design process, like the VLSI chip design process, is iterative in nature and many steps such as global routing and detailed routing are repeated a number of times to acquire an improved layout. Besides, the quality of a result obtained in a step purely depends on the excellence of solution obtained in the previous steps. For example, a *bad* quality placement cannot expect a *good* quality routing. Thus, the earlier steps must have more persuasion on the overall quality of a solution. In other words, partitioning, floorplanning, and placement problems play a more significant role in determining the area and chip performance, as compared to routing and compaction. Since a placement scheme may create an unroutable layout, the chip might need to be re-placed and re-partitioned before another routing can be attempted. By and large, the whole physical design process may be repeated a number of times to achieve the goals of a design, and hence the physical design process is also often known as physical design cycle. The complexity of each step varies depending on the design constraints as well as the design style used [81].

1.4 Channel Routing

The major goal of a routing algorithm is to provide area efficient connections for all the nets present in the circuit of a chip, such that none of the terminals remains disconnected and the overall routing area is minimized. A set of terminals that need to be electrically connected together constitutes a *net*. Terminals of the same net are given the same symbol (or the same integer label).

The process of *routing* builds connections amongst the terminals on the periphery of different blocks (or modules) and also on the boundary of the chip floor. Connections are realized using wire segments assigned to different layers of interconnect. Usually, single layer routing is allowed for some special situations. Some technologies allow only two layers of wiring; this is referred to as *two-layer routing*. Connections between the wire segments assigned to adjacent layers are made using vias.

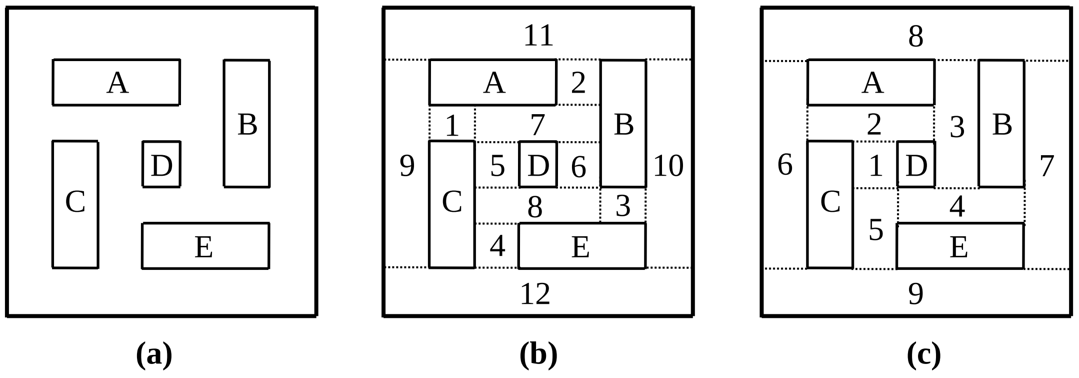


Figure 1.8: **(a)** An (assumed) assignment of five blocks A–E over the chip floor after the placement phase is over. **(b)** The overall routing space is divided into 12 rectangles where the routing regions, 1–6, 9, and 10 are channels, and the remaining routing regions, 7, 8, 11, and 12 are switchboxes. **(c)** An alternative (or better) division of the overall routing space into only nine rectangles where the routing regions, 1–4, 6, and 7 are channels, and the remaining routing regions, 5, 8, and 9 are switchboxes.

To simplify the routing process, the routing regions of a chip are divided into rectangular blocks. The perimeter of these blocks may contain *pins* that need to be connected. In *local routing*, routing within each rectangular routing region is done disjointedly one after another. As shown in Figure 1.6(b), a rectangular routing region with terminals assigned to fixed locations on three (or all four) sides is called a *switchbox* [82]. Producing the detailed connections within a switchbox is called the *switchbox routing problem* [31, 40, 82, 83, 85]. If terminals are assigned to fixed locations only on two opposite sides of a rectangular routing region, then such a region is called a *channel* (see Figure 1.6(a)) [32]. The problem of routing within a channel is called the *channel routing problem* [48, 49, 81, 96]. Note that a channel may contain terminals on either or both of its two (opposite) open ends as well but these are no ways fixed; these are floating terminals. The position of these terminals gets fixed after a routing algorithm is executed for the channel. Moreover, channel routing is a kind of local routing in the detailed routing phase.

A chip floor with only five blocks A–E is shown in Figure 1.8(a), when placement is over. Except the blocks, the remaining space over the chip is known as the overall routing region, which is initially partitioned into 12 (local) rectangular routing regions, out of which there are eight channels, 1–6, 9, 10, and four switchboxes, 7, 8, 11, and 12, as shown in Figure 1.8(b). A different division of only nine rectangles of the overall routing space of the same chip is shown in Figure 1.8(c), where six routing spaces, 1–4, 6, and 7, are identified as channels, and the remaining three rectangles, 5, 8, and 9 are switchboxes. These channels and switchboxes are needed to be routed as the terminals of different nets are there on the periphery of different blocks and also on the boundary of the chip floor.

At this point in time, we may mention that a channel area is adjustable as the blocks (with fixed terminals) may relatively move as needed, they may come closer or go apart, though a major goal of channel routing is to minimize the overall channel area required. This flexibility of relative movement of blocks is not feasible in the case of switchbox routing. In fact, whether a switchbox is completely routable for the nets present therein is the prime issue in routing a switchbox. In our study, we have been acquainted with the fact that routing a channel is a hard problem, and routing a switchbox is even harder than that. So, at the time of dividing the overall routing space other than the blocks on a chip floor, we need to provide a sequence of generating (local) rectangular routing regions (and route them) so that we have, if possible, more channels and less (or no) switchboxes.

As a channel is a rectangular routing region bounded by two parallel rows of fixed terminals, the input to the channel routing problem is usually provided in the form of *channel specification* (or *netlist*) that contains two rows of terminals, TOP and BOTTOM. One such example channel specification is shown below whose net distribution along the length of the channel is depicted in Figure 1.9.

```
TOP     : 3 8 0 0 4 0 0 1 3 0 6 7 0 0 5 0 0 5
BOTTOM  : 0 0 4 2 0 2 0 0 0 8 0 0 6 7 0 0 1 0
```

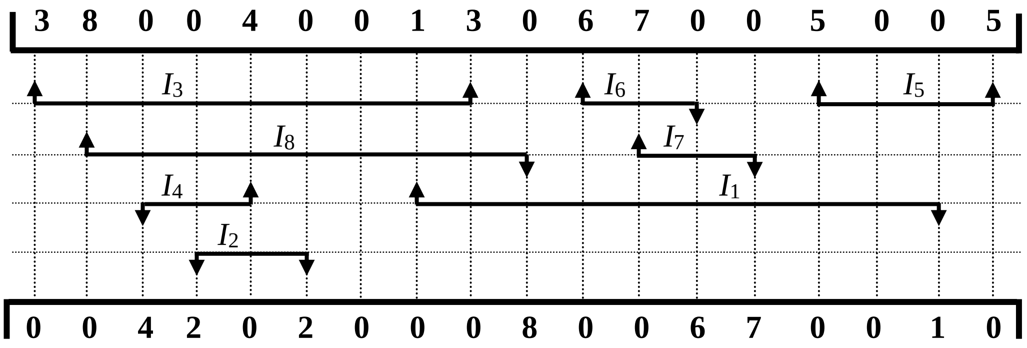


Figure 1.9: An example channel of eight nets. Intervals of the nets are placed in four different tracks. Terminals are vertically aligned along the columns of the channel. The length of the channel (i.e. the number of columns) is 18. Arrows indicate the terminals to be connected, either at the top or at the bottom, to complete the required interconnection of all nets belonging to the channel.

1.4.1 The Structure of a Channel

A channel has two open ends, the left and right sides of the rectangular routing space. The other two sides, i.e. the top and bottom sides of the rectangle have two rows of fixed terminals. The top view of a channel is shown in Figure 1.9. It consists of terminals of eight different nets that are spread over 18 columns along the length of the channel. In other words, the terminals are aligned vertically in *columns*. A set of terminals that need to be electrically coupled together is called a *net*. The terminals of the same net are assigned the same number. Zeros are non-terminals, not required to be connected. Here, the channel contains only two-terminal nets; in general, a net in a channel may have two or more terminals as well.

Characteristically, the connections required within a channel are specified as two equal sized lists of numbers, one for terminals of the upper row of the channel and the other for the terminals of the lower row of the channel. The size of each of these lists is the number of columns in the channel. As we have already mentioned, these lists together are called the *netlist* or *channel specification* [49, 96].

In general, in two-layer channel routing, rectilinear wire segments of the nets are assigned to a minimum number of tracks of a channel. *Tracks* are successively equispaced assumed horizontal lines parallel to two rows of fixed terminals. In order to achieve a routing solution, the horizontal wire segments of all the nets belonging to a channel are assigned to tracks, and the vertical wire segments are assigned to assumed equispaced columns (in succession). In two-layer channel routing, such a kind of assignment of wire segments helps in realizing routing solutions for most of the channels. The structure of a channel is shown in Figure 1.10, and the routing models have been discussed in the following section.

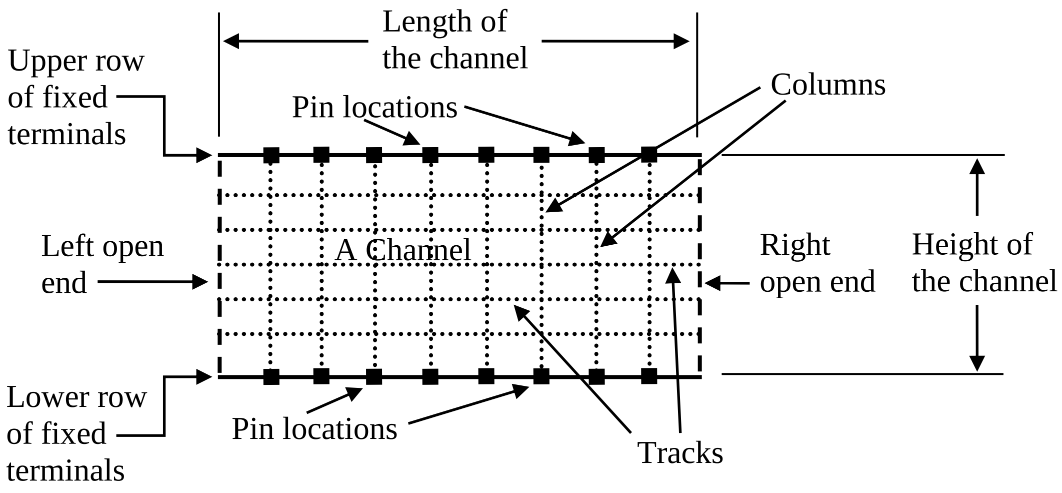


Figure 1.10: The structure of a channel, which is always rectangular in shape with two rows of fixed terminals situated at the pin locations. Rectilinear wire segments of different nets are assigned to different tracks and columns of the channel; tracks are parallel to the rows of fixed terminals and columns are perpendicular to the rows of fixed terminals. The left and right ends of the channel are open ends. If the number of columns (or the number of pin locations) is *c*, then the length of the channel is *c*+1. If the number of tracks required to route the channel is *t*, then the height of the channel is *t*+1. In reality, the height of a channel is determined by the number of tracks required to route the channel (that certainly vary from channel to channel).

1.4.2 Channel Routing Models

A routing algorithm that solves an instance of a channel must follow some routing model to route all its nets [49, 81]. There are several ways of assigning nets to realize a feasible routing solution. Accordingly, the models have their relative advantages and disadvantages. Some of the models are practically more acceptable, whereas some others might be less costly. On the other hand, there are simpler routing models, but the algorithm may fail to generate a high performance routing solution. So, there are trade-offs and debates among the researchers and industry people before accepting a particular model as the most practical one from fabrication and cost as well as performance points of view. Some of the important channel routing models are discussed below.

1. **The Grid-based versus Gridless Routing Model**

In a *grid-based routing model*, a rectilinear grid is superimposed on the routing region, and the wires are restricted to follow paths along the grid lines. In the grid-based routing model, the tracks are equispaced, and the columns are also equispaced. These theoretical separations are finally visualized by a technology supported fabrication tool, which is accepted in general to accomplish a routing solution. This is an abstract routing model, but this routing model is received worldwide in realizing most of the routing solutions.

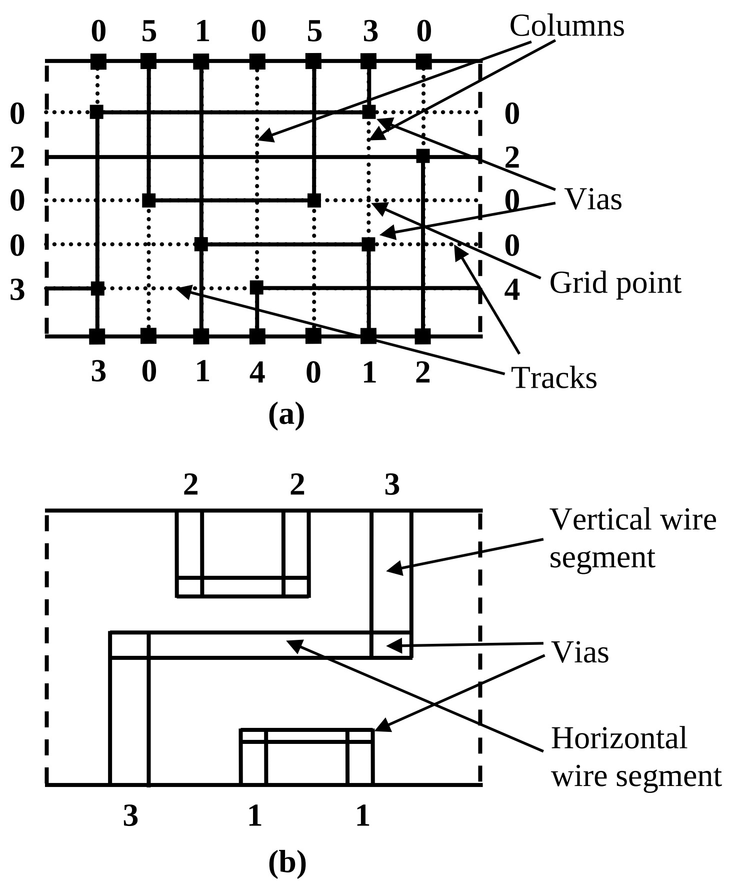


Figure 1.11: (a) A routing solution in a grid-based routing model where two orthogonal wire segments of a net that reach a grid point are connected by a via. **(b)** No superimposed grid is present in a gridless routing model; pin locations are not necessarily equispaced, and the thickness of wire segments may also vary.

A routing solution in a grid-based routing model is shown in Figure 1.11(a) that contains five tracks and seven columns. A dotted grid is superimposed over the routing region. Intersections of orthogonal grid lines are known as grid points where the vias are placed, through which the vertical and horizontal wire segments of a net are connected.

On the other hand, any model that does not follow the 'gridded' structure is referred to as a *gridless routing model*. As a result, the gap between adjacent wire segments, either assigned to tracks or columns, varies from case to case. This routing model might be more acceptable from the real design point of view, where wire segments of different width (or diameter) are used, but a routing solution of this model is more expensive.

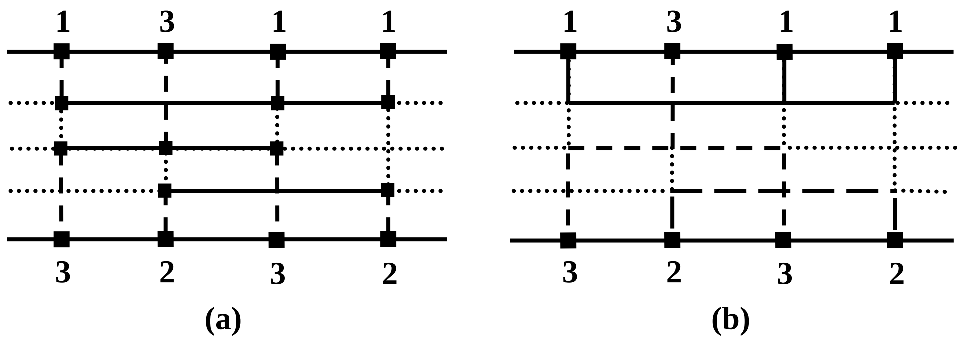


Figure 1.12: (a) Routing in a reserved layer routing model. Here a channel is routed using two layers of interconnect; one layer is reserved for horizontal wire segments (firm segments), and the other layer is reserved for vertical wire segments (dashed segments). Vias are introduced at grid points to connect orthogonal wire segments of respective nets. **(b)** An unreserved layer routing solution for the same channel, where no layer is assigned for a given type of wire segments. In this routing solution, three nets are assigned to three different layers of interconnect, differentiated by the firm, smaller, and bigger dashed segments (while net 2 can also be assigned to the same layer of net 1).

2. The Reserved Layer versus Unreserved Layer Routing Model

In a *reserved layer routing model*, all horizontal wire segments are assigned to a particular layer, known as the *horizontal layer* (H), and all the vertical wire segments are assigned to a separate layer, known as the *vertical layer* (V) in the two-layer VH routing model. In the case of *unreserved layer routing model*, the wire segments may not follow any strict rule of their assignment to tracks; in fact, a routing algorithm for such a routing model is responsible for the assignment of wire segments. Figure 1.12(a) shows a reserved layer routing solution of a channel whereas an unreserved layer routing solution for the same channel is shown in Figure 1.12(b). In this routing solution, net 2 could also be assigned to the same layer of net 1.

3. The Overlap versus Non-Overlap Routing Model

In an *overlap routing model*, the wire segments of two different nets may overlap on adjacent layers. If such overlaps are not allowed, the model is called a *non-overlap routing* model. A routing solution of a channel in the overlap routing model is shown in Figure 1.13(a), whereas that for the same channel in the non-overlap routing model is shown in Figure 1.13(b). A non-overlap routing model is more suitable from the viewpoints of design, fabrication, and performance as a routing solution in an overlap routing model may lead to more electrical hazards.

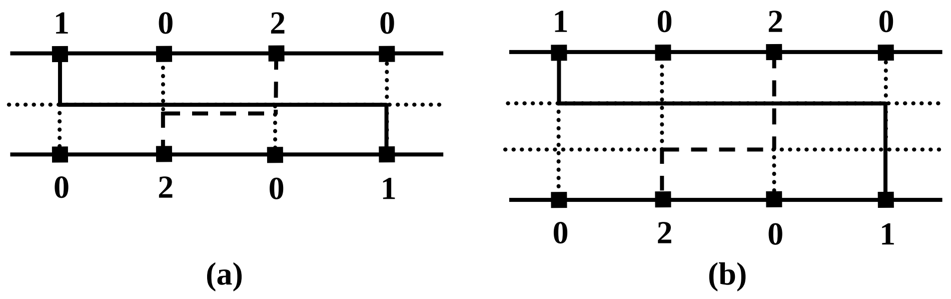


Figure 1.13: (a) A routing solution in an overlap routing model, where a single track is used to assign the horizontal wire segments of two different nets in different layers of interconnects. **(b)** A non-overlap routing solution in an unreserved layer routing model that may require more tracks to route all the nets.

4. The No-Dogleg versus Dogleg Routing Model

In the case of *dogleg routing* (or *doglegging*), the horizontal wire segment of a net may be split into two or more parts and assigned to different tracks, and then the vertical connections are made accordingly using vertical wire segments. Figures 1.14(b) and 1.14(c) show such routing solutions where sub-segments of different nets are assigned to different tracks. If the route for a net is allowed to have only one horizontal wire segment, then it is called a *no-dogleg route*. Figure 1.14(a) shows a no-dogleg routing solution for the same channel instance. If the route for a net is allowed to dogleg only in those columns in which it contains a terminal, then it is called a *restricted dogleg route*. This is explained in Figure 1.14(b), where doglegging is made in a column that contains an intermediate terminal of net 1. Otherwise, it is known as an *unrestricted dogleg route*. An unrestricted dogleg routing solution for the same channel instance is shown in Figure 1.14(c), where net 2 is doglegged in a column containing no terminal of net 2.

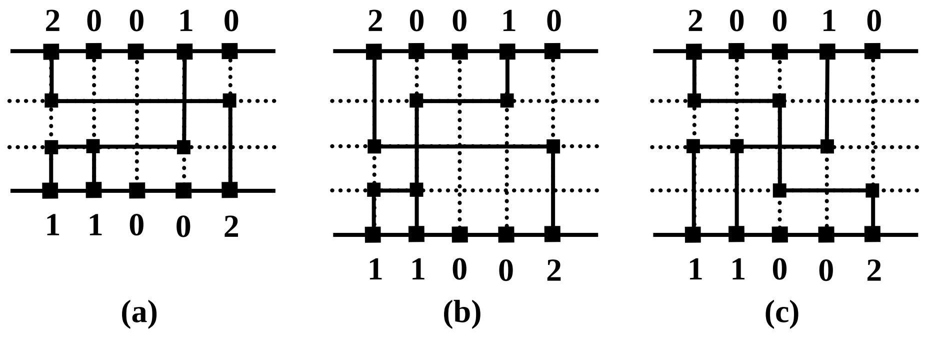


Figure 1.14: **(a)** A no-dogleg routing solution. **(b)** A restricted dogleg routing solution, where net 1 is split (into subnets) in a column that contains a terminal of net 1. **(c)** An unrestricted dogleg routing solution, where net 2 is split (into subnets) in a column that does not contain a terminal of net 2.

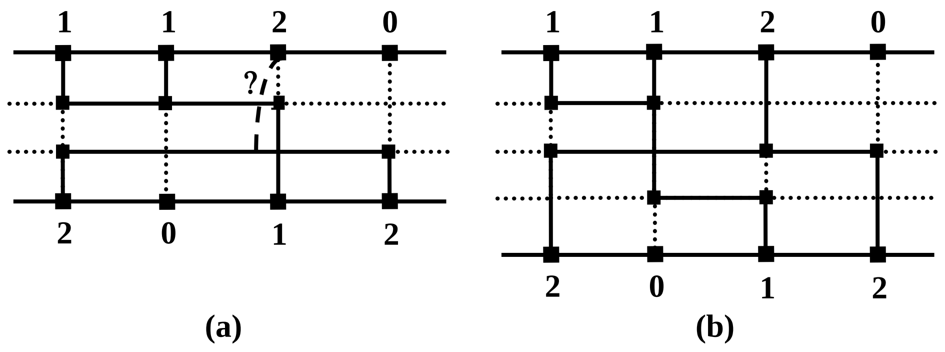


Figure 1.15: **(a)** A routing instance (of a channel) that has no no-dogleg (two-layer) feasible routing solution in the reserved layer routing model. The "?" mark indicates that the third column of the channel is already occupied by the vertical wire segment of net 1 that has been assigned to the top track, and the vertical wire segment of net 2 that has been assigned to the bottom track, is not assignable to this column (to avoid short-circuit). **(b)** A feasible dogleg routing solution of the channel instance, where the horizontal wire segment of net 1 is split and assigned to different tracks (tracks 1 and 3) and the horizontal wire segment of net 2 is assigned to track 2; as a result, more vias are needed.

Often some routing instances may not be routable using only no-dogleg routes in some specified routing model but can be routed using doglegging. One such situation is shown in Figure 1.15. In general, a routing instance may use less number of tracks when the nets present in it are allowed to dogleg. A channel instance with solutions in these two models illustrating this fact is shown in Figure 1.16.

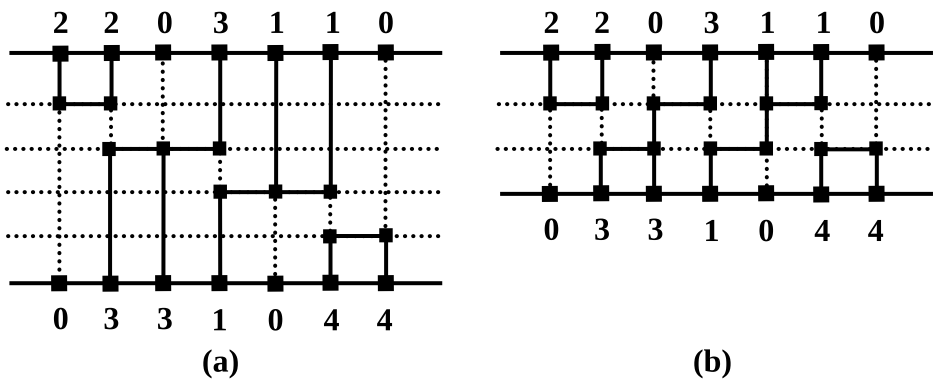


Figure 1.16: **(a)** A no-dogleg routing solution of a channel instance (that requires four tracks, means more area). **(b)** A doglegged routing solution of the same channel instance using only two tracks means less routing area (sacrificing more vias).

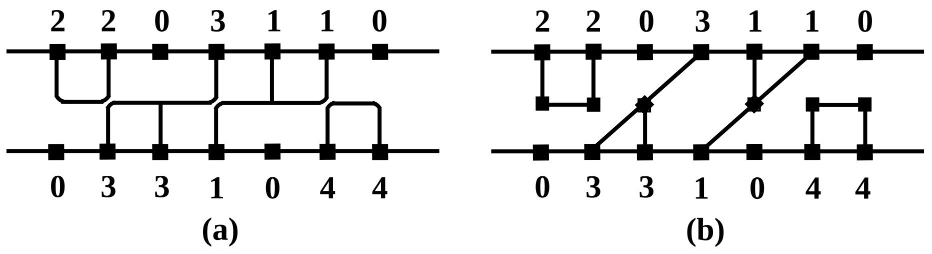


Figure 1.17: **(a)** A routing solution in the knock-knee routing model where a grid point is shared by two nets assigned to various layers of interconnect. **(b)** A solution of the same channel in the Manhattan-diagonal routing model. Vias used in diagonal routing are specially designed; usually, the vias are fabricated in an octagonal shape in the 45°-135° diagonal routing model.

5. The Manhattan versus Knock-knee and Diagonal Routing Models

Knock-knee is usually an unreserved layer routing model that allows two nets to share a grid point if they are in different (adjacent) layers of interconnect. This model may introduce an unavoidable undesired electrical property such as coupling capacitance, caused due to bending of two different nets that overlap and share a grid point. Figure 1.17(a) shows a routing solution of a channel instance shown in Figure 1.16, in the knock-knee routing model that uses only one track to route this channel.

On the other hand, unlike knock-knee, if a grid point is not shared by two different nets for their change of direction, but rectilinear wire segments are allowed to follow grid lines only, the routing model is known as *Manhattan routing model*. All solutions in Figure 1.16 follow the (reserved two-layer) Manhattan routing model. *Diagonal routing model* is the most advanced routing model where wire segments may follow diagonal routes, and often along with Manhattan routing. A routing solution, which allows wire segments to route in rectilinear as well as in diagonal direction, is known as a solution in the *Manhattan-diagonal routing model*. Figure 1.17(b) shows a routing solution for the channel instance shown in Figure 1.16 (as well as Figure 1.17(a)) in the Manhattan-diagonal routing model that also requires only one track to route this channel.

Even then there are other routing models where we like to know the number of interconnecting layers, whether this is two-layer, three-layer, or multi-layer, to route a channel. To route a maximum number of channel instances, only two reserved layers are sufficient in achieving a feasible routing solution. Only for some special cases of channel specifications, computation of a feasible two-layer routing solution is not possible in some models of routing under consideration.

In any case, amongst all these routing models discussed above, there are several trade-offs/issues to select a particular set of models as the most satisfactory rational routing model, as each of such models has their individual advantages and limitations. For example, an unreserved layer overlap routing model may require less routing area, but the performance in the reserved layer non-overlap routing model is much better. Routing algorithms devised for the non-overlap reserved layer routing model as well as the fabrication processes are modular in nature. On the other hand, doglegging may draw designers' attention as these routing solutions are usually area efficient. However, all these routing solutions surely use more vias that increase the fabrication cost, and these solutions are more hazardous too from an electrical performance point of view. Often a routing instance might be most efficiently routable only in the diagonal routing model, but in such a model either octagonal shaped or similar vias are needed to be designed which is complicated and costlier than square vias used in connecting rectilinear wire segments.

So, merits and demerits are there for each of the routing models talked about. Anyways, based on the realistic scenarios and fabrications viewpoint, the most practical, modular, and performance driven routing model is the *grid-based non-overlap reserved layer no-dogleg Manhattan routing model*, in which the majority of the designers, academicians, and industry people have given their attention and shown their curiosity, and published a maximum number of articles on routing [15, 32, 33, 49, 51, 71, 96]. In this thesis too, we like to concentrate on the stated routing model in developing all our theoretical contributions and designing algorithms for crosstalk minimization in two-layer VH channel routing.

1.4.3 Characterization of the Channel Routing Problem

We consider an example channel specification as shown in Figure 1.18. This channel contains eight nets, whose terminals are to be connected to complete the required interconnection for computing a routing solution in the grid-based reserved two-layer no-dogleg Manhattan routing model.

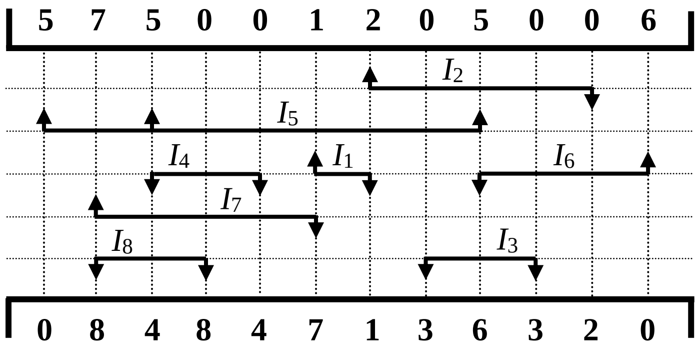


Figure 1.18: An example channel instance, that contains eight nets, is considered for characterizing the channel routing problem.

We know that the primary objective of the channel routing problem is to connect all the terminals using a minimum number of tracks (or channel area), where non-terminals are vacant terminals and not connected. Now, to minimize the number of tracks, of course, we require assigning the horizontal spans of all nets (belonging to a channel) to a minimum number of tracks. This can be done in several possible ways for the instance shown in Figure 1.18. Observe that the nets 4 and 7 can never be assigned to the same track. The reason is that these intervals have horizontal spans that overlap with each other and there is only one (reserved) horizontal layer for assigning all horizontal wire segments in the VH routing model; otherwise, these two spans are to be short-circuited in a track.

Such a constraint between a pair of nets for their assignment to tracks is known as the *horizontal constraint*, and all horizontal constraints in a given instance can be represented by an undirected graph, called the *horizontal constraint graph* (*HCG*). This graph can be constructed as follows. For every net, there is a vertex in the graph. There is an edge between two vertices of the graph if and only if the corresponding intervals have overlapping horizontal spans [49, 96]. Sometimes, this overlapping could be on a single point in a column, where the interval of a net terminates whereas that of another net originates. For example, this is true for the net-pair 1 and 7 in the channel instance shown in Figure 1.18. The HCG thus obtained for the channel instance in Figure 1.18 is shown in Figure 1.19.

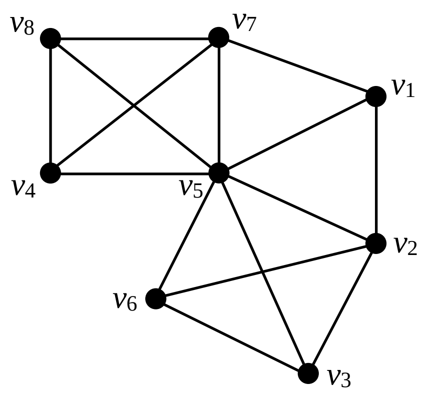


Figure 1.19: The horizontal constraint graph (HCG) of the channel specification shown in Figure 1.18.

Now the question that arises: *Whether the interval (or the horizontal span) of net 6 (i.e. I_6) is assignable to a track above the track to which net 5 (i.e. I_5) to be assigned?* Noticeably, the answer is 'no'; this is because the ninth column of the channel contains a top terminal of net 5 and a bottom terminal of net 6. So, if the interval I_6 is assigned to a track above the track where the interval I_5 is assigned, then it results in an infeasible assignment as shown in Figure 1.20(a). In fact, to get a valid routing solution in the said model at least one track separation is necessary while assigning the intervals to tracks and I_6 is always assigned below I_5, as shown in Figure 1.20(b). This forced constraint in the order of assigning nets from top to bottom along the height of the channel is known as a *vertical constraint* [49, 96]. Here, we say that net 5 is vertically constrained to net 6.

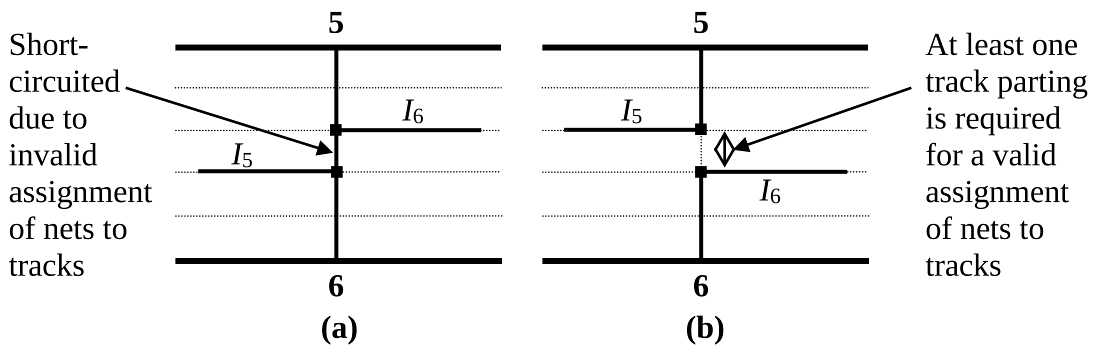

Short-circuited due to invalid assignment of nets to tracks
5
6
I_6
I_5
(a)
5
6
I_5
I_6
(b)
At least one track parting is required for a valid assignment of nets to tracks

Figure 1.20: (a) An infeasible allocation of the intervals of two different nets to tracks due to the presence of a vertical constraint in the ninth column of the channel in Figure 1.18, and as a result the vertical wire segments of the nets get short-circuited. **(b)** A feasible allocation of the intervals of two different nets to tracks where the interval of a net with the top terminal is assigned to at least one track above the interval of a net with the bottom terminal to conform to the vertical constraint present in the column.

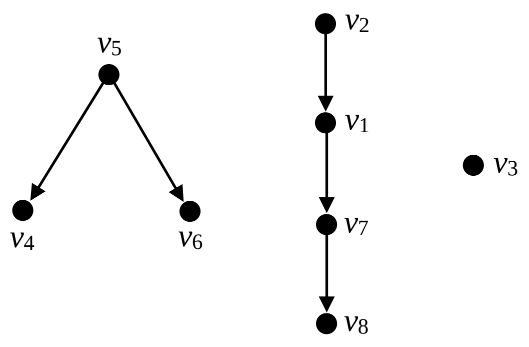


Figure 1.21: The vertical constraint graph of the channel specification shown in Figure 1.18.

The vertical constraints among the nets in a channel are represented by a directed graph, called the *vertical constrains graph* (*VCG*) defined as follows. For every net, we introduce a vertex, and there is a directed edge from v_i to v_j if and only if the corresponding net n_i is vertically constrained to net n_j. The VCG of the channel instance in Figure 1.18, is shown in Figure 1.21.

The horizontal and vertical constraints are two important characterizations of a channel routing instance. The horizontal constraints determine whether two intervals

I_i and I_j of two different nets n_i and n_j, respectively, are assignable to the same track. The vertical constraints determine the order in which the intervals are to be assigned from top to bottom across the height of the channel. An undirected edge $\{v_i, v_j\}$ in the HCG indicates that the corresponding intervals I_i and I_j are not assignable to the same track in routing a solution under the (grid-based) reserved two-layer (VH) (no-dogleg) Manhattan channel routing model. The *channel density* (or only *density*) of a channel is the maximum number of nets passing through a column. Let us denote the channel density by d_{max}. For the channel specification in Figure 1.18, the channel density, d_{max} is four, because at most four nets ($\langle n_5, n_4, n_7, n_8\rangle$ or $\langle n_2, n_5, n_6, n_3\rangle$) are involved in horizontal constraints in a column.

On the other hand, a directed edge (v_i, v_j) in the VCG indicates that the net n_i has to connect a top terminal and the net n_j has to connect a bottom terminal at the same column position. Therefore, the interval I_i must be assigned to a track above the one to which the interval I_j is assigned. For an acyclic VCG, let us denote the length of the longest path in the VCG by v_{max}, where v_{max} is same as the number of vertices belonging to the path. Thus, for the channel instance shown in Figure 1.18, the length of the longest path in the (acyclic) VCG, $v_{max} = 4$, comprising vertices $\langle v_2, v_1, v_7, v_8\rangle$.

1.4.4 A Lower Bound on the Number of Tracks

Typically, there are various assumptions as well as objectives in solving a given channel instance. A routing model itself is a supposition that guides to the other beliefs as well; some of which are briefly mentioned here. As the input of a channel is given, that means the top row of terminals, as well as the bottom row of terminals, is given that are equal in length. Moreover, this implies that the channel length and hence the number of columns are also given, which is fixed for a given channel specification.

A grid is assumed in a grid-based routing model, where rectilinear wire segments are allowed to route; terminals are vertically aligned along the columns. Among several mutually conflicting objectives, the prime objective is to minimize the channel height, which is reflected by minimizing the number of tracks or the channel area.

Hence, we may define the channel routing problem as follows. The *channel routing problem* (*CRP*) is the problem of assigning the horizontal wire segments of a given set of nets (in the form of *netlist* or *channel specification*) to tracks without any conflict (or satisfying constraints), so that the number of tracks required, and thus the channel area, is minimized.

Note that we need at least d_{max} tracks, as well as at least v_{max} tracks to compute a two-layer routing solution under the reserved layer (no-dogleg) non-overlap Manhattan routing model when the channel consisting of a set of fixed terminals and the VCG for the channel is acyclic. Besides, when the VCG of a channel contains a cycle, then there is no question of computing the length of the longest path in the VCG, i.e. v_{max}. In such a case, a net that is belonging to a cycle in the VCG is supposed to be assigned above (or below) to itself, which is meaningless (or ambiguous). Therefore, for an acyclic VCG, $max(d_{max}, v_{max})$ is a lower bound on the number of tracks required for a routing solution in the two-layer VH channel routing model [49, 96]. This is because any two nets belonging to the set of nets in d_{max} must be assigned to two different tracks; the same is true for the set of nets in v_{max}.

We may further make a note of that the HCG of a channel instance is an *interval graph*, which is a kind of perfect graph [8, 29, 34, 36, 70]. Further, it can be easily shown that d_{max}, the density of the channel, is same as the size of the maximum clique of the underlined interval graph [8, 29, 34, 36, 49, 70, 96].

1.4.5 Optimization Issues Involving the Channel Routing Problem

So far, we have viewed the channel routing problem in an abstract framework. Based on this abstraction, there are some usual objectives that we would like to achieve. Some of the *usual factors* (or *objectives*) to optimize the cost of a routing solution are *area minimization*, *wire length minimization*, *via minimization*, *layer minimization*, and so on. All these cost factors are inter-related, and we need to assign priorities on them that can reflect a real situation. Here, we have discussed these factors in brief.

1. **Area:** A primary objective in channel routing is to minimize the total routing area of a channel. Minimization of routing area (space) for each of them eventually, reduces the overall (global) routing area. If routing area is minimized, the cost of production is reduced, and yield is enhanced, as impurities (or defects in the wafer) could be avoided in realizing the same circuit which is smaller in size; and out of the same wafer, more chips are also produced.

2. **Net wire length:** Wire length is another important cost factor, as more wire means more cost. Also, a long wire is responsible for signal propagation delay. For some net, often long wires could be there as a part of design, but for critical nets and nets for power and ground lines, long wires are not allowable at all. Later on, we will find that long wires are hazardous from an electrical point of view as well. In VLSI channel routing, the total wire length minimization is an important problem, and this problem is much more important when the objective is to minimize the longest net.

3. **Via:** A via is introduced to connect two orthogonal wire segments of a net or in a case where the necessity of a change of layer is a must. In a grid-based routing model, a via is placed at the corresponding grid point. Often in dogleg routing, the area is reduced sacrificing more vias. In CRP, vias are minimized to improve the alignment of masks. Besides, vias are often difficult to fabricate. Vias also increase delay and electrical hazards, and therefore, need to be reduced in high performance designs.

4. **Layer:** Increase in the number of layers causes minimization of routing area. However, this increases the size of the chip in the third dimension and the cost of fabrication. Thus, one of the important objectives is not to introduce more layers, if a routing solution is achieved within a specified area using a minimum number of layers of interconnect.

For general-purpose chips, the above objectives are attempted to minimize (or optimize) in isolation or sometimes, in combination. However, there are several high performance issues to make a routing solution reliable and practicable, even if the routing solution is feasible. For designing *high performance chips*, there are some supplementary objectives such as reducing *crosstalk*, *signal delay*, *power consumption*, *heat generation*, *hot spot formation*, and so on and so forth. If we could reduce all these factors in computing a routing solution, then a channel routing solution with high performance is obtained. Some of these are discussed in brief as follows.

1. **Crosstalk:** Crosstalk is one form of noise and is a kind of electrical hazard, which is created due to the mutual capacitance between adjoining wires. Propagation delay increases and logic faults may occur because of increase in

crosstalk. Crosstalk between wires is proportional to the coupling capacitance which in turn is proportional to the coupling length, i.e. the total length of the overlap between (adjacent) wires. Crosstalk is also proportional to the frequency of operation and inversely proportional to the separating distance between wires. Therefore, for high performance routing, it is required to consider all these issues, and through minimization of crosstalk, these could all be reduced below a real limiting value. The aim should be to avoid overlapping wire segments that lie close to each other.

2. **Signal delay:** Delay is another important criterion for high performance routing. In VLSI design, often it is possible that a critical path is cut many times by the partition [81], and thus, the delay in the path may be too long to meet the goals of high performance requirements. The design for high performance routing requires intelligent partitioning algorithms to reduce the cut size (that signifies interconnecting wires) as well as to minimize the delay in the critical path. Often the delay is a consequence of a result of electrical hazards (in the form of crosstalk) as has been mentioned above.
3. **Power consumption:** In high performance routing, the power consumption by any net should be controlled; otherwise, there will be a mismatch of consumed power between nets. Thus, even distribution of power is an important issue in high performance routing. On the contrary, a spot over the chip could become more heated and may get melted. Thus, the proper cooling arrangement is also required for surviving of the chip. This problem is related to wire length minimization and even distribution of wires in routing. Nowadays, low power is one of the most important research domains in devising VLSI chips or any portable electronic gadgets.
4. **Hot spot:** For high performance routing, there should never be a part of a channel which is highly congested. As a matter of fact, the propagation of signals may generate heat that could cross a specified (upper) limit. The spot with crossing heat limit is called a *hot spot*. Congestion of active components on a chip floor and wires in routing, power consumption, and hot spot formation, these are all interlinked. To remove hot spots, rerouting is an alternative way of probable solutions; otherwise, if rerouting fails to reduce congestion (of wires), the adequate cooling arrangement is necessary.

In this thesis, we study the crosstalk minimization problem in two-layer (VH) channel routing, in the routing models specified above, where we assume a pre-computed channel routing solution with reduced area. In other words, our objective is to reduce crosstalk in a given channel routing solution (with a fixed number of tracks or channel area) by computing another routing solution (with reduced crosstalk) keeping the channel area same. In more details, the chapter-wise contribution of the thesis is briefly mentioned in the following section.

1.5 Outline of the Thesis

As fabrication technology advances the feature size reduces. The devices are placed closer to each other, and the interconnecting wire segments are assigned narrower pitches. However, the circuits' operations are realized at higher frequencies. As a result, electrical hazards, viz., *crosstalk* between wire segments are produced. More crosstalk means more noise and more signal delay, and hence, a reduced circuit performance. Therefore, it is desirable to develop channel routing algorithms that not only reduce the number of tracks (i.e. channel area) but also crosstalk. The theoretical aspects of the computational problems along with the presence of crosstalk as an inherent electrical phenomenon in network-on-chip (NoC) and also in mixed-signal ICs in deep submicron digital CMOS technology introduced for devising VLSI circuits have been briefly discussed and reviewed in Chapter 2 of the thesis.

In Chapter 3, we study several crosstalk minimization problems in two-layer VH channel routing in formal frameworks. We first introduce sum *crosstalk minimization problem*. The *sum crosstalk* is the amount of total crosstalk between horizontal wire segments of the nets that are assigned to adjacent tracks in a channel routing solution. Then we introduce *bottleneck crosstalk minimization problem*. The *bottleneck crosstalk* with respect to a feasible routing solution is the maximum amount of crosstalk due to overlapping between any pair of adjacent horizontal wire segments of two different nets. We show that these problems are intractable. This is true irrespective of whether non-overlapping nets are grouped for their assignment to tracks or not. In this chapter, we also address if there is a polynomial time approximation algorithm with guaranteed error bound for the crosstalk minimization problems, and subsequently prove that if P $\neq$ NP, there cannot exist such an approximation algorithm. It is further shown that all these problems are also NP-hard even if doglegging is allowed.

Hardness results obtained in Chapter 3 leaves one option to deal with these problems, design of efficient heuristics for them. In Chapter 4, we develop crosstalk minimization heuristics for the two-layer channel routing, where the channel instances are simple. Then we extend it to two-layer routing for *general* channel instances. For each such instance, two algorithms have been developed; one is *Track Interchange*, and the other one is *Net Change*. The algorithms are efficient enough in reducing crosstalk from 21% up to 36% for simple channel instances and on an average of 12% for general channel instances.

Chapter 5 deals with designing algorithms for the generation of random channel instances, for their use in computing reduced crosstalk channel routing solutions in VLSI physical design. Channel instances are usually of two types: simple and general. A *simple channel instance* does not contain any vertical constraint, whereas a *general channel instance* contains both horizontal as well as vertical constraints. The novelty of the heuristics designed in Chapter 4 can be judged better if it works for a variety of a large number of randomly generated instances of the problem. In this chapter, we develop two random channel instance generators that vary from 14 to more than 33000 in channel length and from 10 to 15000 in the number of nets belonging to a channel. For each net number, we generate 200 simple as well as general random channel instances, with a varying number of terminals per net, and use each of them for reducing crosstalk.

An exhaustive amount of computations has been performed in this research work to support all the algorithms developed in this thesis in Chapters 4 and 5. All results obtained have been included in Chapter 6 as part of experimentation relating to our work. A limited number of sample channel instances have been incorporated in this chapter though we have generated thousands of such instances of varying length. A selected set of routing solutions have also been presented in this chapter where we show an initial routing solution, a reduced crosstalk routing solution after the execution of the algorithm *Track Interchange*, and then a further reduced crosstalk routing solution after the execution of the algorithm *Net Change*. All our experimental results are highly encouraging.

In Chapter 7, we show that the algorithms *Track Interchange* and *Net Change* can be parallelised to get efficient parallel algorithms for simple instances of channel specifications. The *Track Interchange* algorithm uses a sorting algorithm which can also be easily parallelized. This readily gives us a parallel version of algorithm *Track Interchange*. Whereas, in the second algorithm, a processor, which is responsible for a net, searches for a blank space in some other track, if the net is interchangeable, such that this interchange reduces a maximum amount of crosstalk. The algorithms are efficient enough in terms of their computational complexity.

The thesis is concluded in Chapter 8 with some probable open problems for future researchers. In this thesis, several issues relating to two-layer crosstalk minimization problem in channel routing have been considered and resolved. Even then, there are several other issues yet to be studied. Some of them are developing algorithms for computing two-layer channel routing solutions with better (or further reduced) bottleneck crosstalk without sacrificing area, algorithms for reduced crosstalk in two-layer dogleg routing solutions, and parallel algorithms for reducing crosstalk for general channel instances. The crosstalk minimization problem in the three-layer VHV and HVH channel routing models are yet open for potential investigators.

Chapter: 2

Literature Survey on Crosstalk Minimization Problem

2.1 Overview

In this chapter, we briefly survey a few results on channel routing that are relevant to the work presented in this thesis. We view channel routing as a combinatorial problem and survey some fundamental intractability results for the area as well as wire length minimization in channel routing. Since the CRP is NP-hard for both the problems mentioned above, extensive research has been done on designing efficient algorithms for meeting some desired target as usual cost factor(s) to be optimized for routing channels. By the way, mainly in the last two decades, some high performance factors have been addressed in a discrete way, but plenty of work is yet to be done in this domain of research. Only very few researchers have worked on avoiding or minimizing crosstalk and among them almost none on channel routing as a part of high performance VLSI design for circuit synthesis.

In this chapter, we have reviewed the effects and impact that crosstalk has on the performance and reliability of VLSI circuits and systems. With developments in VLSI fabrication technology, as feature size decreases and communicating wires are seated as close as possible, circuits also operate at higher frequencies. Thus, reduction in crosstalk among interconnecting wire segments is becoming an essential concern for VLSI physical design. In this chapter, we have also offered a short and snappy but informative survey of the various methods that researchers worldwide are implementing for a priori crosstalk avoidance or a posteriori crosstalk minimization in VLSI systems from the point of view of fabrication over the past few decades.

In Section 2.2, we converse some basic terms and preliminaries pertinent to the discussion on computational complexity as given in [28, 66]. In this section, we also address some related problems on two-layer (VH) area and wire length minimization in channel routing, either that are polynomial time computable or NP-hard. In Section 2.3, we present a concise survey of the development of evading and/or diminishing crosstalk in several spheres of research including network-on-chip, inbuilt evasion in fabrication, optical network, nanometer design, circuits and communication systems, gate sizing, signal transition, interconnect spacing in analog and digital circuits, frequency domain, error control coding and bus-coding, routing in VLSI domain, and so on and so forth. Some soft computing based methods have also been adopted as has been included in this review. In Section 2.4, we summarize the chapter by making some comments on crosstalk minimization along with mentioning a few established intractable results in two-, three-, and multi-layer channel routing.

2.2 Some Basic Terms and Preliminaries on the Theory of NP-Completeness

A *decision problem* Π is a problem stated in terms of generic parameters for which the solution is either "yes" or "no". That means either the guessed solution to a problem under consideration is a valid solution or the other, but we can check it in polynomial time. In general, all decision problems that can be solved in polynomial time by a nondeterministic computer belong to the class *NP*. Most of the apparently intractable problems encountered in practice, when phrased as decision problems, belong to this class [28, 66].

An instance I_Π of problem Π is obtained by assigning a set of specific values to the parameters of Π. The set of all (valid) instances of Π is denoted by D_Π, whereas $Y_\Pi \in D_\Pi$ is the set of all instances of Π for which the solution is "yes". *P* is the class of all decision problems each of which is polynomial time computable by a *Deterministic Turing Machine* (*DTM*). DTMs are related in polynomial time to the high-level programming languages. Therefore, if an algorithm can be devised for solving Π whose computational space as well as time complexity is polynomial, then $\Pi \in P$. For example, let us consider the following decision problem Π1.

Problem: Two-layer (VH) restricted dogleg channel routing problem of area minimization with multi-terminal nets in the absence of vertical constraints (Π1).

Instance: A channel specification of length *m* (i.e. the length of TOP or BOTTOM sequence of fixed terminals) that contains $n \leq m$ multi-terminal nets such that either the top terminal or the bottom terminal (or both) in each column is a non-terminal (or 0, not to be connected), and a number *t* of tracks between TOP and BOTTOM.

Question: Is there a permissible assignment of the wiring of all nets in the channel in the two-layer (VH) restricted dogleg Manhattan channel routing model that uses no more than *t* tracks?

To judge the nature of the problem Π1, now we consider the following decision problem Π2.

Problem: Two-layer (VH) no-dogleg channel routing problem of area minimization with two-terminal nets in the absence of vertical constraints (Π2).

Instance: A channel specification of length *m* (i.e. the length of TOP or BOTTOM sequence of fixed terminals) that contains $n \leq m$ two-terminal nets such that either the top terminal or the bottom terminal (or both) in each column is a non-terminal (or 0, not to be connected), and a number *t* of tracks between TOP and BOTTOM.

Question: Is there a permissible assignment of the wiring of all nets in the channel in the two-layer (VH) no-dogleg Manhattan channel routing model that uses no more than *t* tracks?

Since there are no vertical constraints for channel instances in this domain of channel routing problem, polynomial time algorithms exist for finding a legal wiring using no more than density number of tracks for the problem Π2 [49, 52, 53]. Thus, one needs only to compare the density of the channel and *t* to determine if a specific instance $I_{\Pi 2} \in Y_{\Pi 2}$. Hence, we conclude by saying that $\Pi 2 \in P$.

Now, it is interesting to observe that if Π1 contains only two-terminal nets, then such an instance of Π1 becomes an instance of Π2 only, as allowing restricted doglegging in the case of multi-terminal nets is analogous to no-doglegging if the number of terminals per net is limited to exactly two. Therefore, we conclude that Π1 also belongs to *P* [49, 52, 53].

Incidentally, for some problem Π, there may not be any (optimal) polynomial time computable algorithm. However, if a guessed solution S_{Π} for some instance I_{Π} (for Π) is given, the guess can be checked in polynomial time to verify if it corresponds to a "yes" or "no" solution in a reasonable amount of time (or in polynomial time) for Π. This is comparable by saying that there exists a polynomial time *Non-Deterministic Turing Machine* (*NDTM*) for solving Π. *NP* is defined to be the class of decision problems each of which can be solved in polynomial time by an

NDTM. It is known that $P \subseteq NP$ [28]; that means a problem which is deterministically polynomial time computable is also non-deterministically polynomial time computable, but the reverse is not true. Thus, we may assert in some other words that when a problem is a polynomial time computable one, then a polynomial time computable non-deterministic algorithm (using NDTM) is there which is more powerful to solve the problem than a polynomial time computable deterministic algorithm (using DTM) to solve the same. However, it is yet to be proved whether $P = NP$ or $P \neq NP$ [28].

Consider the following problem Π3 [41].

Problem: Two-layer (VH) no-dogleg channel routing problem of minimizing area with two-terminal nets (Π3).

Instance: A general channel specification of length *m* (i.e. the length of TOP or BOTTOM sequence of fixed terminals) that contains $n \leq m$ two-terminal nets, and a number *t* of tracks between TOP and BOTTOM.

Question: Is there a permissible assignment of the wiring of all nets in the channel in the two-layer (VH) no-dogleg Manhattan channel routing model that uses no more than *t* tracks?

Π3 is an example of a problem for which there is no known polynomial time algorithm that computes an optimal solution for each and every instance of this problem. However, given a set of routes in a guessed solution $S_{\Pi 3}$ for all the nets of an arbitrary instance $I_{\Pi 3}$ (for Π3), we can check in polynomial time whether all these routes provide feasible (or valid) wiring for the channel. Moreover, the number of tracks present in $S_{\Pi 3}$ can also be compared to *t*. Thus, in polynomial time, a check can be made to verify whether the guessed solution $S_{\Pi 3}$ for the (assumed) random instance $I_{\Pi 3} \in Y_{\Pi 3}$. This implies that $\Pi 3 \in NP$.

In the theory of NP-completeness, there is always a pair of problems that is reducible (or transformable) to each other in polynomial time. In some other words, if a polynomial transformation is applied to an instance of the first problem, a matching instance of the second problem is produced, and if a polynomial transformation is imposed on an instance of the second problem, an analogous instance of the first problem is obtained. Let us consider two decision problems Πx and Πy. A *polynomial transformation* from Πx to Πy is a function $f : D_{\Pi x} \rightarrow D_{\Pi y}$ such that

1. f has a polynomial time computable algorithm, and
2. $I_{\Pi x} \in Y_{\Pi x}$ if and only if $f(I_{\Pi x}) \in Y_{\Pi y}$.

If such a function f exists, then Πx is polynomial time transformable (or reducible) to Πy, which is denoted by $\Pi x \propto \Pi y$. Such a transformation shows (and proves) that Πy must be as hard as Πx. Πx and Πy are *polynomially equivalent* if $\Pi x \propto \Pi y$ and also if $\Pi y \propto \Pi x$. A decision problem Π is *NP-complete* if $\Pi \in NP$ and for every problem $\Sigma \in NP$, $\Sigma \propto \Pi$. NP-complete problems form a class of polynomially equivalent problems such that if one problem in this class has an optimal polynomial time computable algorithm for its solution, then all problems in the class do have the same (means each of the remaining problems also has a polynomial time computable algorithm for its respective optimal solution) [28, 66].

LaPaugh proved that the *circular arc colouring* (*CAC*) problem is transferable to $\Pi3$ in polynomial time [41, 49]. The decision version of the circular arc colouring problem is as follows [28, 49].

Problem: Circular arc colouring (*CAC*).

Instance: A finite set of arcs of a circle and a positive integer k.

Question: Is there an assignment of colours numbered 1 through k to the arcs such that any two arcs that overlap are assigned different colours?

CAC is a well-known NP-complete problem [28]. Therefore, $\Pi3$ is also NP-complete [41, 49].

Now suppose that Πp is a new problem that belongs to *NP*. Also, let Πq is a known NP-complete problem. Moreover, say all instances of Πp can be restricted in the fashion that makes the restricted version $\Pi p'$, exactly the same problem as Πq. Then solving Πp must be as hard as solving Πq. Thus, Πp must also be an NP-complete problem. This technique of proving a (new) problem NP-complete is called *proof by restriction* [28]. Consider the following problem $\Pi4$.

Problem: Two-layer (VH) restricted dogleg channel routing problem of minimizing area with multi-terminal nets ($\Pi4$).

Instance: A general channel specification of length m (i.e. the length of TOP or BOTTOM sequence of fixed terminals) that contains $n \leq m$ multi-terminal nets, and a number t of tracks between TOP and BOTTOM.

Question: Is there a permissible assignment of the wiring of all nets in the channel in the two-layer (VH) restricted dogleg Manhattan channel routing model that uses no more than t tracks?

It is easy to see that Π4 reduces to Π3 by restricting the number of terminals for each net belonging to the channel to two [41]. Therefore, Π4 is NP-complete.

Now we like to mention another problem Π5 for minimizing wire length in two-layer channel routing. The problem is as follows.

Problem: Total wire length minimization in two-layer (VH) no-dogleg channel routing (Π5).

Instance: A general channel specification of multi-terminal nets and a positive integer k.

Question: Does there exist a two-layer (VH) no-dogleg routing solution in the Manhattan channel routing model so that the total wire length is k or less?

In [49, 57, 59], it has been proved that Π5 is an NP-complete problem, and for proving the same the necessary polynomial time reduction is made from an instance of the well-known NP-complete problem *sequencing to minimize weighted completion time* (*SEQ*) [28] to it. Before we declare the sequencing problem, we necessitate explicating a few notations. T is a set of n tasks $t_1, t_2, \ldots, t_n$. For every task t_i, $l(t_i)$ is the duration for which the task t_i runs, $w(t_i)$ is the weight of task t_i, and $\sigma(t_i)$ is the instant of time when the task t_i is sequenced.

Problem: *Sequencing to minimize weighted completion time* (*SEQ*).

Instance: A set $T = \{t_1, t_2, \ldots, t_n\}$ of tasks, a set of partial orders $<$ on T, for each task $t_i \in T$ a duration $l(t_i) \in Z^+$, a weight $w(t_i) \in Z^+$, and a positive integer k.

Question: Does there exist a one processor schedule σ for T that obeys the precedence constraints and for which $\sum_i (\sigma(t_i) + l(t_i)) \cdot w(t_i)$, for $1 \leq i \leq n$, is at most same as k?

2.3 A Review on Crosstalk Avoidance and Minimization

With advancements in fabrication technology, there is a constant reduction in minimum feature size in VLSI chips [2, 14, 88]. Subsequently, interconnecting wires are also being placed in closer proximity to minimize interconnection delay and the real estate required for routing. With the decrease in circuit delay, circuits start operating at higher frequencies. Thus, reduction in crosstalk becomes an important consideration for VLSI physical design. Crosstalk mainly arises due to the coupling capacitances between adjacent interconnecting wires. The crosstalk arising in the course of two wires is found to be proportional to the coupling length between them; the capacitance, in turn, is determined by the relative positions of the wires. The length of the overlapping segments of any two parallel interconnecting wires determines the coupling capacitance between them. The coupling capacitance is also inversely proportional to the distance separating two parallel wires. The coupling capacitance between a pair of orthogonal wires is negligible compared to that between a pair of parallel wires and is thus reasonably assumed to be non-existent. The frequency of the signals travelling through two parallel wires also affects the coupling crosstalk arising between them.

Crosstalk results in noise which may lead to unexpected circuit behaviour. The absolute limit of crosstalk that can be tolerated depends on two things: the power driving the circuit and the sensitivity of the circuit. In order to reduce noise in any particular design, it thus becomes imperative to minimize the total crosstalk. In this chapter, we present a review of various crosstalk avoidance and minimization techniques that researchers have proposed over the past decades. While some of them try to incorporate changes in the design phase to avoid the generation of crosstalk, others attempt to modify the signal using different schemes so as to minimize the crosstalk arising due to the signals.

2.3.1 Crosstalk in IC based Environment

2.3.1.1 Simulated Annealing based Approach [76]

In [76], the authors have devised a simulated annealing (SA)-based high-level synthesis algorithm for the purpose of minimization of crosstalk activity in any given data environment. They have focused on bus-based architectures as bus-lines usually have a well-defined neighbourhood. The main objective of the authors was to minimize the crosstalk in the worst case scenario concerning signal transmission pattern. In order to achieve this, they have also incorporated bus reordering and data transfer invert encoding schemes in addition to synthesis moves.

Experimental results suggest that there is a possibility of significant crosstalk reduction considering both resource and latency constraints. The devised framework has shown an average improvement of 23.5% over non-optimized designs. The authors have also claimed to have achieved up to 75% reduction in bus-lines without requiring shielding in the case of a set of nine DSP benchmarks. Furthermore, they have presented an idea for a circuit for the purpose of detection and elimination of on-chip crosstalk.

2.3.1.2 Signal Transformation Avoidance Technique [87]

The problem of crosstalk manifests itself in integrated circuit design whenever overlapping stretches of wires are present. This is due to the parasitic coupling induced between the adjacent signal conducting wires. It has been intended in [87] to find a solution to this problem using a signal transformation avoidance technique. The authors have used simulated annealing to search for an optimal layout pattern that has the minimum crosstalk. They have modelled the crosstalk involved as a function of energy stored in capacitances, intending to reduce further the crosstalk by rearranging wire signal transition. The proposed framework has been able to bring about a reduction of about 24.4% in energy for the optimal result obtained and about 6.9% for the average result.

2.3.2 In-built Crosstalk Avoidance in Fabrication

2.3.2.1 An Alternative Layout Scheme for Crosstalk Avoidance [45]

Advancements in fabrication technology have steered devices to enter the nanometer regime. This has led to the design of several new logic and memory architectures. The objective of these architectures is to achieve higher packing densities keeping power consumption and delay within acceptable limits compared to contemporary CMOS architectures. The minimum wire spacing attainable and thereby the maximum packing density achievable are constrained by the crosstalk induced in such nano-scale devices.

In [45], the authors have analyzed the crosstalk which is produced in sub-lithographic programmable logic array architectures. They propose an alternative layout scheme with the intention of reducing the effects of crosstalk in adjacent wires. Their proposed framework makes use of an interleaved layout scheme whereby two out-of-phase but non-overlapping clocks prevent simultaneous signal transition in two neighbouring wires.

The results obtained by the authors suggest that their proposed framework provides better tolerance against the induced crosstalk compared to other sub-lithographic programmable logic array architectures (crosstalk of about 30% of V_{DD} compared to the worst case scenario of about 70% of V_{DD}). The authors have also analyzed the effect of different parasitic capacitances on the induced crosstalk.

2.3.2.2 Crosstalk Minimization in Optical Networks [72]

A nonlinear crosstalk minimization algorithm that simultaneously considers self-phase modulation (SPM), cross-phase modulation (XPM), and four-wave mixing (FWM) has been presented and experimentally assessed by the authors. For a passive optical network (PON), in the worst case scenario, a 1 dB power gain is reported by the authors for a bit error rate (BER) of 10^{-9}.

The authors have experimentally demonstrated the effectiveness of their proposed algorithm that is based on Volterra series and genetic algorithm (GA). The authors have reported improvements up to 1.8 dB in the Q factor, in the case of a WDM ring transporting 16×10 Gbps on-off keying non-return-to-zero codes through the standard single-mode fiber, compared to the non-optimized scenario.

2.3.2.3 Crosstalk Avoidance in Nanometer ICs [35]

Application of multi-bit flip-flops (MBFFs) to bring about a reduction in clock power in modern nanometer ICs is a promising lower-power design technique. Researchers have been trying to utilize more MBFFs with as the larger number of bits as possible for more clock power saving. However, an MBFF with a larger number of bits may lead to serious crosstalk problems due to the close spacing of interconnecting wires belonging to different signal nets connected to the same MBFF.

The authors have analyzed, evaluated, and compared the relationship between power consumption and crosstalk on the application of MBFFs with different numbers of bits. To solve the addressed problem, the authors have proposed a novel crosstalk-aware power optimization approach to optimize power consumption while satisfying the crosstalk constraint. Experimental results show that the proposed approach is very effective in crosstalk avoidance when applying MBFFs for power optimization.

2.3.2.4 Crosstalk Reduction in Fabrication Technologies [78]

The authors of this paper have proposed to reduce crosstalk for deep submicron (DSM) and very deep submicron (VDSM) technologies to increase reliability and performance. The proposed methodology aims to reduce the switching activity of the data buses by an efficient data encoding technique. The authors have suggested that their anticipated technique can reduce the total crosstalk delay by around 36% to 40% compared to unencoded data and 0.3% to 32% compared to others techniques for 12-bit, 21-bit, 38-bit, and 71-bit data buses.

2.3.3 Crosstalk Reduction in Circuits and Systems

2.3.3.1 Crosstalk Reduction in Communication Systems [92]

In the context of the design of integrated high-density circuits, crosstalk is one of the major concerns. A wireless communication system can deviate significantly from its intended performance, especially in the high-frequency range. It has been studied in [92] that the relationship between the orientation of and the crosstalk between dual stripline. The authors have analyzed crosstalk effects in the frequency as well as time domains and have concluded that the near-end and far-end crosstalk, respectively, increases and decreases based on a change in the orientation of the stripline from 0 to 90 degrees.

2.3.3.2 A Gate Sizing Technique for Crosstalk Minimization [30]

In [30], the authors suggest a novel gate sizing approach for circuit optimization in the presence of *scarce information* about the distributions of the process variations. The projected framework is said to rely on the concepts of utility theory and risk minimization for the multivariate optimization of parameters, namely, delay, dynamic power, leakage power, and crosstalk noise, via gate sizing.

The authors have compared the results of single-metric optimization for dynamic power, leakage power, and crosstalk noise with equally weighted multi-metric optimization and the results suggest that the proposed algorithm can achieve a multifold speedup in execution times compared to the traditional approaches. The authors have designed the algorithm to allow for selective optimization of the metrics subject to relevant design requirements. To achieve this, the authors have proposed assigning a high weight vector to the particular metric that needs to be optimized. In such cases, the authors have optimized the corresponding metric at the cost of sub-optimizing the other metrics.

2.3.3.3 Encoding Schemes for Reduction in Signal Transition in Crosstalk Minimization [89]

This paper recommends encoding schemes to achieve a reduction in transitions between the previous and present states of wire as well as transitions in adjacent wires. The reduction in transition improves the performance regarding power dissipation and crosstalk. The authors have classified groups of three wires into five types depending upon their respective nature of transitions of signals in the wire: Type-0, Type-1, Type-2, Type-3, and Type-4.

Type-0 coupling occurs when all of the 3-bit wires are in the same state transition, i.e. from 000 to 111 with coupling capacitance being zero in this case. Type-1 coupling occurs when there is a transition in one or two of the wires, and the authors have denoted the coupling capacitance in this case by C_C. A Type-2 coupling occurs when the central wire is in the same state transition with one adjacent wire and the opposite state transition with the other wire. The coupling capacitance assumed by the authors is $2C_C$ in this case. A Type-3 coupling occurs when the central wire undergoes the opposite state transition with one of the other wires while the remaining one remains quiet, the coupling capacitance, in this case, being $3C_C$. In Type-4 coupling, all three wire transitions in the opposite states with respect to each other with a coupling capacitance effect of $4C_C$.

The authors have achieved switching activity reduction of 34.84% and 36.23% using their proposed methodologies Method 1 and Method 2, respectively. The coupling activity reduction of Method 1 and Method 2 is 20.00% and 20.54%, respectively. The encoding methods proposed by the authors apparently reduce the worst crosstalk effects significantly and transforms them to lesser harmful Type-0 and Type-1 couplings. The authors have suggested that Method 1 reduces Type-4, Type-3, and Type-2 couplings by 100%, 77.4%, and 50.0%, respectively and Method 2 achieves a reduction in Type-4, Type-3, and Type-2 couplings by 100%, 77.4%, and 60.0%, respectively.

2.3.4 Crosstalk Reduction in Communication

2.3.4.1 Coding Scheme for Reduction of Signal Transition [77]

This paper proposes a technique which reduces power consumption due to crosstalk in data buses which are fed to DSP / communication devices. The proposed coding technique reduces the transition activity in the input signals. The authors have suggested that this shall consequently result in the reduction of power consumption. A new bus coding technique has been proposed by the authors to achieve less power reduction in transmission.

SPICE simulations have been carried out by the authors for interconnect lines of different dimensions at various technology nodes: 180, 130, 90, and 65 nm. The authors have claimed that their projected model achieves a reduction of 58.25% and 35.00% in the switching activity and the power consumption, respectively, for 16-bit data buses. The authors have studied higher length data buses as well and claim to have achieved a reduction of 45.62% and 27.00% in the switching activity and the power consumption, respectively, for 256-bit data buses.

2.3.4.2 A New Dielectric Structure to Reduce Crosstalk in ICs [46]

To overcome crosstalk noise and delay uncertainty in modern very large scale integration (VLSI) design, a new dielectric structure has been suggested in [46] for integrated circuits. Near- and far-end crosstalk noises are reduced 45.2% and 15.0%, respectively. The authors have named their proposed structure, gradually low-K. The authors also suggest that the structure exhibits negligible side-effects in terms of delay and power consumption.

According to their frequency domain simulations, the crosstalk measure has been reduced by 1 dB in the frequency range of 1–35 GHz for simulated structures with respect to the traditional low-K one. Moreover, according to the time domain simulations done by the authors, the proposed structure reduces the crosstalks: NEN (near-end noise) and FEN (far-end noise), by 45.2% and 15.0% in the test structure, respectively.

2.3.5 Crosstalk Avoidance in Network-on-Chip

2.3.5.1 Crosstalk Avoidance in Network-on-Chips (NoCs) [65]

Network-on-chip is evolving into a revolutionary method for the integration of multiple cores in a single system-on-chip (SoC). Widespread adoption of network-on-chip (NoC) paradigm will only be possible if the signal integrity issues that arise due to crosstalk between adjacent wires are properly addressed. Incorporation of crosstalk avoidance coding (CAC) in NoC data streams can bring about a reduction in the worst case coupling capacitance of inter-switch wire segments and subsequently, a reduction in energy dissipated. The energy savings depend on the distribution of inter-switch wires of different lengths and structure of the packet.

The authors have proposed a method for reducing the energy dissipation by eliminating the need for CAC coding/decoding of payload flits at intermediate switches between communicating NoC cores. It has been observed in [65] that the maximum energy savings have been achieved for the folded-torus architecture due to the uniformly distributed long inter-switch wire segments. The authors have also presented how their method of modifying the packet structure and reducing the coding/decoding overhead makes it possible to achieve higher savings in energy in conjunction with crosstalk avoidance.

2.3.5.2 Flow-Control in NoC for Avoiding Crosstalk [67]

The authors have proposed a flow-control method, for tackling the challenge posed by crosstalk faults in network-on-chips (NoCs), which is power-efficient. The authors have identified opposite direction (OD) transitions as the source of crosstalk faults in NoC communication channels. The proposed method, named FRR (Flit Reordering / Rotation) by the authors, combines three coding techniques to eliminate the OD transitions entirely.

The first coding technique mechanism is known as flit-reordering. It, as the name suggests, reorders flits of every packet to find a flit sequence that generates the least number of OD transitions. The second technique is flit-rotation. It logically rotates the content of every flit in a packet with respect to previously sent flit to achieve an even greater reduction in the number of OD transitions. The third mechanism, known as flit-insertion, investigates flits to find the OD transitions which cannot be removed by the first and the second mechanisms. The third mechanism strives to eliminate the appearance of OD transitions by inserting null-flits between the required flits completely.

The authors have evaluated their proposed FRR method in two ways. First, they have carried out VHDL-based simulations for 16- and 32-bit channels with the constraint that the maximum number of reorderings and rotations in the first and second mechanisms respectively are limited to 2, 4, and 8. Secondly, the authors have developed an analytical model to calculate and compare the expected number of OD transitions in an unprotected NoC as well as an FRR-enabled NoC.

The authors have also suggested that the results obtained from both the simulation and the analytical model, confirm that the FRR method can completely remove crosstalk faults from NoC channels. Also, the authors also suggest that the FRR method provides remarkable power savings because the proposed method reduces the number of transitions in NoC channels by at least 32.8%.

2.3.5.3 Optimization Techniques in NoC Design to Reduce Crosstalk [18]

This paper presents several optimization problems occurring in VLSI interconnect, network-on-chip (NoC) design, and 3D VLSI integration, all possessing closed-form solutions obtained by well-solvable Quadratic Assignment Problems (QAP). The first type of problems deals with the optimal ordering of signals in a bus bundle such that the switching power, delay, and noise interference are minimized. The authors have extended a known solution of ordering the signals in a bus bundle to minimize the impact of crosstalk, i.e. the first order wire-to-wire parasitic capacitance occurring between adjacent wires into a model accounting for the secondary components of wire-to-wire parasitic capacitances as well.

2.3.6 Crosstalk in Analog / Digital Circuits

2.3.6.1 Impact of Crosstalk on Circuit Design [42]

The crosstalk phenomenon and its impact on the design of mixed analog/digital circuits with high accuracy specifications have been demonstrated in this work. Generation of digital disturbs, propagation through the substrate, and effects on analog devices have been considered, with particular emphasis on integrated circuits realized on the heavily doped substrate, where traditional shielding is less effective. Techniques to reduce analog/digital crosstalk have also been reviewed and discussed in [42].

2.3.6.2 Interconnect Spacing Technique for Crosstalk Minimization [47]

Reduction of interconnect delay and interconnect power has become a primary design challenge in recent CMOS technology generations. The spacing between wires can be modified so that line-to-line capacitances can be optimized for minimal power under timing constraints. In [47], the authors have presented a novel algorithm for simultaneous multi-layer interconnect spacing that minimizes the total dynamic power dissipation caused by interconnection while maximum delay constraints are satisfied.

The authors have introduced a multi-dimensional visibility graph to represent the problem, and a layout partitioning technique has been applied to solve the problem efficiently. The algorithm has been evaluated on an industrial microprocessor designed using the 32 nm technology, and the authors claim to have achieved a 5–12% reduction in interconnect switching power.

2.3.7 Crosstalk Avoidance in Interconnects

2.3.7.1 A Frequency Domain Approach for Minimization of Crosstalk in High-Speed Interconnects [19]

In [19], the authors propose a frequency-domain approach to simulate efficiently and minimize the crosstalk between high-speed interconnects. They have discussed several methods in the text for modelling coupled transmission lines, at the same time considering numerous possible simulation strategies. The authors follow a straightforward yet rigorous frequency domain approach. As it exploits a harmonic balance technique, the approach can be used for linearly and non-linearly terminated micro-strip coupled lines.

The authors have simulated a typical example of micro-strip interconnects, and they have compared their results with those obtained using time-domain methods, by other authors. The work suggests that the devised simulation method yields good accuracy. The authors have formulated a crosstalk minimization problem and have implemented the proposed method to solve it.

2.3.7.2 Error Control Coding to Reduce Crosstalk [44]

The authors in [44] have suggested an energy-efficient error control code for the on-chip interconnection link. The proposed code is capable of correcting any error patterns that includes random and burst errors up to 5 bits. The proposed decoding scheme has been designed based on single-error-correction–double-error-detection (SEC–DED) extended Hamming code and standard triplication error correction scheme. The triplication error correction scheme provides crosstalk avoidance by reducing the coupling capacitance of the interconnection wire.

The authors suggest that the proposed code can provide high reliability compared to other error control codes. They have evaluated the performance of the proposed code by codec area, codec power, codec delay, residual flit error rate, link swing voltage, and link power. The proposed code has achieved low residual flit error rate and swing voltage, for any given reliability requirements of 10^{-5} and 10^{-20}. A reduction in the swing voltage, in turn, reduces the link power consumption up to 68% compared to the existing methods. The authors have claimed that the low residual flit error rate achieved and the low link power consumed, make the proposed code appropriate for on-chip interconnection link.

2.3.7.3 Bus-Coding Techniques for Crosstalk Avoidance [21]

RC crosstalk effect in on-chip buses leads to some serious problems as propagation delay and dynamic power dissipation. The authors have presented two efficient bus-coding techniques to reduce simultaneously dynamic power dissipation and wire propagation delay. The authors claim to have achieved improvements in both fronts compared to existing techniques.

Simulation results presented by the authors show that the first proposed technique reduces coupling activity by 26.7–38.2% and switching activity by 3.7–7.0% on 8-bit to 32-bit data buses, respectively. The second proposed coding technique reduces coupling activity by 27.5–39.1% and switching activity by 5.3–9.0% on 8-bit to 32-bit data buses, respectively. The simulated results also suggest that both the proposed methods reduce dynamic power by 23.9–35.3% on 8-bit to 32-bit data buses and total propagation delay by up to 30.7–44.6% on 32-bit data buses, and eliminate the Type-4 coupling. The proposed methods also claim to reduce total power consumption by 23.6–33.9%, 23.9–34.3%, and 24.1–34.6% on 8-bit to 32-bit data buses with the 0.18 mm, 0.13mm, and 0.09 mm technologies, respectively.

2.3.8 Crosstalk Minimization in Routing

2.3.8.1 A Routing Framework for Crosstalk Avoidance [93]

With the exponential reduction in feature size, the inter-wire coupling capacitance becomes the dominant factor of load capacitance. Coupling delay deterioration and crosstalk are two problems that arise with the reduction in feature size. The authors of this paper have proposed a timing-driven global routing algorithm that considers coupling effects and crosstalk avoidance.

They claim that the proposed methodology differs from the existing ones in that the proposed global routing “*framework*” performs well regarding routability, timing, etc. and at the same time also facilitates the detailed routing stage in crosstalk avoidance. The authors have presented experimental results on industrial circuits that suggest how the algorithm leads to substantial delay reduction and effective crosstalk elimination.

2.3.8.2 Switchbox Routing with Crosstalk Constraints [26]

In [26], the authors have investigated the gridded switchbox routing problems with the aim of satisfying crosstalk constraints and minimizing the total crosstalk among the nets. The authors have proposed a new approach to the problems that employs existing switchbox routing algorithms and brings about an improvement in the routing results via respective reassignment of the horizontal and vertical wire segments to rows and columns, in an iterative fashion. The authors claim that this method is applicable to the channel routing problem with crosstalk constraints. The authors have proposed a novel mixed integer linear programming (ILP) formulation and efficient techniques to reduce the number of variables and constraints in the presented ILP formalism. The authors claim to have achieved encouraging experimental results.

2.3.8.3 Crosstalk Minimization in Channel Routing [25, 27]

In [25, 27], the authors have studied the gridded channel routing problem, with the purpose of satisfying crosstalk constraints for the nets. The authors have proposed a new approach that employs existing channel routing algorithms and achieves an improvement upon the routing results by a permutation of the routing tracks. The authors have then presented a novel mixed integer linear programming (ILP) methodology and efficient techniques to reduce the number of variables and constraints in the presented ILP formalism. The authors have claimed to have achieved encouraging experimental results.

2.3.8.4 Crosstalk Minimization in Microring-based Wavelength Routing Matrices [9]

Silicon microring resonators (MRR) can be used for switching operations directly in the optical domain. Nevertheless, MRR-based switching fabrics have the probability of having limited scalability regarding port count because of the crosstalk accumulation caused due to the reuse of spatial wavelengths. The authors have considered an optical switching fabric (OSF) that is built using a wavelength-routing-matrix (WRM), based on MRRs. The authors have highlighted the scalability issues and have proposed a new design. The authors have also suggested two different approaches to limit the spatial reuse of wavelength to enhance the scalability of MRRs making them suitable for future high-capacity OSFs.

The authors have introduced an MRR-based switching fabric, which uses the periodical transfer function and tunability of MRRs to implement multiple wavelength assignments. Subsequently, the authors have presented and analyzed the matrix selection (MS) and matrix combination (MC) strategies to reduce the reuse of wavelength based on the exploitation of the proposed MRR-based WRM. The authors have claimed that the MS approach divides the wavelength reuse factor roughly by two, compared to the single WRM configuration. While on the other hand, the MC strategy has been claimed to reduce further crosstalk, significantly. The authors have described two possible applications for the MC strategy, the exhaustive algorithm (EA) and the greedy algorithm (GA). According to the authors, the complexity of the GA is considerably lower than the MS strategy, and the GA also produces a noticeably lower crosstalk than the MS strategy. The authors have stated that the GA can be believed a good candidate algorithm for controlling the proposed MRR-based WRM.

2.3.8.5 Simulated Annealing based Approach to Crosstalk Minimization in Gridded Channel Routing [39]

With the rapid evolvement of VLSI fabrication technology, the inter-wire spacing in a VLSI is becoming closer. Consequently, it becomes imperative to minimize the crosstalk due to the coupling capacitances between the adjacent wires in the layout design of fast and safe VLSI circuits. The authors have presented an approach based on simulated annealing and segment rearrangement for minimization of crosstalk in an initially gridded channel routing. The authors have compared the proposed technique with previous track-oriented techniques, especially a track permutation technique whose performance is bounded by an exhaustive track permutation algorithm. The authors have claimed that the experimental results obtained indicate the proposed technique to be more effective than the track permutation technique. The efficiency is more pronounced in the case of test examples where there are only a few possibilities of track permutation and which have relatively large number of segments on track. However, the authors have conceded that the time complexity of the simulated annealing based approach is rather high.

2.4 Summary

Crosstalk is a phenomenon that arises when a pair of conductors is placed in close proximity, separated by an insulator. Whenever electrical energy (signals) flows through these conductors, the entire arrangement behaves as a capacitor, resulting in the generation of coupling capacitance. This capacitance is directly proportional to the amount of overlap of the two conductors and is responsible for introducing noise into whatever it is that is being transmitted through the conductors. This phenomenon can be visualized in electrical power lines, communication lines, and in the micro-scale, in electronic circuits, between two interconnecting wires. The amount of noise directly affects the quality of signals being transmitted, and is needed to be kept within a limit or margin, suitably titled as *noise margin*.

In the present scenario, with ever reducing feature sizes and higher speeds of operation, electronic circuit design faces the challenge of overcoming crosstalk noise, one of the most significant factors limiting IC design. Broadly speaking, there are two different approaches towards solving this problem: one involves modifications in the design phase for crosstalk avoidance, and the other involves post-design modifications for crosstalk reduction. In this chapter, we have attempted to make a detailed study of various techniques for both crosstalk avoidance and/or reduction implemented by a number of researchers over the past few decades and the success they have had against this natural vice of electrical/electronic circuits.

Moreover, theoretical studies in terms of knowing (and/or proving) the nature of a problem, whether it is intractable or polynomial time computable, measure the way of thinking in attacking a problem. Some problems that are solvable in polynomial time and some problems that are not, have been discussed at the beginning of this chapter, including a brief idea on the theory of NP-completeness. Now we mention a few more problems in channel routing that are NP-complete before we begin the next chapter on the hardness of crosstalk minimization in two-layer channel routing. We have already mentioned that LaPaugh considered the area minimization problem in two-layer channel routing and proved its NP-hardness [41]. Szymanski established that the following problems in the rectilinear reserved two-layer unrestricted dogleg routing model are NP-hard: (i) minimizing the number of tracks needed to route an arbitrary channel, and (ii) minimizing the total wire length of an arbitrary channel [86].

Schaper demonstrated that under the three-layer HVH routing model, the problem of routing a channel with a minimum number of tracks is NP-hard for the two-terminal no-dogleg case, and subsequently for the multi-terminal restricted dogleg case [80]. The multi-layer channel routing problems of area minimization in the V_iH_i, $2 \leq i < d_{max}$, and V_iH_{i+1}, $2 \leq i < d_{max}-1$ routing models with alternating vertical and horizontal layers of interconnect are known to be NP-hard [49, 55, 56, 58]. Besides, plenty of problems in wire length minimization for two-, three-, and multi-layer channel routing have been considered by researchers, and proved that these problems are also NP-hard [49, 57, 58, 59, 64].

Chapter: 3

Hardness of Crosstalk Minimization in Two-Layer Channel Routing

3.1 Overview

The channel routing problem (CRP) has already been defined in Chapter 1. As mentioned earlier, the main objective is to minimize the channel area while interconnecting all nets belonging to a channel in a specified routing model. Beyond the usual cost factor(s), we consider crosstalk minimization as a high performance issue in computing a routing solution. In the previous chapter, we have seen how the crosstalk is taken care of by researchers worldwide, either by avoiding it as an a priori task or by minimizing it as an a posteriori problem in devising electrical, electronic, and VLSI circuits. By the way, the theoretical study in terms of proving the nature of several crosstalk minimization problems in two-layer channel routing, whether they are tractable or beyond polynomial time computable, is the major concern of this chapter.

We have presumed the problem of crosstalk minimization as a problem of minimizing electrical hazards that should be reduced to in the circuit performance. As fabrication technology advances and feature size reduces, devices are placed increasingly closer to each other, interconnecting wire segments are assigned with a narrower pitch. However, the circuit operations are being realized at even higher frequencies. As a result, electrical hazards, viz., crosstalk between wire segments have significantly evolved. The crosstalk between wire segments is proportional to the coupling capacitance, which in turn is proportional to the coupling length, i.e. the total length of overlap between wire segments of two different nets. The crosstalk is also proportional to the frequency of operation and inversely proportional to the separating distance between wires. More crosstalk means more noise and more signal delay resulting in overall circuit performance. Therefore, it is desirable to develop channel routing algorithms that not only reduce the channel area but also the resulting crosstalk. Work on routing channels with reduced crosstalk is very important from the point of view of high performance requirements in VLSI circuit synthesis.

In this chapter, we show that the crosstalk minimization problem in the reserved two-layer Manhattan routing model is NP-hard for simple and general instances of channel specifications with given partitioning of nets so that the nets in a class of the partition are assigned to the same track. We have also investigated upon the simple as well as general instances of channel specifications with only two-terminal nets, but without any imposed partition of (non-overlapping) nets to tracks. In addition, we introduce the problem of minimizing bottleneck crosstalk in the reserved no-dogleg two-layer channel routing model. We prove that the problem is also NP-hard. We further investigate the existence of polynomial time exact algorithms, and approximation algorithms for the above mentioned problems in two-layer channel routing. We prove that the problems are NP-complete. We show that the problem is hard to approximate, too. In all these cases, the problems have also been studied with doglegging allowed as well.

3.2 Crosstalk Minimization in Two-Layer Channel Routing

3.2.1 Foundation of the Problem

In the channel routing problem, the set of all terminals that need to be electrically connected together is called a *net*. In Figure 3.1, at most two columns of numbers having the same integer value (other than zero) uniquely define a two-terminal net. In other words, a net with only two terminals is called a *two-terminal net*. A vertical wire segment is a wire that lies in a column assigned to the vertical layer, whereas a horizontal wire segment is a wire that lies in a track assigned to the horizontal layer in a two-layer VH routing model. Tracks are horizontal lines that are usually equispaced along the height of the channel, parallel to the two rows of (fixed) terminals.

A route for a net is a collection of horizontal and vertical wire segments spread across different layers, connecting all terminals of the net. A legal wiring of a channel is a set of routes that satisfy all the pre-specified conditions where no two wire segments, used to connect different nets, overlap on the same conducting layer. A legal wiring is also called a feasible routing solution.

The CRP is specified by two m element vectors *TOP* and *BOTTOM* of integers, and a positive integer t. The objective is to find a feasible routing solution for the channel, if it exists, using no more than t tracks. An instance of the CRP is shown in Figure 3.1, where we have an assignment of intervals of the nets present in the channel to four tracks only. Let L_i (R_i) be the leftmost (rightmost) column position of net i, then $I_i = (L_i, R_i)$ is known as the interval (or span) of the net.

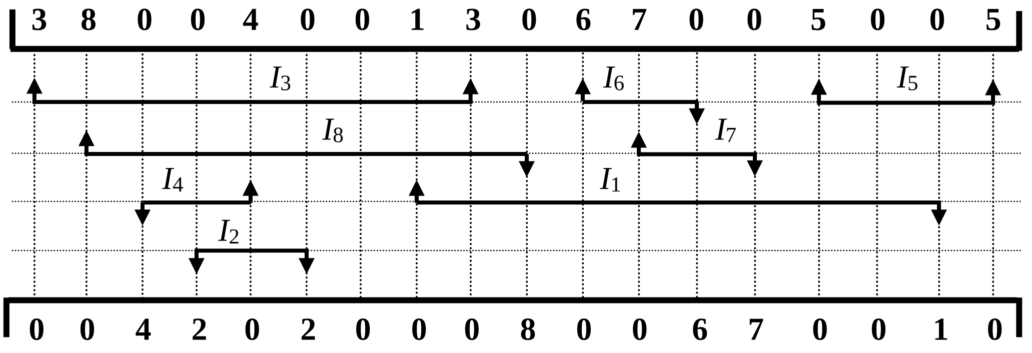


Figure 3.1: An example channel of eight nets. The intervals of the nets are placed in four different tracks. Terminals are vertically aligned along the columns of the channel. The length of the channel (i.e. the number of columns) is 18. Arrows indicate the terminals to be connected, either at the top or at the bottom, to complete the required interconnection of all nets.

3.2.2 Models of Crosstalk Minimization

Routing channels with reduced crosstalk are very important from the viewpoint of high performance [25, 27, 49], and it is desirable to develop channel routing algorithms that not only reduce channel area but also crosstalk. We define the amount of crosstalk between horizontal wire segments (i.e. intervals) of two different nets assigned to (two) adjacent tracks in a given routing solution, to be proportional to the amount of overlap of their horizontal spans. If two intervals do not overlap, there is no horizontal constraint between the nets; therefore, no crosstalk exists between them too. Further, we also assume that two overlapping intervals of two different nets that are not placed in adjacent tracks produces no significant crosstalk and it does not contribute anything to the overall crosstalk. In fact, technology itself is responsible for bringing the amount of this crosstalk within a permissible range of noise margin.

For example, nets n_5 and n_8 would never produce any crosstalk, as the respective intervals, I_5 and I_8 do not overlap with each other in the channel instance shown in Figure 3.1, whereas nets n_5 and n_1 will result in accountable crosstalk as the respective intervals I_5 and I_1 overlap in the channel. The crosstalk between two nets is the same as the amount of overlapping between them if the nets are assigned to adjacent tracks. This is measured in terms of the number of units of overlap between a pair of nets on adjacent tracks in a feasible routing solution. Similarly, if the nets n_3 and n_8 are assigned to (two) adjacent tracks in a solution of the channel shown in Figure 3.1, then the amount of crosstalk between these two nets would be seven units.

We assume that the amount of crosstalk, between vertical wire segments of two different nets placed in adjacent columns, is very small, and hence can be neglected. It is a matter of technology to keep a safe separation between (two) adjacent columns of a channel that crosstalk evolved due to vertical wire segments is always within some limit of tolerance. This is true even if the longest possible adjacency of vertical wire segments of two different nets (as the vertical wire segments are available due to the fixed terminals that cannot be altered) exists along the length of the channel.

It may be noted that if more tracks are introduced into a feasible routing solution, then the amount of crosstalk may be reduced further (as some of the nets that were assigned to adjacent tracks are now mutually nonadjacent). Obviously, if t–1 blank tracks (i.e. the tracks containing no interval of any net) are introduced to a t-track feasible routing solution, where a blank track is placed between two consecutive tracks (of nets), the resulting routing solution will not have any crosstalk. However, this routing solution uses almost twice the area of the initial routing solution, whereas our prime interest is to compute a routing solution of the minimum possible area. So we assume a constraint that a fixed number of tracks are to be used in obtaining a minimum crosstalk routing solution.

3.2.2.1 Types of Crosstalk Minimization

Now we consider the problems of crosstalk minimization in two-layer channel routing. We first introduce the *sum crosstalk minimization* problem. Crosstalk in a reserved two-layer channel routing solution is the sum of all crosstalk between pairs of horizontal wire segments of nets that are assigned to adjacent tracks. The *sum crosstalk minimization* problem is to compute a feasible routing solution with a given number of tracks in which the total amount of crosstalk is minimized. Unless mentioned otherwise, by crosstalk minimization problem we refer to the sum crosstalk minimization problem.

In any routing solution, we define the bottleneck cost (or bottleneck crosstalk) as a maximum of all crosstalk between any pair of nets that are placed in adjacent tracks. There can be several routing solutions that use fixed number of tracks, but having different bottleneck crosstalk. We define the *bottleneck crosstalk minimization* problem as the problem of computing a *k*-track routing solution such that the bottleneck cost is minimized.

Suppose that we are given a *k*-track routing solution. Note that it may be possible to reduce crosstalk further by interchanging some nets between a pair of tracks. This is explained in Figure 3.2, where a new assignment of tracks to the same solution reduces the amount of crosstalk from 15 units (as in Figure 3.1) to 10 units. So, one way of reducing the crosstalk is to suitably interchange the nets assigned to tracks in a given routing solution. Keeping this in view, we define the problem *VHP* of crosstalk minimization in two-terminal no-dogleg two-layer VH channel routing as: *Given an a priori partition of nets such that the nets in a class of the partition can be assigned to a single track, a feasible routing solution can be obtained by assigning each class of nets to one track. This feasible solution has a fixed crosstalk. Every permutation of the classes can be considered a feasible solution with a fixed amount of crosstalk. The objective is to get a feasible solution (a permutation) with the minimum amount of crosstalk.* The problem is formally posed in the next section.

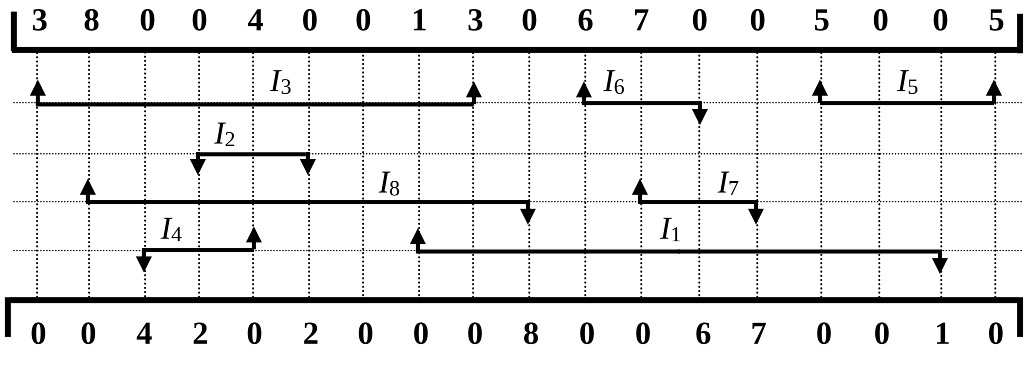


Figure 3.2: Track-wise reassignment of nets of the 4-track assignment of non-overlapping intervals in Figure 3.1 so as to reduce the amount of crosstalk from 15 units to 10 units. Here, intervals of tracks 2, 3, and 4 are now assigned to tracks 3, 4, and 2, respectively.

However, it may not be possible to obtain the minimum crosstalk *k*-track routing solution in this way. This is because such an optimal solution may correspond to a different net grouping. This is illustrated in Figure 3.3. Thus, we address the question: *Is it possible to compute a k-track minimum crosstalk routing solution in polynomial time?*

Keeping this in view, in the next section, we pose two crosstalk minimization problems, *VHS* and *VHG*, of two-terminal no-dogleg two-layer VH channel routing, where the channel instances are free from a specific constraint in one case (i.e. in the absence of vertical constraints) and having both types of constraints present in it in the other case. In both cases, we like to compute a *k*-track two-layer routing solution without any given partition of nets (i.e. there is no class of nets in the form of the partition that is to be assigned to the same track) such that crosstalk is minimized.

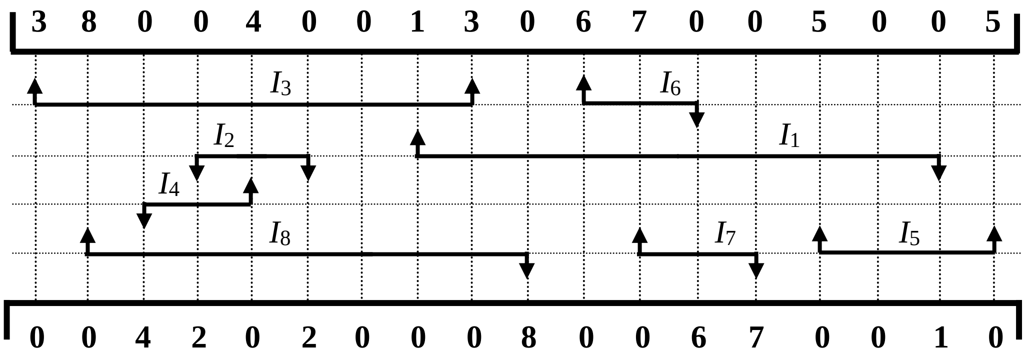


Figure 3.3: Reassignment of nets in order to compute another 4-track routing solution of the channel instance in Figure 3.1, so as to reduce the amount of crosstalk from 15 units to 8 units only. Here, net n_5 is no longer in the group of nets n_3 and n_6; rather n_5 is with nets n_8 and n_7 now. Similarly, leaving n_4 alone, n_1 is with n_2 now.

The crosstalk minimization problems defined above are optimization problems where we need a *k*-track routing solution having a minimum amount of crosstalk. Often, such a routing solution may not be acceptable due to long overlapping of a pair of wire segments (high bottleneck) of two different nets on adjacent tracks. We define the maximum amount of overlap in any routing solution between a pair of adjacent nets as the bottleneck of that solution. Then the bottleneck crosstalk minimization problem is to find a feasible routing solution of given number of tracks such that the bottleneck of that solution is minimized. We define this problem as the bottleneck crosstalk minimization problem, *BVHP*, of two-terminal no-dogleg two-layer VH channel routing, given an a priori partition of nets so that the nets in a class of the given partition are to be assigned to the same track.

Our observation is that this problem of crosstalk minimization is practically more relevant in computing a high performance routing solution in some cases than the sum crosstalk minimization problems. As in the case of the sum crosstalk minimization problem, we can have two variations here as well; one with given partition of nets and the other one where there is no partition of nets. A more formal presentation of all these problems is given in the next section.

3.3 Crosstalk Minimization in Channel Routing

In computing a routing solution, our prime intention is to compute a solution that uses a minimum possible number of tracks (or minimum channel area). In addition to computing a routing solution with reduced area, in high performance routing our interest is also to obtain a routing solution with less electrical hazards (i.e. crosstalk), less signal propagation delay, less power consumption, less or no hot spot formation, and so on and so forth.

The CRP of area minimization is itself an NP-complete problem [41, 49, 56, 58, 61, 80, 86]. However, the problem is polynomial time solvable if the instances are free from any vertical constraint. In fact, there are polynomial time d_{max}-track routing solutions (or routing solutions of density number of tracks) for such instances [32, 49, 52, 53]. Since the problem of minimizing area for instances of routing channel with only horizontal constraints is polynomial time solvable (using exactly d_{max} tracks), we define such instances as the simple instances of channel routing. A general channel instance contains both types of constraints in it. There can be several d_{max}-track minimum area routing solutions for simple channel instances. As discussed in Section 3.2.2, these solutions are of the same area, but of different amount of crosstalk. We also discussed two ways to reduce crosstalk further. One way is to interchange the allotted tracks of all nets among the tracks without changing the net grouping. This way, we could get the improved solution as shown in Figure 3.2 than the one in Figure 3.1. The other way could be by breaking net groupings allotted to tracks, as obtained in Figure 3.3.

Therefore, we have the following decision problem.

Problem: *VHP* (Crosstalk minimization in two-terminal no-dogleg two-layer VH channel routing, given an a priori partition of nets).

Instance: $I = (N, P, K)$, where N is a set of two-terminal nets, P is a partition of N nets into t non-overlapping classes of intervals, and K is a positive integer.

Question: Does there exist an assignment of nets to t tracks such that (i) all nets in the same class in the given partition P are assigned to the same track, and (ii) the total crosstalk is at most K?

In the next section, we prove that this problem is NP-complete. Now we formally pose the problems of crosstalk minimization, *VHS* for simple instances, and *VHG* for general instances of channel specifications in the absence of any partition P of nets, as follows.

Problem: *VHS* (Crosstalk minimization in two-terminal no-dogleg two-layer VH channel routing for simple instances of channel specification).

Instance: A simple channel specification $I = (N, K)$ of N two-terminal nets with density d_{max}, and a positive integer, K.

Question: Does there exist a d_{max}-track no-dogleg two-layer VH routing solution of the given channel specification so that the total crosstalk is at most K?

Problem: *VHG* (Crosstalk minimization in two-terminal no-dogleg two-layer VH channel routing for general instances of channel specification).

Instance: A general channel specification of two-terminal nets, and two positive integers, t and K.

Question: Does there exist a t-track no-dogleg two-layer VH routing solution of the given channel specification so that the total crosstalk is at most K?

In the next section, we prove that these problems are NP-complete. Now we formally present the bottleneck crosstalk minimization problem as follows.

Problem: *BVHP* (Bottleneck crosstalk minimization in two-terminal no-dogleg two-layer VH channel routing, given an a priori partition of nets).

Instance: A simple channel specification with two-terminal nets, a partition, P of nets into classes of non-overlapping intervals, and a positive integer, B.

Question: Does there exist an assignment of nets to $|P|$ tracks such that (i) all nets in the same class in the given partition P are assigned to a track, and (ii) the amount of crosstalk between any two horizontal wire segments (or intervals) of two different nets on adjacent tracks is B or less?

We prove that this problem is also NP-complete in the next section.

3.4 Theoretical Proofs

3.4.1 Hardness of Crosstalk Minimization in the Absence of Vertical Constraint

In this section, we show that *VHP* is NP-complete by reducing a variant of the *Hamiltonian path* (*HP*) problem to *VHP*. The problem *HP* is the following [13, 28, 66].

Instance: An undirected graph $G = (V, E)$.
Question: Does G contain a Hamiltonian path?

Before showing that *VHP* is NP-complete, we need to show that the following variant *HP** of problem *HP* is also NP-complete.

Problem: *HP** (*Weighted Hamiltonian path*).
Instance: An undirected weighted complete graph $G^* = (V, E^*)$, with weight $w(e) = 1$ or 2 for each edge $e \in E^*$.
Question: Does G contain a Hamiltonian path of weight $n-1$, where $n = |V|$?

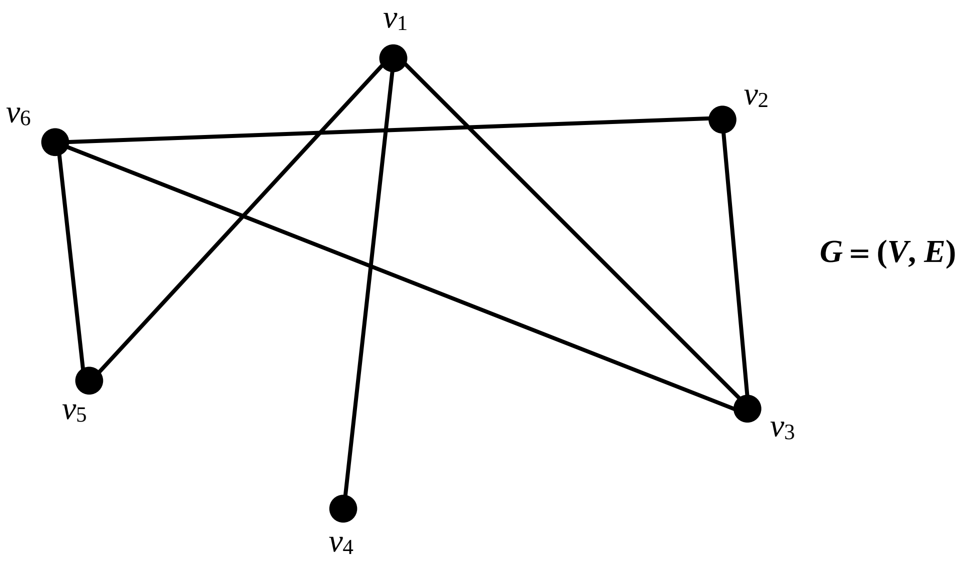
Theorem 3.1: *HP* is NP-complete.*

Proof: We first show that *HP** belongs to NP. Given an instance of the problem, we take a certificate that is a sequence of $n = |V|$ distinct vertices. The verification algorithm checks that this sequence contains each vertex exactly once and the sum of the weights of edges on this path is exactly $n-1$. This algorithm can certainly be executed in polynomial time. Therefore, $HP^* \in$ NP.

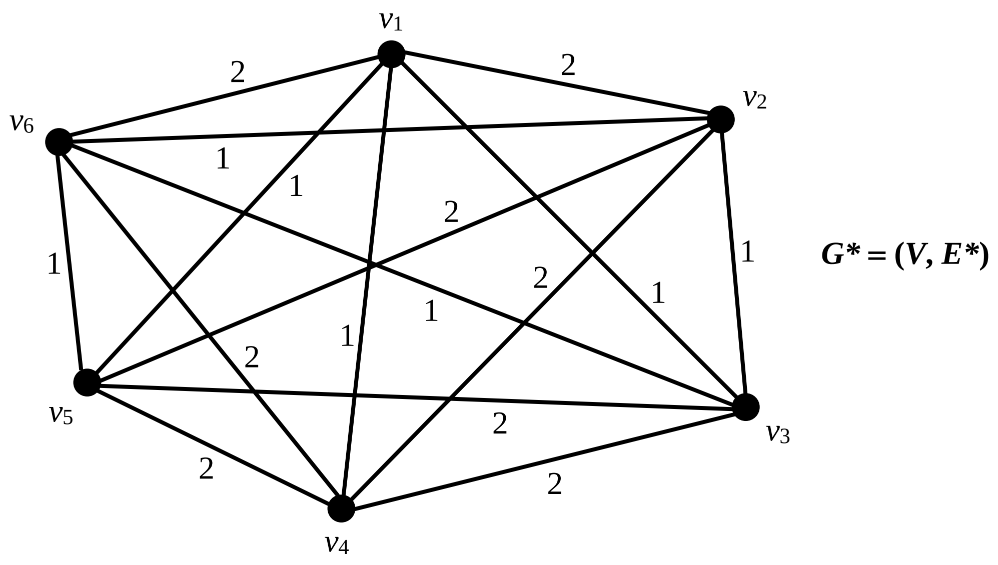
To prove that *HP** is NP-hard, we show that $HP \leq_P HP^*$. Let $G = (V, E)$ be any instance of *HP*. We construct a corresponding instance of *HP** as follows. We compute a complete graph $G^* = (V, E^*)$, where $E^* = \{(v_i, v_j) \mid v_i, v_j \in V\}$, and for every edge $e \in E^*$, we assign weight $w(e) = 1$, if $e \in E$; otherwise, we assign weight $w(e) = 2$. The instance of *HP** can be obtained in polynomial time. The construction has been explained in Figure 3.4.

We now show that the graph G has a Hamiltonian path HP_P if and only if the graph G^* has a weighted Hamiltonian path HP_P^* of weight $n-1$, where $n = |V|$.

Suppose that the graph G has a Hamiltonian path HP_P. The same path is also a Hamiltonian path in G^*. We take $HP_P^* = HP_P$. Each edge of this path has a weight of one. Thus, HP_P^* has weight exactly $n\!-\!1$ in G^*. Hence, if HP_P is a Hamiltonian path in G, then HP_P^* is a weighted Hamiltonian path in G^* with weight $n\!-\!1$.



(a)

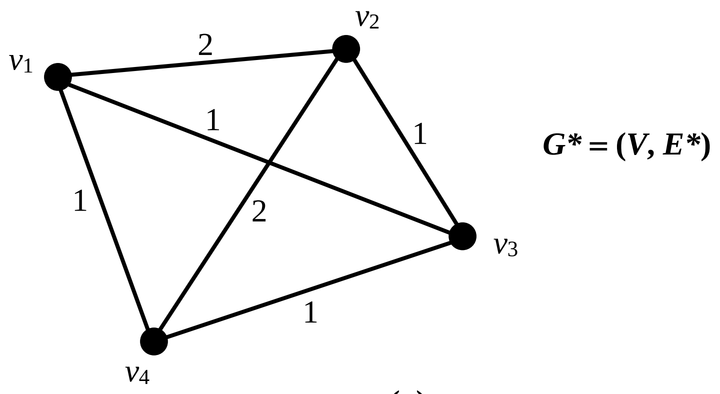


(b)

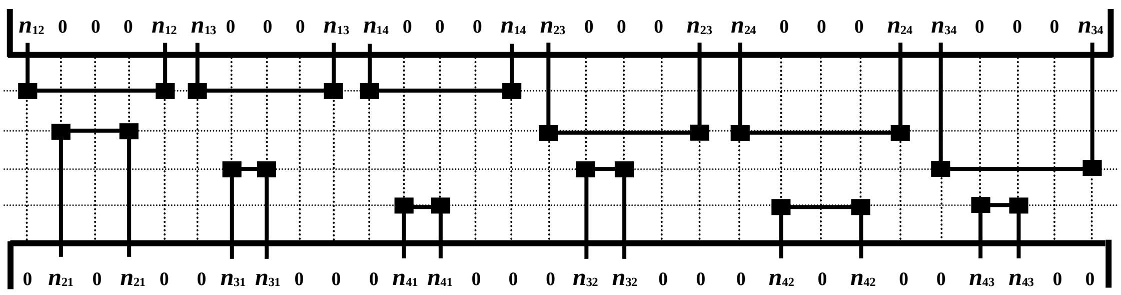
Figure 3.4: (a) A graph instance $G = (V, E)$ of problem HP. **(b)** The graph $G^* = (V, E^*)$ of the corresponding instance of problem HP^*, computed from G.

Conversely, suppose that the graph G^* has a weighted Hamiltonian path HP_P^* of weight $n\!-\!1$. Since HP_P^* has a length of exactly $n\!-\!1$ (edges) and each edge in G^* is of weight one or two, all edges on HP_P^* must have edge weight of exactly one. Thus, all edges on this Hamiltonian path $HP_P{*}$ are edges in E as well. Let HP_P be the set of all these edges in E. HP_P is obviously a Hamiltonian path in G. This completes the proof. ♦

Now we show that *VHP* is NP-complete by reducing *HP** to it. First, we show that $VHP \in$ NP. Given a feasible $|P|$-track no-dogleg two-layer VH routing solution for any instance I' of *VHP*, we can verify in polynomial time whether (i) all the nets of the same class in the given partition P are assigned to the same track, and (ii) the total crosstalk is less than or equal to K, simply by checking the total amount of crosstalk between nets assigned to adjacent tracks. Therefore, $VHP \in$ NP.



(a)



n_{12} 0 0 0 n_{12} n_{13} 0 0 0 n_{13} n_{14} 0 0 0 n_{14} n_{23} 0 0 0 n_{23} n_{24} 0 0 0 n_{24} n_{34} 0 0 0 n_{34}
0 n_{21} 0 n_{21} 0 0 n_{31} n_{31} 0 0 0 n_{41} n_{41} 0 0 0 n_{32} n_{32} 0 0 0 n_{42} 0 n_{42} 0 0 n_{43} n_{43} 0 0

(b)

Figure 3.5: (a) A complete graph $G^* = (V, E^*)$ of instance I of problem *HP**. **(b)** The corresponding channel instance I' of the crosstalk minimization problem *VHP*, where nets n_{ij} *and* n_{ji} are introduced into the channel corresponding to edge (v_i, v_j) $(i < j)$ in G^*. Here, $N = \{n_{ij} \mid j \neq i\}$ and the i-th class $P_i = \{n_{ij} \mid 1 \leq j \leq n, j \neq i\}$, is a set of non-overlapping nets in I'. $\{P_i \mid i = 1, 2, ..., |P|\}$ is the required partition P of nets. $I' = (N, P, K)$, where K is an integer, is the instance of *VHP* obtained.

To show that *VHP* is NP-hard, we consider the following reduction from problem *HP** to *VHP*. We construct an instance $I' = (N, P, K)$ of problem *VHP* from any instance I of the problem *HP** by a polynomial time transformation as follows.

Let the number of vertices of the graph in I be n. For every edge (v_i, v_j) $(i < j)$, we introduce two two-terminal nets n_{ij} and n_{ji} in the channel. We place the two terminals of net n_{ij} at positions $(5\times((2n-i)(i-1)/2 + (j-i-1))) + 1$ and $(5\times((2n-i)(i-1)/2 + (j-i-1))) + 5$ of the top row of terminals. For net n_{ji}, we place the two terminals at positions $(5\times((2n-i)(i-1)/2 + (j-i-1))) + 2$ and $(5\times((2n-i)(i-1)/2 + (j-i-1))) + 2 + w(v_i, v_j)$ of the bottom row of terminals. Assume that $N = \{n_{ij} \mid i \neq j\}$ be the set of all $n\times(n-1)$ nets. Let P_i be the set of nets $\{n_{ij} \mid 1 \leq j \leq n, j \neq i\}$. Observe that $P_i \cap P_j = \emptyset$ and $\bigcup_{i=1}^{n} P_i = N$. Therefore, $P = \{P_i \mid i = 1, 2, \ldots, n\}$ is a partition of all nets N into n classes. By construction, no two nets in the class P_i have any horizontal constraints between them; therefore, they can be assigned to a track. All remaining terminals are vacant terminals and thus, are not required to be connected. Obviously, $I' = (N, P, K)$, where K is an integer, is an instance of *VHP*. This completes the construction of the channel instance I'. It can be constructed in polynomial time. It contains $n\times(n-1)$ nets and the length of the channel is $5n\times(n-1)$. The construction of such a channel instance I' from a graph instance I is explained in Figure 3.5. Observe from the figure that the channel density of I' is two. In the following lemma we prove that it will always be two.

Lemma 3.1: *The channel density d_{max} of the constructed channel instance I' is 2.*

Proof: Let I_{ij} denotes the horizontal span (or interval) of net n_{ij} in I'. There are exactly $|E^*|$ two-terminal nets n_{ij} $(i < j)$, one for each edge (i, j). These nets have both the terminals at the top. By construction, all these nets have disjoint horizontal spans.

There are $|E^*|$ nets n_{ji} $(i < j)$, one for each edge in the graph; n_{ji} has both terminals at the bottom row. By construction, I_{ji} is contained in I_{ij}. Therefore, $I_{ij} \cap I_{ji} \neq \emptyset$. In all other cases, $I_{ij} \cap I_{i'j'} = \emptyset$. Hence, the lemma. ♦

Corollary 3.1: *In fact, $I_{ij} \cap I_{ji} = I_{ji} = w(v_i, v_j)$, where $w(v_i, v_j)$ is the weight of the edge (v_i, v_j). In all other cases $I_{ij} \cap I_{i'j'} = \emptyset$.*

To complete the proof that *VHP* is NP-complete, we now establish the following lemma.

Lemma 3.2: *I has a weighted Hamiltonian path HP_P* of weight n–1 if and only if I' has a feasible n-track no-dogleg two-layer VH routing solution with (i) all nets in a class P_i in the given partition P, assigned to the same track and (ii) the total crosstalk is exactly n–1.*

Proof: Observe that net $n_{ij} \in P_i$ and $n_{ji} \in P_j$. Now as $I_{ij} \cap I_{ji} \neq \emptyset$ and in all other cases $I_{ij} \cap I_{\hat{\imath}\hat{\jmath}'} = \emptyset$, as per the proof of Lemma 3.1, their corresponding intervals overlap. This means that the nets in P_i and P_j cannot be assigned to a single track. Therefore, we need $n = |P|$ tracks in any feasible solution of I'. Thus, all nets in a class are assigned to a track, and no two nets in different classes are assigned to a track in any solution of I'.

Now we calculate the amount of crosstalk, $c_{i,j}$, created between the nets in P_i and P_j if they are assigned to adjacent tracks. Again as $I_{ij} \cap I_{ji} \neq \emptyset$ and in all other cases $I_{ij} \cap I_{\hat{\imath}\hat{\jmath}'} = \emptyset$ (as in the proof of Lemma 3.1), no net in P_i, except n_{ij}, will make any crosstalk with some other net in P_j. By Corollary 3.1, n_{ij} and n_{ji} will produce a crosstalk of amount exactly $w(v_i, v_j)$. Therefore, $c_{i,j} = w(v_i, v_j)$, is the amount of crosstalk created if P_i and P_j are assigned to adjacent tracks.

Suppose that there is a weighted Hamiltonian path $HP_P^* = \langle v_1, v_2, \ldots, v_n \rangle$ in I of weight n–1. We show that there is a feasible n-track no-dogleg two-layer VH routing solution S for I' with (i) all nets in a class P_i are assigned to a track and (ii) the total crosstalk is exactly n–1. To get the feasible routing solution S, we assign all nets in P_i to the i-th track from the top. As the weight of the Hamiltonian path HP_P* is n–1 and HP_P* has exactly n–1 edges, $w(v_i, v_{i+1}) = 1$, $\forall i$. Total crosstalk for this solution S is $\sum_{i=1}^{n-1} c_{i,i+1} = \sum_{i=1}^{n-1} w(v_i, v_{i+1}) = n-1$.

Next, assume that I' has an n-track feasible solution S with a crosstalk n–1 in which all nets in P_i are assigned to a track, $\forall i = 1, 2, \ldots, n$. If P_i and P_j are assigned to (two) adjacent tracks in S, they contribute crosstalk of an amount of $w(v_i, v_j)$, which is either 1 or 2. As the total crosstalk is exactly n–1, each such pair of consecutive tracks contributes exactly one unit of crosstalk in S. Thus, $w(v_i, v_j) = 1$, if P_i and P_j are assigned to adjacent tracks in *S*. We construct a set of edges $HP_P{*}$ as $HP{*} = \{(v_i, v_j) \mid$ where P_i and P_j are adjacent in solution $S\}$. Obviously, *HP** is a Hamiltonian path in *I*. As $w(v_i, v_j) = 1$, *HP** is a Hamiltonian path of total weight $n-1$ in *I*. Hence, the lemma. ♦

We summarize the result obtained in the following theorem.

Theorem 3.2: *VHP is NP-complete.*

The result of the CRP with the partition of nets established in this section equally holds for the general instances of channel specifications (where the partition is provided in such a way that no cyclic vertical constraint is formed). This is because the set of simple instances of channel specifications is a proper subset of the set of general instances of channel specifications considering both constraints are present in it.

3.4.2 Hardness of Other Crosstalk Minimization Problems

In this section, we consider the problems, *VHS* and *VHG*, of crosstalk minimization in two-layer VH channel routing problem where no partition of nets are provided as in *VHP*. Let us first consider the problem *VHS*. As defined in Section 3.3, here we are given a simple channel specification $I = (N, K)$ of *N* two-terminal nets with density d_{max} and a positive integer *K*. *VHS* is required to find out if there exists a d_{max}-track no-dogleg two-layer VH routing solution such that the total crosstalk is at most *K*? In the next theorem, we prove that the problem is NP-complete.

Theorem 3.3: *VHS is NP-complete.*

Proof: A certificate here is a d_{max}-track no-dogleg two-layer VH routing solution *S*. We can find in polynomial time, the total amount of crosstalk in *S*, just by checking and summing up crosstalk between all adjacent tracks. Therefore, $VHS \in$ NP. Next, we show that *VHS* is NP-hard. We use proof by restriction method to prove the hardness of *VHS*. A feasible solution of *VHS* is a partition of nets into d_{max} classes of non-overlapping nets and an assignment of tracks, one to each class. Now there is an exponential number of such partitions of nets into d_{max} classes. If we restrict to a given partition, say *P*, the problem reduces to an instance of problem *VHP*. Now, as *VHP* is NP-hard, we claim that *VHS* is also NP-hard. Hence, the proof. ♦

Now let us consider the problem *VHG*. In this problem, we are given a general channel specification of two-terminal nets, and two positive integers, t and K. We are asked whether there is a t-track solution to this channel instance such that the total crosstalk is at most K? *VHG* is obviously in NP as in the case of *VHS* and *VHP*. We claim NP-hardness of *VHG,* again by using the proof by restriction method. In *VHG*, we restrict to cases where $t = d_{max}$, and there is no vertical constraint. The problem instance reduces to *VHS*. As *VHS* is NP-hard, we claim that *VHG* is also NP-hard. Therefore, we have the following theorem.

Theorem 3.4: *VHG is NP-complete.*

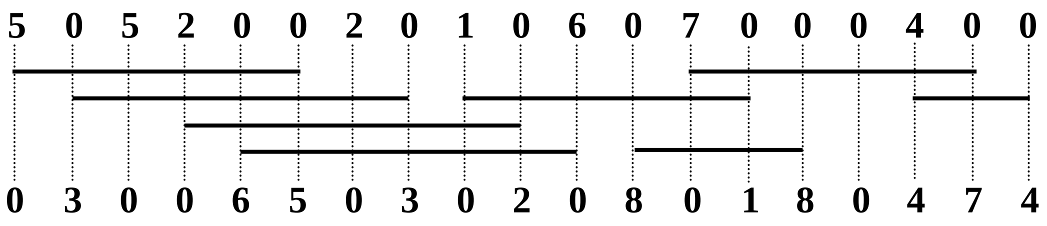
3.4.3 Hardness of Approximating Crosstalk Minimization

In the previous sections, we have shown that the problem of two-layer crosstalk minimization is NP-complete for simple as well as general instances of channel specifications. This means that it is almost impossible to design polynomial time algorithms to solve these problems. A natural question arises: *Whether there is any polynomial time approximation algorithm with guaranteed error bound to solve any one of these problems?* In this section, we prove that the problem of developing such an approximation algorithm for any such problem is also NP-hard.

First, we consider the simplest case, *VHP*. In fact, we show that it is impossible to design an approximation algorithm with ratio error ρ ($\rho > 1$), unless P = NP.

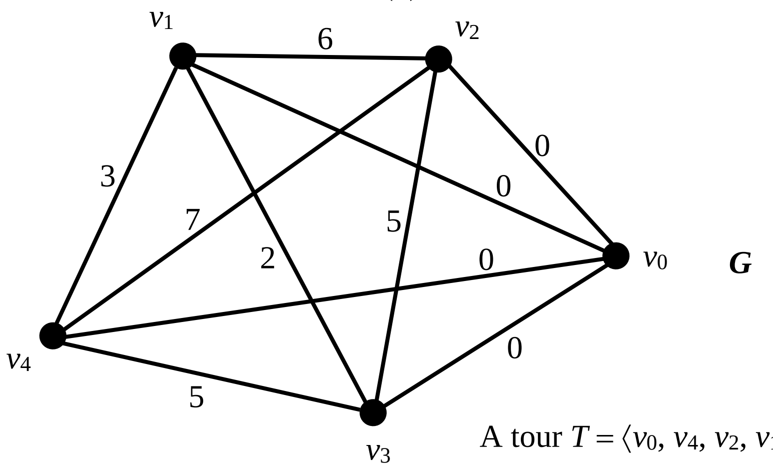
To establish this result, we formulate *VHP* as a general *travelling salesman problem* (*TSP*). The formulation is based on constructing a weighted undirected complete graph G, as described below. Let $I = (N, P, K)$, be the instance of *VHP*, where N is the set of nets, $P = \{P_i \mid i = 1, 2, \ldots, t\}$ is the partition of N into t disjoint classes of non-overlapping nets and K, a positive integer.

We construct a complete graph $G = (V, E)$ of t+1 vertices as follows. For every P_i, $1 \le i \le t$, we introduce a vertex $v_i \in V$ in G. For every pair (v_i, v_j) of vertices, we introduce an undirected edge (v_i, v_j). This edge is assigned a weight $w(i, j) = c_{i,j}$, where $c_{i,j}$ is the total amount of crosstalk created by nets in P_i and P_j when they are assigned to adjacent tracks. We now introduce one more vertex v_0 in V and t edges (v_0, v_i) of weight zero into the graph, where $1 \le i \le t$. This completes the construction of G, a complete graph of t+1 vertices. The construction is illustrated in Figure 3.6.



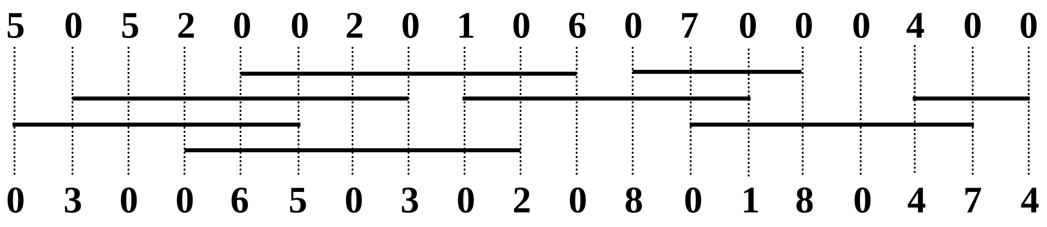
Partition $P = \{C_1, C_2, C_3, C_4\}$,
where $C_1 = \{5, 7\}$, $C_2 = \{3, 1, 4\}$, $C_3 = \{2\}$, and $C_4 = \{6, 8\}$.

(a)



A tour $T = \langle v_0, v_4, v_2, v_1, v_3, v_0\rangle$

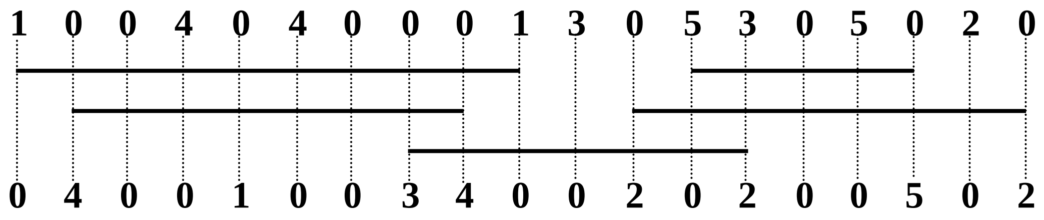
(b)



(c)

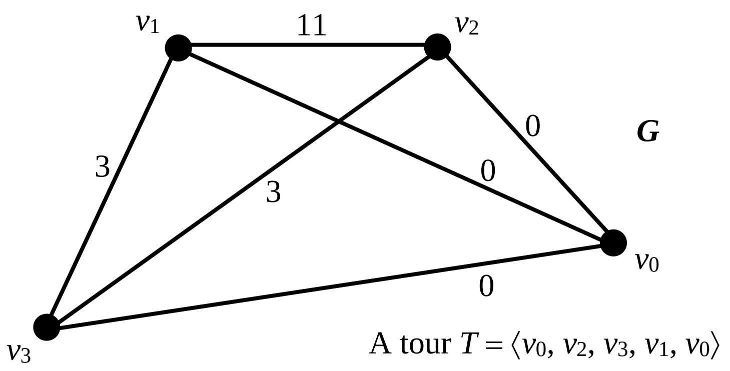
Figure 3.6: (a) A channel specification $I = (N, P, K)$ of *VHP* of eight two-terminal nets. $P = \{P_1, P_2, P_3, P_4\}$, is a partition of P into four non-overlapping class of nets. Nets in P_i can only be assigned exclusively to one track. Note that this is only an input to the problem, not a routing solution. **(b)** The corresponding constructed graph instance $G = (V, E)$ of five vertices of the *TSP* problem. Here v_i is the vertex corresponding to P_i, $\forall i$; v_0 is the other vertex. Weights of all edges adjacent to v_0 are zero. For all other edges, $w(i, j) = c_{i,j}$. A tour $T = \langle v_0, v_4, v_2, v_1, v_3, v_0\rangle$ of the *TSP* problem of the cost of 15 units. **(c)** The assignment of nets corresponding to the tour T, where P_i is assigned to the j-th track from the top if v_i is the $(j+1)$-th vertex in T. This assignment of nets results in a routing solution with exactly 15 units of total crosstalk, the same as the cost of the tour.

In this example, we have considered a channel instance and a partition *P* of nets, as shown in Figure 3.6(a). The corresponding graph *G* for the *TSP* problem is shown in Figure 3.6(b). A tour *T* is also assumed here, and the cost of the tour *c*(*T*) is 15 units. Following the tour (ignoring v_0) of the *TSP* problem, we assign the nets in different classes of *P* of the channel instance to tracks along the height of the channel, as shown in Figure 3.6(c). The total amount of crosstalk of this assignment of nets as a routing solution is 15 units, which is the same as the cost of the tour. Now it is interesting to note the following lemma.



Partition $P = \{P_1, P_2, P_3\}$,
where $P_1 = \{1, 5\}$, $P_2 = \{4, 2\}$, and $P_3 = \{3\}$.

(a)



(b)

Figure 3.7: (a) A channel specification of *VHP* of five nets. Let $P = \{P_1, P_2, P_3\}$ be the given partition of *P*, which is not a solution to the instance. **(b)** The corresponding instance of general *TSP* problem, where the triangle inequality is not satisfied. This is because the cost of edge (v_2, v_3) (i.e. 3 units) plus the cost of edge (v_3, v_1) (i.e. 3 units) is less than the cost of edge (v_1, v_2) (i.e. 11 units).

Lemma 3.3: *For every tour, starting with v_0, of the travelling salesman problem there is a unique |P|-track no-dogleg two-layer VH routing solution, and the amount of crosstalk in this routing solution is same as the cost of the tour.*

Proof: Let $T = \langle v_0, v_1', v_2', \ldots, v_t', v_0\rangle$ be a tour for an instance of the *travelling salesman problem* (*TSP*) obtained with cost $c(T)$. From tour T, if we delete vertex v_0 and its adjacent edges, the cost of the path from v_1' through v_t' remains same as $c(T)$. This is because, from the construction of graph G for a *TSP* instance, the weights of the edges (v_0, v_1') and (v_t', v_0) are zero. Accordingly, we could assign the sets of nets from top to bottom along the height of the channel, obeying t classes of the partition P, such that the nets in a class are assigned to the same track. In this assignment, nets corresponding to v_i' are assigned to track i, $1 \le i \le t$, from the top of the channel. Thus, the nets corresponding to v_t' are assigned to the bottommost track, where $t = |P|$. Hence, a $|P|$-track no-dogleg two-layer VH routing solution is obtained.

Now we prove that the amount of crosstalk of this routing solution is the same as $c(T)$. Here we have just stated how the nets are assigned to tracks following their classes in P, along with the height of the channel. So for two consecutive vertices v_i' and v_{i+1}' in the path obtained (ignoring v_0), $1 \le i \le t-1$, the corresponding sets of nets are assigned to two successive tracks i and $i+1$, respectively, from the top of the channel. According to the construction of the graph, the weight of edge (v_i', v_{i+1}') is same as the total amount of overlapping between the nets in the corresponding classes of P. Hence, the total crosstalk of the routing solution is same as the cost of the path consisting of vertices v_1' through v_t', which is the same as $c(T)$, as the weights of the edges (v_0, v_1') and (v_t', v_0) are zero. ♦

It turns out from the above lemma that *VHP* can be formulated as a general *TSP* problem. Observe that an instance of the *TSP* problem thus obtained may not satisfy the triangle inequality, as explained in Figure 3.7. Also, we know that there is no approximation algorithm for the general *TSP* problem, with ratio error ρ ($\rho > 1$), unless P = NP [13, 28]. Therefore, we have the following theorem.

Theorem 3.5: *Unless P = NP, it is impossible to design an approximation algorithm for the no-dogleg two-layer VH channel routing problem of crosstalk minimization with the partition of nets for simple instances of channel specifications (VHP), with ratio error ρ ($\rho > 1$).*

As the problem of crosstalk minimization with a partition in two-layer channel routing for simple instances of channel specifications is a special case of the general two-layer channel routing problem of crosstalk minimization, we claim the following.

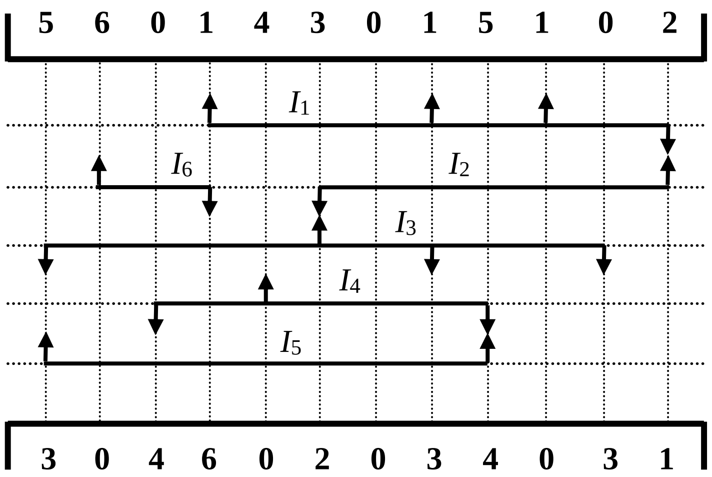
Theorem 3.6: *Unless $P = NP$, it is impossible to design an approximation algorithm for the no-dogleg two-layer VH channel routing problem of crosstalk minimization without any partition of nets for general instances of channel specifications (VHG), with ratio error $\rho\ (\rho > 1)$.*

3.4.4 Hardness of Bottleneck Crosstalk Minimization

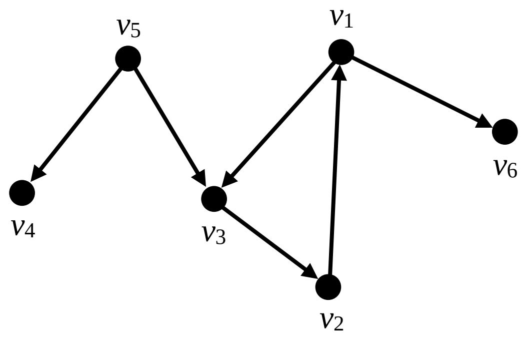
Now we prove that the bottleneck crosstalk minimization problem, *BVHP* is NP-complete. Input here is the same as *VHP*. Let $I' = (N, P, B)$ the input instance of *BVHP*. We need to find if there is a feasible $|P|$-track assignment such that exactly one track is used to assign all nets in P_i and the maximum amount of crosstalk between two nets in (two) adjacent tracks of the solution is at most B. Obviously, a certificate, i.e. an assignment of the track to each P_i can be checked in polynomial time for the existence of nets in an adjacent track having overlap at most B. This proves that *BVHP* is in NP.

To show the NP-hardness of *BVHP*, we use the same reduction as used in *VHP*. Let I be the original instance of HP_P* and $I' = (N, P, B)$ be the constructed channel instance. By Lemma 3.2, I has a Hamiltonian path P of length $n-1$ if and only if the corresponding track assignment of I', as mentioned in the lemma, produces exactly one unit of crosstalk between each pair of adjacent tracks. Let P_i and P_j be the classes of nets assigned to adjacent tracks in this solution. By construction, n_{ij} and n_{ji} are the only nets in these classes that can overlap. Thus, these two nets will produce crosstalk of one unit, and all the other nets will not produce any crosstalk. Therefore, *the instance I will have a Hamiltonian path of length $n-1$ if and only if instance I' has an n-track solution with bottleneck crosstalk 1*. Note that I' has many feasible solutions with bottleneck crosstalk two.

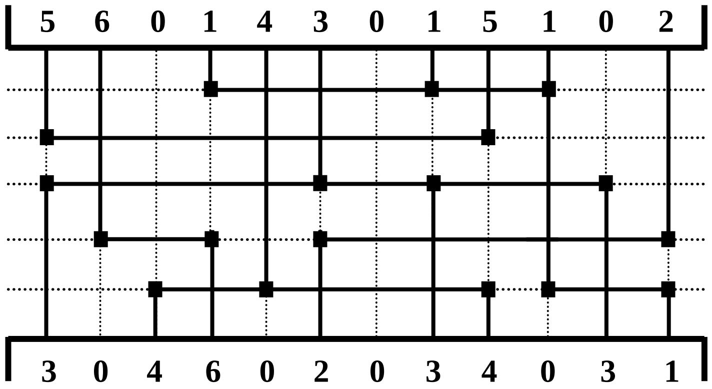
Theorem 3.7: *BVHP is NP-complete.*



(a)



(b)



(c)

Figure 3.8 (a) A channel instance. **(b)** The VCG of the channel instance. **(c)** A restricted dogleg routing solution for the channel instance in (a), where net 1 is doglegged and its horizontal sub-segments are assigned to the first track and the fifth track of the channel, from top to bottom. Vias are also shown, where two orthogonal wire segments of the same net intersect; these are used for changing layers of interconnect.

It becomes obvious that the bottleneck crosstalk minimization problem is NP-complete without any partition of nets for simple as well as general instances of channel specifications.

3.4.5 Hardness of Crosstalk Minimization in Doglegging

So far, we have considered several crosstalk minimization problems in computing a routing solution for two-layer VH channel routing, using the shape of no-dogleg routes only. No-dogleg routing is simple, and it requires a minimum number of vias used for changing layers of interconnect. However, there are instances of CRP for which we may not have a feasible routing solution using only the shapes of no-dogleg routes. One such channel instance is shown in Figure 3.8(a). The VCG of this channel instance contains a cycle comprising vertices v_1, v_3, and v_2 as shown in Figure 3.8(b).

Thus, there is no feasible no-dogleg two-layer VH routing solution in the reserved Manhattan routing model in this case. Rather, in order to compute a feasible two-layer VH routing solution in the specified routing model, we need to split the horizontal span of a net into subnets (or subintervals) for their assignment to different tracks. This is known as doglegging, and the route is known as a dogleg route [15, 49]. Even when a no-dogleg solution exists, use of doglegging may also help in computing a routing solution with a lesser number of tracks (or less channel area), sacrificing a few more vias [15, 49].

Usually, there are two kinds of doglegging within the span of a net. If the route for a net is allowed to dogleg only in those columns in which it contains a terminal, it is called a restricted dogleg route [15, 49]; otherwise, it is known as an unrestricted dogleg route [73]. In restricted dogleg routing, the horizontal span of a net is permitted to dogleg only at columns where it has a terminal. A restricted dogleg two-layer VH routing solution for the channel instance in Figure 3.8(a) is shown in Figure 3.8(c). Here, net 1 is doglegged and assigned to two different tracks, the topmost and bottommost tracks of the channel.

We now consider channel routing with restricted doglegging for the instances with multi-terminal nets. We extend the NP-completeness of the results proved so far to this model by considering no-dogleg routing of instances restricted to two-terminal nets. Now it is easy to see that any problem with multi-terminal restricted dogleg two-layer VH channel routing reduces to two-terminal no-dogleg two-layer VH channel routing, by restricting the number of terminals for each net to two [41, 49]. So all the above results of NP-completeness proved in this chapter using no-dogleg routing imply that the problems of computing minimum crosstalk routing solutions (of given area) using restricted doglegging (with multi-terminal nets) are also NP-complete.

All these results of crosstalk minimization equally hold for the general instances of channel specifications consisting of both horizontal and vertical constraints. This is because the set of instances of channel specifications without any vertical constraint is a proper subset of the set of general instances of channel specifications.

3.5 Summary

In this chapter, we have considered the problem of crosstalk minimization in two-terminal no-dogleg two-layer VH channel routing and have proved that the problem is NP-complete for (i) the simple instances of channel specifications where there is no vertical constraint, and (ii) the general instances of channel specifications with both types of constraints present in it, with a partition of nets so that the nets in a class of the given partition are to be assigned to the same track.

The problem of crosstalk minimization for simple, as well as general instances of channel specifications in two-terminal no-dogleg two-layer VH channel routing, has also been proved NP-complete, even if there is no such partition of nets (so that the nets in a class of the partition are to be assigned to the same track). We have considered the issue of the existence of polynomial time approximation algorithms for the CRP and proved that it is impossible to design such an approximation algorithm. The bottleneck crosstalk minimization problem has also been considered and proved to be NP-complete with and without any partition of nets. In addition, all these problems have been proved NP-hard, even if restricted doglegging is allowed.

As the problem of crosstalk minimization is a hard problem, it is unlikely that a polynomial time algorithm can be developed to solve the problem; rather, devising heuristic algorithm(s) could be a probable solution strategy to solve available channel instances with mostly reduced crosstalk. In this thesis, in Chapter 4, we have developed heuristics for obtaining reduced crosstalk routing solutions. Experimental results based on the heuristics are computed that show a lot of improvement over existing routing solutions of reduced area; these results have been included in Chapter 6 of this thesis. The problem of crosstalk minimization in three-layer and multi-layer channel routing, however, is still an open problem in any routing model.

Chapter: 4

Algorithms for Computing Two-Layer Reduced Crosstalk Channel Routing Solutions

4.1 Overview

In the previous chapter, we have shown that the crosstalk minimization problem for the reserved two-layer (VH) Manhattan channel routing is NP-hard for simple instances of channel specifications (i.e. the channels without any vertical constraint). It remains NP-hard for general instances of channel specifications which involve both horizontal and vertical constraints. In this chapter, we present two heuristics for computing reduced crosstalk two-layer channel routing solutions on given routing solutions of minimum area for simple as well as general channel instances. The performance of our algorithms is encouraging enough for most of the existing benchmark channels, and reduction in crosstalk for these channels is up to 28.34% for a given routing solution. By the way, for all the relevant instances, simple as well as general, generated in the next chapter, both the associated algorithms devised in this chapter have been executed, and results computed are included in Chapter 6.

4.2 Area and Crosstalk Minimization in CRP

Channel routing has been studied extensively in the layout of integrated chips in last four-and-a-half decades. The CRP of area minimization is an NP-hard problem [41, 49, 80, 86]; several heuristic algorithms have been designed for routing channels in different routing models [10, 12, 15, 33, 49, 51, 54, 71, 73, 96]. The problem is polynomial time computable if the instances are free from any vertical constraint and we are interested only in resolving horizontal constraints in the two-layer VH channel routing model [32, 49]. The same algorithm is applicable in computing routing solutions for any channel instances in the $V_{i+1}H_i$ routing models, where $1 \leq i \leq d_{max}$, with alternating vertical and horizontal layers of interconnect [49, 52, 53, 57].

Since the problem of minimizing area for the instances of routing channels without any vertical constraint is polynomial time solvable using only d_{max} tracks, such instances are termed as the *simple* channel specifications. Hashimoto and

Stevens [32] proposed a scheme for solving this problem. According to Schaper [80], it can be implemented in O(n(log n + d_{max})) time, where d_{max} is the channel density and n is the number of nets in the channel. Later on, Pal *et al.* [49, 52, 53] developed and analyzed two different minimum clique cover based algorithms, *MCC1* and *MCC2*, based on the scheme developed by Hashimoto and Stevens [32].

The first algorithm *MCC1* is devised on a graph theoretic approach that runs in O(n + e) time, where n is the number of nets and e is the size of the horizontal non-constraint graph (HNCG) [49, 51, 52, 53, 59], the complemented graph of HCG. The second algorithm, *MCC2* is developed using a balanced binary search tree data structure that takes O(n log n) time for a channel with n nets [49, 52, 53]. Though routing solution of only d_{max} tracks is guaranteed for the simple channel instances in the stated routing models, it may not be a *good* routing solution from the resulting crosstalk point of view.

We reiterate the presence of crosstalk between nets (or intervals) assigned to different tracks in a two-layer channel without any vertical constraint. If two intervals do not overlap, there is no horizontal constraint between them. That is, if there is a horizontal constraint between a pair of nets, there is a possibility of having a *measurable* crosstalk between them. We quantify crosstalk in terms of the number of units a pair of nets overlaps on adjacent tracks in a feasible routing solution.

Consider the problem of minimizing crosstalk in a two-layer VH routing model. Suppose we have three intervals a, b, and c as shown in Figure 4.1(a), in a feasible routing solution of three tracks only.

Since all three nets, a, b, and c overlap, we are compelled to assign them to three different tracks on the same horizontal layer in any feasible routing solution. However, the most interesting feature we can point out is that in Figure 4.1(a), nets b and c share 11 units of horizontal span in the channel, and nets c and a share 2 units; whereas in Figure 4.1(b), we have a net sharing of 4 units of horizontal span in total just by reassigning the nets to tracks. It is inevitable that the assignment of nets to tracks in Figure 4.1(b) produces a reduced crosstalk routing solution; in fact, it is the minimum crosstalk three-track routing solution in this particular case. On the other hand, we identify a channel specification as *general*, if both the constraints are present in it. We now consider the presence of vertical constraint in a channel, and the situation evolved, due to this constraint.

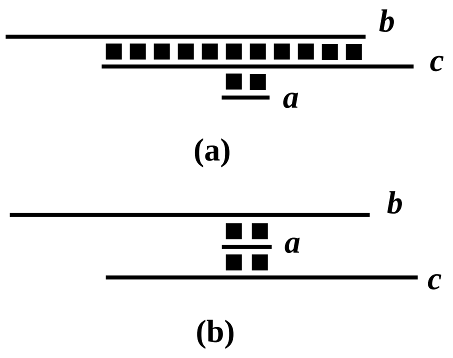


Figure 4.1: Crosstalk minimization problem in two-layer VH channel routing, in the absence of vertical constraints. **(a)** A feasible three-track routing solution with three intervals of three different nets *a*, *b*, and *c* that are overlapping to each other. Nets *b* and *c* share 11 units of horizontal span in the channel (as they are assigned to adjacent tracks), and nets *c* and *a* share 2 units, amounting a total of 13 units' cross coupling length. **(b)** Another feasible three-track routing solution for the same channel instance, with a total net sharing of 4 units of horizontal span; hence, a minimized crosstalk routing solution is obtained just by reassigning the nets to tracks.

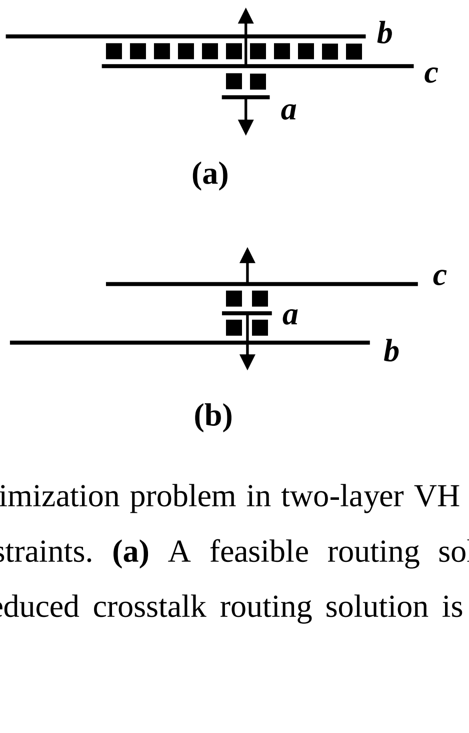


Figure 4.2: Crosstalk minimization problem in two-layer VH channel routing, in the presence of vertical constraints. **(a)** A feasible routing solution with a vertical constraint (*c*, *a*). **(b)** A reduced crosstalk routing solution is gratifying the vertical constraint.

Now, suppose that there is a vertical constraint (*c*, *a*) as shown in Figure 4.2(a), as in some column we have a terminal of net *c* at the top row and a terminal of net *a* at the bottom row. In this case, we cannot alter the sequence of assigning the intervals *c* and *a* in order to compute a reduced crosstalk feasible routing solution, as we did in the case of Figure 4.1. Rather in any feasible routing solution of this instance, we have to assign the interval of net *c* to a track above the track to which the interval of net *a* is assigned. In this case, in order to minimize crosstalk, we can alter the assignment of interval *b* to any of the three tracks conforming the vertical constraint. In fact, a minimum crosstalk feasible routing solution of this example is shown in Figure 4.2(b).

In this chapter, we have developed two heuristic algorithms for minimizing the crosstalk in the reserved two-layer (VH) Manhattan channel routing model, where our intention is to minimize sum crosstalk. The sum crosstalk is the amount of total crosstalk between horizontal wire segments of different nets pair-wise that are assigned to adjacent tracks. The sum crosstalk minimization problem is to compute a feasible routing solution with a given number of tracks in which the total amount of crosstalk is minimized. In Chapter 3, we have proved that the problem of sum crosstalk minimization is NP-hard even if the channels are free from vertical constraints. Thus, developing deterministic polynomial time algorithms for minimizing the crosstalk is implausible. Developing heuristic algorithm(s) is the way out for dealing with the problem.

4.3 Algorithms for Crosstalk Minimization

In the previous section, we have mentioned that the crosstalk minimization problem in two-layer channel routing is NP-hard, even if the channel instances are *simple*. For this reason, we first develop a crosstalk minimization algorithm in two-layer channel routing, where instances are free from any vertical constraint. Then we extend it to two-layer routing with *general* channel instances. Before that, we reexamine the assignment of intervals in order to reduce crosstalk in two-layer (VH) channel routing, as has been illustrated in Figure 4.1, where the amount of crosstalk in Figure 4.1(b) is reduced to 30.77% to that of in Figure 4.1(a). Hence, we have the following observation.

Observation 4.1: *The amount of crosstalk can be reduced if a net (or interval) of a smaller span is sandwiched by two nets (or intervals) of larger spans, or vice versa.*

This observation is the motivation for developing the first algorithm.

NP-completeness proofs, given in Chapter 3, of crosstalk minimization in two-layer channel routing, with and without vertical constraints imply that a polynomial time algorithm for any one of these is unlikely to exist. Furthermore, results obtained in Section 3.4.3 entail that getting an approximation algorithm for the problem is also NP-hard. Hence, it makes sense to design appropriate heuristics for this problem. In this section, we present two heuristics for computing *near optimal* crosstalk routing solutions from a given routing solution of minimum area for simple instances of two-layer CRP. The heuristics have also been subsequently generalized using a novel technique to compute a *near optimal* crosstalk routing solution from a routing solution of given area, for a general instance of two-layer CRP.

We start with a two-layer feasible routing solution S of t tracks and compute another feasible routing solution S' of the same area (i.e. using exactly t tracks) with reduced total crosstalk. For the simple instances of CRP, we take the routing solutions, S, of d_{max} tracks, obtained by the algorithm *Minimum_Clique_Cover_1* (*MCC1*) [49, 52, 53]. On the other hand, for a general channel specification, we start with an existing routing solution obtained from the algorithm *Track_Assignment_Heuristic* (*TAH*) [49, 51].

In the first heuristic, we reassign the nets in P_i, track-wise, in a given routing solution S, so as to reduce the amount of overall crosstalk in computing S'. In the second one, the nets that are reassignable to some other track(s) are shifted such that the total crosstalk is further reduced. Intelligently adapting these reassignments, we are able to design the heuristic for the general instances of CRP.

4.3.1 The First Heuristic: Algorithm *Track_Change*

4.3.1.1 The Basic Approach Used

Let P_i be the set of nets assigned to the i-th track in the given solution S. This heuristic obtains a permutation $P' = (P_1', P_2', \ldots, P_t')$ of $P = (P_1, P_2, \ldots, P_t)$ and assigns P_i' to the i-th track, for all $i = 1, 2, \ldots, t$. Note that the groups of nets obtained in the area minimization problem are not changed here. Reduction in total crosstalk due to such reassignment of net groups can be observed in Figures 3.1 through 3.2. To get the permutation P' (of P), the heuristic starts with the *reduced vertical constraint graph* (*RVCG*), introduced in [49, 56, 58, 59], of the minimum area routing solution S, and in t iterative steps, it reassigns the net groups track-wise from the top to the bottom along the height of the channel.

We often represent vertical constraints by the RVCG, a graph that represents all vertical constraints between groups of nets, where each group has a set of non-overlapping intervals representing a clique in the *horizontal non-constraint graph* (*HNCG*) [49, 51, 52, 53, 59]. Note that an HNCG is the complement of the HCG of a given channel, and a clique of the HNCG corresponds to a set of non-overlapping intervals that may safely be assigned to a track in a routing solution. As the nets in a track of a routing solution S correspond to a clique in the HNCG, the RVCG can be used to represent the vertical constraints among the classes in P. The RVCG under consideration is computed for a given feasible two-layer VH channel routing solution S, and hence acyclic (or a DAG).

The algorithm is ‘greedy’ that works iteratively. In the first iteration, it selects a source vertex (a vertex with indegree zero) in the RVCG whose nets are of the maximum horizontal span. Then it deletes the vertex along with its adjacent edges (if any) from the RVCG. In the i-th iteration, $2 \le i \le t$, the algorithm selects a source vertex, say s, in the current RVCG (i.e. the modified RVCG at the beginning of the i-th iteration) such that the nets in s are best fitted (in terms of reduction in crosstalk) for their assignment to the i-th track from the top row of the channel. After assignment of all the nets in s to the i-th track, s and its adjacent edges, if any, are deleted from the current RVCG and the modified RVCG is obtained for the next iteration.

Clearly, for simple instances of CRP, the number of tracks t in S is the same as the density, d_{max}, of the channel. Further, the RVCG, in this case, contains no edges, as there is no vertical constraint in the channel instance I; therefore, it consists of exactly d_{max} isolated vertices (corresponding to d_{max} disjoint sets of non-overlapping intervals). Thus, we have to handle only the horizontal constraints present in I. The solution technique is similar to the solution of *Interval Containment Problem* (*ICP*) [16, 22]. Therefore, we need to present *ICP* and the algorithm for solving it.

Problem: Interval Containment Problem (*ICP*).

Instance: A set of n intervals such that for any pair of intervals i and j, $1 \le i, j \le n$, either i contains j or j contains i.

Objective: Obtain an ordering of all *n* intervals on a real line, that the total overlapping of the consecutive intervals in the order is the minimum.

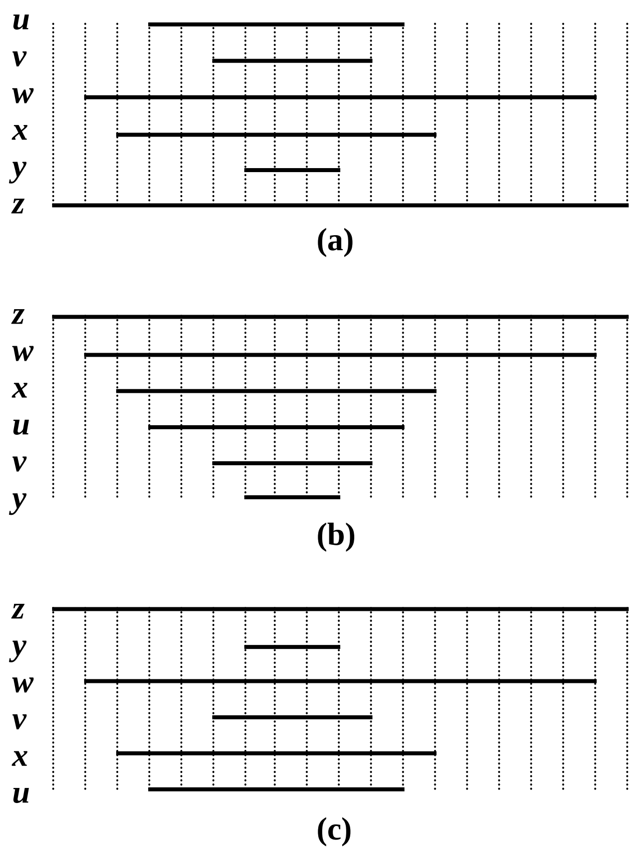


Figure 4.3: **(a)** An instance of problem *ICP* with six intervals *u* through *z*. **(b)** The intervals are sorted based on their spans in descending order. **(c)** A reassignment of intervals to tracks with reduced crosstalk. The sequence of reassignment is as follows: Net with the largest span, net with the smallest span, net with the second largest span, net with the second smallest span, and so on. Here, the amount of total overlapping is 24 units, which is the lowest amount (of overlying based on adjacency of the intervals).

Without any loss of generality, we assume that all 2*n* end points are distinct. To design a polynomial time algorithm for an instance of *ICP*, we perform the following two steps one after the other. First, we sort the intervals based on their spans and then reorder them in such a way that the amount of total overlapping of consecutive intervals in the new order is minimized. To get the intended ordering, we place the interval with the maximum span to the first position, the interval with the minimum span to the second position, the interval with the next to maximum span to the third position, the interval with the next to minimum span to the fourth position, and so on. This is exactly the algorithm that we have followed in reordering the intervals. It is obvious that this ordering minimizes the sum of overlapping of consecutive intervals. This has been explained in Figure 4.3. Therefore, we have the following lemma.

Lemma 4.1: *A minimum cost ordering of intervals of the ICP of n intervals can be obtained if the intervals are assigned to a new order in the following way: (i) The interval with the maximum span is assigned to the first position, (ii) the interval with the minimum span is assigned to the second position, (iii) the interval with the next to maximum span is assigned to the third position, (iv) the interval with the next to minimum span is assigned to the fourth position, (v) and so on. If the intervals are arranged in non-ascending order of their spans, then the amount of total overlapping is same as twice the sum of spans of last (or smallest)* $\lfloor n/2 \rfloor$ *intervals, if n is odd; otherwise, if n is even, that is same as twice the sum of spans of last (or smallest) n/2 – 1 intervals plus the span of the (n/2+1)-th interval.*

Proof: The amount of overlap between two consecutive intervals is same as the span of the smaller one. After reassignment, we obtain an interval with smaller span flanked by two intervals of larger spans, and this is acquired after sorting the intervals based on their spans. Thus, the amount of total overlapping of the consecutive intervals in the new order is as follows.

Clearly, if n is odd, then each of the $\lfloor n/2 \rfloor$ intervals with their smaller relative spans is flanked by a pair of the remaining $\lceil n/2 \rceil$ intervals with their larger relative spans. Therefore, the total overlap of intervals, in this case, is *twice the sum of total span of* $\lfloor n/2 \rfloor$ *smaller intervals.* On the other hand, if n is even, then each of the $n/2–1$ intervals with their smaller relative spans is flanked by a pair of (the elongated) $n/2$ intervals with their larger relative spans, and the remaining interval (i.e. the ($n/2$)-th smallest interval) is assigned to the last position (or below but) adjacent to the ($n/2$)-th longest interval. Thus, the total amount of overlap, in this case, is *twice the sum of spans of last (or smaller) n/2–1 intervals plus the span of the (n/2+1)-th interval* (if the intervals are arranged in non-ascending order of their spans).

Observe that, the total overlap of consecutive intervals obtained by the algorithm matches the lower bound. None of the other orders, evidently, produces total overlap better than the solution obtained. Hence, we conclude the lemma. ♦

The problem of minimizing the crosstalk in CRP becomes the same problem of minimizing the total overlapping in *ICP* when instances are free from any vertical constraint, and the span of every net is contained in the span of some other net. In general, unlike *ICP*, we can have two or more non-overlapping intervals in each class P_i for their assignment to a track in CRP. The solution method used for *ICP* is not expected to produce an optimal solution for the minimum crosstalk CRP. The solution obtained is obviously an approximated routing solution.

In designing a heuristic for the simple instances of CRP, we do the following. Assume a single hypothetical interval I_i for each P_i. The *effective span* of I_i is assumed to be the sum of spans of all nets in P_i. Further, we assume that the amount of overlap between I_i and I_j be $c_{i,j}$, the amount of crosstalk created between P_i and P_j if they are assigned to adjacent tracks. The abstract *ICP* instance obtained is solved using the algorithm mentioned above. If j is the position of I_i in the final *ICP* order, all nets in P_i are assigned to the j-th track from the top. Note that the *ICP* instance obtained here may not satisfy the containment property. However, it can still be used to compute a reduced crosstalk routing solution. This gives us the first heuristic *Track_Change_Simple*. The execution of the algorithm has been explained in Figure 4.3.

The initial partition P_i of nets required by *Track_Change_Simple* is computed using the d_{max}-track routing solution S obtained by executing any one of the algorithms, *Left Edge Algorithm* (*LEA*) [32], or *MCC1* or *MCC2* [49, 52, 53], for a simple instance of channel specification. First, we compute the effective spans of intervals I_i for each P_i. The effective span tells about the use of a track; how much it is occupied or how much it is sparse.

Subsequently, for a d_{max}-track routing solution S, we sort the set of all I_i's in non-increasing order according to their effective spans. If we find two or more classes of nets having the same effective span, they are placed in the sorted sequence in non-decreasing order of their total spans. The *total span of nets* belonging to a class I_i is the span between the starting column of the first net and the ending column of the last net in I_i. Total span tells us how the nets are relatively distributed and/or separated over a track when two or more tracks are there with a same effective span of intervals.

After having the desired sorting of all P_i's, in non-increasing order of their effective span, we reassign them track-wise in the way as it is done in the case of *ICP* (Figure 4.3). Thus, in the final solution obtained, the first two sets of the sorted sequence having large effective spans, sandwich the d_{max}-th set (of least effective span). The second and third sets sandwich the (d_{max}–1)-th set of this sequence, and so on. The sandwich of a track with a less effective span of intervals by a pair of flanked tracks with more effective spans of intervals is absolutely motivated by the geometry of the channel and the initial routing solution provided (or computed by us) as input to run the heuristic. This is all about the algorithm *Track_Change_Simple*, which is designed for a simple instance of CRP.

4.3.1.2 Modification Introduced

Now we introduce some changes in devising the algorithm *Track_Change_General* so that the modified heuristic is able to compute a two-layer reduced crosstalk VH routing solution starting from a (given) two-layer VH routing solution, S of t tracks, for a general instance of CRP. In this case, we need to satisfy vertical constraints in addition to the horizontal constraints present in the channel instance, in order to compute a desired feasible routing solution S'. As a consequence, we are not in a position to compute a sorted sequence of vertices in the RVCG that would certainly lead to a two-layer feasible routing solution (as there are vertical constraints among the nets). In addition, the number of feasible solutions for some given routing solution, S can also be exponential with respect to the number of nets. Thus, instead of computing a sorted sequence, we devise two versions of this algorithm: In the first version, we consider all groups of nets belonging to the set of source vertices in a current RVCG and apply algorithm *Track_Change_Simple* for this set of vertices in isolation, and in the second version, we introduce a balanced binary search tree data structure, both in an iterative-cum-greedy manner. We now briefly describe the first version of the algorithm as follows.

Based on the given routing solution S of t tracks for some general channel instance I, we first compute the RVCG, which is not edge-free now. We know that the RVCG represents vertical constraints between groups of nets, where each group has a set of non-overlapping intervals assigned to a track in S. In other words, the nets in a track in S correspond to a clique in the HNCG, and t is the size of a clique cover of the underlying interval graph (based on the spans of nets as intervals), satisfying vertical constraints present in I.

Thus, (i) the RVCG is used to represent the vertical constraints in S, and it contains exactly t vertices, as S is a routing solution of t tracks, (ii) a vertex in the RVCG corresponds to a set of non-overlapping nets assigned to a track in S, and (iii) there is a directed edge (u, v) in the RVCG, if there is a net $n_i \in u$ and another net $n_j \in v$, such that (n_i, n_j) (or (v_i, v_j)) is a directed edge in the VCG. Obviously, the VCG as well as the RVCG, is cycle-free (or DAG), as S is a feasible two-layer VH routing solution.

The first version of the algorithm iterates for p times, where $p \leq t$ for a given t-track two-layer VH routing solution S. For a feasible S, an acyclic RVCG is there, that must contain at least one source vertex at the beginning of each successive iteration. We may note that if there are $k \geq 1$ source vertices in RVC_1 (i.e. the initial RVCG), all (k groups of) nets belonging to these source vertices are set-wise assignable to the topmost k tracks of S' (i.e. the solution we like to compute with reduced crosstalk). For this, we apply the algorithm *Track_Change_Simple* that has been devised in the previous section, only for the set of k source vertices in RVC_1. Next, we delete all these vertices (along with adjacent edges) from RVC_1 and obtain RVC_2 (i.e. the modified RVCG at the beginning of the second iteration).

Thus, we get a new set of source vertices in RVC_2 to again apply the algorithm *Track_Change_Simple* on this set (of vertices) for computing the desired sequence of the corresponding groups of nets. In this case, it is necessary to check which end set (of non-overlapping intervals) of the computed sequence introduces less crosstalk with the set of intervals already assigned to the k-th track; then from the (k+1)-th track onwards the groups of nets are assigned to subsequent tracks using the sequence computed above, and the RVCG is further updated by deleting the source vertices (along with adjacent edges) in RVC_2 to obtain RVC_3 (if any, for the next iteration). This process is continued till the RVCG is exhausted; that means all the t groups of (non-overlapping) nets are assigned to tracks and a reduced crosstalk routing solution S' is computed. This version of the algorithm is viewed at a glance in the next section.

The method devised for the second version of the algorithm is now described as follows.

This version of the algorithm *Track_Change_General* starts with the RVCG of a given minimum area routing solution, *S* of *t* tracks for a general channel instance *I*, and in *t* iterative steps, track-wise it reassigns the nets from top to bottom along the height of the channel. Let RVC_i be the RVCG at the beginning of the *i*-th iteration, $1 \le i \le t$, and S_i be the set of source vertices in RVC_i. S_i must contain at least one element at the beginning of the *i*-th iteration. This is because the given routing solution *S* is a feasible two-layer routing solution in the specified routing model without any cyclic vertical constraint and the corresponding RVCG is also acyclic. In the *i*-th iterative step, we select a source vertex, say *s*, from RVC_i, so that the nets in *s* are best fitted (in terms of reduction in crosstalk) for their assignment to the *i*-th track from the top row of the channel. After assignment of all the nets in *s* to the *i*-th track, *s* and its adjacent edges (if any) are deleted from RVC_i, and the RVCG for the next iteration is obtained. The algorithm terminates exactly after *t* iterations and computes S', for a given routing solution, *S* of *t* tracks.

4.3.1.3 More on Implementation Details

Now we state how each iterative step of the second version of the algorithm *Track_Change_General* works in computing S'. In the first iteration of this version of the algorithm, we assign the nets corresponding to the source vertex, *s* such that the effective span of intervals of all the nets in *s* among the source vertices in RVC_1 (i.e. the initial RVCG) is maximum. The idea of selecting such a source vertex, *s* for the topmost track is justified by the following fact. Without loss of generality, we may assume that the amount of crosstalk between the nets in the first track and the fixed terminals at the top row of the channel is negligible.

From the second track onwards in successive iterations, we select such a source vertex, *s* from RVC_i for assigning the (non-overlapping set of) nets (assigned to a track) in *S* to the *i*-th topmost track of the channel, $2 \le i \le t$, that renders a minimum amount of crosstalk with the nets already assigned to the $(i-1)$-th track. Now we state how a source vertex, *s* from RVC_i is selected so that the corresponding nets deserve their assignment to the *i*-th topmost track.

To compute *s* efficiently, we maintain a balanced binary search tree data structure among the set of source vertices in RVC_i based on their effective spans of intervals. From this binary search tree, we particularly trace two source vertices having the minimum and the maximum effective spans of intervals. Both elements can be computed in O(log *t*) time. According to this heuristic, the nets either belonging to the source vertex with a maximum effective span of intervals or the source vertex with a minimum effective span of intervals are best assignable to the *i*-th track. This can be computed in a constant amount of additional time by separately computing and comparing crosstalk between the nets already assigned to the ($i-1$)-th track and the nets belonging to two end vertices (in inorder) in the balanced binary search tree amongst the source vertices in RVC_i.

If these two effective spans of intervals in RVC_i are same (this is true only when all the source vertices in RVC_i are having the same effective span of intervals), we compute their total spans of intervals, as defined earlier. It may also happen that a few (two or more but not all) source vertices are having the same effective span of intervals in RVC_i. In that case, we arrange these source vertices in reverse order based on their total spans of intervals within the same balanced binary tree structure.

The computation of total spans of intervals is motivated by considering the design issues of (i) percentage utilization of a track, (ii) congestion of nets over the region near the *i*-th track, (iii) amount of overlapping between the nets in adjacent tracks, and (iv) sometimes, vertical wire length minimization, if situation supports. All these aspects are suitably incorporated in all the phases of the heuristics designed in this chapter, and these are extremely important in synthesizing VLSI physical design from high performance routing point of view. Nonetheless, if the source vertices in RVC_i are differentiated by creating another balanced binary search tree based on their total spans of intervals, we select the vertex that is the best fit (in terms of reduction in crosstalk) to the *i*-th topmost track; otherwise, we assign the nets of any one of them arbitrarily.

It is clear from the heuristics illustrated above that the final solution S' computed using algorithm *Track_Change_General* is a feasible routing solution of exactly *t* tracks, as it always assigns the nets from the top to the bottom of the channel and in each iteration of assigning the nets, it selects a source vertex from the current

RVCG. This completes the design of the heuristic *Track_Change_General*. The steps of both the versions of this algorithm are now presented in the following section.

4.3.1.4 Algorithms at a Glance

In this section, we first view the algorithm *Track_Change_Simple* and then view both the versions of the second heuristic, i.e. algorithm *Track_Change_General*. In each of the cases, the input to the algorithm is a feasible two-layer VH routing solution of a respective channel instance, and the output is a reduced crosstalk channel routing solution. The algorithms at a glance are as follows.

```
Algorithm Track_Change_Simple()
Input: A simple channel instance and a feasible two-layer VH routing solution of the
       channel.
Output: A reduced crosstalk routing solution.
Begin
Step 1: For ($i$ = 1 to $d_{max}$) do
        Begin
        Step 1.1: If ($i \leq \lfloor d_{max}/2 \rfloor$), then
                Assign the $i$-th set of non-overlapping intervals to the ($2i$–1)-th track
                Else
                Assign the $i$-th set of non-overlapping intervals to the $2\times(d_{max}-i+1)$-th
                track.
                End if
        End
        End for
End
```

```
Algorithm Track_Change_General_Version_I()
Input: A general channel instance and the RVCG of a feasible two-layer VH routing
       solution of the channel.
Output: A reduced crosstalk routing solution.
Begin
Step 1: Set $p$ = 1.
Step 2: Set $RVC_p$ = RVCG.
Step 3: Repeat until the RVCG is exhausted
       Begin
       Step 3.1: Select the source vertices in $RVC_p$ and put them in a set $S_p$.
               Let $S_p = \{t_1, t_2, \ldots, t_k \mid$ each $t_i$, $1 \le i \le k$, is a source vertex in $RVC_p\}$.
       Step 3.2: Apply Track_Change_Simple() on set $T_k$ of tracks containing non-
               overlapping sets of intervals in $S$, corresponding to set $S_p$ of vertices in
               $RVC_p$.
       Step 3.3: Assign the sets of nets associated with the vertices in $S_p$ as has been
               sequenced as the output of Step 3.2 (or the reverse sequence) for the
               set $T_k$ of tracks to the current topmost $|T_k|$ (empty) tracks, for which the
               amount of crosstalk is minimum.
       Step 3.4: Delete the set $S_p$ of vertices (along with their edges) from $RVC_p$, and
               modify the graph (i.e. $RVC_p$).
       Step 3.5: Set $p = p+1$.
       End
End
```

Algorithm *Track_Change_General_Version_II*()

```
Input: A general channel instance and the RVCG of a feasible two-layer VH routing
    solution of the channel.
Output: A reduced crosstalk routing solution.
Begin
Step 1: Set $RVC_i$ = RVCG.
Step 2: For ($i$ = 1 to $t$) do
    Begin
    Step 2.1: Construct a balanced binary search tree, $BST_i$ of the source vertices
            in $RVC_i$ based on their (sets of non-overlapping nets') effective spans
            of intervals to tracks (and total spans of intervals, whenever required).
    Step 2.2: Assign a desired set of non-overlapping intervals associated to a
            vertex $v_i$ in $BST_i$ to the $i$-th track.
            Delete vertex $v_i$ along with its adjacent edges, if any, from $RVC_i$.
    End
    End for
End
```

4.3.1.5 Computational Complexity

We first derive the computational time complexity of the algorithm *Track_Change _Simple*. As sorting is the prime task of this algorithm over d_{max} sets of non-overlapping nets based on their effective spans of intervals and if needed, total spans of intervals, the time required by this algorithm is O(d_{max} log d_{max}) in the worst case. Thus, we conclude the following.

Theorem 4.1: *The algorithm Track_Change_Simple computes a feasible two-layer VH routing solution S′with a reduced crosstalk for a given two-layer VH routing solution S of a simple channel instance, where a pair of nets in S′is assigned to the same track that was assigned to the same track in S. The time complexity of the algorithm is O(d_{max} log d_{max}), where d_{max} is the density of the simple channel.*

We now analyze the time complexity of algorithm *Track_Change_General*. As this algorithm (in its first version) executes the tasks of the algorithm *Track_Change_Simple*, so algorithm *Track_Change_General* requires at least the same time that of *Track_Change_Simple*. Note that the value of d_{max} is same as O(n) in the worst case when each set contains a single net (or on an average a constant number of nets). Moreover, the RVCG is computed from the given routing solution, *S* of *t* tracks in O(t + l) = O(n) time, as both t and l are same as O(n), where l is the length of the given channel specification comprising n nets in total. As the size of the set of source vertices in one iteration is O(n), the best fit source vertex, *s* from RVC_i is computed in time O(log n), as these vertices have already been organized in a balanced binary tree (in the second version of the algorithm), whose computation time is O(n log n). There can be at most 2t insertions and deletions in total as every vertex of the RVCG can only be inserted and deleted exactly once in the balanced binary tree data structure. Modification of the RVCG, as a whole, takes O(t) time. Now as the heuristic iterates for t times and t = O(n), the overall computational complexity of the algorithm *Track_Change_General* is O(n log n) time for a general channel specification of n nets, in the worst case. Hence, we conclude the following.

Theorem 4.2: *The algorithm Track_Change_General computes a feasible two-layer VH routing solution S′with reduced crosstalk for a given two-layer VH routing solution S of a general channel instance, where a pair of nets in S′is assigned to the* *same track that was assigned to the same track in S. The time complexity of the algorithm is O(n log n), where n is the number of nets belonging to the channel.*

4.3.2 The Second Heuristic: Algorithm *Net_Change*

Algorithm *Track_Change_General* is efficient enough in minimizing the maximum amount of crosstalk belonging to a given routing solution of a channel by reassigning the nets track-wise, obeying vertical constraints. The initial net grouping is never changed. However, it may so happen that an optimal solution actually corresponds to a different net grouping. Thus, algorithm *Track_Change_General*, presented in the previous section, can further be improved by interchanging nets among different classes in the partition *P*. In the next heuristic, our objective is to interchange a pair of nets assigned to two different tracks only when (i) the nets are horizontally constrained to each other, (ii) the interchange does not introduce any horizontal constraint violation due to overlapping with other nets, (iii) the interchange does not introduce any vertical constraint violation in computing S', and (iv) the resulting crosstalk (after interchanging the nets) is reduced.

4.3.2.1 The Method Used

Notice that the tasks mentioned above are not easy at all. Moreover, we do not know the sequence of interchanging pairs of nets such that a maximum amount of crosstalk can be reduced. Furthermore, a particular net may be interchanged O(*n*) times among the tracks without giving any remarkable gain in overall crosstalk, and making the complexity of the algorithm very high. That is why in this heuristic instead of allowing net-to-net swapping, we shift a net to some other track where a suitable blank space is available and this shifting results in a reduction of overall crosstalk. For some net *x*, if several such shifting is possible, we perform the particular shifting of *x* that maximizes the reduction in crosstalk.

In this heuristic, we sort the nets that are interchangeable (or exchangeable with a blank space in some other track) from left to right in *S* with respect to their starting column positions in the channel. Then we consider such exchangeable nets one after another, and for some particular exchangeable net *x* we search out a track where the net is best fitted in terms of the overall crosstalk minimization without violating any vertical constraint. The sequence of substituting a net with a suitable blank space in some other track plays the most important role in reducing the amount of crosstalk. In general, there are an exponential number of such sequences, and it is not possible to presume all of them to compute an optimal solution. Thus, we consider a constant number of such sequences to allow interchanging of position (of the span or interval) of a net with a blank space on some other track.

Such sequences may be computed in several ways. In our algorithm, we compute the sequences by sorting of exchangeable nets (or blank spaces) (a) from top-left to bottom-right and (b) from bottom-left to top-right in a solved two-layer routing solution, S with respect to their starting column positions in the channel. Similarly, such sequences may also be allowed by sorting of exchangeable blank spaces (or nets) (a) from top-right to bottom-left and (b) from bottom-right to top-left in a solved two-layer routing solution, S with respect to their starting column positions from right to left along the length of the channel. In this context, it is imperative to mention that several such constant numbers of sequences may be identified and allowed to go through this heuristic before we accept the routing solution S' with the least amount of total crosstalk in it. Moreover, this heuristic may repeatedly be executed a constant number of times, if the reduction in crosstalk can further be achieved in each case. Next, we view the algorithm at a glance in the following section.

4.3.2.2 Algorithm *Net_Change* at a Glance

We may note that this version of the algorithm is written in a generalized manner, which is applicable for simple channel instances as well as general channel instances. In any case, we assume a valid two-layer routing solution of t tracks, where in each track there are $N \geq 1$ nets (as a variable). The steps of the algorithm are as follows.

```
Algorithm Net_Change()
Input: A feasible two-layer routing solution, $S$ of $t$ tracks, i.e. a valid assignment of
       tracks for all nets belonging to a channel.
Output: A reduced crosstalk routing solution, $S'$.
Begin
Step 1: For $i = 1$ to $t$ do
            For $j = 1$ to $N$ do
```

```
        Begin
        Step 1.1: Crosstalk = 0.
        Step 1.2: Calculate the crosstalk incurred by net $n_j$ in track $t_i$ due to
                overlapping of $n_j$ at track $t_i$ with the nets assigned at track $t_i$–1 (if $t_i$ is
                not the first track) and track $t_i$+1 (if $t_i$ is not the last track). Store the
                sum of these crosstalk for $n_j$ in $ct_{old}$.
        Step 1.3: If $ct_{old} > 0$, then
                Begin
                Step 1.3.1: Find out the range ($t_p$, $t_q$) of tracks, $p \le q$, such that net $n_j$
                        can be interchanged with a suitable blank span in a track
                        between $t_p$ and $t_q$, both tracks inclusive, and no vertical
                        constraint violation is introduced.
                Step 1.3.2: Find out track $t_s$, $p \le s \le q$, if any, such that the assignment
                        of net $n_j$ to $t_s$ produces a minimum sum of total crosstalk with
                        the nets in its adjacent (upper and lower) tracks, and vertical
                        constraints with other nets are also maintained. Store this sum
                        of crosstalk for $n_j$ in $ct_{new}$.
                Step 1.3.3: If $ct_{new} < ct_{old}$, then remove the horizontal span (i.e.
                        interval) of net $n_j$ from $t_i$ and place it to $t_s$. Vertical wire
                        segments for net $n_j$ are connected accordingly.
                End
        Step 1.4: Crosstalk = Crosstalk + $ct_{new}$.
        End
        End for
        End for
End
```

Without loss of generality, we may assume that the channel instance is simple, and the number of tracks required for such an instance is the same as the density, d_{max} of the channel, and in that case, t is replaced by d_{max} in Step 1 above. In addition, in the case of simple channel instances, there is no question of vertical constraint violation, as the channels are free from any vertical constraint; so this checking is superfluous in such a case (see Steps 1.3.1 and 1.3.2 of the algorithm). Furthermore, if t_i is the first track, we may assume that t_i–1 is the row of top terminals rendering no crosstalk in practice; similarly, if t_i is the last track, we may assume that t_i+1 is the row of bottom terminals rendering no crosstalk (see Step 1.2). Notice that the range (t_p, t_q) of tracks, as mentioned in Step 1.3.1, is applicable for a general channel instance where net n_j assigned to a track t_i is sandwiched by vertical constraints (by other nets that are assigned above and below to n_j); otherwise, for the given routing solution of a simple channel instance, t_p may be assumed as the topmost track whereas t_q as the bottommost track.

4.3.2.3 Computational Complexity

Now we analyze the time complexity of algorithm *Net_Change*. For a given t-track routing solution S of n nets, the heuristic requires O(t) iterations, each requiring O(n) time. Now since t = O(n), the algorithm takes O(n^2) time in the worst case as for each of the O(n) interchangeable nets (or blank spaces) in S, the heuristic searches blank spaces (or nets) in at most O(n) tracks of the given routing solution S of the channel. For the case of simple channel specifications, the worst case complexity is O(nd_{max}), as for each of the n nets (or blank spaces) in the channel we have to search blank spaces (or nets) in at most O(d_{max}) tracks in a given density routing solution, S of the channel. Therefore, we have the following theorem.

Theorem 4.3: *The algorithm Net_Change computes a feasible two-layer VH routing solution with near optimal crosstalk for a given routing solution of a general channel instance in O(n^2) time, where n is the number of nets belonging to the channel.*

4.4 Experimental Results

In this section, we present the performance of our heuristic algorithms. First of all, we have dealt with only the simple instances of channel specifications. Unfortunately, such instances are hardly available in practice. That is why we randomly generate a large number of such instances of channel specifications. We have used all these instances to compute the amount of crosstalk in the routing solutions using the algorithms *MCC1* (to compute the initial crosstalk of the routing solutions), *Track_Change* (to compute the drastically reduced crosstalk routing solutions from the routing solutions computed by *MCC1* [49, 52, 53]), and *Net_Change* (to compute the further reduced near optimal crosstalk routing solutions).

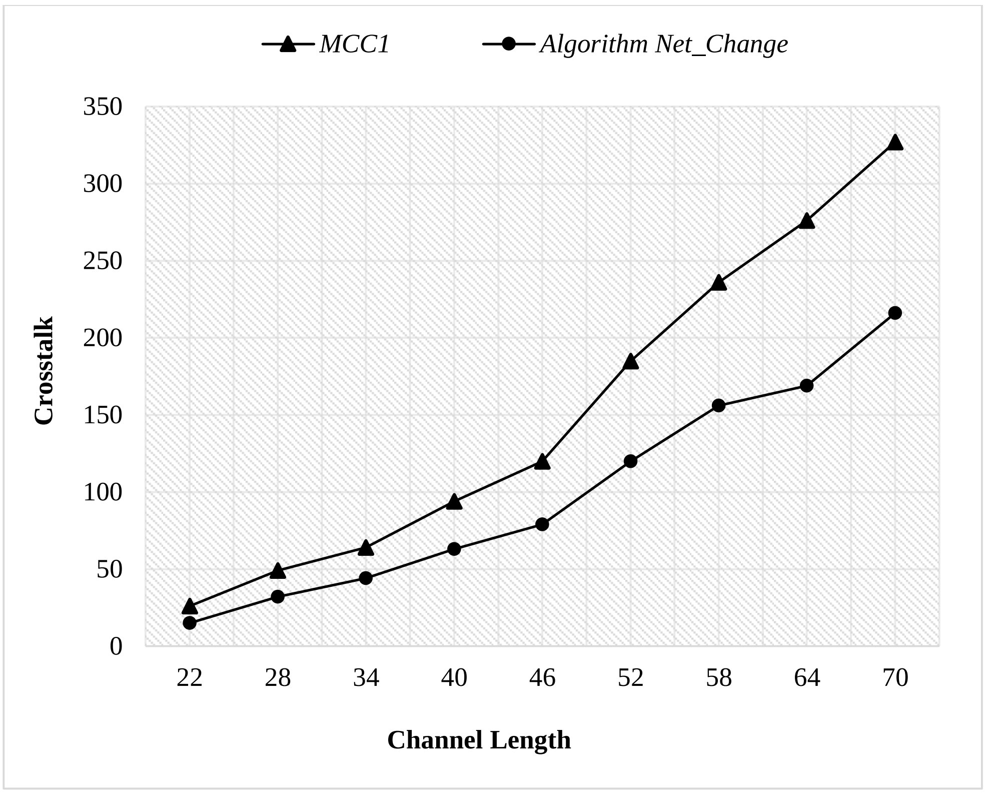


Figure 4.4: Performance graph for crosstalk minimization in channel routing for simple channel instances.

According to the generation of simple channel instances, a channel of length 2*p* has exactly *p* nets, and each column of each of the constructed instances contains a non-terminal either at the top or at the bottom. For each channel length, we have generated 60 random instances, each of which is used to compute crosstalk using the algorithms stated above one after another. The total crosstalk is measured in each case, and an average over the total crosstalk is computed for each of the algorithms for a fixed channel length. These results are included in Table 4.1. The overall reduction in crosstalk using algorithm *MCC1* through algorithm *Net_Change* helps to compute the percentage reduction in crosstalk on an average. The data obtained in the last column of the table are truly interesting. On an average for all the 540 instances of randomly generated channel specifications, the overall reduction in crosstalk is 35.24%. The performance graph in Figure 4.4 shows the variation of crosstalk on an average as the channel length increases, for the routing solutions computed using algorithm *MCC1* through algorithm *Net_Change*. The graph also helps to visualize the amount of overall reduction in crosstalk for the channel instances of the same length using our algorithms.

Table 4.1: Average crosstalk in the computed routing solutions for some randomly generated simple channel instances using different algorithms and percentage reduction in overall crosstalk.

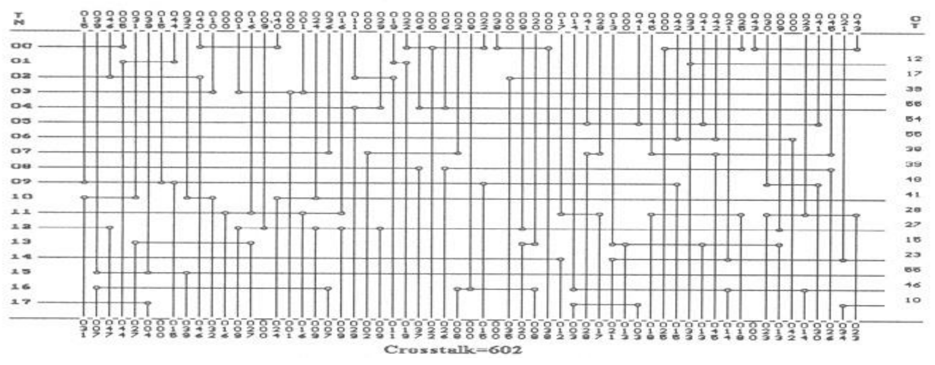
Channel Length	**Amount of Crosstalk**			
	MCC1	***Track_Change_Simple***	***Net_Change***	**% Reduction**
22	26	16	15	42.31
28	49	33	32	34.69
34	64	45	44	31.25
40	94	64	63	32.98
46	120	82	79	34.17
52	185	122	120	35.14
58	236	160	156	33.90
64	276	174	169	38.77
70	327	220	216	33.95

Now, we summarize the performance of our algorithms for several existing channel instances as follows. Here, we have dealt with the general instances of channel specifications, and there are a number of such instances of benchmark channels including the famous *Deutsch's Difficult Example* (*DDE*) [49, 96]. We have considered all these instances to quantify the amount of reduction in crosstalk in the computed routing solutions starting from the two-layer routing solutions obtained using *TAH*, the well-known *Track_Assignment_Heuristic* that is designed for computing minimum area routing solutions [49, 51]. We have considered all two-layer no-dogleg routing solutions computed using *TAH*, as given in Table 4.1 (in page # 102) and shown in different solutions (Figures 4.4-4.17 in pages 103-112) given by Pal [49]. We have assumed all these solutions as the initial routing solutions and executed them using the algorithms developed in this chapter one after another. In other words, we have computed the reduced crosstalk routing solutions for all the aforesaid benchmark examples following a two-phase implementation, where in the first phase solutions are computed through algorithm *Track_Change_General*, and these solutions are subsequently used as input to compute further reduced crosstalk routing solutions following algorithm *Net_Change*. All the results are included in Table 4.2.

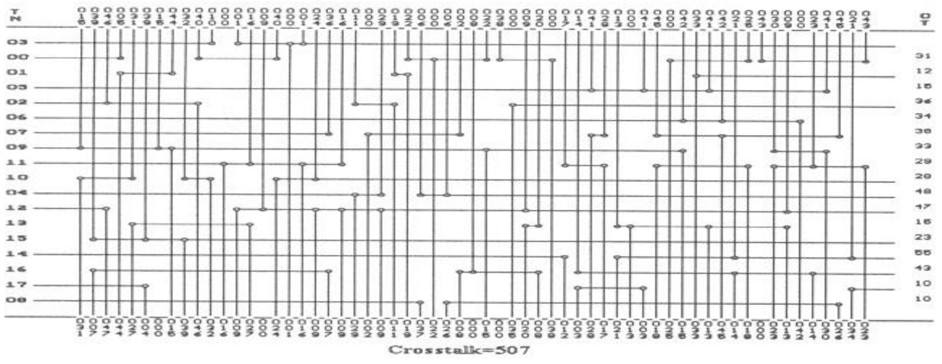
Table 4.2: Amount of crosstalk computed after each of the algorithms and percentage reduction in the overall crosstalk.

Example	**Number of Tracks**	**Amount of Crosstalk**			
		TAH	***Track_Change_General***	***Net_Change***	**% Reduction**
Ex. 1	12	201	201 (0)	196 (1)	02.49
Ex. 2	15	414	397 (1)	396 (1)	04.35
Ex. 3(a)	16	564	519 (1)	506 (1)	10.28
Ex. 3(b)	18	602	507 (1)	502 (1)	16.61
Ex. 3(c)	19	795	771 (1)	732 (3)	07.92
Ex. 4(b)	19	953	901 (1)	845 (2)	11.33
Ex. 5	20	942	724 (1)	675 (2)	28.34
DDE	29	1510	1435 (1)	1181 (3)	21.79
r1	23	1519	1482 (1)	1421 (2)	06.45
r2	20	1071	1034 (1)	1034 (0)	03.45
r3	18	784	728 (2)	690 (3)	11.99
r4	18	1262	1260 (1)	1206 (1)	04.44
Ex. 3(b).1	21	518	504 (1)	393 (2)	24.13
Ex. 3(c).1	18	846	818 (1)	775 (3)	08.39

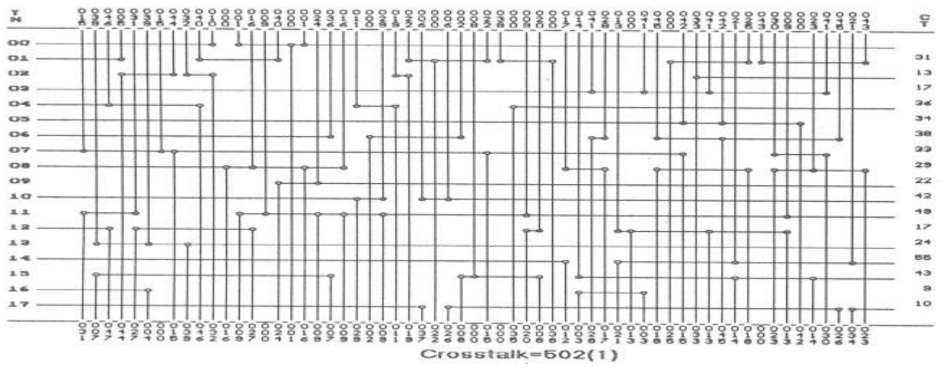
The amount of crosstalk using the algorithm *TAH* is the initial crosstalk [49, 51] as shown in column *TAH*. Each of the relevant columns, *Track_Change_General* and *Net_Change*, shows the computed crosstalk obtained using the corresponding algorithm. The first algorithm provides significantly reduced crosstalk routing solutions from the initial routing solutions computed using *TAH* in the first phase of implementation, and the second algorithm computes the further reduced crosstalk routing solutions based on the routing solutions computed using the first algorithm, in the second phase of implementation of our algorithm. Numbers within parentheses in these columns indicate the additional number of times the corresponding algorithm is executed in obtaining a better routing solution of minimum crosstalk. Percentage reduction column is obtained by computing the overall reduction in crosstalk, starting from the initial routing solution computed using *TAH* [49, 51].



(a)

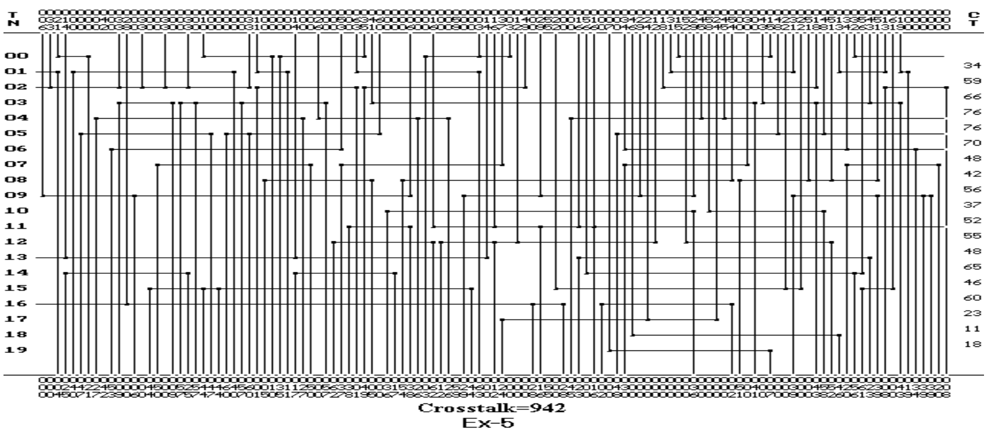


(b)

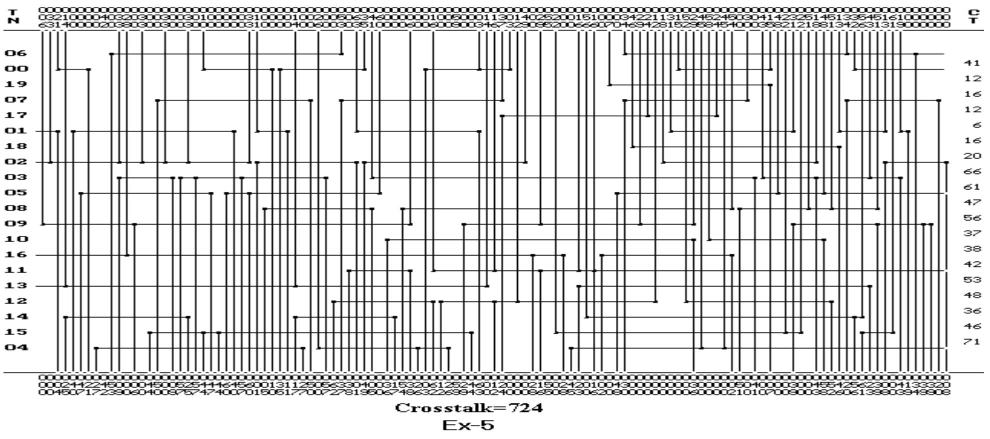


(c)

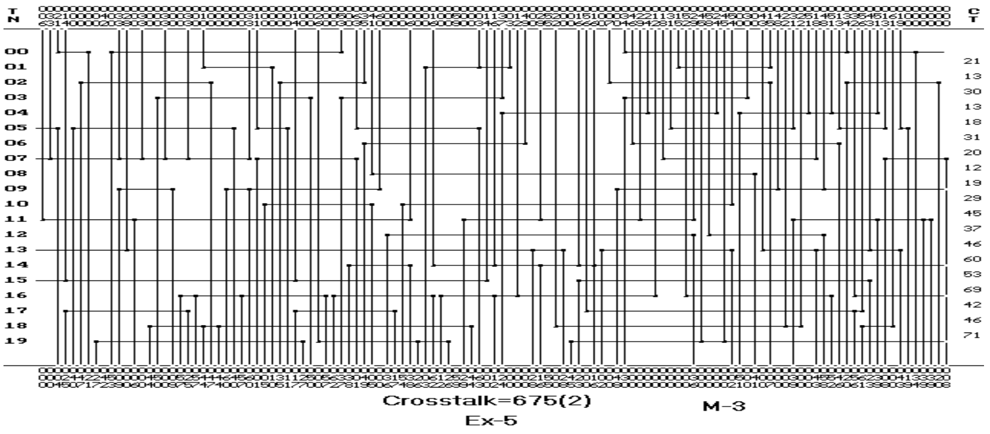
Figure 4.5: (a) A minimum area routing solution for *Ex. 3(b)* using algorithm *TAH* [49, 51]. **(b)** A minimum crosstalk routing solution for *Ex. 3(b)* using algorithm *Track_Change_General*. **(c)** A minimum crosstalk routing solution for *Ex. 3(b)* using algorithm *Net_Change*.



(a)

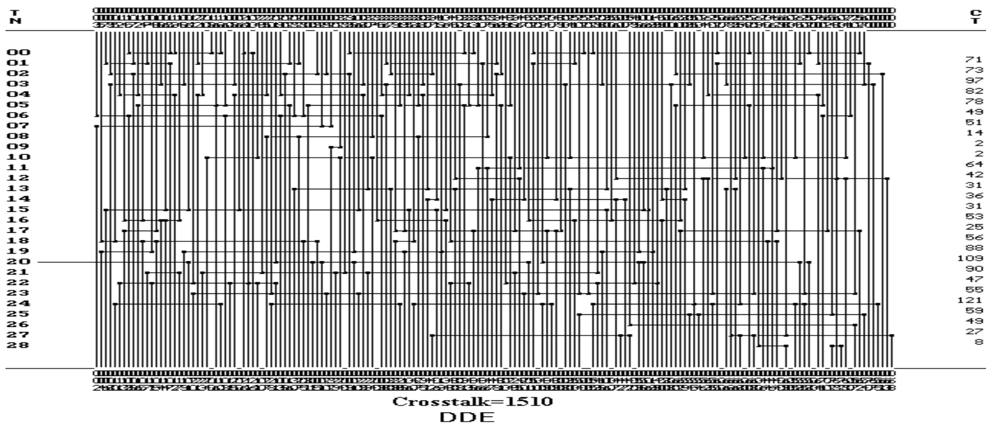


(b)

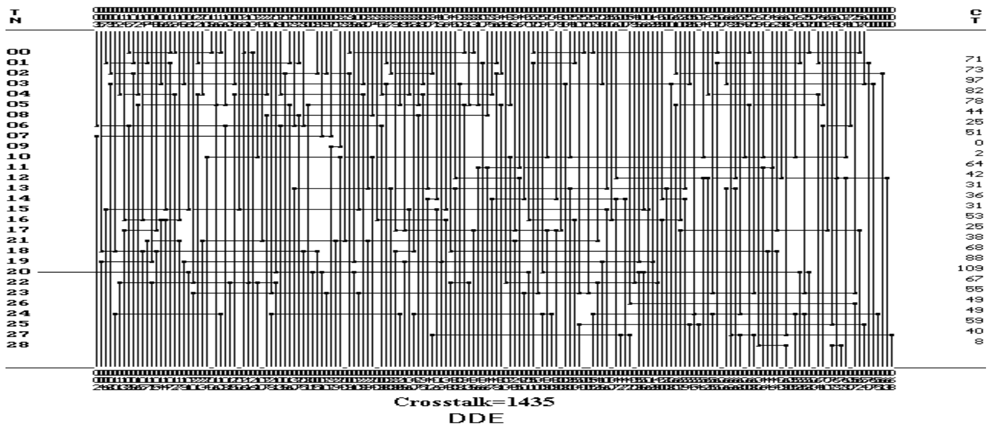


(c)

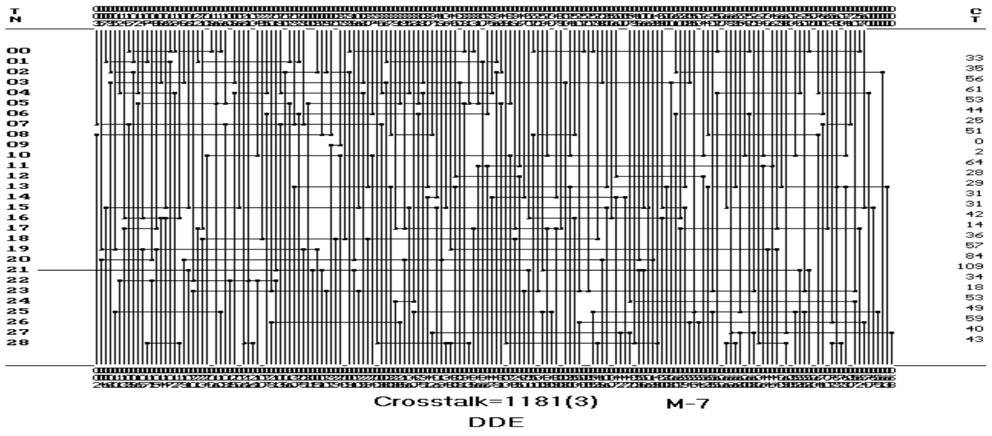
Figure 4.6: (a) A minimum area routing solution for the *Ex. 5* using algorithm *TAH* [49, 51]. **(b)** A minimum crosstalk routing solution for the *Ex. 5* using algorithm *Track_Change_General*. **(c)** A minimum crosstalk routing solution for the *Ex. 5* using algorithm *Net_Change*.



(a)

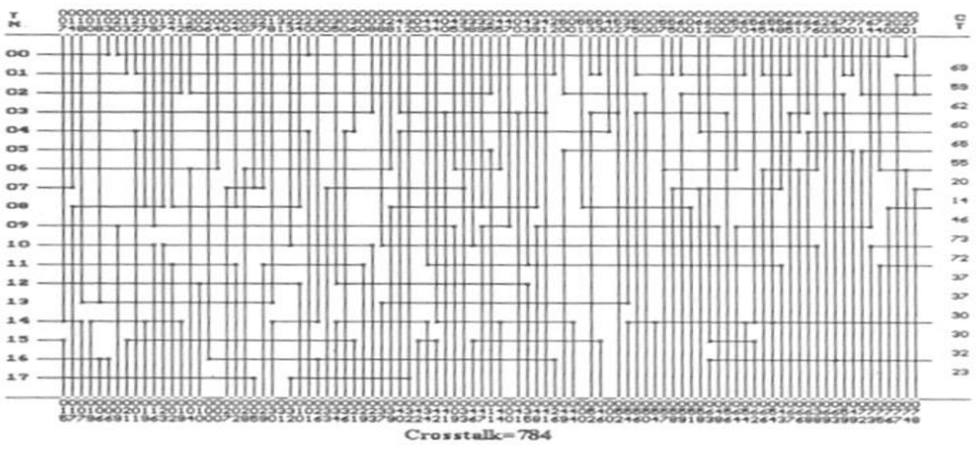


(b)

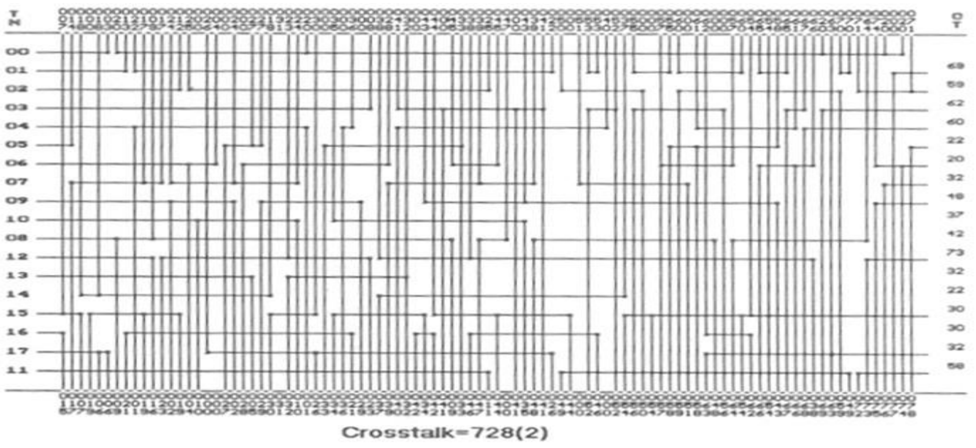


(c)

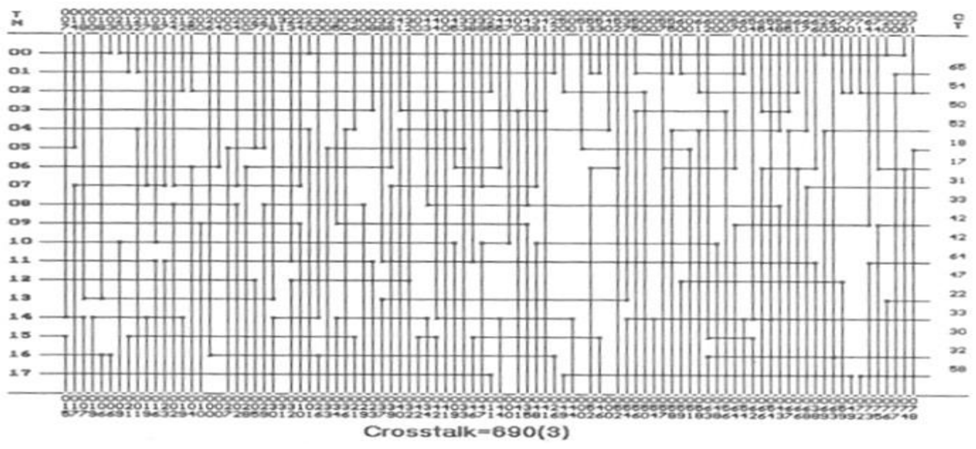
Figure 4.7: **(a)** A minimum area routing solution for the *DDE* using algorithm *TAH* [49, 51]. **(b)** A minimum crosstalk routing solution for the *DDE* using algorithm *Track_Change_General*. **(c)** A minimum crosstalk routing solution for the *DDE* using algorithm *Net_Change*.



(a)

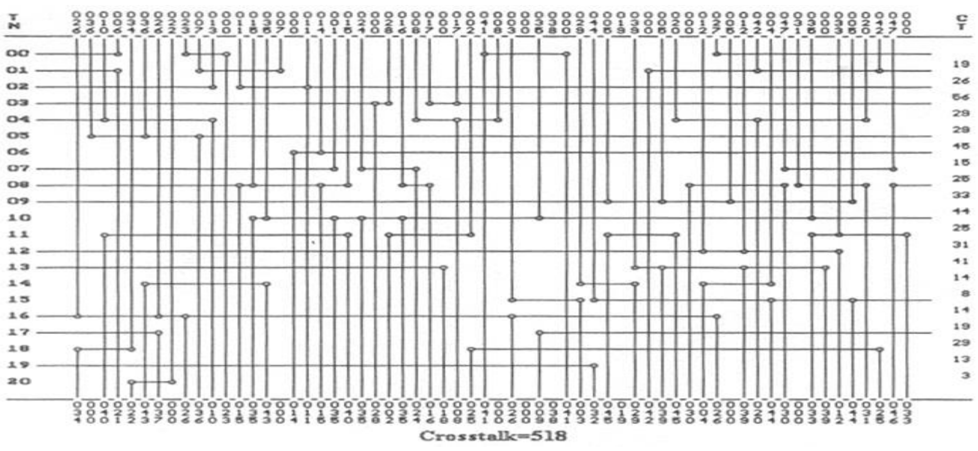


(b)

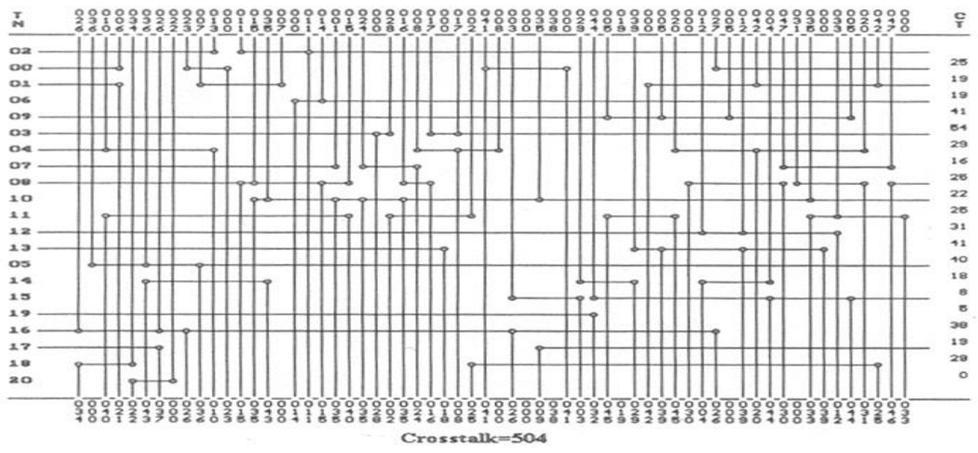


(c)

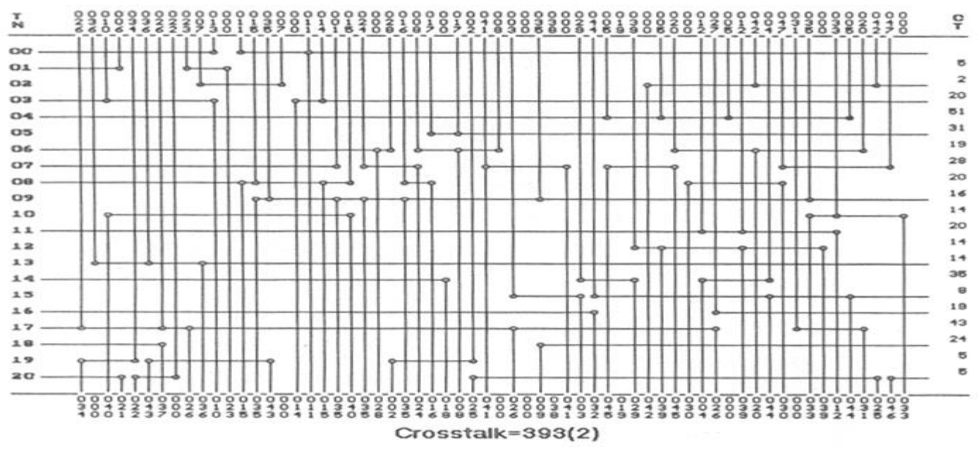
Figure 4.8: (a) A minimum area routing solution for the *r3* using algorithm *TAH* [49, 51]. **(b)** A minimum crosstalk routing solution for the *r3* using algorithm *Track_Change_General*. **(c)** A minimum crosstalk routing solution for the *r3* using algorithm *Net_Change*.



(a)



(b)



(c)

Figure 4.9: (a) A minimum area routing solution for the *Ex. 3(b).1* using algorithm *TAH* [49, 51]. **(b)** A minimum crosstalk routing solution for the *Ex. 3(b).1* using algorithm *Track_Change_General*. **(c)** A minimum crosstalk routing solution for the *Ex. 3(b).1* using algorithm *Net_Change*.

The results obtained here are highly encouraging. For example, in the case of *Ex. 5*, the reduction in overall crosstalk is 28.34%, which is the maximum among the example channels under consideration; see Figure 4.6 for all these routing solutions. For the famous *DDE*, the overall reduction in crosstalk is 21.79%, which is very inspiring and motivating from the point of view of performance driven routing for VLSI circuit synthesis. The routing solutions for the *DDE* are shown in Figure 4.7. Three other sets of similar routing solutions for instances *Ex. 3(b)*, *r3*, and *Ex. 3(b).1* are shown in Figures 4.5, 4.8, and 4.9, respectively. In all these figures, the solution in (b) shows the reassignment of tracks indicating the initial track numbers (*TN*) in the allied solution in (a) to the left of each channel. Besides, the amount of crosstalk (*CT*) between the nets assigned to adjacent tracks is shown to the right of the solutions.

4.5 Summary

In Chapter 3, we have considered several crosstalk minimization problems in two-layer (VH) channel routing and proved that the problems are NP-hard for simple as well as general instances of channel specifications, with or without any partition of nets such that the nets in a class of the given partition are to be assigned to the same track, or there is no such partition. In the same chapter, the issue of the existence of polynomial time approximation algorithms for CRP has also been considered and proved that the design of such an algorithm is also not plausible. The bottleneck crosstalk minimization problem has also been considered, and so on and so forth. Thus, devising heuristic algorithm(s) is a reasonable solution strategy for computing reduced crosstalk routing solutions that has been considered in this chapter.

More specifically, in this chapter, we have devised two prime algorithms (with their variations) for minimizing crosstalk in two-layer channel routing, both for simple as well as general channel instances. As simple instances are hardly available in the literature, we have randomly generated only nine sets of smaller simple channels (using the associated algorithm developed in Chapter 5), each set containing 60 in number, for computing desired routing solutions. For general channel instances, we have presumed two-layer routing solutions of 14 benchmark channel instances existing in the literature, and for each of them, we have executed subsequent crosstalk minimization algorithms for computing mostly reduced crosstalk routing solutions. All these results have been included in this chapter.

By the way, a very large number of simple as well as general channel instances have been generated, executing the instance generators devised in the next chapter, each of which has been considered for computing some initial routing solution, and then carried out for obtaining a reduced crosstalk routing solution in Chapter 6. Truthfully, this is not possible to include all those instances along with all hardcopy routing solutions in this small span of thesis; rather, a very selective number of instances have been included as generated randomly and also a very few routing solutions have been incorporated in Chapter 6 to illustrate the depth of experimentation executed in this thesis. The experimental results based on the heuristics are computed that show lot of improvement over existing routing solutions of reduced area.

Chapter: 5

Algorithms for Generation of Random Channel Specifications

5.1 Overview

In this chapter, we have developed algorithms for generating random channel instances for their use in computing channel routing solutions in VLSI physical design. Channel instances are usually of two types: simple and general, and there are usually two kinds of inherent constraints involving channel routing problem: horizontal constraint and vertical constraint. Simple channel instances do not contain any vertical constraint, whereas, general channel instances contain both horizontal as well as vertical constraints.

Most of the optimization problems in two-, three-, and multi-layer channel routing are NP-hard and, in fact, very few are polynomial time computable. Hence, for each of the NP-hard problems in channel routing, it is unlikely to design a polynomial time deterministic algorithm. Developing heuristic algorithm may be a rational way out that hopefully provides good solutions for most of the instances available in the literature. The novelty of a heuristic algorithm is judged better if it works for a variety of a large number of randomly generated instances of the problem.

5.2 A Review on Channel Instance Generation

Channel routing performs a dominant role in VLSI physical design. The number of active components on a chip has significantly increased nowadays in order to meet the growing demands of functionality. Maximum layout schemes begin with the positioning of modules on a chip and subsequently moves on to wiring terminals, to be electrically linked to separate modules, together. A useful tactic for resolving such a problem is partitioning the chip hierarchically into a set of rectangular channels, subsequently routing each channel one after the other. This successfully splits a difficult problem into smaller and easily solvable (similar) subproblems. Due to its significance from the point of view of layout automation, the channel routing problem (CRP) has been studied comprehensively. With regard to channel routing problem, Deutsch's examples [15, 49, 96] have been extensively used as benchmarks for the purpose of evaluation of the performance of proposed algorithms, with special attention given to the so-called *Deutsch's Difficult Example* (*DDE*) [49, 96]. Researchers have been motivated by the following facts to find ways of generating a large number of random channel instances for use as routing standards.

- The available benchmarks represent a really small subset of real problems. As a result, they ideally, may not represent the existing complexity of the contemporary as well as future designs.
- The number of active components on a chip keeps increasing at a considerably rapid rate. Hence, testing on only traditional benchmarks may prove to be inadequate for the purpose of evaluation of the performances of new channel routing algorithms, proposed by researchers.
- It is quite possible that a proposed algorithm works pretty well for the benchmarks, yet not for other instances.

One of the earlier efforts has been made in [80]. The author in this dissertation has generated a large number of random channel instances with specific characteristics for the purpose of performing experiments on channel routers. The author has used Rivest's random channel generator (rewritten by Eustace in C [20]). The inputs to the channel generator used are as follows:

- n – the number of columns in the channel
- d – the density of the channel
- f – the fraction of used pins
- r – the average number of terminals per net
- a – the Boolean flag controlling production of cyclic vertical constraints
- s – the initial random seed

The fraction of used pins points out the percentage of pins that actually are connected to a circuit. The flag, a, has a value TRUE if no cyclic vertical constraints are required to be generated. As for example, a call to Rivest $\langle n = 150, d = 30, f = 0.75, a = \text{TRUE}, s = 0\rangle$ would produce an output file with vectors TOP, BOTTOM, LEFT, and RIGHT for a channel with 150 columns and density 30. The channel would have 75 percent of its pins connected to circuits, and there would be no cyclic vertical constraints.

In the article [11], the authors have developed a random channel routing generator. The proposed system is capable of generating difficult channel routing instances of random size. The authors have also introduced and explained the major constraints on a CRP. The proposed algorithm is able to generate CRP instances which can be routed without any doglegs. The authors have suggested that the proposed algorithm should prove to be useful for the purpose of testing the performance of new algorithms for channel routing. Moreover, due to the arbitrary size of the generated CRPs, they quickly become intractable with the increase in the number of nets. As a result, exhaustive search techniques become infeasible. The authors have stated that for some of the generated examples, there is a significant difference between the optimal solution and the traditional lower bounds. The authors have also indicated that consideration of the interaction of constraints is significant for the purpose of developing channel routing algorithms.

In another article [5], the authors have tackled the same problem of random channel routing instances but with the help of a genetic algorithm. The authors have suggested that for all the generated cases, they have found better specifications (or channel instances) compared to well-known existing benchmarks. The random channels generated in this article have been claimed to be difficult to route due to them having higher horizontal and vertical constraints. For example, compared to the *DDE* [49, 96] having 72 nets and 174 columns, the five random channels generated by the authors all have been shown to be more difficult to route. The authors have claimed that the proposed algorithm can also be extended to generate even more difficult channels, those that cannot be routed using two-layer no-dogleg routing algorithms and can only be routed using two-layer dogleg, or three- or multi-layer routing algorithms.

5.3 A Prelude to the Generation of Random Channel Instances

In this chapter, we are interested in generating random channel specifications for channel routing problem. To do so, we first generate channel specifications having two-terminal nets only, where none of the channel specifications contains any vertical constraint. Such channel specifications that do not have any vertical constraint are known as *simple channel specifications*. Then we modify our requirement for computing *general* channel instances with multi-terminal nets so that we can generate channel specifications containing two- or more than two-terminal nets; also containing vertical constraints.

We know that the two-layer channel routing problem of area minimization is polynomial time solvable for simple channel specifications [32, 49, 52, 53], but the problem of crosstalk minimization in two-layer channel routing is NP-hard even if the channel specifications are free from any vertical constraint. So, it is unlikely that there exists a polynomial time algorithm for computing two-layer minimum crosstalk channel routing solutions even for the *simple* instances of channel specifications. Incidentally, such channel instances are hardly available in the literature. A very few channel instances are available in the literature, usually known as *benchmark channel specifications*, each of which contains both horizontal as well as vertical constraints [49, 96].

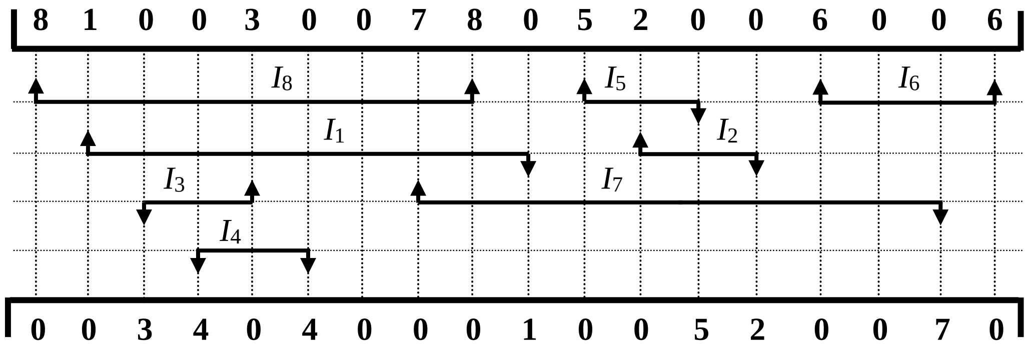


Figure 5.1: An example channel of eight nets; zeros are non-terminals or vacant terminals, not to be connected. Intervals of the nets are placed in four different tracks. Terminals are vertically aligned along the columns of the channel. The length of the channel is 18. Arrows indicate that the terminals to be connected, either at the top or at the bottom, to complete the required interconnection of all the nets present in the channel.

A *channel specification* is usually obtained in the form of two *m*-element vectors TOP and BOTTOM indicating the *top* and *bottom terminal lists*, respectively, of a channel. We often may use *channel specification* and *channel instance* interchangeably. A *channel* is a rectangular routing region containing two sets of fixed terminals on two of its opposite sides, say top and bottom, and the other two sides, the left and right sides of the rectangle, are open ends. The left and right sides of the rectangle may contain terminal(s) of net(s), but the terminal position(s) is (are) not fixed before having a routing solution. The fixed terminals at the top and the bottom are (usually) aligned vertically in *columns*. The set of terminals that need to be electrically connected together is called a *net*. A channel specification with a set of eight two-terminal nets in a channel is shown in Figure 5.1; the length of the channel is 18.

In order to obtain simple channel specifications, in this chapter, we develop an algorithm for generating instances of such (simple) channel specification in random. There are several problems in channel routing that are hard in nature even if the channel specifications are free from any vertical constraints; such randomly generated simple channel specifications can be utilized to compute (desired) routing solutions while evaluating the performance of the heuristic algorithms designed for those problems. In the next section (i.e. in Section 5.4), we formulate the steps necessary for developing the algorithm to generate simple routing channel instances. Then we generalize the algorithm in generating general channel routing instances in Section 5.5. All the experimental results computed for generation of simple as well as general channel instances are included in Section 5.6. Usefulness in developing such algorithms in Sections 5.4 and 5.5 are briefly described in Section 5.7. In Section 5.8, we summarize the chapter with a few remarks.

5.4 Generation of Simple Channel Specifications

Let us assume, the number of nets be n in a channel that is being generated. So the channel length would be of $2n$ if no *blank column* (containing no terminal at the top as well as at the bottom) or *trivial column* (containing both the terminals of the same net) is introduced into the channel. In generating random channel specifications of simple in nature, we introduce a non-terminal (i.e. a vacant terminal) either at the top or at the bottom randomly, and the other side of the column would contain a terminal of a net. Our next task is to obtain the starting and the ending column positions of a two-terminal net, and on which side it would be (i.e. to obtain the terminal position of the net), at the top or at the bottom, randomly along the length of the channel. In this respect, we like to fix up and maintain a set of criteria while generating the random channel specifications, as follows.

- Though it is needless to mention that the net numbers are nothing but symbols to differentiate themselves, we like to obtain channel instances where the net numbers would present randomly along the length of the channel, i.e. the nets are not sorted in succession based on their starting column positions from left to right or from right to left (along the length of the channel) or something of that sort.

- Nets should be of different spans (or intervals), and they would present randomly along the length of the channel. If L_i (R_i) is the leftmost (rightmost) column position of net n_i, then $I_i = (L_i, R_i)$ is known as the *span* (or *interval*) of the net. As for example, the span of net 7 in Figure 5.1 is 9, and that of net 5 is 2. This criterion tells that all the smaller (or larger) nets are not accumulated on a side along the length of the channel.

- Generally, in practice, a channel contains smaller nets in large number and larger nets a few. Nets are defined smaller or larger based on their relative spans (or intervals) over the length of the channel. However, the number of nets with a fixed span (say, 5 nets of span 3 each, 2 nets of span 7 each, etc.) is not fixed; otherwise, the randomness of the channel to be generated may suffer. Only we can say that there would be a large number of smaller nets and a few larger nets, but we do not precisely specify the exact spans of different nets or the number of nets of a fixed span. Obviously, the number of nets with some intermediate spans is neither more nor less. Essentially, this criterion helps us in generating random channel instances where the number of nets gradually reduces as their spans increase along the length of the channel.

In order to generate the desired channel specifications, as guided by the criteria stated above, we now describe the channel specification generating procedure as follows. Let us assume the channel instance we want to generate is C that contains n nets, and its length is $m = 2n$ (as each of the n nets is a two-terminal net and we are not introducing any blank column or trivial column into C). We can simply generate two column numbers randomly for each net, and immediately place them at the top or at the bottom of the selected columns randomly. The problems that may arise in this method are that of generating the same number (as a net) more than twice (that means the net that has already been assigned) and the same column position c in C, $1 \leq c \leq m$, is accessed for assigning net j, which has already been occupied by net i, $j \neq i$, after starting the procedure (that can be considered as a *collision*). Finding out of a *free* column (where no net has yet been assigned) in some iteration might enhance the cost of the procedure (in terms of computational complexity) beyond some limit, and even if we use *linear probing technique* for such *collision resolution* (like *hashing*), it may result in some channel specification, which may not satisfy one or more criteria, mainly the third criterion described above.

Thus, generation of random column numbers for the nets is not a problem at all; even different methods for generating random numbers are easily available in the literature, among which we can use any one of them. The problem gets apparent whenever a collision occurs because we cannot predict anything about the generation of random numbers. To get rid of this collision situation, we may think of the problem in a different way so that randomness remains unaffected.

5.4.1 Formulation of the Problem

In the preceding part of this section, we formulate the problem towards developing the algorithm to generate simple channel routing specifications, based on the overview of generating the same as discussed earlier. The channel specification may be thought as a linear *list* of two-tuple elements, where each element corresponds to a column in the channel (that is being generated) and the two tuples correspond to the top and the bottom terminal positions at each column. The length of the lists is same as the length of the channel (say m), where n-number of nets are to be introduced when the channel is completely generated. For each list element we generate a net number; put it in one of the two-tuples, i.e. either at the top or at the bottom, and the other side would contain a non-terminal.

At the very beginning, all the elements of the linear list of the two-tuple assumed above are initialized with zeros. In our algorithm, we replace $2n$ ($= m$) zeros in total in this two-tuple list, in order to generate a desired simple channel instance, where columnwise only one zero is randomly replaced by a net number. In order to generate such a channel specification, we assume another list of length m, which is same as the length of the channel specification that is being generated. Initially, the list contains the natural numbers starting with zero onwards (i.e. 0 through m–1), i.e. each element i in the list contains the number i–1, $1 \leq i \leq m$.

We consider a singly linked linear list L of m nodes and initialize the *information field* of node i ($L(i) \rightarrow info$) by i–1, where $1 \leq i \leq m$. In other words, the linear list L contains the column numbers 0 through m–1 of channel C to be generated. Initially, all the columns are free, as we have not yet assigned any net to a column. If some net is assigned to column c, $0 \leq c \leq m-1$, then the node with information c is deleted from L and keeps L as the list of free or available columns for the nets yet to be assigned. For each net i, $1 \leq i \leq n$, we randomly choose a node N from the existing list L of free nodes, retrieve its information part, which is a free column number in C, and store i in the free column. Thus, if N contains c in L, we replace c by i and delete N from L. In effect, channel C would contain a terminal of net i at column c. Furthermore, in identifying the side (TOP or BOTTOM) of the channel, we choose another random number that signifies whether i is assigned at the top or at the bottom of c so that the other side of c is allotted for a non-terminal (i.e. 0). We treat this terminal as the position of the *initial terminal* of net i.

The problem of selecting a particular node in L may also arise for multiple times, if we do not delete the selected node from L. That is why we assign the current net to the column obtained by randomly selecting a node in L as stated above and delete the node from L. In the same iteration (i.e. for net i) we generate the *final terminal* for net i too.

While generating the position of the *final terminal* of net i, $1 \leq i \leq n$, we have to keep in mind the last criterion of generating random channel specification stated above (in Section 5.4), which is relating to spans of nets to be introduced in C. Here our intention is to generate many smaller nets and a few nets with larger spans. In order to obtain a random channel specification, we generate some smaller (positive) random numbers that would be added to or subtracted from column number c that is already selected randomly as the initial column position of net i. At first, we generate smaller random numbers and assign the nets with lesser spans along the length of the channel. Gradually, L becomes sparse as the nets are assigned to C, and even for randomly generated smaller numbers nets with larger spans are evolved during later iterations.

As every time a new node with a new column number (as its information part) is selected, we can always be able to use the column successfully for a new net without any conflict, and in this way, the problem of multiple occurrences of the same random number could be resolved.

5.4.2 The Algorithm and Its Complexity

This is the way how a random channel C of n two-terminal nets is generated, where the length m of the channel specification is same as $2n$, and C contains no vertical constraint in it. The algorithm based on the tentative formulation of the problem (towards developing the algorithm) described in the previous subsection, is given below.

```
Algorithm: Simple_Random_Channel_Generator
Input: Number of nets ($n$) to be introduced into the channel ($C$) to be constructed.
Output: A random simple channel specification of length $m = 2n$.
Begin
Step 1: Set $free\_node = m$. // It is the variable that keeps the number of nodes (i.e.
    free columns) currently existing in $L$.
    Create a singly linked linear list ($L$) of $m = 2n$ nodes. Each node $i$ has two fields,
    where $L(i)$^info contains the column number of the channel to be constructed, and
    $L(i)$^link contains the address of the next node.
Step 2: For ($i = 1$ to $m$) do
    Begin
    Set $L(i)$^info $= i-1$
    End
Step 3: Select a suitable random number generator, Random(), to generate the
    random numbers and initialize the generator by a valid (may be a varying
    quantity) seed value.
Step 4: For ($i = 1$ to $n$) do
    Begin
    Step 4.1: Set initial = Random(free_node).
    Step 4.2: Search $L$ linearly, and get the content of $L(initial)$^info
        Set initial_col = $L(initial)$^info.
    Step 4.3: Delete node $L(initial)$.
        Set $free\_node = free\_node - 1$.
    Step 4.4: Select any suitable function to determine $max\_offset$.
        Set $offset = Random(max\_offset)$.
    Step 4.5: If ($initial + offset \le free\_node$), then
            Set $final = initial + offset$
        Else
            If ($initial - offset \ge 0$ and $initial - offset \le free\_node$), then
                Set $final = initial - offset$
            Else
                Set $final = Random(free\_node)$
            End if
        End if
    Step 4.6: Search $L$ linearly, and get the value of $L(final)$^$info$.
        Set $final\_col = L(final)$^$info$.
    Step 4.7: Delete node $L(final)$.
        Set $free\_node = free\_node - 1$.
    Step 4.8: $initial\_col$ and $final\_col$ contain the two terminals of the net under
        consideration (i.e. net $i$). Select top/bottom randomly, and assign $i$ to these
        two columns.
    End
End
```

Now we compute the computational complexities in generating such a random channel specification C as follows. In C, we have n nets in total. In addition, m is also of the order of n. Initially, we create a singly linked linear list L of order n. For each net i, $1 \le i \le n$, to be introduced into the channel, we randomly select a node N from the existing list L of free nodes and retrieve its information part to determine the initial terminal position of net i in a column in C. This linear search on the linked linear list L (of length $2n$) takes time $O(n)$. At the same iteration, we also determine the final terminal position of net i in a similar way, and that also takes time $O(n)$. So for each net i, $1 \le i \le n$, $O(n)$ time is required for finding out its terminal positions (in two different columns) in generating C. Thus, for a channel C that contains n nets, the total time required is O(n^2). Hence, the time complexity in generating *C* is O(n^2). Needless to mention that in generating *C*, we have used O(*n*) dynamic space of computer memory. We summarize the complexity results in the following theorem.

Theorem 5.1: *Algorithm Simple_Random_Channel_Generator randomly generates simple channel specifications of only two-terminal nets without introducing any vertical constraint in it. The algorithm takes time O(n^2), and space O(n), where n is the number of nets introduced into the channel.*

Now we illustrate the algorithm in generating a simple channel specification in the following subsection.

5.4.3 An Illustration

In this subsection, we illustrate the algorithmic procedure developed in this chapter and motivate the steps while creating a simple channel specification, *C*. Let us consider *C* would contain n = 10 nets. Therefore, the length of the channel, *m* is supposed to be 20. Now we create a singly linked linear list, *L* of length 20 (that means *L* contains 20 nodes), where node *i* is initialized with number i–1, $1 \leq i \leq 20$. Hence, the initial list *L* with its information fields in successive nodes is as follows, which are same as the column numbers of the channel instance *C* under construction.

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19

For the sake of generating channel instance *C*, initially we assume that *C* contains only non-terminals, and the initial node numbers contain all the nodes as shown below.

Initial Node Numbers:	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Remaining Column Numbers:	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
TOP:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
BOTTOM:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

In the overview in Section 5.1, we have discussed that we are intended to introduce a large number of smaller spanned nets and a small number of larger spanned nets (and a few neither larger nor smaller in their spans). In the algorithm, developed in this chapter, we have introduced the nets in increasing order, 1 through *n*. So, naturally the smaller numbered nets are also smaller in their spans and larger numbered nets are eventually larger in spans. Though net numbers are having no special significance other than differentiating themselves, in order to break this usual trend in generating channel specifications, we follow a statistical measure assumed as follows.

In Table 5.1, the percentage of free columns (of the channel to be generated), i.e. the percentage of free nodes available in the node list is introduced in the left column. The right column contains the allowable percentage of remaining free nodes for computing the maximum offset (i.e. *max_offset*) of a net under consideration in an iteration. The maximum offset is a number that determines the maximum span (or interval) of the net that at most we like to provide for the net under consideration. Offset (i.e. *offset*) is a randomly generated number in the range of zero through *max_offset* that we essentially employ as the span (or interval) of the allied net. After obtaining the terminal locations in two different columns for a net, we again generate two random numbers; if the number is odd (even) for a column, then at that column we assign the terminal position at the top (at the bottom) for the net under consideration.

Table 5.1: An assumption on the allowable percentage of remaining free nodes for computing *max_offset*, based on available free nodes in the list (or available free columns in the channel) in percentage.

Percentage of Free Nodes in the List (or Percentage of Free Columns in the Channel)	**Allowable Percentage of Remaining Free Nodes for Computing *max_offset***
91 – 100	0 – 10
81 – 90	0 – 20
71 – 80	0 – 30
61 – 70	0 – 40
51 – 60	0 – 50
41 – 50	0 – 60
31 – 40	0 – 70
21 – 30	0 – 80
11 – 20	0 – 90
1 – 10	0 – 100

Now we briefly describe the basis of introducing *max_offset*. *Max_offset* is the maximum allowable range of span (or interval) of a net. The *offset* is a randomly generated number within the range of *max_offset* (starting from zero). Thus, certainly a smaller numbered net may have a span 10% (as *offset*) of the total length of the channel to be generated, whereas a larger numbered net may have a span of 0% (i.e. a minimum of one unit span) of the same (at some other iteration, as *offset*).

Table 5.1 is self-explanatory when we consider a channel of length 100 in determining the allowable range of *offset* (i.e. zero through *max_offset*) for a net in subsequent iterations of the algorithm. This has been shown in Table 5.2. Based on our assumption in Table 5.1, the distribution of the allowable *max_offset* follows a normal curve.

Now, we go back to our illustration when the channel specification under generation of length 20 had only non-terminals at the top as well as at the bottom. The first iteration starts with net number 1. Say, *initial* node, randomly selected, is 4. The content of this node is 3, i.e. the column number of the initial terminal is 3. It is a free node and is assigned as the initial terminal of net 1. Now the node (i.e. node 4) is deleted from *L*, and after deletion of node 4, the status of the list (i.e. *L*) is as follows.

Table 5.2: Allowable range of *offset* for a typical channel of length 100, based on available free nodes in the list (or available free columns in the channel).

Free Nodes in the List or Free Columns in the Channel	**Allowable Range of *Offset* (0 – *max_offset*)**
91 – 100	0 – 10
81 – 90	0 – 18
71 – 80	0 – 24
61 – 70	0 – 28
51 – 60	0 – 30
41 – 50	0 – 30
31 – 40	0 – 28
21 – 30	0 – 24
11 – 20	0 – 18
1 – 10	0 – 10

Remaining Node Numbers:	1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
Remaining Column Numbers:	0 1 2 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19

Now the list has 19 free nodes. We take a variable *free_node* to keep this free node count. Each time when a node is deleted from the list, the value of *free_node* is decremented by 1. According to Table 5.1, the *max_offset* of the node to be selected randomly for the *final* terminal of the net is 10% of 19 free nodes, which belongs to the range of percentage of free nodes 91 – 100, which is same as 1.9. Taking the floor of this value, we get the *max_offset* that equals to 1. (Taking the ceiling may work equally good or another for this experimentation in computing the value of *max_offset*.)

Based on this value of *max_offset*, the allowable range of *offset* is 0 – 1. Say, the number selected randomly for computing the final column position of net 1 is 0 (from the range of the above *offset*). To determine the final column position for net 1, we add 0 to 4, which is the initial node number for this net. So, the final column position of net 1 is the content of the fourth remaining free node of list *L*, which is also same as 4. So, the span of net 1 is from column (number) 3 to column (number) 4. Node 4 is deleted from the list (i.e. *L*), and variable *free_node* is decremented to 18. The exact terminal positions for net 1 at these columns are selected randomly, as stated above. Say, the terminal position at column 3 is at the top (as an odd number is randomly generated) and at column 4 is at the bottom (as an even number is randomly generated). The generated channel is currently as follows.

Column Number:	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
TOP:	0	0	0	**1**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
BOTTOM:	0	0	0	0	**1**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Now the status of the remaining free nodes and the remaining column numbers are as shown below.

Remaining Node Numbers:		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
Remaining Column Numbers:	0	1	2	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	

Say, in a similar way we have executed six more iterations in assigning nets 2 through 7 into the channel being constructed, and the generated channel is currently as follows.

Column Number:	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
TOP:	0	0	3	1	0	7	2	0	0	0	0	0	0	0	4	0	0	6	4	5
BOTTOM:	7	2	0	0	1	0	0	3	0	0	0	6	0	5	0	0	0	0	0	0

Furthermore, let us consider the status of the remaining free nodes and the unassigned column numbers, before introducing net 8, are as follows.

Remaining Node Numbers:	1	2	3	4	5	6
Remaining Column Numbers:	8	9	10	12	15	16

The remaining three nets are 8, 9, and 10, and the number of free columns is six. Then for net number 8, following the algorithm, say, *initial* = 4. Therefore, *initial_col*, i.e. the initial column number for net 8 is 12. Now the available *free_node* is five, and the list of remaining column numbers does not contain column number 12. Next, in a similar way, we compute the *max_offset* for net 8, which is same as four. So, the range of *offset* is 0 – 4. Say, three is randomly selected as the value of *offset*. Thus, *final* is found by adding three to *initial*, i.e. *final* = 4+3 = 7, which is greater than the number of the available free nodes (i.e. 5). In this case, we subtract *offset* from *initial* to get the other node. Therefore, *final* = 4–3 = 1, and *final_col* = 8. So, selecting sides (top or bottom) for the terminal positions of net 8 randomly and after assigning them to the channel, the generated channel specification is as follows.

Column Number:	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
TOP:	0	0	3	1	0	7	2	0	0	0	0	0	**8**	0	4	0	0	6	4	5
BOTTOM:	7	2	0	0	1	0	0	3	**8**	0	0	6	0	5	0	0	0	0	0	0

Before consideration of net 9, the status of the remaining free nodes and the column numbers where nets have not yet assigned, are as shown below.

Remaining Node Numbers:	1	2	3	4
Remaining Column Numbers:	9	10	15	16

Two nets (9 and 10) are still to be assigned, and there are four free columns available for these assignments. Say, *initial* = 4, and hence, *initial_col* = 16. Node 4 is deleted; so, *free_node* = 3. The *max_offset* calculated is 2; say, 0 is generated as *offset*. Hence, *final* = 4+0 = 4, which is, in turn, greater than the value of *free_node* available. Thus, we follow the first *Else*-part of Step 4.5 of algorithm *Simple_Random_Channel_Generator*. Even if we go backward, i.e. subtract 0 from 4, the value of *final* remains unchanged, and therefore, we do not get any valid node as node 4, as currently, the list is containing only three free nodes. In such cases, where both addition of *offset* to the *initial* and then subtraction of *offset* from the *initial* result in invalid nodes (as *final* > *free_node* as well as *final* < 1), we may select any node from *L* randomly, which gives a valid node and a valid column number. Say, the randomly selected node for *final* is 3. Thus, node 3 gives column number 15; hence, *final_col* = 15. Say, the side for the initial terminal of net 9 is randomly selected as the top, and that of the final terminal is randomly selected as the bottom. After assignment of net 9, the channel is as follows.

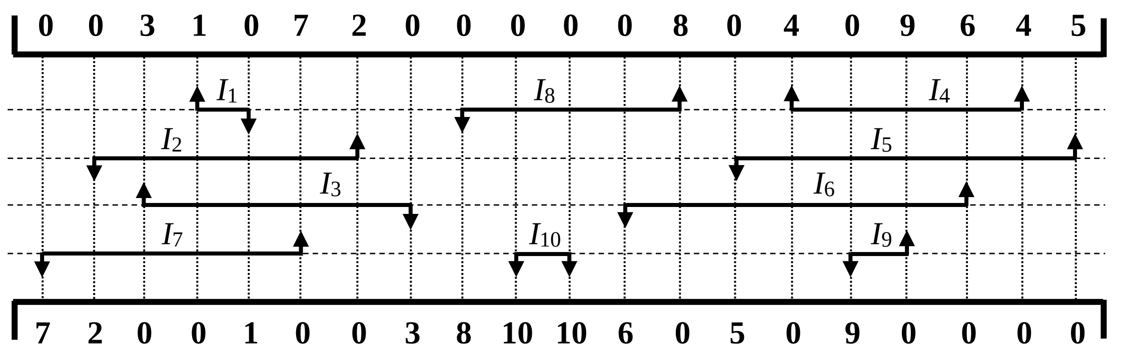
Column Number:	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
TOP:	0	0	3	1	0	7	2	0	0	0	0	0	8	0	4	0	**9**	6	4	5
BOTTOM:	7	2	0	0	1	0	0	3	8	0	0	6	0	5	0	**9**	0	0	0	0

For net 10, i.e. the last net to be introduced only two columns are free of the channel being constructed. The status of the remaining free nodes and the unassigned column numbers, before introducing net 10, are as follows.

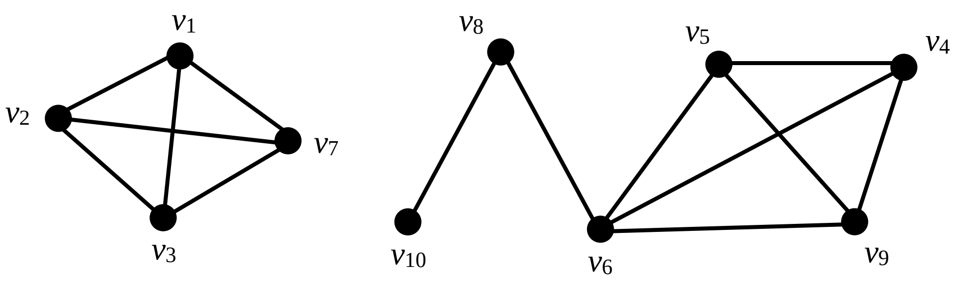
Remaining Node Numbers:	1	2
Remaining Column Numbers:	9	10

This is the case where only two free columns remaining to be assigned and the *L* has exactly two nodes (node 1 and node 2). Then we randomly compute the initial column number for this net, and compute *max_offset*, randomly select *offset*, and compute so on and so forth for the final column position of net 10, as stated above. As at the beginning of this iteration, we had only two free columns, so for the assignment of terminal positions for the last net (i.e. net 10), eventually we must assign the terminals in these two columns only; of course, sides of these two terminals are again selected randomly. The finally generated channel specification is as follows.

Column Number:	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
TOP:	0	0	3	1	0	7	2	0	0	0	0	0	8	0	4	0	9	6	4	5
BOTTOM:	7	2	0	0	1	0	0	3	8	10	10	6	0	5	0	9	0	0	0	0



(a)



(b)

Figure 5.2: (a) The generated simple channel specification that contains 10 nets; the length of the constructed channel is 20. **(b)** The horizontal constraint graph of this (generated) simple channel instance comprises two components, as none of the nets in this channel instance passes through both columns 8 as well as 9.

Now we analyze the channel specification generated above. No doubt that this channel specification is a simple channel specification, as in each column of this channel a non-terminal is present either at the top or at the bottom. Nets with their spans (or intervals) belonging to the channel are shown in Figure 5.2(a); Figure 5.2(b) shows the horizontal constraint graph of this channel. In a *horizontal constraint graph*, *n* vertices are introduced corresponding to *n* nets belonging to a channel, and two vertices are connected by an edge if and only if the spans (or intervals) of the associated nets overlap in the channel [49, 96]. Table 5.3 shows how the nets in the channel generated are spanned over the length of the channel, as the net number increases. Nets are also randomly stretched over the length of the channel. In addition, the table shows the leftmost column position and the rightmost column position of each of the nets that are randomly generated and introduced into the channel. From this table, it is clear that the spans of the nets also do not increase as the net numbers increase; rather *max_offset* as well as *offset* play their roles in obtaining the desired random spans (or intervals) of the nets to be introduced into the randomly generated channel specifications.

Table 5.3: Spans (or intervals) of different nets introduced into the randomly generated channel specification, shown in Figure 5.2(a).

Net Number	Leftmost Column Position	Rightmost Column Position	Span (or Interval)
1	4	5	1
2	2	7	5
3	3	8	5
4	15	19	4
5	14	20	6
6	12	18	6
7	1	6	5
8	9	13	4
9	16	17	1
10	10	11	1

5.5 Generation of General Channel Specifications

5.5.1 An Overview

In the previous section, we have described how to generate random channel instances, containing only two-terminal nets, and there are no vertical constraints between any pair of nets. In general, (benchmark) channel specifications that are found in literature may have vertical constraints and nets may contain more than two terminals, i.e. there may be one or more than one terminals in between two end terminals of a net. Thus, the procedure described above is applicable only to some special type of channel specifications that are known as *simple channel instances.* However, we can tune the procedure explained above, so that it becomes capable of generating channel instances having vertical constraints and nets may contain more than two terminals. In generating such general channel specifications, we assume that there is an upper bound on the number of terminals and this number to be no more than six for the few nets that are introduced into the channel being constructed.

5.5.2 Formulation of the Problem

Suppose, we like to generate a random channel instance that consists of N nets ($N > 0$). So, the only input to be given here is the number of nets, i.e. N. Each net is assumed to be of k terminals ($2 \leq k \leq 6$). We need to place these N nets along the length of the channel. Also, there may be some non-terminals spread over the channel that should be incorporated in the specification. In order to generate a random general channel specification, first, we have to estimate the length of the channel (L) in a way such that it is long enough to accommodate proposed terminals of all the nets as well as non-terminals. If k_i is the number of terminals of net i, then the total number of terminals for all the nets can be computed by adding the number of terminals for each net. For example, if there are 10 nets in the channel then the total number of terminals for 10 nets is $K = k_1 + k_2 + \ldots + k_{10}$. If NT is the number of non-terminals, then the total number of terminals is $P = K + NT$. These P terminals are evenly distributed over top and bottom positions of the channel specification along the length of the channel, in a random fashion.

Now we can calculate the estimated length of the channel as $L = \lceil P/2 \rceil$. As an illustration, say, there are five nets having two terminals each, three nets having three terminals each, and there are a 5-terminal and a 6-terminal net of a general channel specification to be generated of ten nets. Also, say, the number of non-terminals (NT) is seven. Thus, the total number of terminals is calculated as follows:

$P = K + NT = (2\times5 + 3\times3 + 5\times1 + 6\times1) + 7 = 30 + 7 = 37.$

Hence, $L = \lceil P/2 \rceil = 19$.

So, the channel of length L is now capable of accommodating K terminals and NT non-terminals. In general, there are more nets having fewer terminals, and the random distribution of terminals should follow this constraint. Now, we have to locate the terminals of the nets at top/bottom positions in different columns in a random fashion. In order to place them properly, we have used three auxiliary lists, described as follows.

1. *NET_LIST*: The length of this list is also N. Initially, the list is filled in with positive integers from 1 to N. An instance of the list is given below.

NET_LIST: 1 2 3 4 5 6 7 8 9 10

This list tells that the nets to be introduced into the random general channel specification that is to be generated.

2. *TERM_COUNT*: The length of this list is *N*, i.e. same as the number of nets. Each location of this list is used to store the number of terminals (*k*) for a net. For example, if *N* = 10, then *TERM_COUNT* may be looked like as:

TERM_COUNT: 2 2 2 2 2 3 3 3 5 6

This means that there are five nets of two terminals each, three nets of three terminals each, and two nets having five and six terminals only.

3. *TERM_LIST*: The length of this list is 2*L*, i.e. same as the sum of the total number of top plus total number of bottom terminal positions. Initially, the list is populated by positive integers started from 1 up to 2*L*. For the above example, as 2*L* = 38, so *TERM_LIST* must have positive integers from 1 to 38 as shown below.

TERM_LIST: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38

Now, it is the time to describe how a channel specification, *C* is generated with the help of these three auxiliary lists. At the very beginning of generating a desired random general channel specification, we initialize all the 2*L* top and bottom terminal positions of the channel by zeros. Then we start with generating the channel specification in *N* successive iterations. In each iteration, we select a net number, *i*, randomly and locate its terminals in different columns (*c*) in *C*, and again randomly determine the terminal position, either at the top or at the bottom in column *c* for net *i*. First, we randomly select an element *p* from *NET_LIST* and remove the element from the list immediately; the length of *NET_LIST* is reduced accordingly. Thus, *p* is the net, whose terminals are to be assigned to different columns in *C*. Now, an element *k* from *TERM_COUNT* is selected randomly, and the entry is deleted, and the size of this list is also reduced as before. Thus, *p* is considered as a *k*-terminal net. Say, we have selected the sixth entry from *NET_LIST* and the ninth entry from *TERM_COUNT*; so *p* = 6 and *k* = 5, i.e. *p* (or net 6) is a 5-terminal net, whose terminal positions are to be determined now.

For each of these k terminals we generate a random number, r and select the r-th element (which we denote by another number s) from *TERM_LIST* and delete the element as before, which results in a reduction of the length of *TERM_LIST*. The number s is now located at column $c = \lceil s/2 \rceil$; if s is odd (even), then p (the net under consideration) is placed at the top (bottom) position in column c. When all the terminals of net p are assigned in different columns, we are left with *TERM_LIST* of length reduced by k, i.e. the length of *TERM_LIST* is same as $2L–k$, at the beginning of the second net/iteration under consideration.

In this way we extract elements from the three lists and using them, we fill in the top and bottom positions of column c, and after N iterations, we obtain the desired channel specification. Note that based on the initialization of the channel specification of length L (with all terminal positions containing only zeros), ultimately $2L–K$ terminal positions are left unassigned with some net terminals; so, these terminal positions contain only non-terminals for the generated channel specification. A formal description of this algorithm is stated in step-by-step in the following subsection.

5.5.3 The Algorithm

This is the way how a random general channel instance of multi-terminal nets is generated, where the length of the channel specification is L. The algorithm based on the tentative formulation of the problem (towards developing the algorithm) described in the previous subsection, is given below.

```
Algorithm: General_Random_Channel_Generator
Input:  Number of nets ($N$) to be introduced into the channel to be constructed.
Output: A random general channel specification as a vector (Channel) of length $L$.
    Top (Bottom) list is denoted by Channel.top (Channel.bottom).
Begin
Step 1: [Determination of length $L$ of channel $C$ to be generated]
    Step 1.1: For each net, $i$ ($1 \le i \le N$), randomly estimate a number $k_i$ as the
        number of terminals.
        Set $K = k_1 + k_2 + \ldots + k_N$.
    Step 1.2: Randomly estimate the total number of non-terminals ($NT$).
        Set $P = K + NT$.
    Step 1.3: If $P$ is odd, then
```

```
        Begin
        Step 1.3.1: Set $NT = NT + 1$.
        Step 1.3.2: Set $P = P + 1$.
        End
    Step 1.4: Set $L = P/2$.
Step 2: [Initialize TERM_COUNT, NET_LIST, and TERM_LIST]
    Step 2.1: For $i = 1$ to $N$ do
        $TERM\_COUNT[i] = k_i$.
    Step 2.2: For $i = 1$ to $N$ do
        $NET\_LIST[i] = i$.
    Step 2.3: For $i = 1$ to $P$ do
        $TERM\_LIST[i] = i$.
Step 3: [Create the channel specification]
    Step 3.1: For $i = 1$ to $L$ do
        Begin
        Step 3.1.1: Set $Channel[i].top = 0$.
        Step 3.1.2: Set $Channel[i].bottom = 0$.
        End
    Step 3.2: For $i = 1$ to $N$ do
        Begin
        Step 3.2.1: Randomly select an element $p$ from NET_LIST.
            Delete $p$ from NET_LIST, and reduce the length of NET_LIST by 1.
        Step 3.2.2: Randomly select an element $k$ from TERM_COUNT.
            Delete $k$ from TERM_COUNT and reduce the length of TERM_COUNT.
        Step 3.2.3: For $j = 1$ to $k$ do
            Begin
            Step 3.2.3.1: Randomly select an element $s$ from TERM_LIST.
                Delete $s$ from TERM_LIST and reduce its length.
            Step 3.2.3.2: Set $c = \lceil s/2 \rceil$.
                If $s$ is odd, then
                    $Channel[c].top = p$
                Else
                $Channel[c].bottom = p$
            ***End***
        ***End***
***End***
```

Now we illustrate the algorithm in generating a general channel specification in the following subsection.

5.5.4 An Illustration

Let us illustrate the algorithm with the help of the example considered in Section 5.4.3; see Figure 5.2(a). Here, the number of nets (N) = 10, the number of terminals (K) = 30, the number of non-terminals (NT) = 8. Thus, P = 38 and the length of the channel (L) = 19. The initial state of the three auxiliary lists and the status of the channel to be generated are as shown below.

NET_LIST	: 1 2 3 4 5 6 7 8 9 10
TERM_COUNT	: 2 2 2 2 2 3 3 3 5 6
TERM_LIST	: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38

Column #	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
TOP	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
BOTTOM	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Let us assume, at the first iteration, the sixth entry from *NET_LIST* is selected randomly, which is 6, and the ninth element is randomly selected from *TERM_COUNT*, which is 5. So, p = 6 and k = 5. Therefore, net 6 is a 5-terminal net. Now we need to select randomly five elements from *TERM_LIST*, one after another; those are required in knowing the columns where net 6 has its terminals. Say, the selected elements from *TERM_LIST* are 5, 12, 15, 22, and 25. Thus, the column positions for the terminals of net 6 are calculated as follows. The first column position of net 6 is $c = \lceil 5/2 \rceil = 3$, and as 5 is an odd number, this terminal of net 6 is assigned to the top at the third column of the channel. Similarly, other four terminal positions for net 6 are column 6 at the bottom, column 8 at the top, column 11 at the bottom, and column 13 at the top. The lists are modified accordingly. After the first iteration, the status of the three lists and the generated channel after introducing net 6 are as follows.

NET_LIST	:	1 2 3 4 5 7 8 9 10
TERM_COUNT	:	2 2 2 2 2 3 3 3 6
TERM_LIST	:	1 2 3 4 6 7 8 9 10 11 13 14 16 17 18 19 20 21 23 24 26 27 28 29 30 31 32 33 34 35 36 37 38

Column #	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
TOP	0	0	**6**	0	0	0	0	**6**	0	0	0	0	**6**	0	0	0	0	0	0
BOTTOM	0	0	0	0	0	**6**	0	0	0	0	**6**	0	0	0	0	0	0	0	0

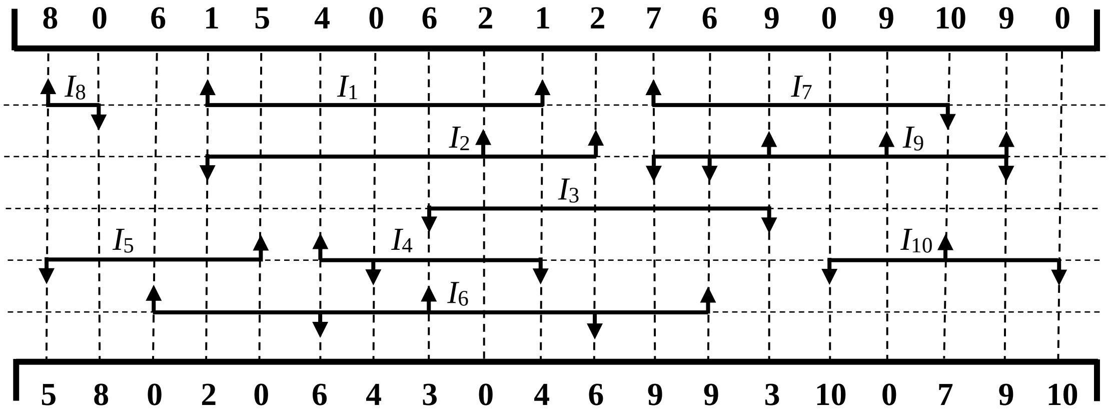
In the second iteration, say, the fourth entry from *NET_LIST* (i.e. 4) and the sixth entry from *TERM_COUNT* (i.e. 3) are randomly selected. It means that net 4 is a 3-terminal net that is to be introduced into the newly constructed channel *C*. Say, the elements that are chosen randomly from *TERM_LIST* are 11, 14, and 20, which implies that the terminal positions of net 4 are column 6 at the top, column 7 at the bottom, and column 10 at the bottom. The lists are modified accordingly. After the second iteration, the modified lists are as shown below.

NET_LIST	:	1 2 3 5 7 8 9 10
TERM_COUNT	:	2 2 2 2 2 3 3 6
TERM_LIST	:	1 2 3 4 6 7 8 9 10 13 16 17 18 19 21 23 24 26 27 28 29 30 31 32 33 34 35 36 37 38

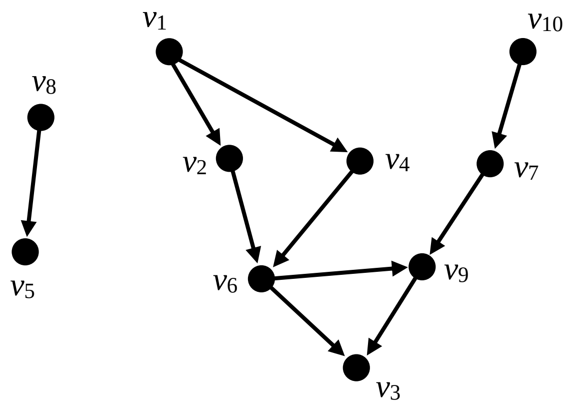
Column #	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
TOP	0	0	**6**	0	0	**4**	0	**6**	0	0	0	0	**6**	0	0	0	0	0	0
BOTTOM	0	0	0	0	0	**6**	**4**	0	0	**4**	**6**	0	0	0	0	0	0	0	0

In this way, if we follow 10 successive iterations, the first two auxiliary lists become empty, and eventually we generate a desired general channel specification, whose all nets' terminals are assigned to different columns along the length of the channel. The finally generated channel specification may be some sort of the following channel.

Column #	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
TOP	8	0	6	1	5	4	0	6	2	1	2	7	6	9	0	9	10	9	0
BOTTOM	5	8	0	2	0	6	4	3	0	4	6	9	9	3	10	0	7	9	10



(a)



(b)

Figure 5.3: (a) The generated general channel specification whose length is 19 and that comprises 10 nets. **(b)** The vertical constraint graph of this generated general channel specification. Incidentally, it consists of two components and does not contain any cyclic vertical constraint.

The channel with 10 nets is shown in Figure 5.3(a). The vertical constraint graph of this channel is shown in Figure 5.3(b). Vertical constraints among a set of nets determine a relative ordering over the nets along the height of the channel [49, 96]. The *vertical constraint graph* (*VCG*) consists of *n* vertices for *n* nets present in a channel. If a column of the channel contains terminals of two different nets, say *a* and *b*, at the top and at the bottom, respectively, then in the vertical constraint graph, a directed edge is introduced from vertex v_a to vertex v_b.

Table 5.4 shows how the nets in the generated channel are spanned over its length, as the net number increases. Furthermore, terminal count, i.e. the number of terminals per net is also shown in the table. Data show the randomness of the generated channel instance.

Table 5.4: Spans (or intervals) of different nets introduced into the randomly generated general channel specification, shown in Figure 5.3(a). The number of terminals per net is also shown in this table in the rightmost column of Terminal Count.

Net Number	Leftmost Column Position	Rightmost Column Position	Span (or Interval)	Terminal Count
1	4	10	6	2
2	4	11	7	3
3	8	14	6	2
4	6	10	4	3
5	1	5	4	2
6	3	13	10	5
7	12	17	5	2
8	1	2	1	2
9	12	18	6	6
10	15	19	4	3

5.5.5 The Complexity of the Algorithm

We now analyze the computational complexities of algorithm *General_Random_ Channel_Generator*. Suppose N nets are introduced in generating a general channel instance; so, the number of iterations required is N. In each iteration, we need to search some random locations from the three auxiliary lists. The length of each of the lists *TERM_COUNT* and *NET_LIST* is same as the number of nets, i.e. N, and also the length ($2L$) of *TERM_LIST* is linearly dependent on N, i.e. L is $O(N)$. If we represent all these lists by singly linked linear lists, then the sequential search time for each list is dependent on their length. As the length of each of these (three) lists is $O(N)$, and as they are searched in a sequential manner, the total time to search the lists is $O(N) + O(N) + O(N)$, i.e. $O(N)$. So, for N iterations, as we like to introduce N nets into the channel, the total time complexity is $O(N^2)$. Also, the space complexity is dependent on the length of each of all these lists, which is $O(N)$. Consequently, the space requirement is bounded by O(*N*). We summarize the complexity results in the following theorem.

Theorem 5.2: *Algorithm General_Random_Channel_Generator randomly generates general channel specifications of multi-terminal nets that contain both horizontal as well as vertical constraints in it. The algorithm takes time* $O(n^2)$*, and space* $O(n)$*, where n is the number of nets introduced into the channel.*

5.5.6 Removal of Cyclic Vertical Constraints

In this subsection, we like to mention an additional checking that is often necessary to know whether a generated general channel instance contains any cyclic vertical constraints. This is because there are some routing models, like the *reserved two-layer no-dogleg Manhattan channel routing model* [15, 49, 51, 52, 53, 54, 80, 96], where vertical constraint violation occurs if a general channel instance contains a cyclic vertical constraint. So, in order to make the vertical constraint graph of a generated general channel instance acyclic we do the following task that we explain here with the help of a suitable example.

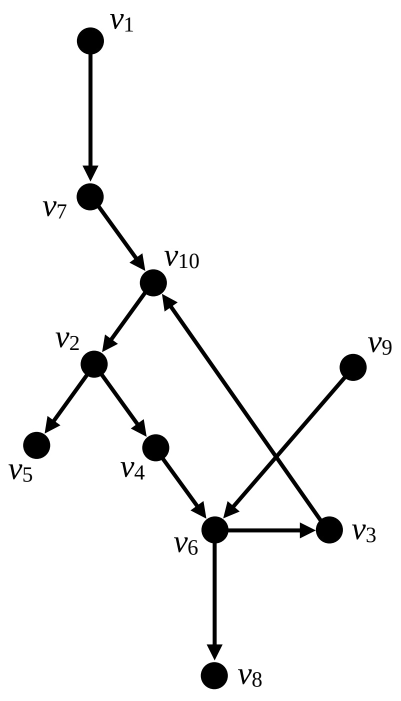


Figure 5.4: The vertical constraint graph of a randomly generated general channel instance that contains a cycle among nets 10, 2, 4, 6, and 3.

Let us consider a randomly generated general channel instance is as follows.

2	0	6	5	9	1	4	0	10	10	6	8	7	10	4	0	2	0	3	10
5	9	3	0	6	7	0	6	2	0	8	0	10	0	6	8	4	1	10	0

The vertical constraint graph (VCG) of this channel instance is shown in Figure 5.4 that contains a cycle. We identify the cycle(s) using the well-known *depth-first search* (*DFS*) algorithm on the directed VCG in time O(n+e), where n is the number of nets in the generated channel instance and e is the number of edges in the VCG.

After a cycle in the VCG is detected, we dissolve the cycle by destroying a vertical constraint belonging to this cycle. We may destroy any one of the vertical constraints in this cycle based on some logic so that the resulting VCG is acyclic. Anyway, following DFS with source vertex v_1, edge (v_3, v_{10}) may be identified as a back edge that we could destroy by splitting the column containing a terminal of net 3 at the top and a terminal of net 10 at the bottom into two, as shown underlined below in two consecutive columns of the channel.

2	0	6	5	9	1	4	0	10	10	6	8	7	10	4	0	2	0	$\underline{3}$	$\underline{0}$	10
5	9	3	0	6	7	0	6	2	0	8	0	10	0	6	8	4	1	$\underline{0}$	$\underline{10}$	0

In this way, the channel length is increased by one. So, for a randomly generated general channel instance with p cycles in its VCG, at most p columns may require being introduced in order to generate a desired channel instance without any cycle in its VCG.

5.6 Experimental Results

In this section, we include all the experimental results that are computed for generation of simple as well as general channel instances. In practice, the instances we have generated randomly contain 10 through 15000 nets in a channel, and for each number of nets 200 such instances are generated. Results that are shown in different tables and curves in this section are computed by making an average on all such 200 randomly generated channel instances for a specific number of nets. As for example, consider Table 5.5, where we show the experimental results obtained following algorithm *Simple_Random_Channel_Generator*. This table shows the minimum as well as the maximum span of nets out of 200n nets in generating 200 random simple channel instances of n nets each. Incidentally, we may observe in the first row of the table, the minimum span is one unit only whereas the maximum span of a net (out of 2000 nets) is almost spanned over the length of the channel, which is 19 units. The average span per net is also shown in Table 5.5, and in the last row of this table, we see that the average span per net is 4.852 units only (that is actually the average span of 2000 nets here).

Table 5.5: Experimental results of randomly generated simple channel instances; 200 instances are generated for each number of nets, and a row in this table shows the data out of all these 200 instances for a given net number.

Number of Nets	Minimum / Maximum Span per Net	Average Span per Net
10	1 / 19	4.852000
20	1 / 39	10.599500
40	1 / 79	21.971500
60	1 / 103	29.509000
80	1 / 159	41.075125
100	1 / 175	51.174000
150	1 / 277	74.469733
200	1 / 360	98.519850
300	1 / 536	145.046333
400	1 / 730	200.074325
500	1 / 951	252.453620
600	1 / 1143	303.903500
700	1 / 1336	362.000243
800	1 / 1502	406.109000
900	1 / 1757	463.381778
1000	1 / 1976	516.205790
1500	1 / 2930	763.650047
2000	1 / 3898	1048.163660
4000	1 / 7844	2063.192635
6000	1 / 11895	3049.836172
8000	1 / 15670	3995.323902
10000	1 / 19622	5052.101507
12000	1 / 23783	6006.648183
15000	1 / 29814	7422.525149

The maximum span of a net out of 30,00,000 nets that are introduced in generating 200 random simple channel instances, each of which contains 15000 nets, is 29814 units whereas the average span per net of all the nets introduced in these channels is 7422.525149 units only; see the last row of Table 5.5. In another form, we view these computed results as shown in Figure 5.5. In this figure, the number of nets is available along X-axis, and the average span per net is obtained along Y-axis of the plot. The figure shows regularity in changing the average span per net as the number of nets introduced into the channels changes. The relation is very close to linear, as the average span per net is almost half of the number of nets introduced into the channels of the same net number (also see Table 5.5).

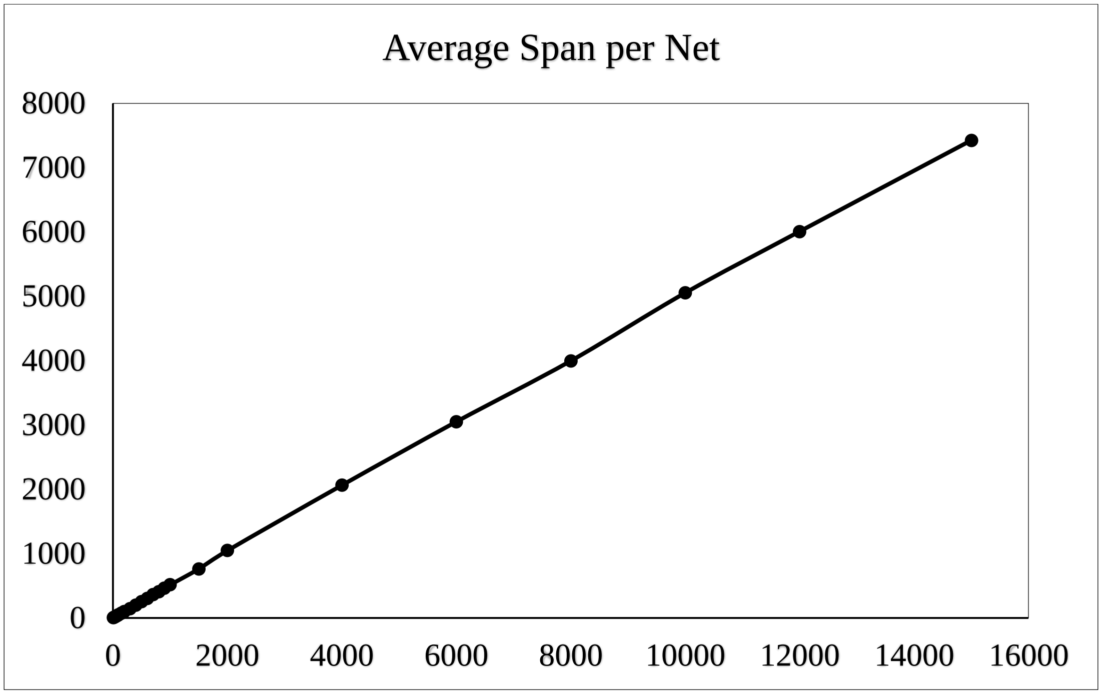


Figure 5.5: The variation of average span per net over the number of nets introduced into a randomly generated simple channel instance. Here essentially 200 instances of a particular net number are generated randomly, and the average span per net is obtained by making an average of spans of all those nets that are introduced into the said channels.

Now we consider the experimental results that are computed following algorithm *General_Random_Channel_Generator*. The results are included in Table 5.6. Here also, we randomly compute 200 general channel instances for a given number of nets. Note that here in generating general channel instances the channels that are generated for a particular net number may have different channel lengths; this is unlike in generating simple channel instances, where the channel length is always same as 2*n* for a channel with *n* nets (as we introduced 50% non-terminals there to avoid vertical constraints). Here the generated channel lengths vary from around 1.5 to 2.2 times (as minimum to maximum channel lengths obtained) of the number of nets belonging to the channel; see the second column of Table 5.6. Incidentally, the average channel length (of 200 generated channel instances of a given net number) is approximately 1.8 – 1.9 times of the number of nets introduced into the channel; see the third column of the table.

Table 5.6: Experimental results of randomly generated general channel instances; 200 instances are generated for each number of nets, and a row in this table shows the data out of all these 200 instances for a given net number.

Net Number	Min / Max Channel Length	Average Channel Length	Average Number of Terminals per Net	Average Number of Non-Terminals per Channel	Average Span per Net
10	14 / 20	16.485000	2.592500	7.045000	5.539000
20	29 / 42	35.765000	2.807500	15.380000	9.914000
40	59 / 86	73.290000	2.881875	31.305000	18.232125
60	90 / 131	111.075000	2.911583	47.455000	27.340167
80	121 / 175	148.760000	2.922750	63.700000	36.178312
100	154 / 222	188.245000	2.959200	80.570000	45.017050
150	229 / 330	280.715000	2.941833	120.155000	65.774600
200	307 / 444	376.395000	2.959200	160.950000	86.965850
300	461 / 666	564.755000	2.959200	241.750000	127.570717
400	614 / 888	752.845000	2.959200	322.010000	168.568050
500	768 / 1110	941.090000	2.959200	402.580000	209.976840
600	921 / 1331	1129.190000	2.959200	482.860000	251.071517
700	1075 / 1553	1317.510000	2.959200	563.580000	291.667157
800	1228 / 1775	1505.690000	2.959200	644.020000	334.257887
900	1382 / 1997	1693.960000	2.959200	724.640000	374.271572
1000	1535 / 2219	1882.115000	2.959200	805.030000	414.723735
1500	2303 / 3329	2823.130000	2.959200	1207.460000	622.300127
2000	3071 / 4438	3764.190000	2.959200	1609.980000	822.438110
4000	6141 / 8876	7528.400000	2.959200	3220.000000	1628.330819
6000	9212 / 13314	11292.625000	2.959200	4830.050000	2432.932756
8000	12282 / 17752	15056.740000	2.959200	6439.880000	3186.042123
10000	15353 / 22190	18821.120000	2.959200	8050.240000	3948.734045
12000	18423 / 26628	22585.100000	2.959200	9659.800000	4736.121187
15000	23029 / 33285	28231.470000	2.959200	12074.940000	5918.711587

More specifically, we like to mention the following. In generating a general channel instance, we have introduced only 2- to 6-terminal nets with the following five sets of percentage range that are also selected randomly in our implementation. The percentage ranges are {50, 30, 10, 7, 3}, {40, 35, 15, 5, 5}, {60, 20, 15, 3, 2}, {40, 30, 20, 6, 4}, and {45, 30, 15, 7, 3}. Observe in each of these sets a higher percentage is usually used for nets with a smaller number of terminals, and the vice-versa. As for example, consider the third set where 60% 2-terminal nets, 20% 3-terminal nets, 15% 4-terminal nets, 3% 5-terminal nets, and 2% 6-terminal nets are to be introduced. As a result of this in generating a channel having lesser number of nets, like 10, 20, etc., the higher terminal net may not be introduced there in generating general channel instances.

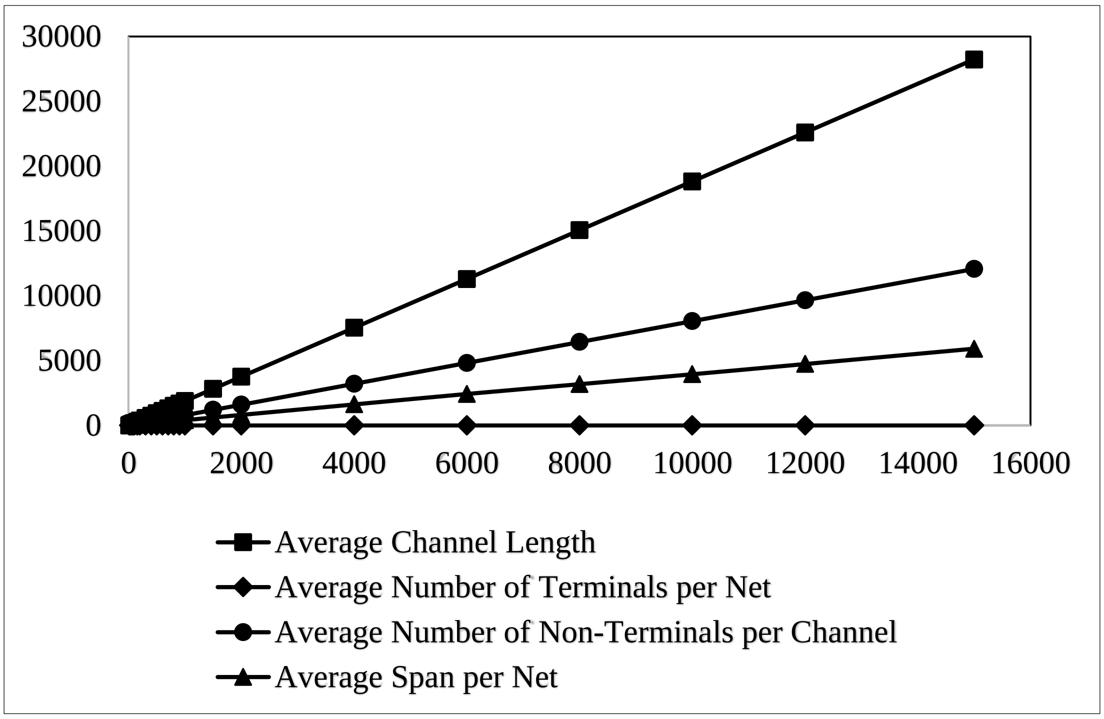


Figure 5.6: The variation of different parameters that are obtained as experimental results in generating general channel instances randomly as the number of nets introduced into the generated channels increases. The parameters that are considered here are average channel length, average number of non-terminals per channel, average span per net, and the average number of terminals per net. In reality, here 200 instances of a particular net number are generated randomly, and a parameter is computed by making an average of the said parameter of all those nets that are introduced into the said channels of a given net number.

Hence, the question arises about the average number of terminals per net, the average number of non-terminals per channel, and the average span per net for the randomly generated general channel instances, where a specific number of nets are introduced. These parameters are also computed in our implementations that are included in columns 4 through 6 in Table 5.6. Incidentally, the average number of terminals per net approximately varies from 2.6 to 2.96 (which is almost constant), the average number of non-terminals per channel is roughly 80% of the net number, and the average span per net is 50 – 40% (in units) of the net number, as the number of nets increases in generating general random channel instances. We also may view these results, along with their variation, as shown in Figure 5.6.

Both the algorithms *Simple_Random_Channel_Generator* and *General_Random_Channel_Generator* have been implemented in C on an Intel Pentium Dual-Core machine having a 794 MHz clock. Furthermore, the implementations are done using Microsoft Visual C++ 6.0 on a platform of Microsoft Windows XP Professional (Version 2002) with CPU T2080@1.73GHz and with a support of 504 MB RAM. The CPU times required are not shown in the tables, but these are mostly negligible up to channels with 10000 nets in generating 200 channel instances of the same number of nets introduced there.

We have used the seed increment for generating a channel as 65000 over the channel count plus one. Here the channel count is 200, as we like to generate 200 random channel instances over a continuum of 65000 (as the maximum seed value), where no two generated instances are identical to each other; rather, the instances generated are likely to be evenly distributed over the continuum. Experimental results show the beauty of the data obtained based on different parameters of generated channel instances, as most of the parameters vary linearly as the number of nets increases in generating random channel instances.

5.7 Usefulness in Developing the Algorithms

In channel routing problem, usually, the primary objective of a router is to compute a routing solution for some channel instance that uses a minimum number of tracks (or minimum channel area). In addition, in high performance routing our interest is also to compute a routing solution with less electrical hazards (i.e. crosstalk), less signal propagation delay, less power consumption, less or no hotspot formation, and so on and so forth.

Crosstalk is one of the most important high performance optimization criteria in (channel) routing that is to be reduced to achieve better performance out of the routing solution. There are some techniques for minimizing crosstalk in a different channel and switchbox routing models [23, 24, 25, 39]. Usually, there are two types of crosstalk minimization problem in channel routing, namely, *sum crosstalk minimization* and *bottleneck crosstalk minimization*; see Chapter 3. *Sum crosstalk* is the amount of total crosstalk between horizontal wire segments of the nets that are assigned to adjacent tracks. The sum crosstalk minimization problem is to compute a feasible routing solution with a given number of tracks in which the total amount of crosstalk is minimized.

On the other hand, *bottleneck crosstalk* with respect to a feasible routing solution is the maximum amount of crosstalk due to overlapping between any pair of horizontal wire segments of two different nets that are assigned to (two) adjacent tracks in a routing solution. Thus, the bottleneck crosstalk minimization problem is the problem of finding a feasible routing solution with a given number of tracks, such that the amount of bottleneck crosstalk is minimized. Incidentally, all these problems along with some other problems of crosstalk minimization have been proved NP-hard, and all these results have been included in Chapter 3 of this thesis.

The channel routing problem of area minimization being an NP-hard problem [41, 49, 50, 60, 61, 63, 80, 86], several heuristic algorithms have been proposed for routing channels in different routing models [10, 12, 33, 49, 51, 54, 71, 73, 80, 96]. The problem is polynomial time solvable if the channel instances are free from any vertical constraint, and there are algorithms for computing a routing solution using exactly density number of tracks for each of such instances [32, 49, 52, 53]. Since the problem of minimizing area for an instance of channel routing with only horizontal constraints is polynomial time solvable (in computing a routing solution using exactly density number of tracks), we call them as *simple channel instances* of channel routing. We define a channel specification as *general* if both the constraints are present in it.

However, the crosstalk minimization problem for two-layer channel routing, both in the case of *simple* as well as *general* channel instances, is NP-hard, as has been proved herein. That is, there exists no polynomial time algorithm for computing a reduced crosstalk two-layer channel routing using the routing model under consideration (i.e. the reserved two-layer (VH) Manhattan channel routing model), even if the instances are free from any vertical constraint. As a result, several heuristic algorithms have been developed to minimize crosstalk for two-layer channel routing for *simple* as well as *general* instances of channel specifications, which have been included in Chapter 4 of this thesis. Incidentally, there are a few *general* channel instances available in the literature as benchmark channel instances [49, 96]; however, those are not sufficient in executing the heuristic algorithms developed for reducing crosstalk. On the other hand, no *simple* channel instances are available in the literature. So, it is an obvious necessity in developing algorithms so that a huge number of *simple* and *general* channel instances can be generated (randomly) for using them in executing the heuristic algorithms developed for several NP-hard channel routing problems. The algorithms developed in this chapter meet that requirement.

Furthermore, it may be mentioned that following the illustration in Section 5.4.3, we show how a random simple channel instance is generated using algorithm *Simple_Random_Channel_Generator* whose vertical constraint graph does not contain any directed arc; the associated horizontal constraint graph is shown in Figure 5.2(b). On the other hand, algorithm *General_Random_Channel_Generator* generates a general channel instance whose illustration is available in Section 5.5.4; the associated vertical constraint graph is shown in Figure 5.3(b). In computing a general channel instance, a cycle may also be evolved in its vertical constraint graph. Section 5.5.6 resolves such constraints using an algorithmic technique so that eventually cycles are removed from a generated general channel instance. This is how a huge number of simple and (desired) general channel instances are randomly generated following the algorithms developed in this chapter. All generated instances are utilized in computing reduced crosstalk channel routing solutions for the algorithms developed in the previous chapter, and all associated results have been reported in the next chapter (i.e. in Chapter 6).

5.8 Summary

In this chapter, we have developed algorithms for generating random channel instances for both simple and general in nature for channel routing problem in VLSI physical design. Simple channel instances do not contain any vertical constraint, whereas general channel instances contain both horizontal as well as vertical constraints [48, 49, 80, 81, 96]. Simple channel instances that are generated in this chapter are all containing only two-terminal nets; simple channel instances of multi-terminal nets, based on some requirements, can also be generated. General channel instances that are randomly generated in this chapter contain two-terminal as well as multi-terminal nets. These general channel instances may also contain cyclic vertical constraints; however, for the channel routing model under consideration (that is the reserved two-layer Manhattan channel routing model), we have removed those cyclic vertical constraints such that a solution in two-layer routing is always guaranteed. This has also been explained in this chapter.

The algorithms that are developed in this chapter are able to generate a very large number of channel instances of any (possible) number of nets. In any case, the said algorithms are extremely useful when we require a large number of channel instances. As most of the problems in two-, three-, and multi-layer channel routing are computationally hard to solve [41, 49, 50, 55, 56, 57, 58, 59, 61, 63, 80, 86], for each of these problems it is unlikely to design a polynomial time deterministic algorithm. Rather, developing heuristic algorithm may be a probable way out that hopefully provides good solutions for most of the instances available in the literature. Incidentally, a very few (general) benchmark channel instances are available in the literature [49, 96]. So, to execute all these heuristic algorithms (that are developed for NP-hard channel routing problems) and to judge their novelty, a variety of a large number of similar kind of channel instances are required that we may generate with the help of the algorithms developed in this chapter. In fact, the convergence of results of a heuristic algorithm is well established when the algorithm of a problem is executed for a substantial number of randomly generated similar (channel) instances, and the final result is computed making an average on all of them.

Chapter: 6

Experimental Results on Generating Random Channel Instances and Computing Two-Layer Reduced Crosstalk Channel Routing Solutions

6.1 Overview

In this thesis, we have considered the problem of crosstalk minimization in two-layer channel routing. We have already developed the previous three chapters for dealing with the same. In Chapter 3, we have marked out several problems in crosstalk minimization and subsequently proved their hardness. In order to acquire a large number of channel instances, we have devised two algorithms for generating random channels: one for the simple instances of channel specifications and the other for the general instances of channel specifications, in Chapter 5. The crosstalk minimization problem in two-layer (VH) channel routing is NP-hard. Thus, if a routing solution, S of t tracks, is given and we are supposed to compute a reduced crosstalk channel routing solution, S' using the same t number of tracks (i.e. without enhancing the area of routing), then the task that we can think of immediately is to design heuristic algorithms to solve most of the instances of the problem under consideration.

In this light of thinking, in this thesis, we have devised two principal algorithms for reducing crosstalk in Chapter 4. These algorithms have also been implemented in Chapter 4 for a smaller number (540 only, in nine sets containing 60 channels in each set for a given number of nets) of randomly generated simple channel instances and also for a set of 14 existing benchmark channel instances; all these results are also included there.

Four noteworthy algorithms have been developed in the previous two chapters of this thesis, and we have randomly generated an enormous number of channel instances of both kinds. Even then, in order to comprise a particularly tiny subset of some of these as representative generated instances as well as representative hardcopy routing solutions produced by the subsequently reduced crosstalk channel routing algorithms, we essentially include this chapter in the form of computed experimental results as has been depicted in different tables.

6.2 Generation of Random Channel Instances

The random channel instance generating algorithms have been developed in Chapter 5. We have developed two algorithms for generating such instances; one for the simple channel instances and the other for the general channel instances. The simple channel instances do not contain any vertical constraint whereas the general channel instances do have both the constraints. The length of a randomly generated simple channel specification of *n* nets is exactly 2*n*, and each of the nets present in such a channel is a two-terminal net. In this context, a general channel specification contains two-terminal as well as multi-terminal nets with varying length of the channel. In more detail, these are explained in the following two subsections by depicting some of the randomly generated instances, the status of the routing solutions computed, and some of the selected sets of hardcopy routing solutions after execution of each of the algorithms.

As simple channel instances are hardly available in the literature, and general (benchmark) channel instances are not much, in this work we have created a large number of random channel instances of both types. In making the generated instances as random as possible along the length of the channel, we follow some measures of assumed standard, as has been briefly mentioned again.

(i) In our randomly generated channel instances the nets (that are made different by numbers) appear randomly, i.e. the nets are not sorted in succession based on their starting column positions from left to right (or from right to left) along the length of the channel. We may remember that the net numbers signify nothing but symbols to discriminate themselves.

(ii) Nets of different spans (of intervals) are supposed to appear randomly along the length of the channel. This criterion tells that all the smaller (or larger) nets are not accumulated (or concentrated) on a side of the channel.

(iii) We may assume that as a general practice, a channel includes a large number of smaller nets and less large nets. Here the smaller or larger nets are differentiated by their relative spans (or intervals). Observably, the number of nets with some intermediate spans is neither more nor less. Thus, this criterion tells that the number of nets steadily decreases as their spans increase along the length of the channel.

(iv) Also, a general channel instance contains a percentage of non-terminals.

6.2.1 Generation of Simple Channel Instances

In this work, as we concentrate on minimizing crosstalk in two-layer channel routing, for net pairs assigned to adjacent tracks (in a routing solution), that depends on only the horizontal spans of different nets, we usually do not bother how much the vertical wire segments of two different nets overlap on two adjacent columns. In fact, as the pin-to-pin separation is determined by the allowable process technology, and we usually have no control over a given channel specification (that are the TOP and BOTTOM vectors of a channel), we do not like to impose any additional constraint in assigning terminal locations, either at the top or at the bottom, of a two-terminal net while generating a simple channel instance for a given number of nets.

We have already mentioned that for a given number of nets we have generated 200 random simple channel instances (where the number of nets introduced in a channel varies from 10 through 15000; see Table 5.5), but there is no scope to show all these instances by making this thesis unnecessarily thicker, and that could be boring as well to a reader to go through. Besides, larger instances might take pages to depict. Thus, as a matter of fact, we only include a reasonably small number of such instances (that containing a smaller number of nets), with their associated information for simple channel specifications. First, we include only six simple sample channels each containing a total number of 10 nets. For all the nets belonging to all the 200 randomly generated instances, the average span of nets obtained is 4.8520 (see Table 5.5).

Sample simple channel # 1 that contains only 10 nets:

Column #	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
TOP	2	0	5	2	8	0	10	0	0	0	4	0	0	0	10	7	4	0	9	6
BOTTOM	0	9	0	0	0	5	0	8	1	1	0	3	7	3	0	0	0	6	0	0

The total span of nets = 48 and the average span of nets of this channel = 4.80.

Sample simple channel # 2 that contains only 10 nets:

Column #	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
TOP	10	0	9	0	0	0	6	0	7	4	7	0	0	0	3	0	0	0	1	0
BOTTOM	0	8	0	10	8	9	0	4	0	0	0	5	5	6	0	3	2	1	0	2

The total span of nets = 26 and the average span of nets of this channel = 2.60.

Sample simple channel # 3 that contains only 10 nets:

Column #	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
TOP	4	0	0	0	4	0	0	0	10	0	6	8	0	1	2	3	9	0	0	0
BOTTOM	0	10	7	9	0	5	6	7	0	5	0	0	1	0	0	0	0	2	3	8

The total span of nets = 52 and the average span of nets of this channel = 5.20.

Sample simple channel # 4 that contains only 10 nets:

Column #	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
TOP	3	9	1	0	2	0	5	3	8	0	0	0	8	0	10	0	4	0	0	0
BOTTOM	0	0	0	1	0	2	0	0	0	5	10	4	0	7	0	7	0	6	9	6

The total span of nets = 46 and the average span of nets of this channel = 4.60.

Sample simple channel # 5 that contains only 10 nets:

Column #	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
TOP	0	0	9	0	1	0	7	8	6	0	10	5	3	0	0	0	0	0	2	0
BOTTOM	9	4	0	4	0	1	0	0	0	7	0	0	0	8	6	10	3	5	0	2

The total span of nets = 36 and the average span of nets of this channel = 3.60.

Sample simple channel # 6 that contains only 10 nets:

Column #	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
TOP	1	6	10	0	4	0	9	5	6	0	3	10	8	0	2	0	0	0	0	1
BOTTOM	0	0	0	4	0	8	0	0	0	3	0	0	0	5	0	2	9	7	7	0

The total span of nets = 62 and the average span of nets of this channel = 6.20.

Now, we include four simple sample channels each containing a total number of 20 nets, two channels each holding 40 nets, two similar channels each having 60 nets and only one channel of 80 nets. The average span per net for all nets belonging to all 200 randomly generated instances for a given number of nets is included in Table 5.5.

Sample simple channel # 1 that contains only 20 nets:

Column #	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
TOP	14	20	18	0	0	15	13	0	2	0	0	0	5	0	0	18	1	0	12	6	10
BOTTOM	0	0	0	10	19	0	0	20	0	16	2	6	0	14	9	0	0	1	0	0	0

Column #	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40
TOP	0	3	0	0	0	8	0	0	0	7	0	0	0	11	0	4	15	16	0
BOTTOM	13	0	19	5	9	0	17	7	3	0	12	4	17	0	8	0	0	0	11

The total span of nets = 224 and the average span of nets of this channel = 11.20.

Sample simple channel # 2 that contains only 20 nets:

Column #	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
TOP	0	0	20	7	0	0	18	0	19	5	3	0	3	0	0	0	10	0	0	0	18
BOTTOM	14	4	0	0	5	10	0	4	0	0	0	15	0	13	7	20	0	16	14	8	0

Column #	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40
TOP	0	13	0	17	0	8	0	2	0	2	6	15	0	0	11	9	0	0	0
BOTTOM	12	0	19	0	12	0	17	0	11	0	0	0	16	6	0	0	9	1	1

The total span of nets = 168 and the average span of nets of this channel = 8.40.

Sample simple channel # 3 that contains only 20 nets:

Column #	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
TOP	0	0	19	0	11	0	0	0	2	15	16	0	14	9	10	0	13	0	0	0	3
BOTTOM	12	9	0	15	0	18	2	8	0	0	0	17	0	0	0	3	0	12	8	5	0

Column #	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40
TOP	0	20	7	13	0	4	6	0	0	1	0	0	0	10	0	17	6	7	0
BOTTOM	11	0	0	0	18	0	0	1	20	0	4	5	16	0	14	0	0	0	19

The total span of nets = 278 and the average span of nets of this channel = 13.90.

Sample simple channel # 4 that contains only 20 nets:

Column #	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
TOP	0	0	14	8	20	0	0	0	0	3	20	0	18	0	1	0	4	0	17	0	9
BOTTOM	19	8	0	0	0	17	3	18	5	0	0	4	0	1	0	11	0	14	0	16	0

Column #	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40

TOP	5	13	0	13	0	7	0	10	0	0	0	9	7	19	0	12	6	2	2
BOTTOM	0	0	16	0	11	0	12	0	10	6	15	0	0	0	15	0	0	0	0

The total span of nets = 154 and the average span of nets of this channel = 7.70.

Sample simple channel # 1 that contains only 40 nets:

Column #	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
TOP	18	0	20	4	28	0	6	0	13	6	0	0	37	1	33	0	31	0	0	0	28
BOTTOM	0	39	0	0	0	20	0	34	0	0	4	40	0	0	0	25	0	30	1	38	0

Column #	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41
TOP	38	12	0	23	0	40	0	0	3	37	0	13	8	26	0	29	0	9	39	24
BOTTOM	0	0	22	0	18	0	3	8	0	0	12	0	0	0	19	0	26	0	0	0

Column #	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61
TOP	0	11	0	27	0	22	11	0	0	36	0	21	0	25	0	23	0	30	0	19
BOTTOM	33	0	9	0	24	0	0	29	14	0	15	0	34	0	15	0	31	0	16	0

Column #	62	63	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80
TOP	35	32	0	27	0	0	0	35	0	17	0	21	0	5	7	2	0	10	0
BOTTOM	0	0	14	0	17	5	36	0	32	0	7	0	16	0	0	0	2	0	10

The total span of nets = 598 and the average span of nets of this channel = 14.95.

Sample simple channel # 2 that contains only 40 nets:

Column #	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
TOP	39	28	34	0	37	0	0	0	18	0	0	30	38	0	38	0	23	6	40	2	37
BOTTOM	0	0	0	40	0	33	25	31	0	21	6	0	0	28	0	22	0	0	0	0	0

Column #	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40
TOP	0	2	17	26	0	11	0	0	0	1	0	0	4	4	0	33	0	27	24
BOTTOM	12	0	0	0	15	0	29	26	1	0	39	24	0	0	17	0	5	0	0

Column #	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61
TOP	11	0	12	0	0	8	29	0	21	0	10	5	13	0	36	0	34	0	15	3	25
BOTTOM	0	22	0	16	7	0	0	18	0	10	0	0	0	9	0	7	0	8	0	0	0

Column #	62	63	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80
TOP	0	23	31	20	0	14	0	0	0	0	0	32	0	36	0	14	0	0	0
BOTTOM	3	0	0	0	19	0	30	9	35	13	19	0	27	0	16	0	35	20	32

The total span of nets = 804 and the average span of nets of this channel = 20.10.

Sample simple channel # 1 that contains only 60 nets:

Column #	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
TOP	3	0	27	6	58	0	51	0	0	0	26	0	45	0	56	0	0	26	48	0	0
BOTTOM	0	36	0	0	0	39	0	54	9	41	0	49	0	50	0	17	5	0	0	5	7

Column #	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41
TOP	0	42	0	49	0	17	14	32	0	2	0	27	0	0	0	31	0	34	0	41
BOTTOM	56	0	43	0	42	0	0	0	37	0	14	0	7	37	2	0	52	0	33	0

Column #	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62
TOP	0	24	0	60	31	25	0	22	0	0	0	47	0	44	0	16	0	54	0	0	0
BOTTOM	19	0	19	0	0	0	18	0	59	3	18	0	4	0	60	0	53	0	6	57	28

Column #	63	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80	81	82
TOP	0	0	11	0	36	0	46	0	47	0	0	0	23	0	13	0	35	0	0	0
BOTTOM	4	38	0	59	0	53	0	30	0	48	20	24	0	29	0	52	0	55	58	38

Column #	83	84	85	86	87	88	89	90	91	92	93	94	95	96	97	98	99	100	101
TOP	39	0	21	11	16	0	0	0	12	0	13	0	32	1	57	10	35	0	55
BOTTOM	0	34	0	0	0	40	9	22	0	33	0	30	0	0	0	0	0	44	0

Column #	102	103	104	105	106	107	108	109	110	111	112	113	114	115	116
TOP	0	50	0	0	0	0	23	28	0	12	25	29	0	43	0
BOTTOM	8	0	1	21	45	10	0	0	20	0	0	0	46	0	15

Column #	117	118	119	120
TOP	8	0	15	51
BOTTOM	0	40	0	0

The total span of nets = 1980 and the average span of nets of this channel = 33.00.

Sample simple channel # 2 that contains only 60 nets:

Column #	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
TOP	53	0	0	0	0	0	11	0	0	0	13	0	3	12	39	0	54	0	13	59	20
BOTTOM	0	46	57	56	26	58	0	31	11	3	0	24	0	0	0	51	0	24	0	0	0

Column #	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41
TOP	0	41	0	0	0	50	0	42	0	49	0	0	0	37	0	0	0	37	0	0
BOTTOM	29	0	49	45	12	0	48	0	43	0	42	29	26	0	57	6	33	0	52	20

Column #	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62
TOP	0	0	0	31	0	6	0	0	0	47	0	0	0	60	2	5	0	56	0	0	0
BOTTOM	14	28	41	0	60	0	33	53	5	0	18	14	44	0	0	0	54	0	38	30	46

Column #	63	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80	81	82
TOP	34	0	0	0	0	0	10	0	1	0	9	0	0	0	48	0	25	0	35	55
BOTTOM	0	10	1	9	2	30	0	36	0	17	0	47	16	23	0	52	0	18	0	0

Column #	83	84	85	86	87	88	89	90	91	92	93	94	95	96	97	98	99	100	101
TOP	0	0	0	0	32	0	40	21	15	0	28	27	59	0	17	0	0	0	16
BOTTOM	32	58	21	38	0	39	0	0	0	7	0	0	0	8	0	35	32	44	0

Column #	102	103	104	105	106	107	108	109	110	111	112	113	114	115	116
TOP	0	55	19	50	0	4	0	36	0	0	0	15	0	25	4
BOTTOM	7	0	0	0	27	0	45	0	43	22	34	0	40	0	0

Column #	117	118	119	120
TOP	22	0	51	19
BOTTOM	0	8	0	0

The total span of nets = 1680 and the average span of nets of this channel = 28.00.

Sample simple channel # 1 that contains only 80 nets:

Column #	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
TOP	0	0	49	0	0	0	77	76	74	13	78	0	46	0	0	27	58	0	79	0	0
BOTTOM	48	67	0	47	66	50	0	0	0	0	0	28	0	24	26	0	0	71	0	37	61

Column #	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41
TOP	0	77	21	13	0	41	0	61	0	0	0	0	0	25	0	52	0	63	57	20
BOTTOM	53	0	0	0	65	0	24	0	69	21	34	70	54	0	62	0	60	0	0	0

Column #	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62
TOP	0	0	0	36	0	0	0	44	0	0	0	38	0	11	4	42	0	2	0	0	0
BOTTOM	11	51	9	0	39	4	68	0	55	69	45	0	28	0	0	0	9	0	43	34	80

Column #	63	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80	81	82
TOP	2	0	0	0	18	0	20	0	59	72	12	0	22	0	42	0	0	10	29	0
BOTTOM	0	26	44	68	0	27	0	39	0	0	0	67	0	47	0	12	29	0	0	64

Column #	83	84	85	86	87	88	89	90	91	92	93	94	95	96	97	98	99	100	101
TOP	31	0	0	0	56	0	30	17	46	0	58	0	0	0	0	0	73	0	33
BOTTOM	0	48	17	66	0	23	0	0	0	80	0	6	45	37	1	25	0	22	0

Column #	102	103	104	105	106	107	108	109	110	111	112	113	114	115	116
TOP	50	6	0	1	0	31	49	74	0	52	0	0	0	15	3
BOTTOM	0	0	76	0	33	0	0	0	10	0	19	41	18	0	0

Column #	117	118	119	120	121	122	123	124	125	126	127	128	129	130	131	132
TOP	38	0	0	0	0	0	65	14	32	0	72	0	78	8	16	0
BOTTOM	0	75	16	23	36	55	0	0	0	19	0	3	0	0	0	59

Column #	133	134	135	136	137	138	139	140	141	142	143	144	145	146	147	148
TOP	54	0	0	0	73	0	53	64	35	0	62	0	43	0	0	0
BOTTOM	0	15	30	60	0	8	0	0	0	51	0	14	0	57	5	79

Column #	149	150	151	152	153	154	155	156	157	158	159	160
TOP	32	40	75	0	70	0	7	0	0	0	63	0
BOTTOM	0	0	0	56	0	35	0	40	5	7	0	71

The total span of nets = 3968 and the average span of nets of this channel = 49.60.

In this section, we have included only 15 sample simple channel instances (out of 4800 instances) of smaller in size that we have generated randomly using algorithm *Simple_Random_Channel_Generator* developed in Chapter 5 of this thesis. We may observe that in each column of all these generated instances a non-terminal is there that is not to be connected, and all the nets are only two-terminal nets. Multi-terminal nets can also be generated randomly with necessary modification in the devised algorithm, but as we allow only no-dogleg routing, from the crosstalk minimization point of view, it does not matter whether the nets introduced are either two-terminal nets or multi-terminal nets. Thus, the simple channel instances that we have generated are all two-terminal nets only. In the next section, we show some example general channel instances (of smaller in size) that we have generated randomly using the generalized version of the algorithm.

6.2.2 Generation of General Channel Instances

General_Random_Channel_Generator is the algorithm that we have devised in Chapter 5 to generate general channel instances each of which contains both horizontal as well as vertical constraints. Generated instances also contain multi-terminal nets with the number of terminals varying from 2 through 6. In such a case, we are supposed to introduce an additional number of non-terminals (as a percentage of total number of terminals of different nets), where the pins along a column can both be non-terminals (or 0's), as a column can also be a trivial column (containing terminals of the same net) [49]. Moreover, as the number of terminals per net is not fixed (including the number of non-terminals), the channel length for a given number of nets is also varying.

We may further note that in generating a general channel instance a cycle may be introduced into the vertical constraint graph of the channel under construction, and for the exclusion of each such cyclic vertical constraint, we have added one extra column into the channel to break a vertical constraint belonging to the cycle. This is also a part of the generation of a general channel instance of our desire, as a channel specification containing cyclic vertical constraint is not fully routable in the assumed reserved two-layer no-dogleg Manhattan channel routing model.

To execute the algorithm *General_Random_Channel_Generator* in this thesis, we have randomly generated 4800 total random channels for 24 sets of nets each having 200 instances of our choice; the outcome of this experimentation is included in Table 5.6. Interestingly, we have the following observation out of the table. On an average, the minimum channel length is 1.5 times the number of nets, and the maximum length of a channel is approximately 2.25 times, although the average channel length is less than two times. The most interesting observation is that the average number of terminals per net approaches 2.9592 as the number of nets increases; initially, it starts from 2.5925 when the number of nets is 10 only. The average number of non-terminals per channel is roughly 21% of the total number of pin locations belonging to a channel. Moreover, as the number of nets increases, the average span per net decreases, and this attenuation starts from approximately 33.5% of the average length of the channel and reduces up to 21% of the same, where it becomes almost steady.

It has already been mentioned that as we did in generating general channel instances, for a given number of nets we have produced 200 random channel instances, where the number of nets introduced in a channel varies from 10 through 15000 (see Table 5.6). Next, in this thesis, we include only a rationally small number of such instances (that containing a smaller number of nets), with their associated information for general channel specifications. First, we include only six sample general instances each containing 10 nets per channel; for all nets belonging to all 200 randomly generated instances, the average span of nets obtained is 5.5390 (see Table 5.6).

Sample general channel # 1 that contains only 10 nets:

Column #	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
TOP	10	6	10	6	0	3	8	7	1	9	2	0	3	5	4	0	10	0	8
BOTTOM	0	0	0	9	5	3	1	0	2	8	3	0	8	7	4	5	0	4	4

The total span of nets = 68 and the average span of nets of this channel = 6.80. The number of terminals of this channel = 28; thus, the number of non-terminals = 10 and the average number of terminals per net = 2.8, as the channel length is 19.

Sample general channel # 2 that contains only 10 nets:

Column #	1	2	3	4	5	6	7	8	9	10	11	12	13	14
TOP	6	4	9	4	0	4	8	6	10	2	3	0	9	2
BOTTOM	7	0	10	4	8	5	7	1	5	1	5	8	3	0

The total span of nets = 53 and the average span of nets of this channel = 5.30. The number of terminals of this channel = 24; thus, the number of non-terminals = 4 and the average number of terminals per net = 2.4, as the channel length is 14.

Sample general channel # 3 that contains only 10 nets:

Column #	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17

TOP	2	9	0	10	0	5	3	10	1	6	3	9	4	8	0	7	6
BOTTOM	0	0	2	0	5	1	7	0	3	8	4	0	5	8	0	10	8

The total span of nets = 64 and the average span of nets of this channel = 6.40. The number of terminals of this channel = 25; thus, the number of non-terminals = 9 and the average number of terminals per net = 2.5, as the channel length is 17.

Sample general channel # 4 that contains only 10 nets:

Column #	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
TOP	0	0	0	1	0	1	6	3	0	8	9	0	5	3	10	7	2	0	4
BOTTOM	10	8	1	1	8	8	9	0	6	3	0	0	4	7	7	2	5	2	0

The total span of nets = 57 and the average span of nets of this channel = 5.70. The number of terminals of this channel = 27; thus, the number of non-terminals = 11 and the average number of terminals per net = 2.7, as the channel length is 19.

Sample general channel # 5 that contains only 10 nets:

Column #	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
TOP	0	0	9	8	10	1	1	0	0	8	4	7	5	0	9
BOTTOM	0	10	7	4	5	6	8	1	2	2	3	6	2	8	3

The total span of nets = 65 and the average span of nets of this channel = 6.50. The number of terminals of this channel = 24; thus, the number of non-terminals = 6 and the average number of terminals per net = 2.4, as the channel length is 15.

Sample general channel # 6 that contains only 10 nets:

Column #	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
TOP	0	0	9	6	10	0	5	0	8	3	3	2	2	0	7	3
BOTTOM	8	8	4	6	1	1	6	1	4	10	7	2	3	2	5	9

The total span of nets = 58 and the average span of nets of this channel = 5.80. The number of terminals of this channel = 27; thus, the number of non-terminals = 5 and the average number of terminals per net = 2.7, as the channel length is 16.

Among other smaller randomly generated general channel instances, now we include merely four general sample channels each containing a total number of 20 nets, four similar channels each holding 40 nets, two similar channels each having 60 nets, two such channels each holding 80 nets, and only one channel of 100 nets; the average span per net for all nets belonging to all 200 randomly generated instances for a given number of nets along with other associated data are included in Table 5.6.

Sample general channel # 1 that contains only 20 nets:

Column #	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
TOP	19	5	0	18	5	11	18	1	11	0	17	0	0	9	2	16	2	13	8	0	0
BOTTOM	0	19	0	5	0	19	5	0	6	6	1	6	18	2	0	12	17	18	0	9	4

Column #	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41
TOP	12	0	7	13	17	8	0	17	0	13	14	10	0	0	10	0	3	3	15	0
BOTTOM	4	14	4	8	0	0	0	17	18	20	15	10	7	12	17	3	10	3	20	16

The total span of nets = 190 and the average span of nets of this channel = 9.50. The number of terminals of this channel = 60; thus, the number of non-terminals = 22 and the average number of terminals per net = 3.0, as the channel length is 41.

Sample general channel # 2 that contains only 20 nets:

Column #	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
TOP	0	0	0	10	20	9	9	9	9	19	0	6	4	6	16	5	8	14	16	7	15
BOTTOM	19	2	14	15	2	13	8	17	16	13	4	5	17	14	6	10	0	11	14	15	0

Column #	22	23	24	25	26	27	28	29	30
TOP	0	7	12	11	1	20	12	15	0
BOTTOM	18	3	0	12	18	3	1	18	0

The total span of nets = 155 and the average span of nets of this channel = 7.75. The number of terminals of this channel = 50; thus, the number of non-terminals = 10 and the average number of terminals per net = 2.5, as the channel length is 30.

Sample general channel # 3 that contains only 20 nets:

Column #	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
TOP	19	19	16	0	8	0	0	10	2	8	7	12	17	17	10	0	9	15	3	14	5
BOTTOM	0	0	13	0	14	9	8	2	0	14	2	2	0	8	8	7	7	11	0	18	19

Column #	22	23	24	25	26	27	28	29	30	31	32	33	34	35
TOP	6	20	5	1	15	1	20	17	6	18	4	12	4	18
BOTTOM	13	1	16	10	0	3	1	19	11	4	5	4	15	0

The total span of nets = 250 and the average span of nets of this channel = 12.50. The number of terminals of this channel = 58; thus, the number of non-terminals = 12 and the average number of terminals per net = 2.9, as the channel length is 35.

Sample general channel # 4 that contains only 20 nets:

Column #	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
TOP	9	9	19	6	17	19	0	20	0	18	15	15	0	10	8	0	10	8	19	12	8
BOTTOM	11	9	9	0	16	17	6	1	18	18	1	0	0	15	8	8	8	11	13	7	0

Column #	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38
TOP	16	14	7	10	0	12	3	5	0	2	4	4	14	0	5	20	0
BOTTOM	18	18	0	0	0	7	17	2	0	2	2	3	4	4	5	14	13

The total span of nets = 209 and the average span of nets of this channel = 10.45. The number of terminals of this channel = 60; thus, the number of non-terminals = 16 and the average number of terminals per net = 3.0, as the channel length is 38.

Sample general channel # 1 that contains only 40 nets:

Column #	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
TOP	0	0	0	13	0	9	34	23	13	0	0	24	11	0	12	30	20	0	9	27	12
BOTTOM	11	0	0	34	34	0	35	0	11	0	39	34	18	0	35	31	15	13	26	21	21

Column #	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41
TOP	20	39	0	33	18	26	24	24	0	34	1	0	6	1	26	22	31	40	34	23
BOTTOM	18	20	12	24	15	20	35	6	6	18	6	0	28	0	35	33	17	15	40	0

Column #	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61
TOP	29	19	19	28	14	7	35	16	8	37	8	37	38	3	17	37	2	32	38	2
BOTTOM	10	19	36	19	37	14	25	40	31	21	38	2	26	2	10	8	28	3	7	4

Column #	62	63	64	65	66	67	68	69	70	71	72	73
TOP	37	14	28	4	25	5	37	5	29	0	38	29
BOTTOM	21	27	29	22	32	0	8	16	5	30	0	36

The total span of nets = 782 and the average span of nets of this channel = 19.55. The number of terminals of this channel = 123; thus, the number of non-terminals = 23 and the average number of terminals per net = 3.075, as the channel length is 73.

Sample general channel # 2 that contains only 40 nets:

Column #	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
TOP	0	0	15	0	0	0	36	1	1	0	26	17	35	0	0	0	15	0	34	24	40
BOTTOM	35	0	0	0	0	0	0	1	36	1	19	0	15	12	36	17	29	38	32	26	19

Column #	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40
TOP	24	4	21	30	0	13	2	34	12	34	0	30	0	13	13	39	20	0	14
BOTTOM	10	4	4	4	32	31	10	39	0	21	2	20	38	23	25	35	29	20	20

Column #	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61
TOP	23	0	37	29	11	25	29	5	0	6	3	21	0	0	23	3	27	27	27	32	0
BOTTOM	33	0	25	14	28	34	0	18	5	0	31	0	11	0	9	20	35	0	0	22	7

Column #	62	63	64	65	66	67	68	69	70	71
TOP	27	6	37	9	0	8	16	40	16	28
BOTTOM	8	28	0	0	18	0	22	0	7	33

The total span of nets = 590 and the average span of nets of this channel = 14.75. The number of terminals of this channel = 103; thus, the number of non-terminals = 39 and the average number of terminals per net = 2.575, as the channel length is 71.

Sample general channel # 3 that contains only 40 nets:

Column #	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
TOP	38	23	37	0	23	0	37	0	34	12	31	0	0	34	16	2	0	2	36	32	30
BOTTOM	0	0	23	8	35	0	0	0	0	8	30	33	12	17	30	0	29	0	0	31	0

Column #	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41
TOP	16	38	28	28	31	16	1	0	11	16	14	0	22	24	4	0	15	6	0	9
BOTTOM	26	1	1	12	16	11	9	0	33	29	24	35	32	39	25	37	0	22	0	40

Column #	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61
TOP	21	21	21	3	3	15	3	3	14	14	28	7	0	38	18	20	34	13	20	32
BOTTOM	19	4	6	3	3	22	0	17	21	19	29	26	37	40	35	20	20	25	10	15

Column #	62	63	64	65	66	67	68	69	70	71	72	73	74	75	76	77
TOP	0	7	27	27	27	35	0	5	10	0	19	13	0	0	5	0
BOTTOM	27	27	34	40	25	5	5	5	36	18	39	29	5	0	0	0

The total span of nets = 838 and the average span of nets of this channel = 20.95. The number of terminals of this channel = 120; thus, the number of non-terminals = 34 and the average number of terminals per net = 3.0, as the channel length is 77.

Sample general channel # 4 that contains only 40 nets:

Column #	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
TOP	38	38	0	31	0	39	0	3	0	27	31	30	0	0	30	25	32	18	38	0	0
BOTTOM	21	21	21	27	21	0	34	0	3	37	0	18	34	8	8	16	34	27	4	4	6

Column #	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40
TOP	0	22	26	4	10	23	39	0	28	24	23	31	10	6	13	40	10	36	35
BOTTOM	4	8	34	10	4	33	14	13	20	20	16	20	14	22	35	38	40	0	36

Column #	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61
TOP	20	20	40	19	11	7	12	40	29	15	15	33	5	9	2	24	5	7	28	12	2
BOTTOM	34	9	20	30	0	0	39	26	11	22	19	28	0	5	32	17	35	15	25	29	11

Column #	62	63	64	65
TOP	0	0	1	1
BOTTOM	37	36	17	0

The total span of nets = 704 and the average span of nets of this channel = 17.60. The number of terminals of this channel = 110; thus, the number of non-terminals = 20 and the average number of terminals per net = 2.75, as the channel length is 65.

Sample general channel # 1 that contains only 60 nets:

Column #	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
TOP	0	22	0	0	54	0	54	55	0	44	40	0	40	44	47	27	40	0	9	0	29
BOTTOM	0	0	0	59	22	0	22	22	54	0	50	0	0	27	0	9	48	9	0	0	7

Column #	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41
TOP	34	9	10	46	45	45	29	22	0	0	23	26	0	0	23	31	36	0	0	43
BOTTOM	26	29	0	29	44	41	7	44	50	22	0	50	32	40	0	0	0	50	35	31

Column #	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62
TOP	50	34	30	41	27	50	51	32	17	23	17	39	0	17	24	21	0	14	42	48	13
BOTTOM	37	0	10	23	34	0	30	41	8	43	30	0	8	47	18	8	33	33	25	15	14

Column #	63	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80	81	82
TOP	36	25	0	38	21	24	19	13	13	13	12	0	57	47	60	25	0	43	49	39
BOTTOM	33	19	49	12	28	0	47	42	0	25	0	15	0	49	0	18	0	0	24	0

Column #	83	84	85	86	87	88	89	90	91	92	93	94	95	96	97	98	99	100	101
TOP	38	19	43	34	52	0	0	0	0	58	3	3	3	0	20	38	0	58	0
BOTTOM	48	33	12	19	32	38	30	39	36	55	39	11	58	20	24	43	3	3	59

Column #	102	103	104	105	106	107	108	109	110	111	112	113	114	115	116
TOP	56	2	37	35	46	6	53	2	58	5	0	52	53	0	51
BOTTOM	0	57	3	0	2	0	5	58	45	20	1	58	0	0	1

Column #	117	118	119	120	121	122	123	124	125	126	127
TOP	60	6	53	11	0	0	1	0	0	16	20
BOTTOM	0	1	0	28	0	16	4	0	56	53	4

The total span of nets = 1791 and the average span of nets of this channel = 29.85. The number of terminals of this channel = 187; thus, the number of non-terminals = 67 and the average number of terminals per net = 3.1167, as the channel length is 127.

Sample general channel # 2 that contains only 60 nets:

Column #	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
TOP	0	39	42	25	30	25	0	42	25	30	0	0	29	39	0	34	0	54	5	35	21
BOTTOM	41	48	25	41	0	34	58	59	23	26	43	0	44	0	0	60	46	0	40	5	15

Column #	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41
TOP	35	60	0	32	45	16	15	49	50	36	51	58	7	35	54	35	33	11	35	16
BOTTOM	5	15	43	0	40	21	0	7	39	52	41	38	47	15	26	54	44	23	28	27

Column #	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62
TOP	0	27	27	40	28	29	20	57	31	0	3	11	8	22	41	17	48	52	24	0	0
BOTTOM	28	53	45	39	1	20	0	35	24	3	0	1	3	53	38	8	3	9	37	0	50

Column #	63	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80	81	82
TOP	17	54	20	22	19	48	55	2	19	10	12	48	4	4	12	9	18	46	58	52
BOTTOM	56	6	0	32	48	37	4	6	4	0	0	49	37	59	56	2	0	51	47	14

Column #	83	84	85	86	87	88	89	90	91	92
TOP	36	33	18	0	60	13	58	24	14	55
BOTTOM	31	0	19	18	13	47	10	13	13	57

The total span of nets = 1515 and the average span of nets of this channel = 25.25. The number of terminals of this channel = 157; thus, the number of non-terminals = 27 and the average number of terminals per net = 2.6167, as the channel length is 92.

Sample general channel # 1 that contains only 80 nets:

Column #	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
TOP	80	0	38	0	17	30	0	0	61	68	0	76	66	8	17	8	50	0	38	17	8
BOTTOM	50	44	48	0	73	62	79	79	0	73	58	74	8	71	53	0	30	0	59	52	13

Column #	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41
TOP	63	35	0	6	0	78	75	64	44	5	1	6	1	0	33	57	0	35	33	56
BOTTOM	35	0	0	35	53	57	76	24	62	54	62	44	33	5	47	54	14	3	50	13

Column #	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62
TOP	42	12	72	19	0	37	51	48	78	24	67	4	67	58	12	0	28	70	0	74	74
BOTTOM	0	0	38	44	25	12	12	0	3	34	53	57	34	31	0	64	34	53	65	28	14

Column #	63	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80	81	82
TOP	19	41	67	7	65	67	7	20	63	20	76	0	27	59	66	21	55	9	25	61
BOTTOM	65	4	32	9	0	26	79	60	23	41	75	7	26	7	22	52	74	80	16	71

Column #	83	84	85	86	87	88	89	90	91	92	93	94	95	96	97	98	99	100	101
TOP	62	16	21	49	59	36	45	78	56	47	68	0	16	29	67	37	10	55	67
BOTTOM	31	77	0	0	49	26	54	15	27	26	16	21	63	59	18	40	40	29	22

Column #	102	103	104	105	106	107	108	109	110	111	112	113	114	115	116
TOP	0	78	26	39	15	0	77	77	23	69	60	32	0	45	0
BOTTOM	40	36	0	70	69	11	2	51	10	42	11	0	46	43	77

Column #	117	118	119	120	121	122	123
TOP	49	0	76	72	2	78	46
BOTTOM	18	43	39	46	39	0	43

The total span of nets = 2611 and the average span of nets of this channel = 32.6375. The number of terminals of this channel = 210; thus, the number of non-terminals = 36 and the average number of terminals per net = 2.625, as the channel length is 123.

Sample general channel # 2 that contains only 80 nets:

Column #	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
TOP	33	0	28	0	0	75	0	32	45	48	49	0	34	46	0	55	0	69	45	63	80
BOTTOM	76	20	76	31	0	73	38	48	34	0	73	38	64	33	46	11	46	0	0	55	0

Column #	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41
TOP	38	39	80	33	80	61	0	31	43	0	0	78	6	74	0	31	0	23	0	50
BOTTOM	46	33	80	0	31	11	16	20	28	61	63	39	39	0	0	67	0	0	48	16

Column #	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62
TOP	37	55	13	16	13	19	0	78	53	6	0	0	0	72	32	0	55	72	0	76	19
BOTTOM	0	13	0	55	61	0	6	0	16	52	59	53	79	38	39	38	38	72	61	79	0

Column #	63	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80	81	82
TOP	37	53	0	0	14	39	14	0	23	65	75	22	17	49	30	73	24	10	12	29
BOTTOM	39	14	37	59	45	0	0	14	48	0	53	22	72	12	22	22	17	73	0	37

Column #	83	84	85	86	87	88	89	90	91	92	93	94	95	96	97	98	99	100	101
TOP	0	66	0	71	70	0	0	25	10	0	73	0	0	0	0	15	8	43	71
BOTTOM	0	0	0	29	10	29	10	0	0	4	9	27	35	24	8	4	8	35	9

Column #	102	103	104	105	106	107	108	109	110	111	112	113	114	115	116
TOP	75	25	60	25	73	0	64	15	0	25	78	54	51	18	74
BOTTOM	8	1	64	18	74	55	18	0	50	64	1	67	0	57	51

Column #	117	118	119	120	121	122	123	124	125	126	127	128	129	130	131	132
TOP	21	0	56	0	52	0	0	18	44	35	60	30	0	26	59	29
BOTTOM	29	70	41	50	0	70	68	42	58	62	0	0	69	0	27	60

Column #	133	134	135	136	137	138	139	140	141	142	143	144	145	146	147	148
TOP	40	78	0	77	0	7	54	58	44	71	7	26	5	65	5	7
BOTTOM	70	0	77	42	77	59	77	0	57	0	70	54	58	57	21	77

Column #	149	150	151	152	153	154	155	156	157	158	159	160	161	162
TOP	5	0	0	0	0	62	58	57	68	0	0	2	0	60
BOTTOM	41	5	47	36	0	0	77	36	47	0	66	3	0	36

Column #	163	164	165	166	167	168	169	170
TOP	0	56	40	68	0	66	2	3
BOTTOM	51	26	2	0	0	0	0	2

The total span of nets = 2927 and the average span of nets of this channel = 36.5875. The number of terminals of this channel = 250; thus, the number of non-terminals = 90 and the average number of terminals per net = 3.125, as the channel length is 170.

Sample general channel # 1 that contains only 100 nets:

Column #	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
TOP	42	0	0	35	0	67	42	49	0	0	0	0	0	46	8	42	0	72	0	0	0
BOTTOM	46	0	10	21	0	46	0	0	0	10	9	0	10	71	35	0	0	20	79	0	8

Column #	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41
TOP	0	46	0	81	71	88	9	26	26	83	0	0	71	0	0	0	0	0	47	0
BOTTOM	0	0	88	81	9	0	42	76	26	6	35	46	0	98	67	81	0	21	35	88

Column #	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62
TOP	84	59	81	68	38	77	95	0	0	0	67	2	2	92	49	20	66	0	0	0	0
BOTTOM	0	65	61	94	24	76	18	18	20	6	74	78	2	2	0	74	24	11	30	26	75

Column #	63	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80	81	82
TOP	18	0	0	0	41	0	97	0	93	64	58	52	97	91	3	52	72	0	0	50
BOTTOM	97	65	53	75	53	0	91	97	97	11	0	0	32	87	32	0	0	0	0	0

Column #	83	84	85	86	87	88	89	90	91	92	93	94	95	96	97	98	99	100	101
TOP	77	0	52	0	91	0	60	54	74	0	3	0	34	0	45	30	0	34	58
BOTTOM	43	40	99	76	32	71	0	0	79	0	0	52	64	44	38	98	0	0	44

Column #	102	103	104	105	106	107	108	109	110	111	112	113	114	115	116
TOP	62	47	58	45	56	72	44	89	0	41	62	0	4	90	0
BOTTOM	45	0	76	44	59	0	64	89	4	87	96	0	99	59	80

Column #	117	118	119	120	121	122	123	124	125	126	127	128	129	130	131	132
TOP	83	34	45	89	0	85	43	84	51	48	61	44	86	89	4	40
BOTTOM	83	0	65	78	76	44	92	48	87	50	31	79	70	62	16	0

Column #	133	134	135	136	137	138	139	140	141	142	143	144	145	146	147	148
TOP	57	65	100	54	0	23	0	51	0	54	12	12	12	28	0	51
BOTTOM	63	45	0	66	94	53	68	31	28	29	15	82	55	23	80	0

Column #	149	150	151	152	153	154	155	156	157	158	159	160	161	162
TOP	51	90	51	31	0	0	73	96	0	19	25	0	69	56
BOTTOM	28	0	77	80	73	1	55	60	51	23	74	29	19	14

Column #	163	164	165	166	167	168	169	170	171	172	173	174	175	176
TOP	0	0	0	1	23	16	94	25	70	37	1	78	0	54
BOTTOM	87	14	57	27	64	36	82	95	98	73	77	96	60	27

Column #	177	178	179	180	181	182	183	184	185	186	187	188	189	190
TOP	14	94	82	7	7	17	86	13	5	99	15	5	37	5
BOTTOM	0	0	85	77	13	7	22	0	48	27	0	5	85	62

Column #	191	192	193	194	195	196	197	198	199	200	201	202	203	204
TOP	5	0	48	36	0	33	13	22	0	69	87	100	93	73
BOTTOM	92	90	0	5	69	0	99	0	87	0	39	27	70	22

Column #	205	206	207	208	209	210	211	212	213
TOP	0	0	80	17	39	37	0	57	33
BOTTOM	0	0	0	22	0	94	63	0	0

The total span of nets = 5023 and the average span of nets of this channel = 50.23. The number of terminals of this channel = 309; thus, the number of non-terminals = 117 and the average number of terminals per net = 3.09, as the channel length is 213.

In this section, we have included only 19 sample general channel instances (out of 4800 instances) of smaller in size that we have generated randomly using algorithm *General_Random_Channel_Generator* developed in Chapter 5 of this thesis. Even in these example general channel specifications, we may observe that there are *blank columns* (containing only non-terminals) or *trivial columns* (pins hold terminals of the same net) that have also been introduced randomly into the channel. In general, the channel lengths vary for a given number of nets, as the number of terminals per net is not fixed, and the number of non-terminals is also determined randomly. The minimum number of terminals for each net introduced is restricted to two, and the maximum is six. All these generated channels contain vertical constraints and never contain any cyclic vertical constraints.

Based on a different objective under consideration, viz., wire length minimization or routing for multi-layer channels, different sets of channels with dissimilar or special criteria can also be devised with suitable changes as desired in the instance generating algorithms developed in this thesis (in Chapter 5). Now, in the next section, we include some of the hardcopy routing solutions (for channel instances that are smaller in length), for both simple as well as general, that we have computed after execution of the algorithms devised in Chapter 4. For a selected set of random channel instance generated, first of all, we have computed an initial routing solution (using some relevant algorithm), then we have implemented crosstalk minimization algorithms *Track_Change* and *Net_Change* one after another, and obtained minimized crosstalk routing solutions. Results we have acquired are highly encouraging.

6.3 Results of Crosstalk Minimization Algorithms

Crosstalk minimization algorithms *Track_Change* and *Net_Change* have been designed in Chapter 4, and most of the implemented results are included in this chapter. Results for only 540 simple channel instances (nine sets of simple channels each comprising 60 smaller randomly generated channel specifications) have been depicted in Table 4.1 along with 14 existing benchmark (general) channel instances (in Table 4.2), and only five hardcopy routing solutions are included there.

In this section, we incorporate most of the exhaustive computations we have made as a part of our research in the form of tables, where each datum in each of the tables represents an average value of a vast number of instances of a similar kind. Parenthetically, as the number of instances we have handled is large, scopes for showing all routing solutions is very little (as we have included even less than 0.31% of all channel instances we have randomly generated in the previous section). However, we have illustrated a reasonable number of hardcopy routing solutions that we may find next, mostly for (randomly generated) smaller channel specifications.

Table 6.1: Performance of crosstalk minimization algorithms *Track_Change_Simple* and *Net_Change* after their successive execution for computing reduced crosstalk routing solutions for simple channel instances in two-layer no-dogleg (channel) routing.

Number of nets	**Initial crosstalk after *MCC1***	**Crosstalk after algorithm *Track_Change _Simple***	**Reduction in crosstalk after *Track_Change _Simple* (%)**	**Crosstalk after algorithm *Net_Change***	**Reduction in crosstalk after *Net _Change* (%)**
10	25	16	36.00	16	36.00
15	69	47	31.88	46	33.33
20	131	90	31.30	88	32.82
25	221	155	29.86	153	30.77
30	322	226	29.81	223	30.75
35	465	330	29.03	326	29.89
40	611	439	28.15	434	28.97
45	797	569	28.61	562	29.49
50	1012	733	27.57	726	28.26
60	1483	1089	26.57	1079	27.24
70	2045	1522	25.57	1509	26.21
80	2745	2031	26.01	2016	26.56
90	3484	2618	24.86	2598	25.43
100	4290	3196	25.50	3173	26.04
110	5264	3945	25.06	3919	25.55
120	6284	4784	23.87	4754	24.35
130	7450	5590	24.97	5557	25.41
140	8661	6588	23.93	6549	24.39
150	9999	7570	24.29	7527	24.72
160	11430	8721	23.70	8675	24.10
170	12877	9881	23.27	9832	23.65
180	14533	11224	22.77	11166	23.17
190	16261	12462	23.36	12400	23.74
200	18163	13928	23.32	13860	23.69
220	21869	16855	22.93	16782	23.26
240	26109	20145	22.84	20051	23.20
260	30828	23855	22.62	23764	22.91
280	36235	28104	22.44	28009	22.70

Number of Nets	**Initial Crosstalk after *MCC1***	**Crosstalk after algorithm *Track_Change _Simple***	**Reduction in crosstalk after *Track_Change _Simple* (%)**	**Crosstalk after algorithm *Net_Change***	**Reduction in crosstalk after *Net_ Change* (%)**
300	41255	32078	22.24	31962	22.53
320	47153	36617	22.34	36484	22.63
340	53362	41378	22.46	41235	22.73
360	60281	46714	22.51	46562	22.76
380	67050	52369	21.90	52205	22.14
400	74467	58258	21.77	58073	22.02
420	82586	64563	21.82	64362	22.07
440	90378	70731	21.74	70520	21.97
460	98804	77236	21.83	77001	22.07
480	108033	84561	21.73	84317	21.95
500	117339	92054	21.55	91800	21.77
600	168910	132363	21.64	132024	21.84
700	231763	182698	21.17	182235	21.37
800	303586	238811	21.34	238273	21.51
900	383901	304147	20.77	303468	20.95
1000	474072	374947	20.91	374182	21.07

6.3.1 Results of Crosstalk Minimization Algorithms for Simple Channel Instances

In this section, we include the results of crosstalk minimization algorithms for simple channel instances we have devised in Chapter 4 of this thesis. Specifically, for this experimentation, we have randomly generated 200 instances for a given number of nets, and the data in a row are obtained by making an average of each set of relevant 200 executed data, where the number of nets varies from 10 through 1000 only. In carrying out this test, we have not considered even larger channel instances as we interestingly observed that the percentage reduction in crosstalk is getting saturated to a value a little below 21% after execution of algorithm *Track_Change_Simple* and a value a bit above 21% after execution of algorithm *Net_Change*, when the algorithms are implemented in succession, over the initial amount of crosstalk measured in the first density routing solutions after execution of algorithm *Minimum_Clique_Cover_1* (*MCC1*) [49, 52, 53], as the number of nets goes above 800. The amount of average crosstalk (after execution of respective algorithm) is rounded off to its nearest integer value (for a given number of nets), and the percentage reduction in crosstalk is also calculated up to two decimal places. All these results have been included in Table 6.1 as the performance of both the crosstalk minimization algorithms developed in this thesis and employed one after the other.

The variation in crosstalk minimization (as shown in Table 6.1) is graphically depicted in Figure 6.1, where the percentage reduction in crosstalk in respective routing solutions is included after each of the algorithms *Track_Change_Simple* and *Net_Change*. Now we include some of the selected hardcopy routing solutions that we obtained after execution of each of the algorithms *MCC1*, *Track_Change_Simple*, and *Net_Change*. These routing solutions are shown in Figures 6.2 through 6.19 for channels containing nets 10 through 150. In each of these figures, (a) displays the routing solution obtained after *MCC1*, (b) depicts the significantly reduced crosstalk routing solution after execution of *Track_Change_Simple*, and (c) includes the mostly reduced crosstalk routing solution obtained after execution of algorithm *Net_Change*.

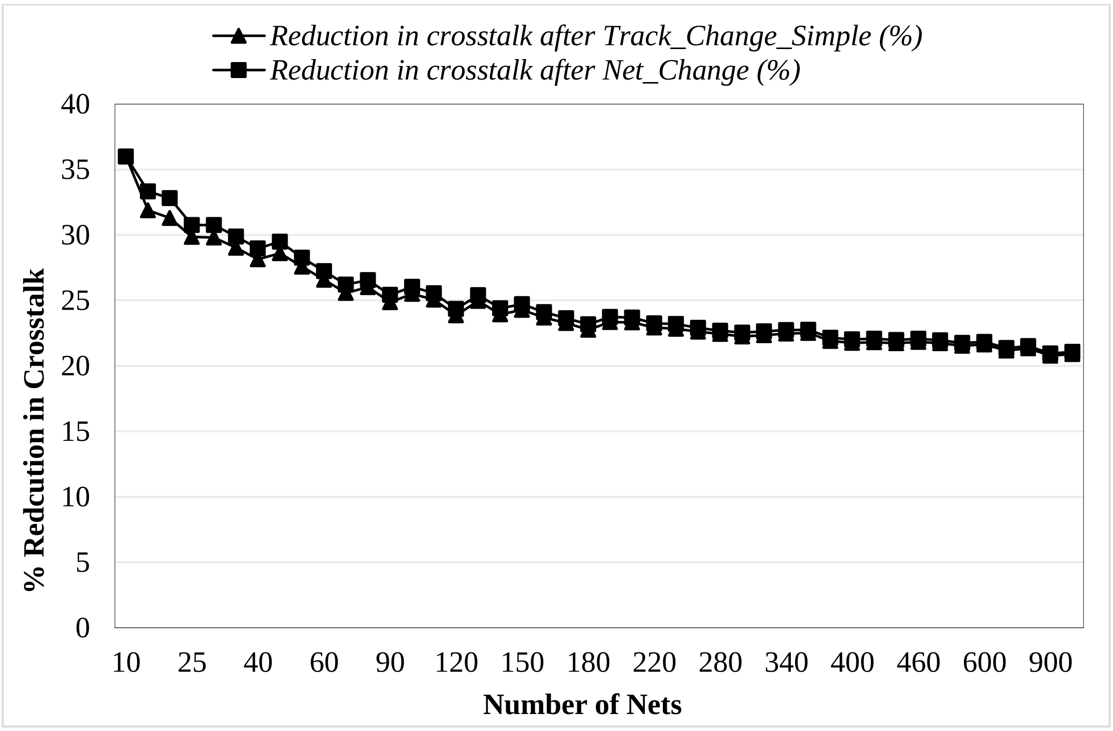
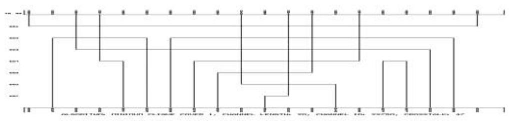
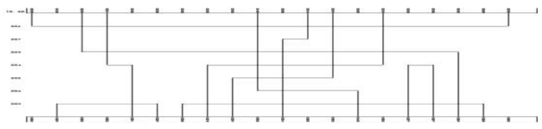


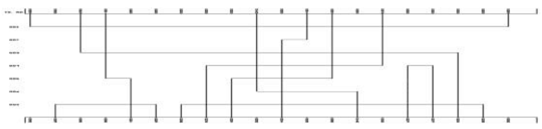
Figure 6.1: Percentage reduction in crosstalk (of routing solutions) versus the number of nets after successive executions of each of the algorithms, *Track_Change_Simple* and *Net_Change* over the initial amount of crosstalk of routing solutions computed after algorithm *Minimum_Clique_Cover_1* for simple instances in two-layer no-dogleg channel routing.



(a)

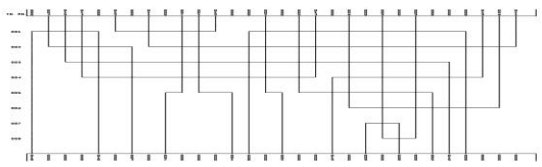


(b)

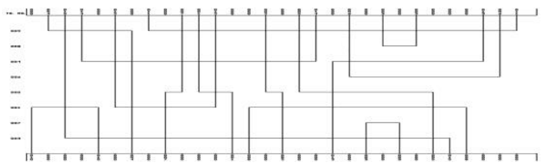


(c)

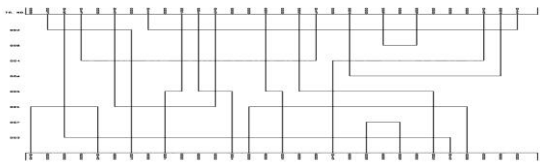
Figure 6.2: (a) The initial crosstalk of a simple channel instance comprising 10 nets (and channel length 20) is 47 units after execution of *Minimum_Clique_Cover_1*. **(b)** Crosstalk after execution of algorithm *Track_Change_Simple* is 22 units. **(c)** Crosstalk after execution of algorithm *Net_Change* is 21 units only.



(a)

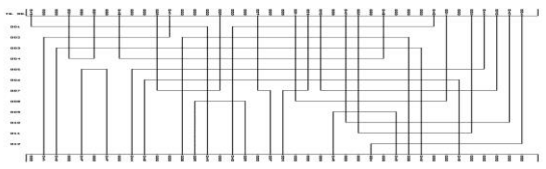


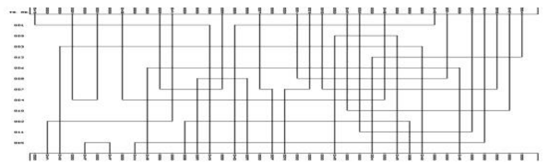
(b)



(c)

Figure 6.3: (a) The initial crosstalk of a simple channel instance comprising 15 nets (and channel length 30) is 83 units after execution of *Minimum_Clique_Cover_1*. **(b)** Crosstalk after execution of algorithm *Track_Change_Simple* is 32 units. **(c)** Crosstalk after execution of algorithm *Net_Change* is also 32 units only.


(a)


(b)

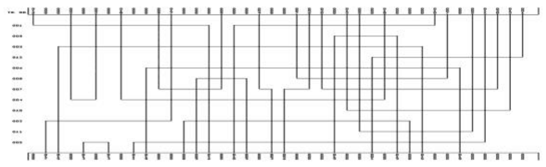
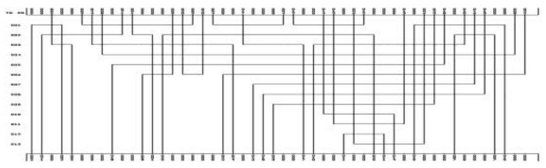
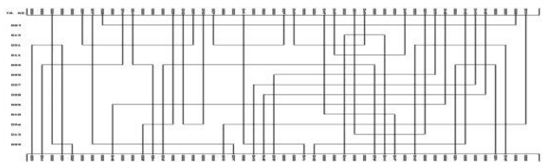

(c)

Figure 6.4: **(a)** The initial crosstalk of a simple channel instance comprising 20 nets (and channel length 40) is 180 units after execution of *Minimum_Clique_Cover_1*. **(b)** Crosstalk after execution of algorithm *Track_Change_Simple* is 84 units. **(c)** Crosstalk after execution of algorithm *Net_Change* is also 84 units only.


(a)


(b)

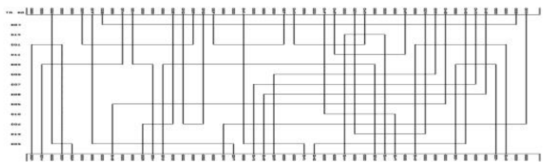
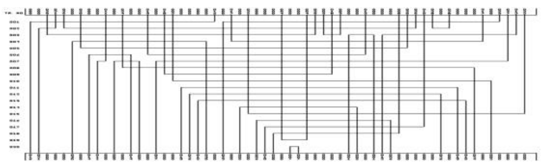
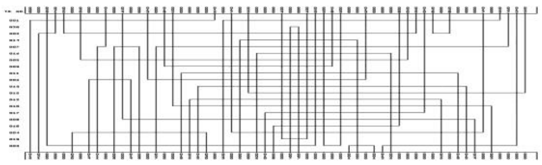

(c)

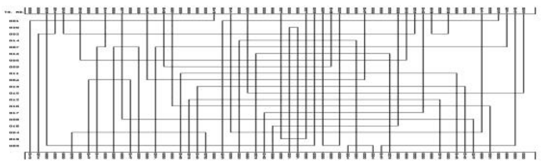
Figure 6.5: **(a)** The initial crosstalk of a simple channel instance comprising 25 nets (and channel length 50) is 229 units after execution of *Minimum_Clique_Cover_1*. **(b)** Crosstalk after execution of algorithm *Track_Change_Simple* is 106 units. **(c)** Crosstalk after execution of algorithm *Net_Change* is also 106 units only.



(a)

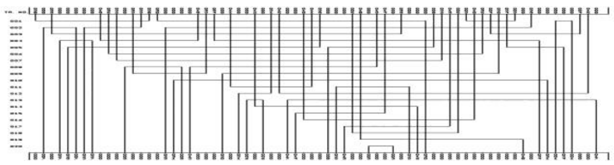


(b)

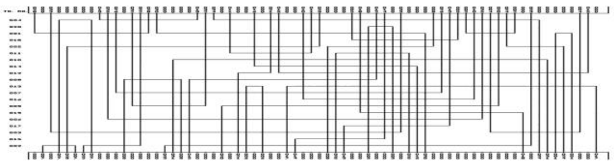


(c)

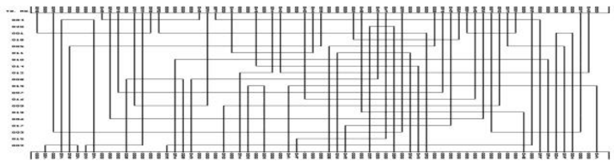
Figure 6.6: (a) The initial crosstalk of a simple channel instance comprising 30 nets (and channel length 60) is 473 units after execution of *Minimum_Clique_Cover_1*. **(b)** Crosstalk after execution of algorithm *Track_Change_Simple* is 295 units. **(c)** Crosstalk after execution of algorithm *Net_Change* is also 295 units only.



(a)

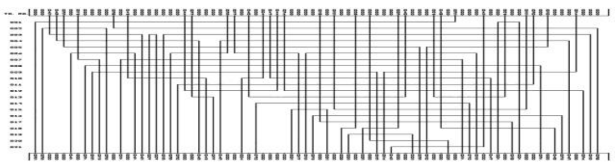


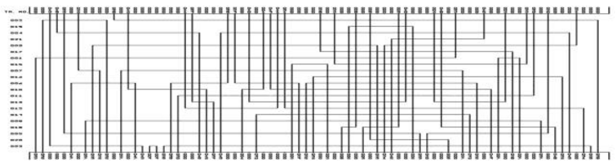
(b)



(c)

Figure 6.7: (a) The initial crosstalk of a simple channel instance comprising 35 nets (and channel length 70) is 510 units after execution of *Minimum_Clique_Cover_1*. **(b)** Crosstalk after execution of algorithm *Track_Change_Simple* is 264 units. **(c)** Crosstalk after execution of algorithm *Net_Change* is also 264 units only.


(a)


(b)

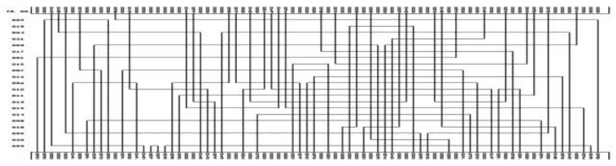
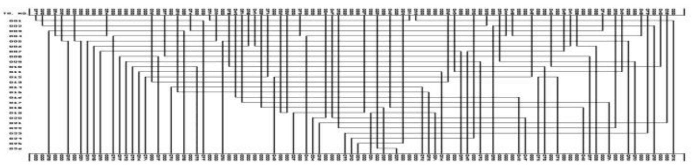
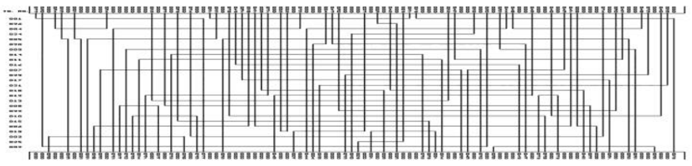

(c)

Figure 6.8: (a) The initial crosstalk of a simple channel instance comprising 40 nets (and channel length 80) is 776 units after execution of *Minimum_Clique_Cover_1*. **(b)** Crosstalk after execution of algorithm *Track_Change_Simple* is 450 units. **(c)** Crosstalk after execution of algorithm *Net_Change* is 448 units only.


(a)


(b)

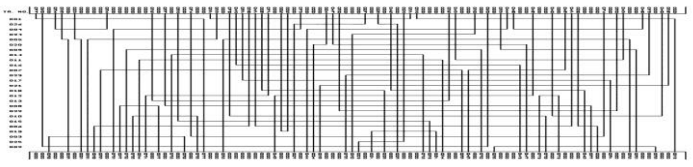
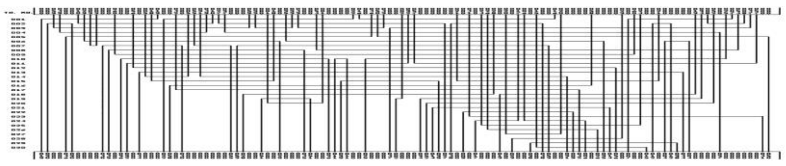
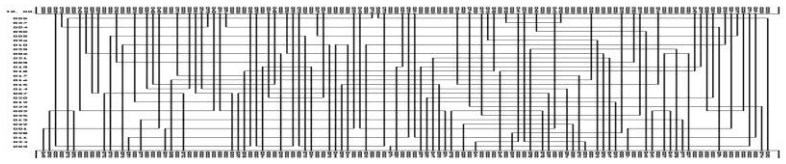

(c)

Figure 6.9: (a) The initial crosstalk of a simple channel instance comprising 50 nets (and channel length 100) is 1242 units after execution of *Minimum_Clique_Cover_1*. **(b)** Crosstalk after execution of algorithm *Track_Change_Simple* is 771 units. **(c)** Crosstalk after execution of algorithm *Net_Change* is 770 units only.


(a)


(b)

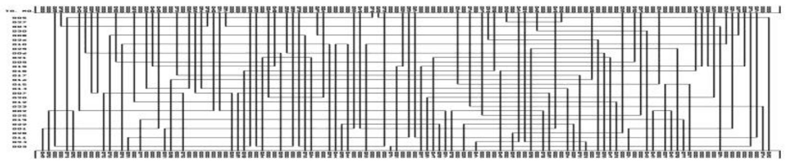
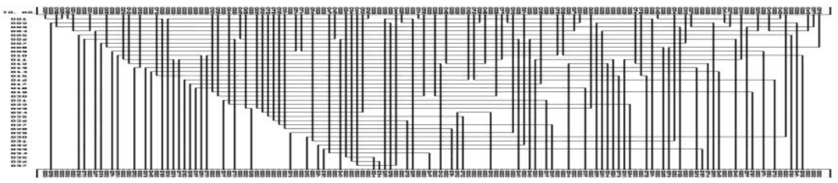
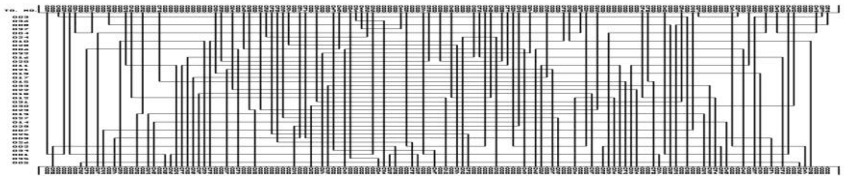

(c)

Figure 6.10: **(a)** The initial crosstalk of a simple channel instance comprising 60 nets (and channel length 120) is 1571 units after execution of *Minimum_Clique_Cover_1*. **(b)** Crosstalk after execution of algorithm *Track_Change_Simple* is 813 units. **(c)** Crosstalk after execution of algorithm *Net_Change* is 793 units only.


(a)


(b)

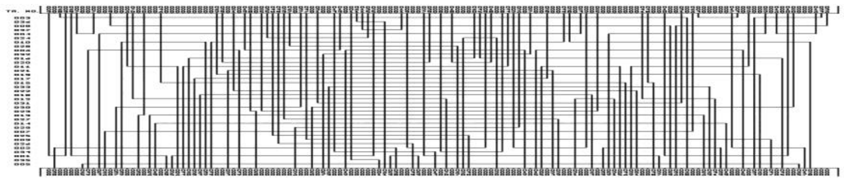
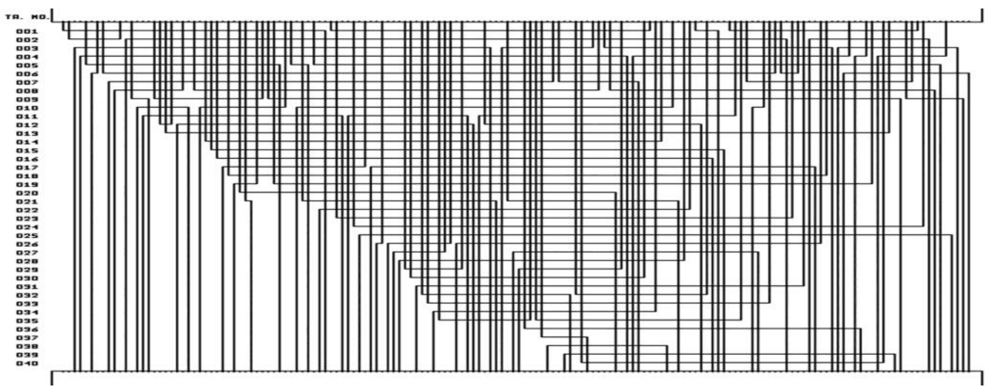
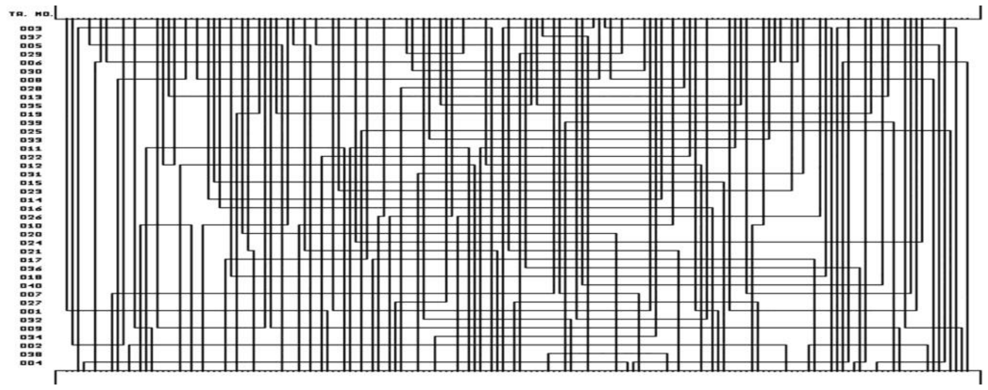

(c)

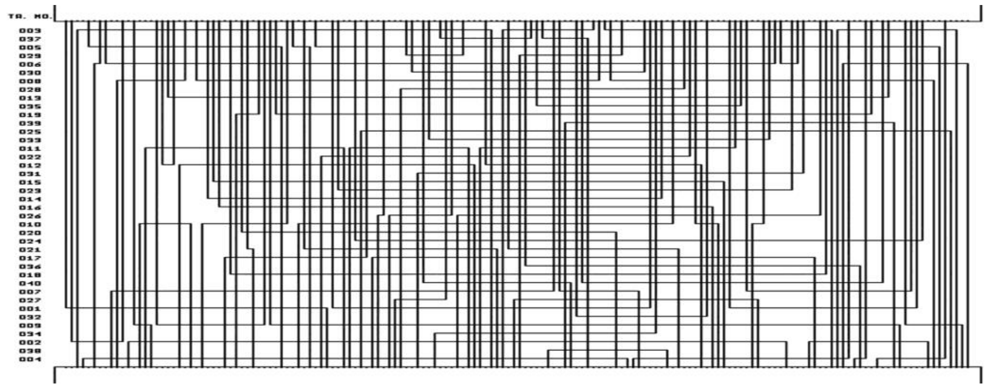
Figure 6.11: **(a)** The initial crosstalk of a simple channel instance comprising 70 nets (and channel length 140) is 2291 units after execution of *Minimum_Clique_Cover_1*. **(b)** Crosstalk after execution of algorithm *Track_Change_Simple* is 1308 units. **(c)** Crosstalk after execution of algorithm *Net_Change* is 1304 units only.



(a)

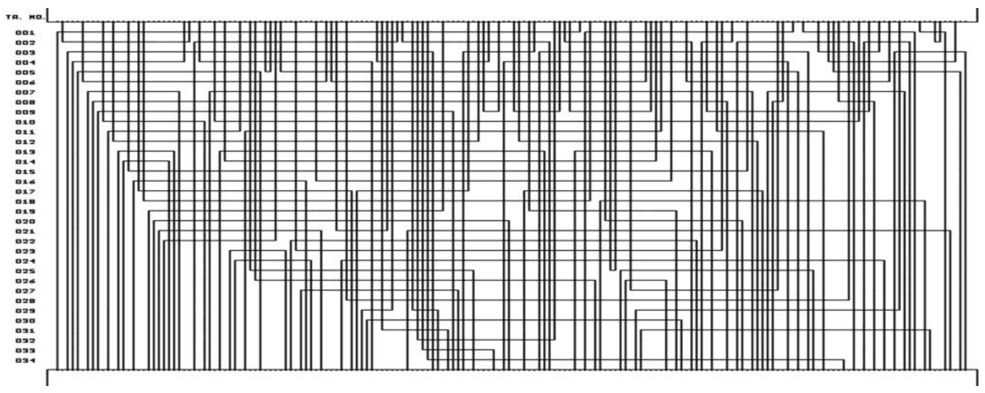


(b)

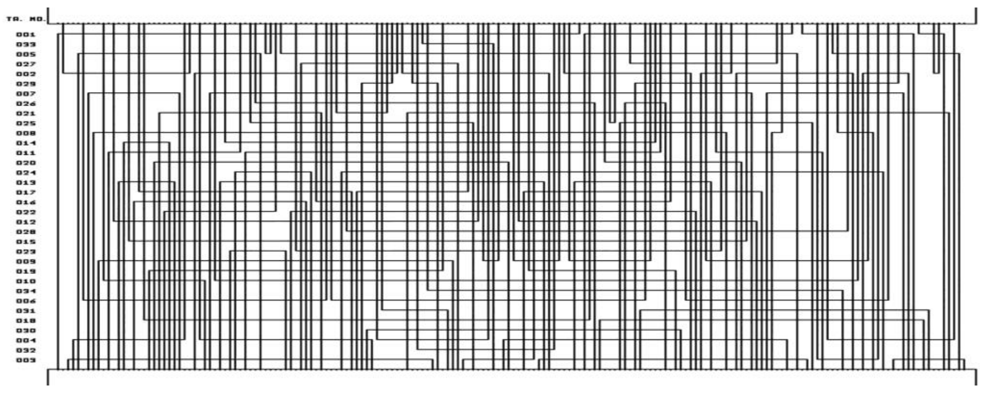


(c)

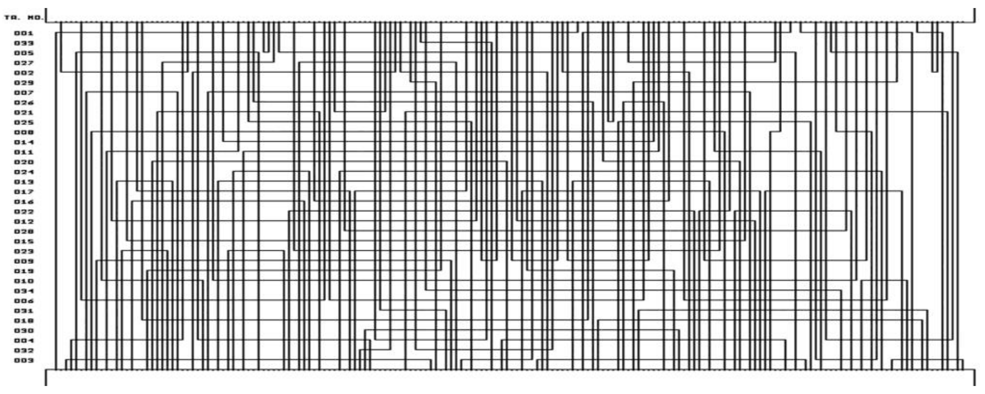
Figure 6.12: (a) The initial crosstalk of a simple channel instance comprising 80 nets (and channel length 160) is 2921 units after execution of *Minimum_Clique_Cover_1*. **(b)** Crosstalk after execution of algorithm *Track_Change_Simple* is 1810 units. **(c)** Crosstalk after execution of algorithm *Net_Change* is 1805 units only.



(a)

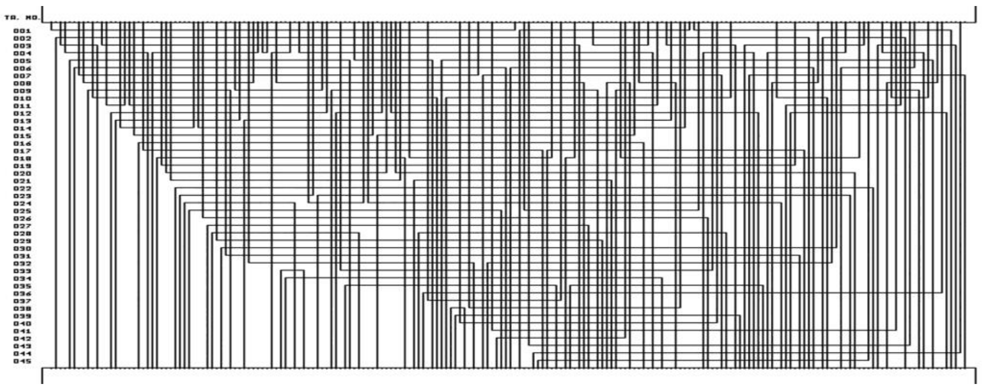


(b)

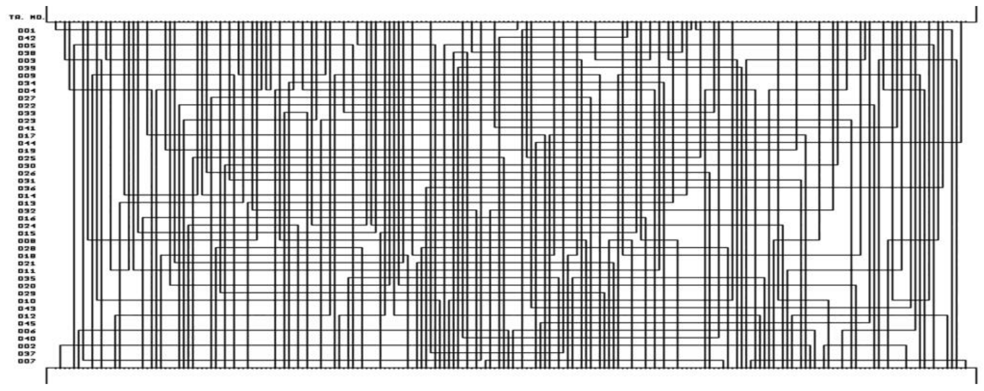


(c)

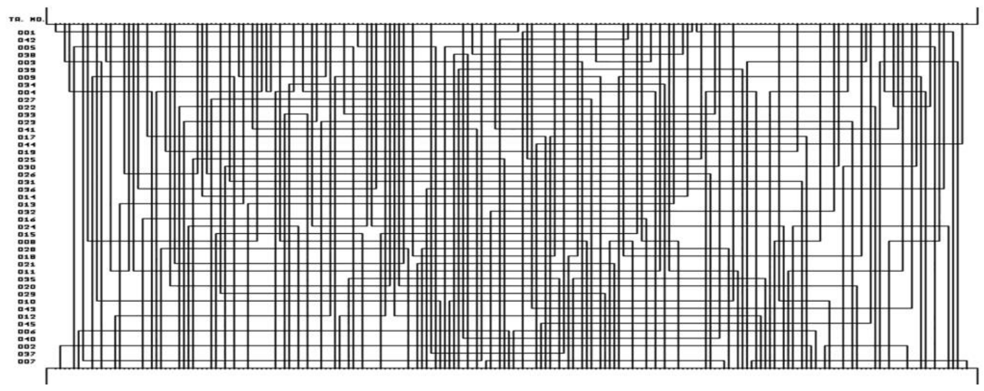
Figure 6.13: (a) The initial crosstalk of a simple channel instance comprising 90 nets (and channel length 180) is 2917 units after execution of *Minimum_Clique_Cover_1*. **(b)** Crosstalk after execution of algorithm *Track_Change_Simple* is 2270 units. **(c)** Crosstalk after execution of algorithm *Net_Change* is 2199 units only.



(a)

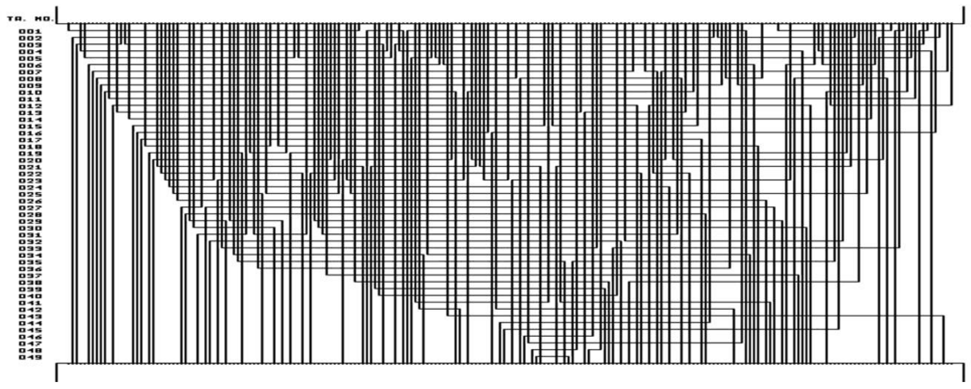


(b)

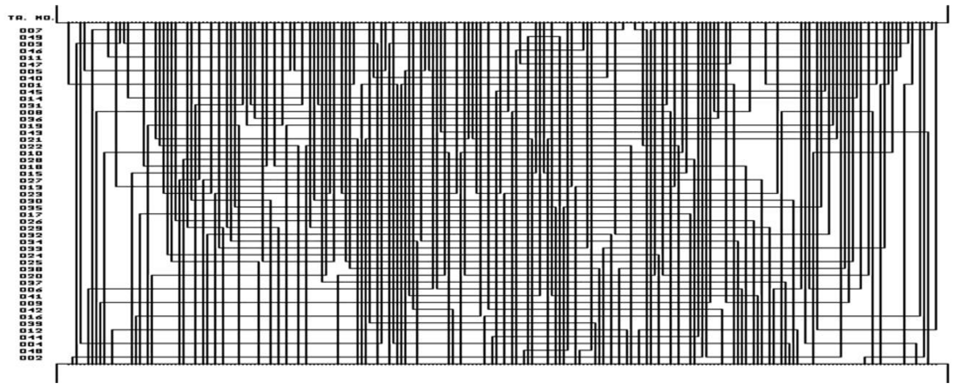


(c)

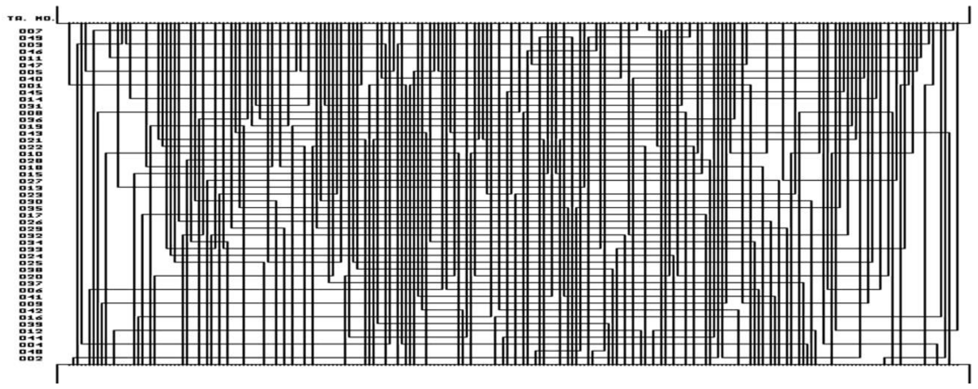
Figure 6.14: (a) The initial crosstalk of a simple channel instance comprising 100 nets (and channel length 200) is 4525 units after execution of *Minimum_Clique_Cover_1*. **(b)** Crosstalk after execution of algorithm *Track_Change_Simple* is 3223 units. **(c)** Crosstalk after execution of algorithm *Net_Change* is 3132 units only.



(a)

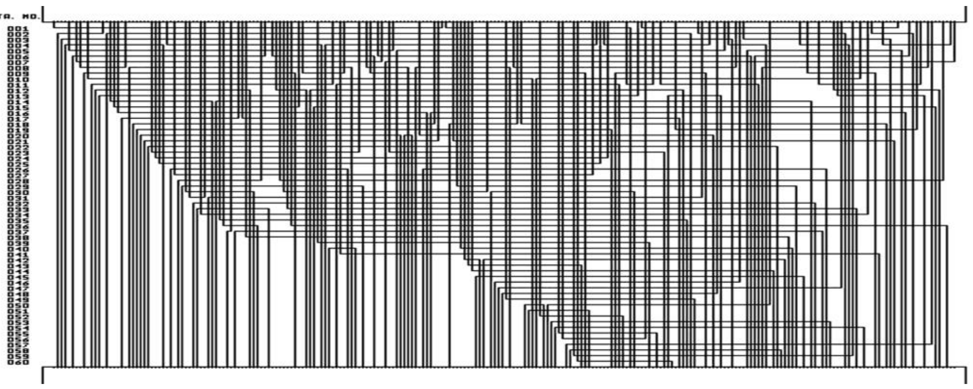


(b)

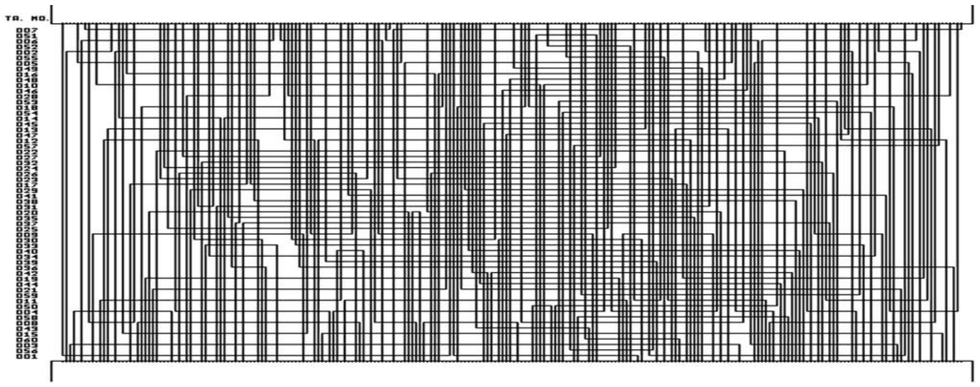


(c)

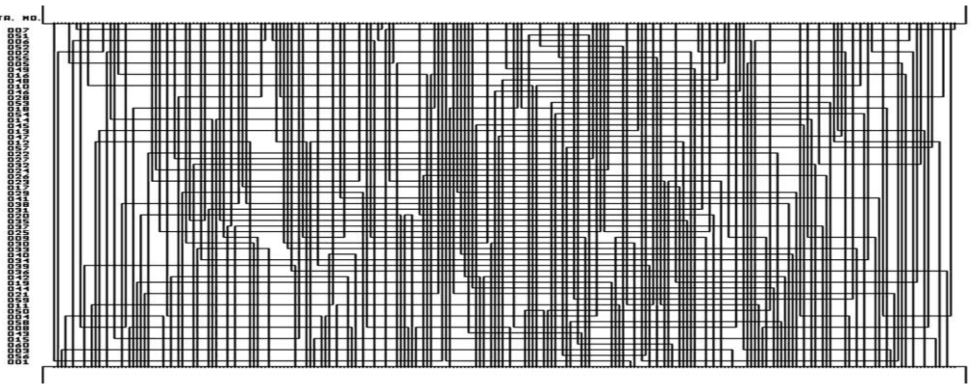
Figure 6.15: (a) The initial crosstalk of a simple channel instance comprising 110 nets (and channel length 220) is 5772 units after execution of *Minimum_Clique_Cover_1*. **(b)** Crosstalk after execution of algorithm *Track_Change_Simple* is 4281 units. **(c)** Crosstalk after execution of algorithm *Net_Change* is 4208 units only.



(a)

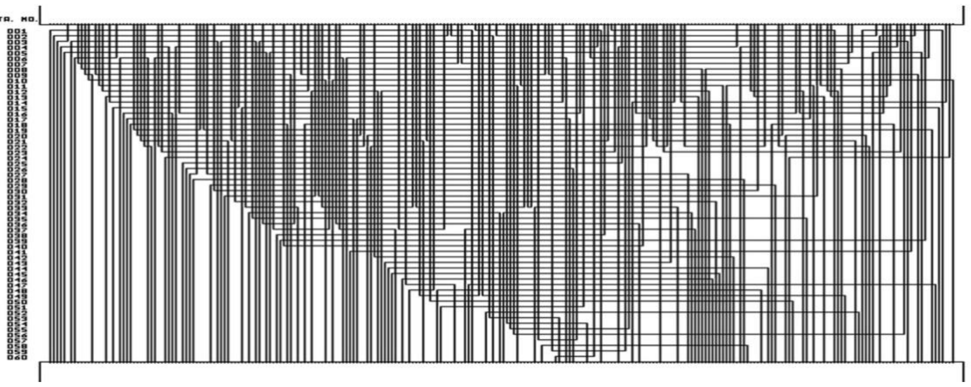


(b)

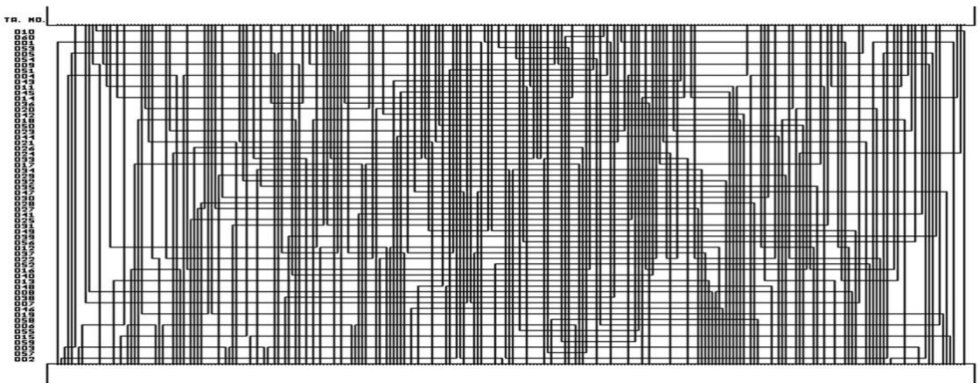


(c)

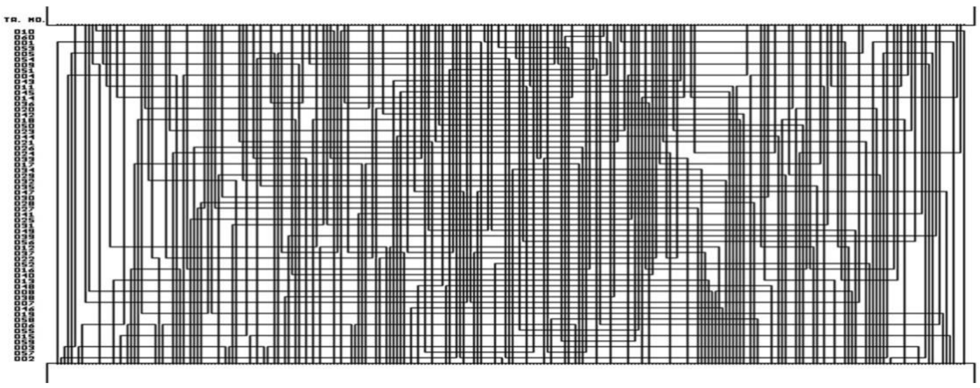
Figure 6.16: (a) The initial crosstalk of a simple channel instance comprising 120 nets (and channel length 240) is 6725 units after execution of *Minimum_Clique_Cover_1*. **(b)** Crosstalk after execution of algorithm *Track_Change_Simple* is 4151 units. **(c)** Crosstalk after execution of algorithm *Net_Change* is 4031 units only.



(a)

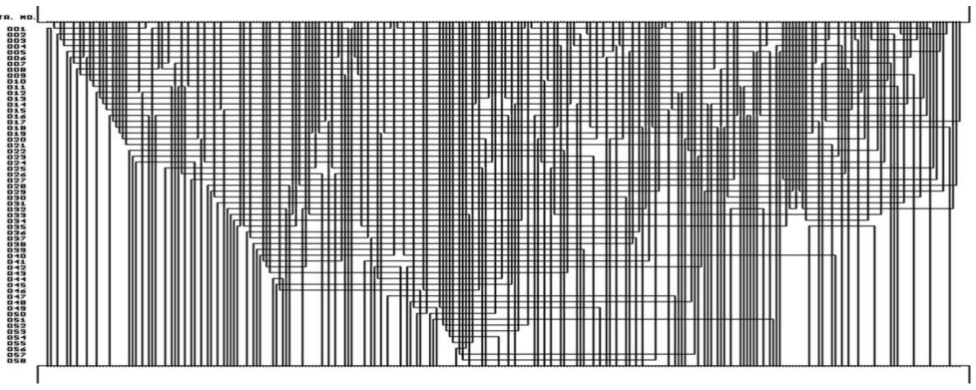


(b)

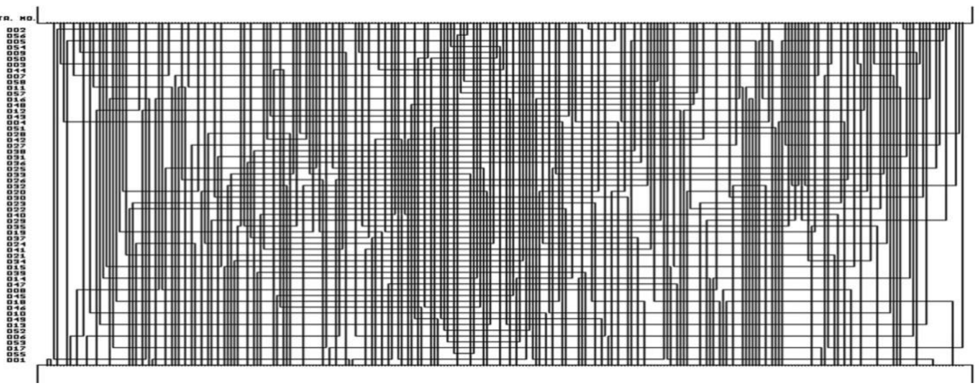


(c)

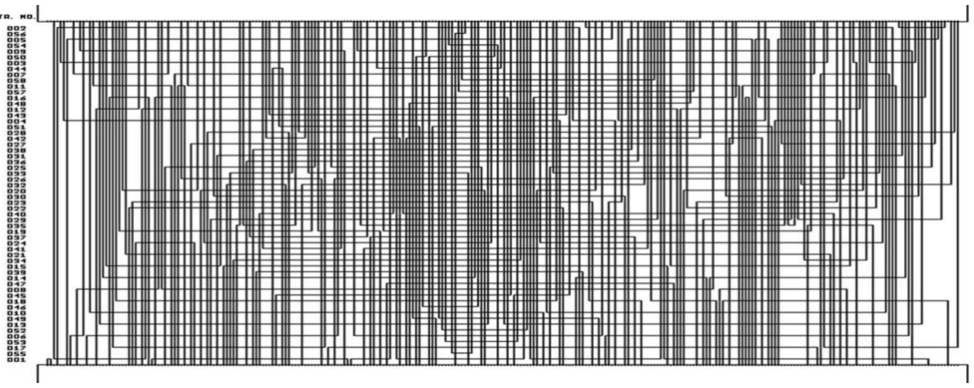
Figure 6.17: (a) The initial crosstalk of a simple channel instance comprising 130 nets (and channel length 260) is 7648 units after execution of *Minimum_Clique_Cover_1*. **(b)** Crosstalk after execution of algorithm *Track_Change_Simple* is 4750 units. **(c)** Crosstalk after execution of algorithm *Net_Change* is 4689 units only.



(a)

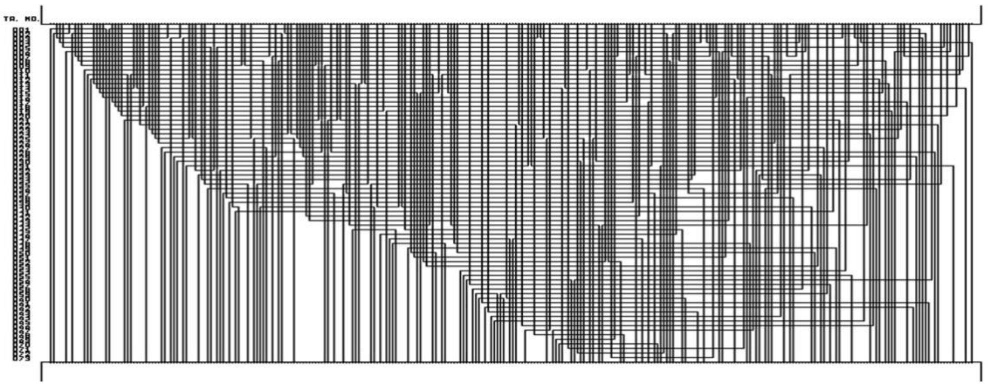


(b)

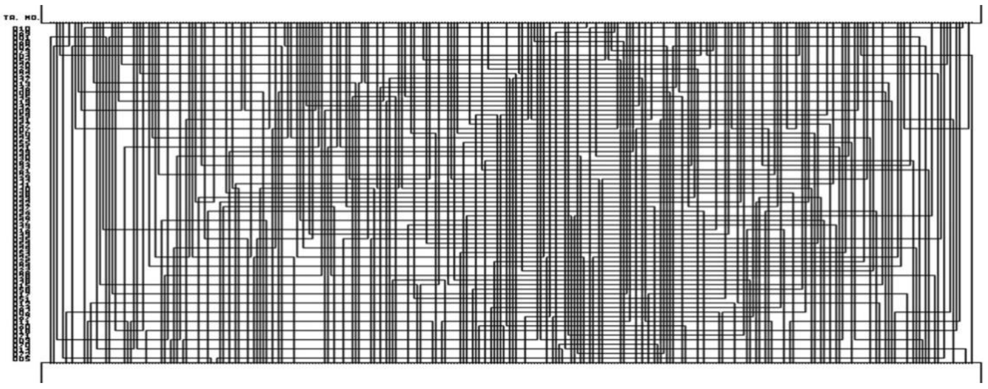


(c)

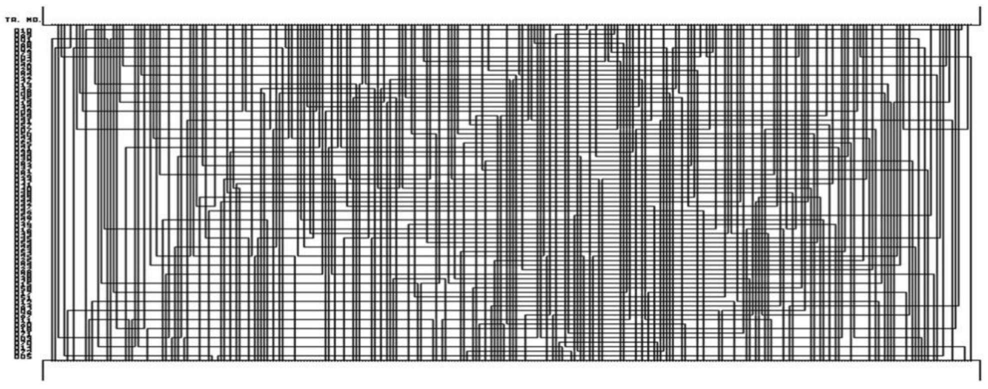
Figure 6.18: (a) The initial crosstalk of a simple channel instance comprising 140 nets (and channel length 280) is 8054 units after execution of *Minimum_Clique_Cover_1*. **(b)** Crosstalk after execution of algorithm *Track_Change_Simple* is 4718 units. **(c)** Crosstalk after execution of algorithm *Net_Change* is 4697 units only.



(a)



(b)



(c)

Figure 6.19: (a) The initial crosstalk of a simple channel instance comprising 150 nets (and channel length 300) is 11316 units after execution of *Minimum_Clique_Cover_1*. **(b)** Crosstalk after execution of algorithm *Track_Change_Simple* is 7049 units. **(c)** Crosstalk after execution of algorithm *Net_Change* is 6978 units only.

6.3.2 Results of Crosstalk Minimization Algorithms for General Channel Instances

In this section, we include the results of crosstalk minimization algorithms for general channel instances we have devised in Chapter 4 of this thesis. Incidentally, in this thesis, we have developed two versions of the algorithm *Track_Change_General*; in each case starting from a two-layer no-dogleg routing solution that has been computed after execution of algorithm *Track_Assignment_Heuristic* (*TAH*) [49, 51] devised for computing minimum area channel routing solutions for general instances of channel specifications. To start with a minimum area two-layer channel routing solution of our desire, one may implement any one of many other existing channel routing algorithms as well [10, 33, 49, 51, 71, 73, 96]. Nevertheless, after a feasible two-layer routing solution is obtained, we compute the RVCG, which is a directed acyclic graph (DAG) of the solution.

We may recall that both the versions of algorithm *Track_Change_General* are iterative in nature. In the first version of the algorithm, we consider all source vertices of the current RVCG in some iteration, and apply algorithm *Track_Change_Simple* (using the concept of effective span of intervals of the tracks, and total span of intervals, if needed) for that set of source vertices for their assignment to the current available topmost tracks as directed by the step of reassignment of the algorithm. In the second version, in each iteration, a most suitable source vertex from the current RVCG is identified for its assignment to the current topmost track such that the overall crosstalk is assumed to be reduced. This algorithm is greedy in nature.

In particular, the first version of algorithm *Track_Change_General* has been implemented in Chapter 4 and executed for the existing 14 benchmark channel instances; results are included in Table 4.2. We may straightforwardly observe from the table that the results computed are encouraging enough. However, as the number of existing general channel instances is not much and we have developed a random channel instance generator in Chapter 5, both for creating simple as well as general channel instances, for the implementation of both the versions of algorithm *Track_Change_General*, we have randomly produced 200 general channel instances and executed each of them to obtain the performance of our algorithms. These results have been included in Tables 6.2 and 6.3. Results in Table 6.2 show the average implemented data for a given number of nets (varying from 20 to 2000) after execution of algorithm *Track_Change_General_Version_I* and results in Table 6.3 include the same after execution of algorithm *Track_Change_General_Version_II*.

Table 6.2: The performance of crosstalk minimization algorithms *Track_Change _General_Version_I* (with the concept of *Track_Change_Simple*) and *Net_Change* after their successive execution for computing reduced crosstalk routing solutions for general channel instances in two-layer no-dogleg (channel) routing.

Number of Nets	**Initial crosstalk after *TAH***	**Crosstalk after algorithm *Track_Change _General***	**Reduction in crosstalk after *Track_Change _General* (%)**	**Crosstalk after algorithm *Net_Change***	**Reduction in crosstalk after *Net_ Change* (%)**
20	135	125	7.41	118	12.59
40	564	530	6.03	494	12.41
60	1363	1291	5.28	1202	11.81
80	2475	2347	5.17	2183	11.80
100	3997	3808	4.73	3534	11.58
150	9042	8626	4.60	8028	11.21
200	16260	15498	4.69	14395	11.47
250	25293	24174	4.42	22492	11.07
300	36612	35004	4.39	32577	11.02
350	49708	47552	4.34	44228	11.02
400	65032	62164	4.41	57691	11.29
450	82857	79077	4.56	73323	11.51
500	102922	98324	4.47	91530	11.07
600	148035	141247	4.59	131355	11.27
700	201940	192513	4.67	179071	11.32
800	265574	252957	4.75	235523	11.32
900	336455	320481	4.75	298856	11.18
1000	412991	393305	4.77	366015	11.37
1500	938379	890801	5.07	831324	11.41
2000	1665471	1579930	5.14	1468851	11.81

In performing this test, we have not considered even larger channel instances (containing number of nets more than 2000) as we intriguingly observed that the percentage reduction in crosstalk is approaching to a value of approximately 5% after execution of algorithm *Track_Change* and a little above 11% after execution of algorithm *Net_Change*, when we implement the first version (using the concept of effective span of intervals) of the algorithm. This comparison is visibly different in the case of implementing the second version (using greedy method) of the algorithm. In this case, the percentage reduction in crosstalk is becoming saturated to a bit above 2%, which is much less, after execution of algorithm *Track_Change* and to a value of approximately 11% after execution of algorithm *Net_Change*.

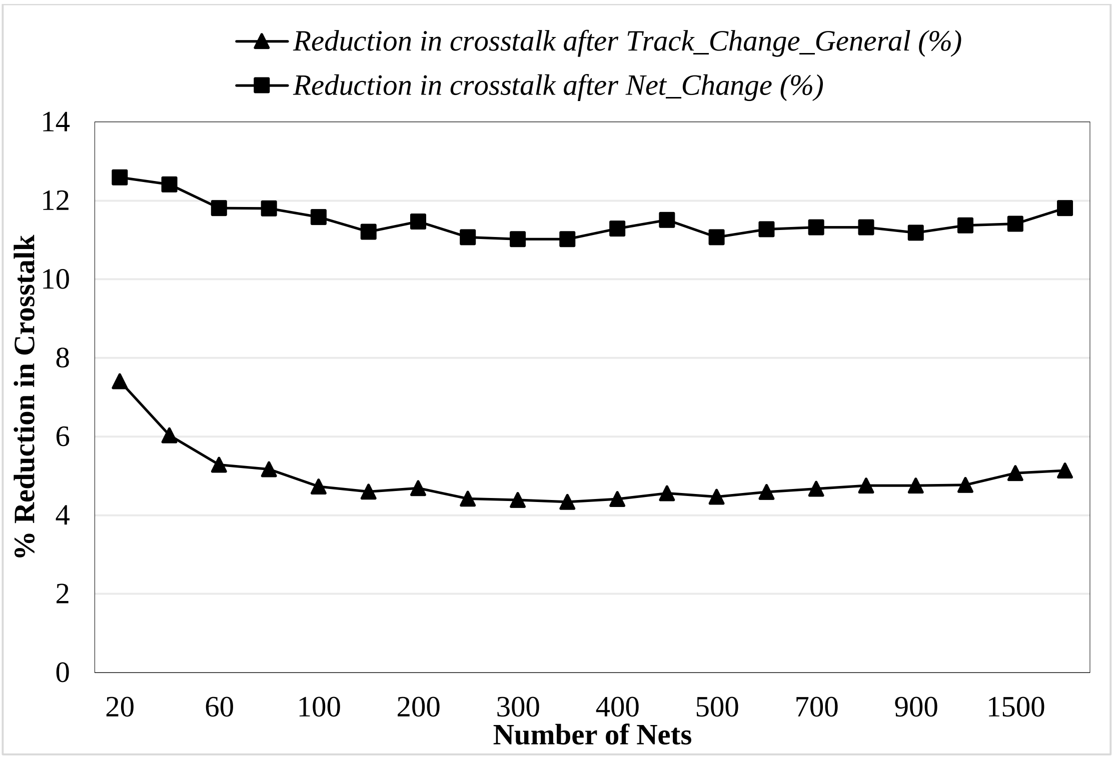


Figure 6.20: Percentage reduction in crosstalk (of routing solutions) versus number of nets after successive execution of each of the algorithms, *Track_Change_General* (with the concept of *Track_Change_Simple*) and *Net_Change* over the initial amount of crosstalk of routing solutions computed after algorithm *Track_Assignment _Heuristic* (*TAH*) for general instances in two-layer no-dogleg channel routing.

Thus, the first version of the algorithm (i.e. *Track_Change_General_Version _I*) performs better over its second version (i.e. *Track_Change_General_Version_II*), though after execution of algorithm *Net_Change* for all routing solutions obtained after each of the versions of algorithm *Track_Change_General*, the percentage reduction of crosstalk is roughly the same for a huge number of randomly generated channel instances, as the number of nets in a channel increases. To be clearer, as we may observe in the tables, all these algorithms have been implemented in a sequence, over the initial amount of crosstalk measured in the optimal or near-optimal routing solutions computed after execution of algorithm *Track_Assignment_Heuristic* (*TAH*) [49, 51]. The amount of average crosstalk (after execution of respective algorithm) is rounded off to its nearest integer value (for a given number of nets), and the percentage reduction in crosstalk is also calculated up to two decimal places. All these results have been included in Tables 6.2 and 6.3 as the performance of all the crosstalk minimization algorithms developed in this thesis and executed one after the other.

Table 6.3: The performance of crosstalk minimization algorithms *Track_Change _General_Version_II* (using *Greedy* approach) and *Net_Change* after their successive execution for computing reduced crosstalk routing solutions for general channel instances in two-layer no-dogleg (channel) routing.

Number of nets	**Initial crosstalk after *TAH***	**Crosstalk after algorithm *Track_Change_ General* (*Greedy*)**	**Reduction in crosstalk after *Track_Change _General* (%)**	**Crosstalk after algorithm *Net_Change***	**Reduction in crosstalk after *Net_ Change* (%)**
20	135	127	5.93	120	11.11
40	564	540	4.26	499	11.52
60	1363	1312	3.74	1214	10.93
80	2475	2386	3.60	2198	11.19
100	3997	3875	3.05	3568	10.73
150	9042	8787	2.82	8100	10.42
200	16260	15851	2.52	14516	10.73
250	25293	24648	2.55	22658	10.42
300	36612	35760	2.33	32824	10.35
350	49708	48596	2.24	44546	10.38
400	65032	63666	2.10	58105	10.65
450	82857	81039	2.19	73881	10.83
500	102922	100788	2.07	92086	10.53
600	148035	144816	2.17	132193	10.70
700	201940	197677	2.11	180150	10.79
800	265574	259729	2.20	236978	10.77
900	336455	329553	2.05	300532	10.68
1000	412991	404210	2.13	368171	10.85
1500	938379	918524	2.12	836159	10.89
2000	1665471	1624401	2.47	1471997	11.62

The deviation in crosstalk minimization (as shown in Tables 6.2 and 6.3) is graphically depicted in Figures 6.20 and 6.21, where the percentage reduction in crosstalk in allied routing solutions versus number of nets is included after each of the algorithms *Track_Change_General* (for both of its versions I and II, respectively) and *Net_Change*, over the initial two-layer channel routing solutions computed after *Track_Assignment_Heuristic* (*TAH*). Now we include some of the selected hardcopy routing solutions that we obtained after execution of each of the algorithms *TAH*, *Track_Change_General*, and *Net_Change*. These routing solutions are shown in Figures 6.22 through 6.32 for channels containing nets 20 through 250. In each of these figures, (a) displays the routing solution obtained after *TAH*, (b) depicts the significantly reduced crosstalk routing solution after execution of *Track_Change _General*, and (c) includes the mostly reduced crosstalk routing solution obtained after execution of algorithm *Net_Change*.

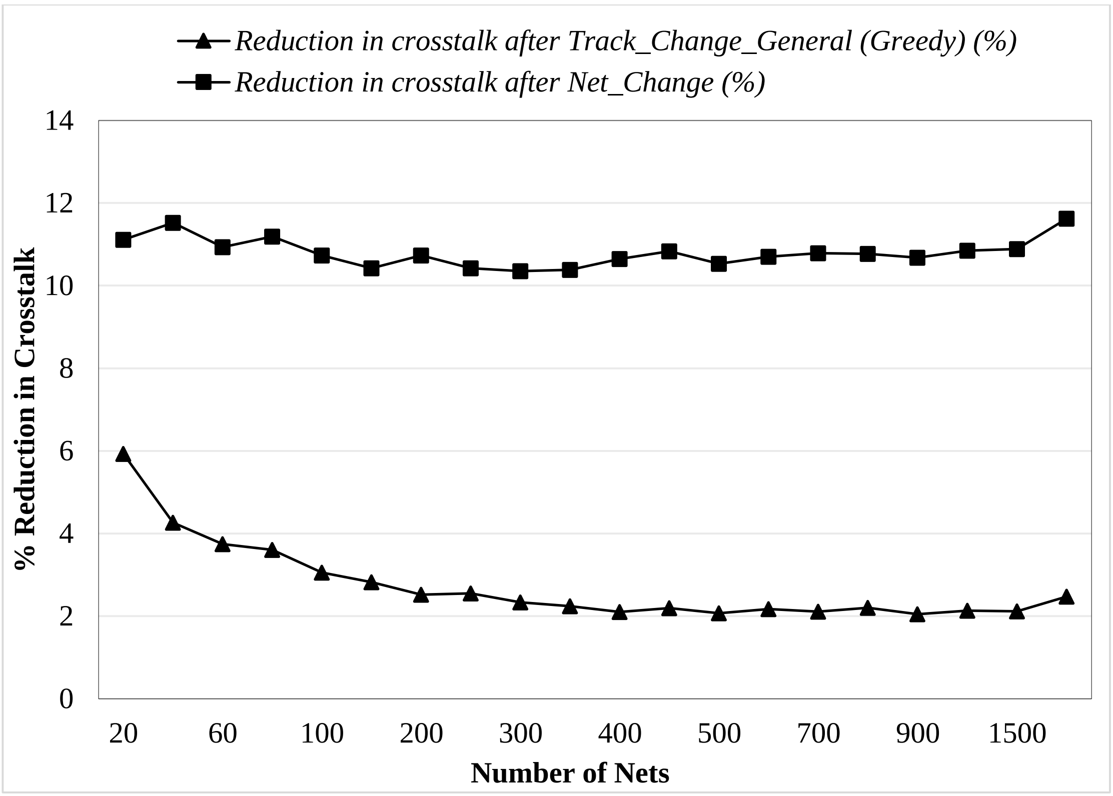
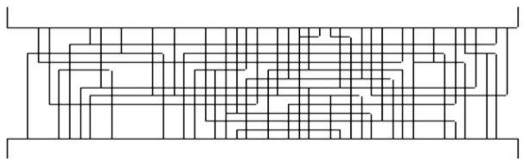
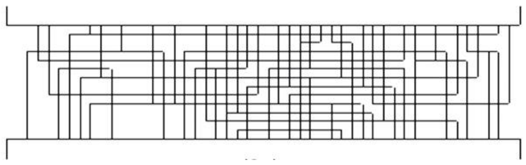


Figure 6.21: Percentage reduction in crosstalk (of routing solutions) versus the number of nets after successive execution of each of the algorithms, *Track_Change _General* (using *Greedy* approach) and *Net_Change* over the initial amount of crosstalk of routing solutions computed after algorithm *Track_Assignment_Heuristic* (*TAH*) for general instances in two-layer no-dogleg channel routing.


(a)


(b)

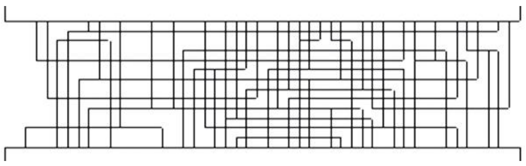
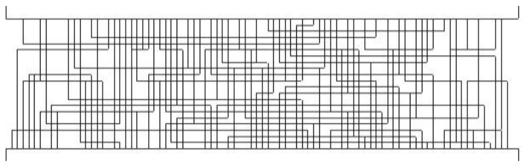
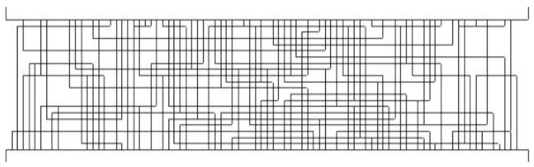

(c)

Figure 6.22: (a) The initial crosstalk of a general channel instance comprising 20 nets (track number 12 and channel length 48) is 228 units after execution of *Track_Assignment_Heuristic*. **(b)** Crosstalk after execution of algorithm *Track_Change_General* is 181 units. **(c)** Crosstalk after execution of algorithm *Net_Change* is 165 units only.


(a)


(b)

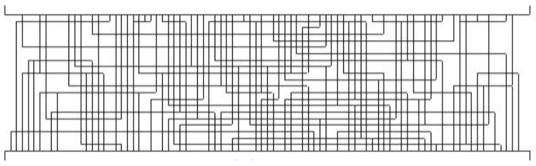
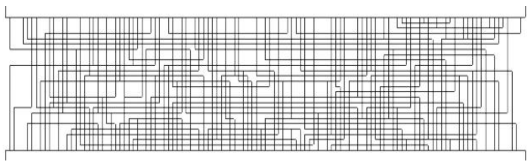
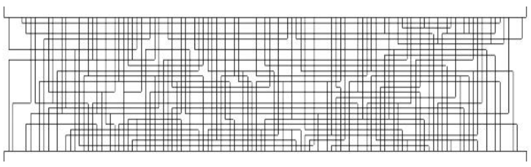

(c)

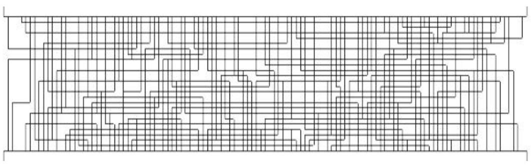
Figure 6.23: (a) The initial crosstalk of a general channel instance comprising 40 nets (track number 20 and channel length 89) is 565 units after execution of *Track_Assignment_Heuristic*. **(b)** Crosstalk after execution of algorithm *Track_Change_General* is 452 units. **(c)** Crosstalk after execution of algorithm *Net_Change* is 437 units only.



(a)

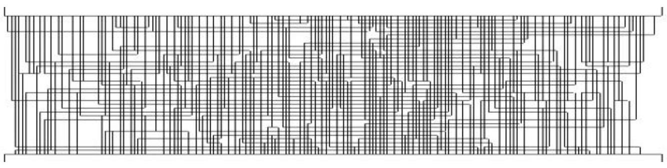


(b)

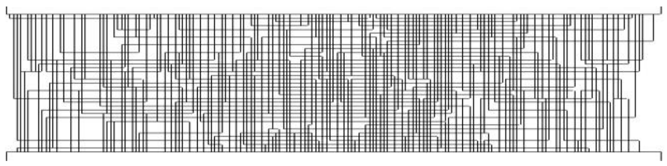


(c)

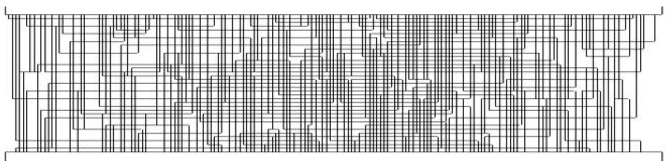
Figure 6.24: (a) The initial crosstalk of a general channel instance comprising 60 nets (track number 24 and channel length 117) is 1338 units after execution of *Track_Assignment_Heuristic*. **(b)** Crosstalk after execution of algorithm *Track_Change_General* is 1077 units. **(c)** Crosstalk after execution of algorithm *Net_Change* is 1059 units only.



(a)

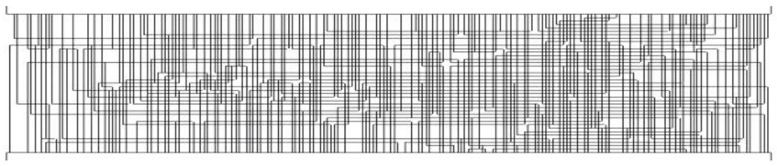


(b)

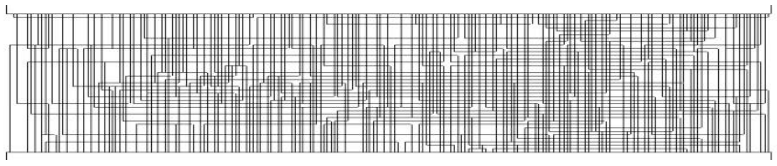


(c)

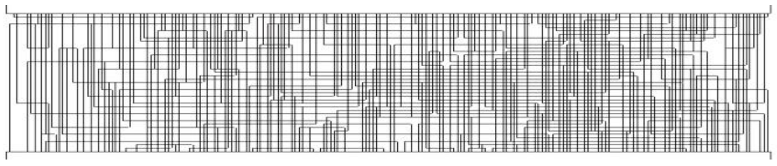
Figure 6.25: (a) The initial crosstalk of a general channel instance comprising 80 nets (track number 35 and channel length 181) is 2676 units after execution of *Track_Assignment_Heuristic*. **(b)** Crosstalk after execution of algorithm *Track_Change_General* is 2404 units. **(c)** Crosstalk after execution of algorithm *Net_Change* is 2253 units only.



(a)

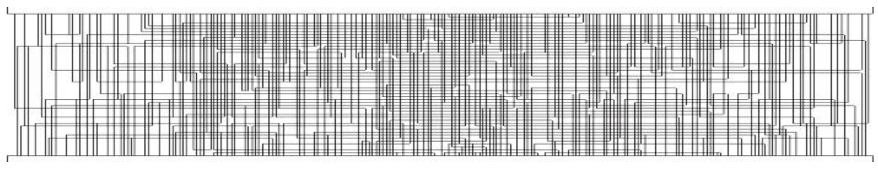


(b)

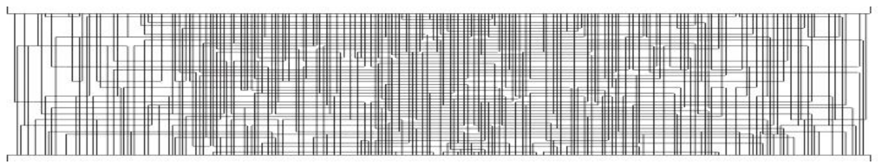


(c)

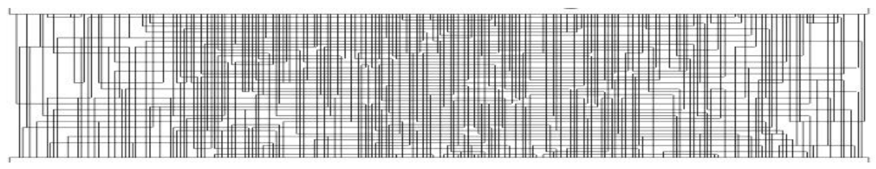
Figure 6.26: (a) The initial crosstalk of a general channel instance comprising 100 nets (track number 39 and channel length 216) is 3375 units after execution of *Track_Assignment_Heuristic*. **(b)** Crosstalk after execution of algorithm *Track_Change_General* is 2890 units. **(c)** Crosstalk after execution of algorithm *Net_Change* is 2488 units only.



(a)

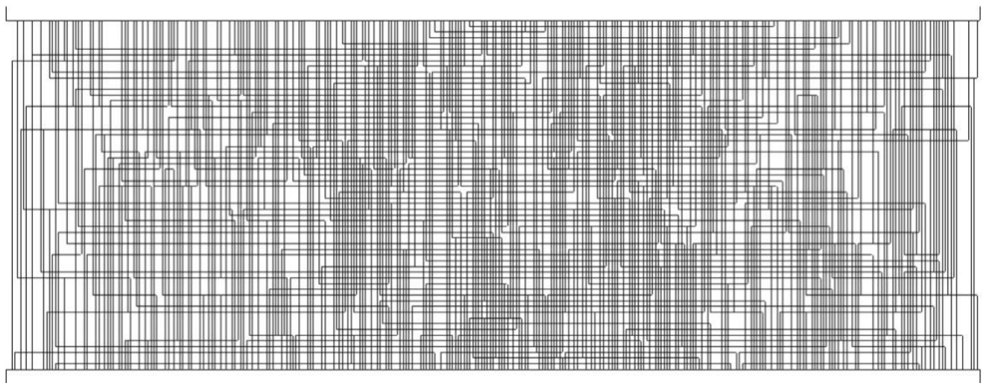


(b)

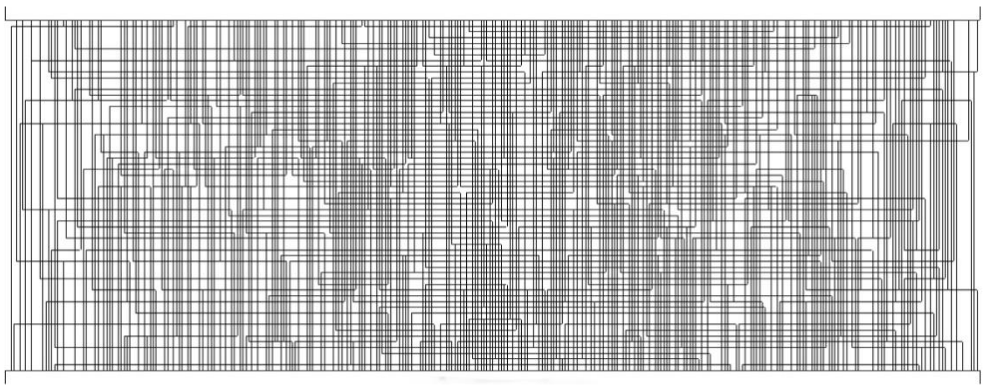


(c)

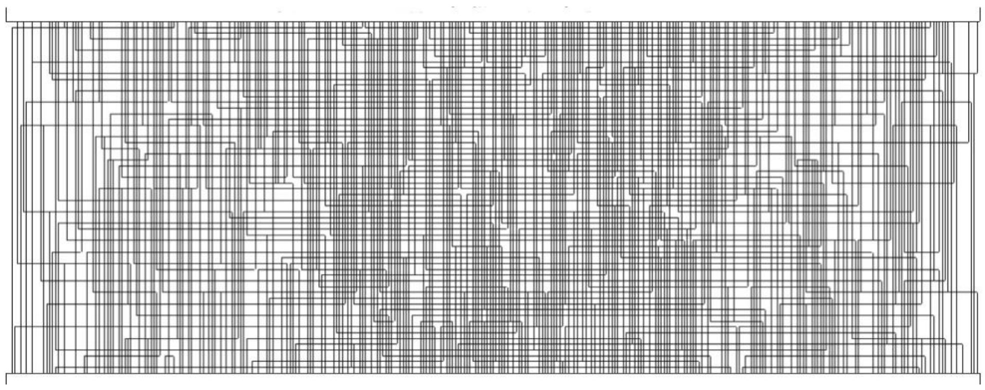
Figure 6.27: (a) The initial crosstalk of a general channel instance comprising 120 nets (track number 47 and channel length 250) is 4832 units after execution of *Track_Assignment_Heuristic*. **(b)** Crosstalk after execution of algorithm *Track_Change_General* is 4269 units. **(c)** Crosstalk after execution of algorithm *Net_Change* is 3942 units only.



(a)

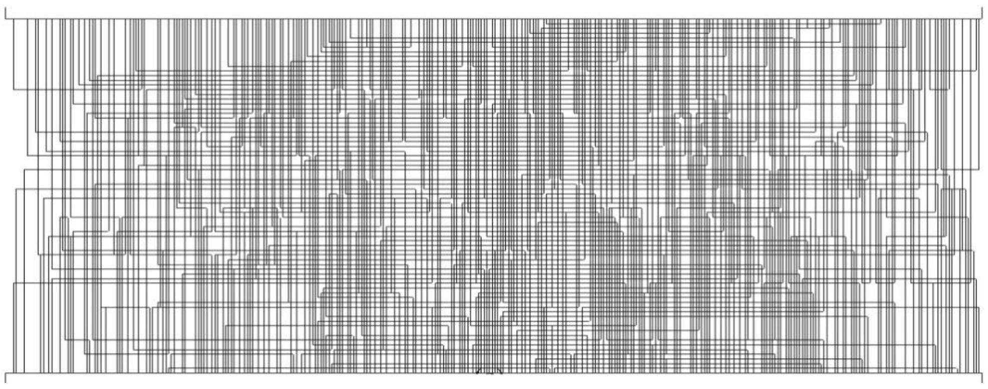


(b)

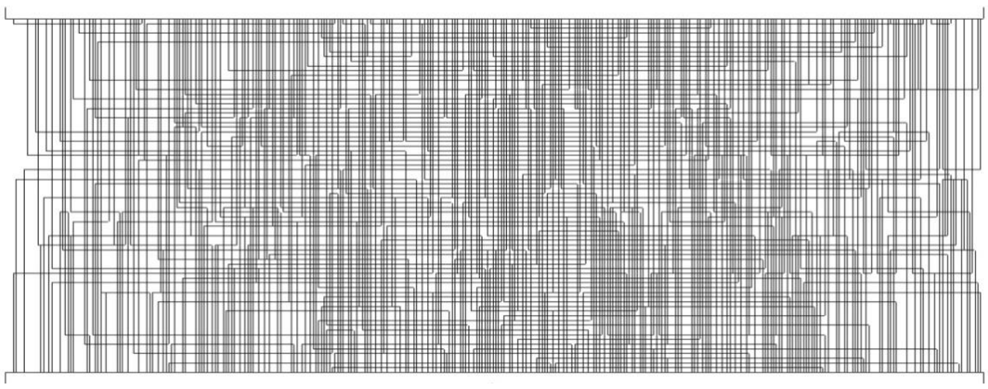


(c)

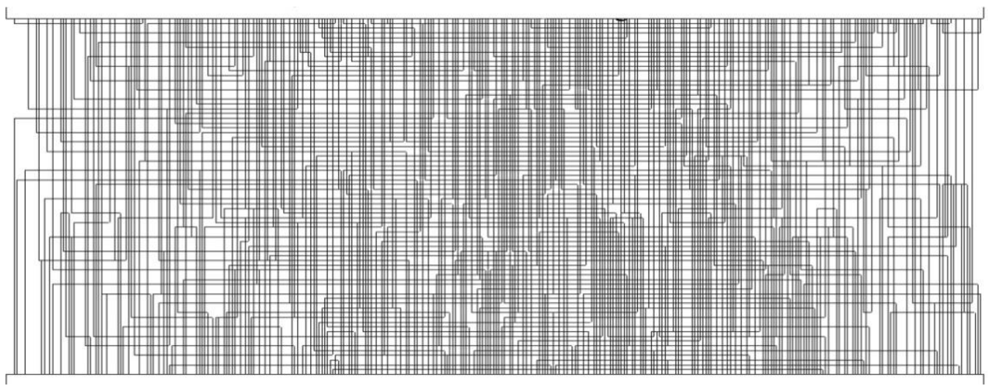
Figure 6.28: (a) The initial crosstalk of a general channel instance comprising 150 nets (track number 60 and channel length 335) is 9314 units after execution of *Track_Assignment_Heuristic*. **(b)** Crosstalk after execution of algorithm *Track_Change_General* is 8490 units. **(c)** Crosstalk after execution of algorithm *Net_Change* is 7977 units only.



(a)

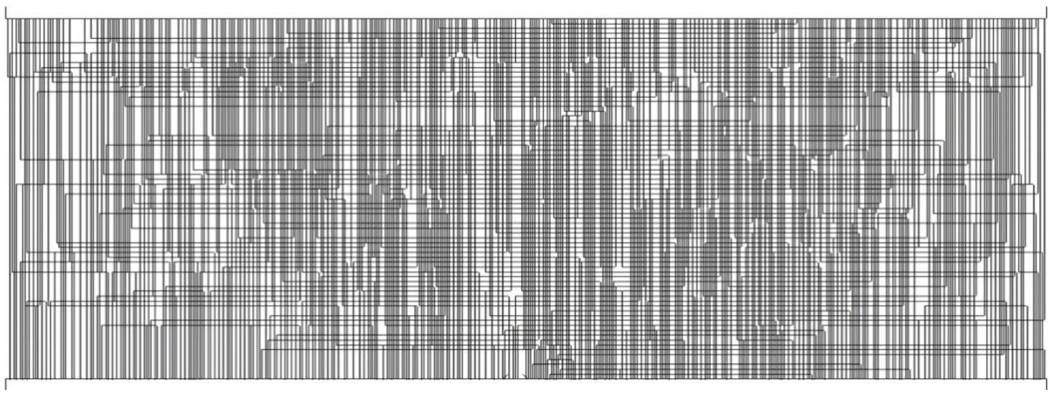


(b)

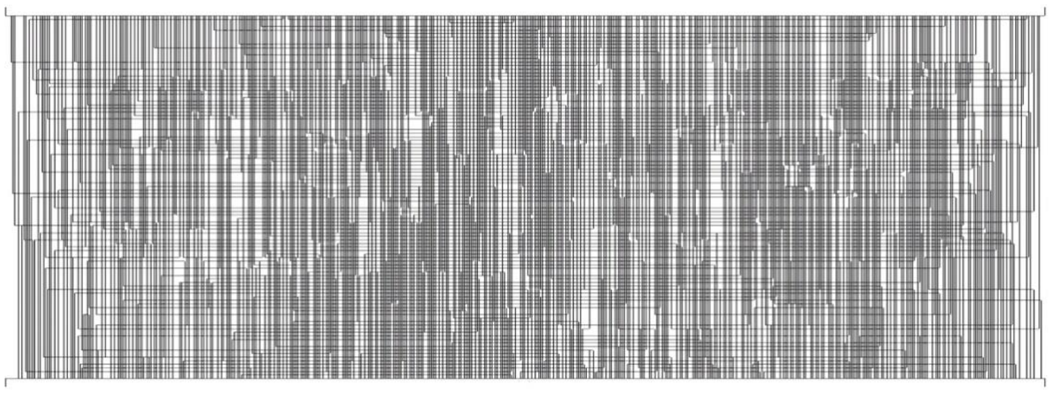


(c)

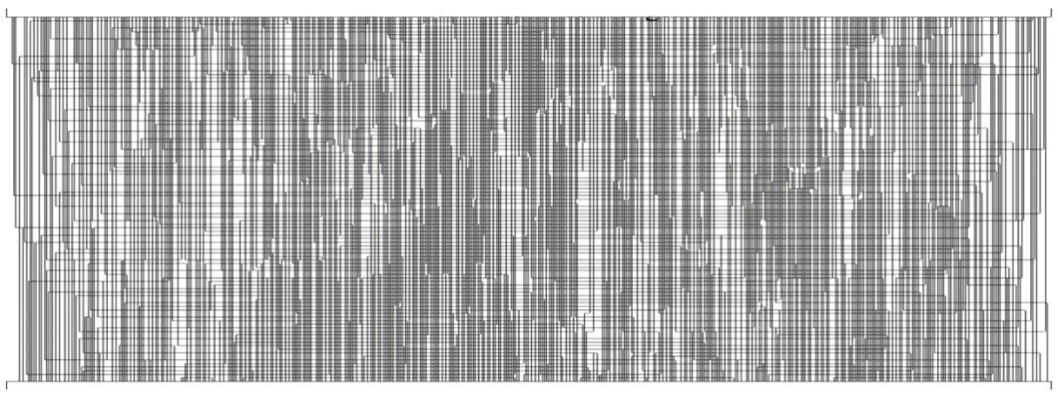
Figure 6.29: (a) The initial crosstalk of a general channel instance comprising 180 nets (track number 74 and channel length 358) is 12077 units after execution of *Track_Assignment_Heuristic*. **(b)** Crosstalk after execution of algorithm *Track_Change_General* is 10974 units. **(c)** Crosstalk after execution of algorithm *Net_Change* is 10116 units only.



(a)

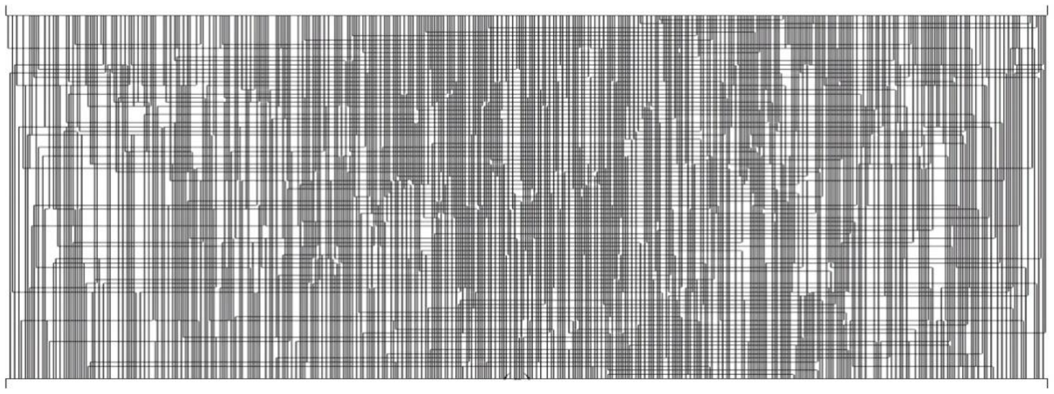


(b)

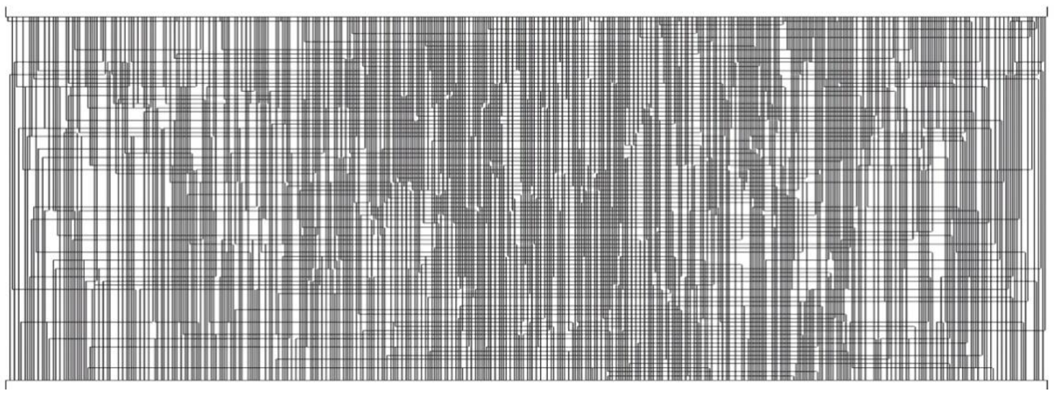


(c)

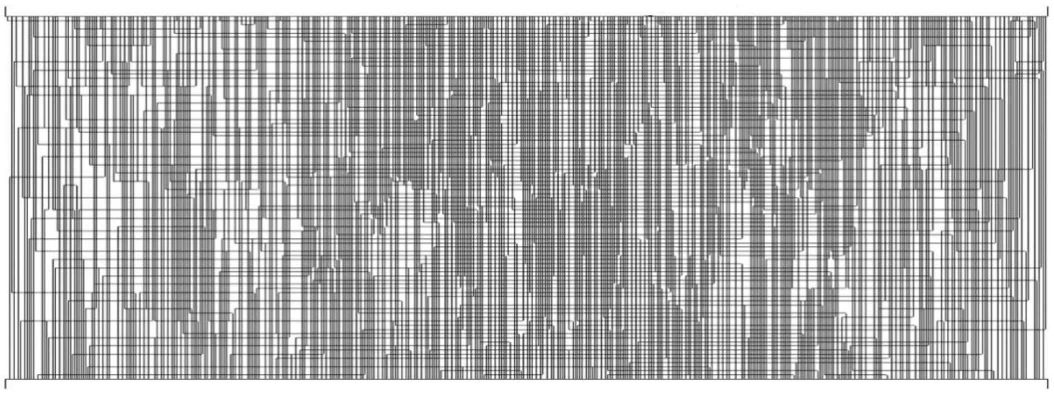
Figure 6.30: (a) The initial crosstalk of a general channel instance comprising 200 nets (track number 73 and channel length 487) is 17769 units after execution of *Track_Assignment_Heuristic*. **(b)** Crosstalk after execution of algorithm *Track_Change_General* is 17116 units. **(c)** Crosstalk after execution of algorithm *Net_Change* is 14750 units only.



(a)

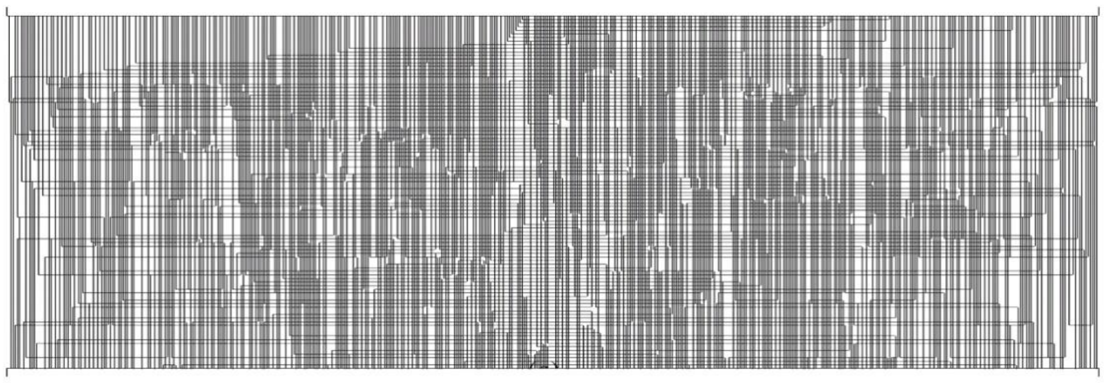


(b)

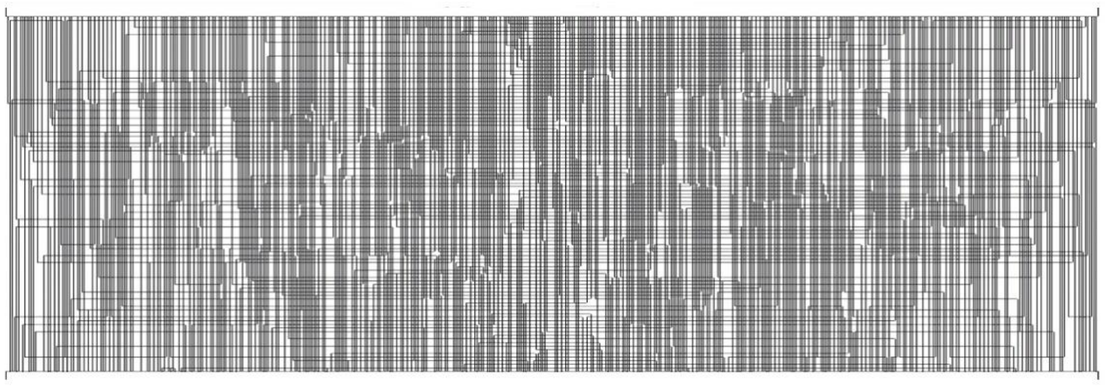


(c)

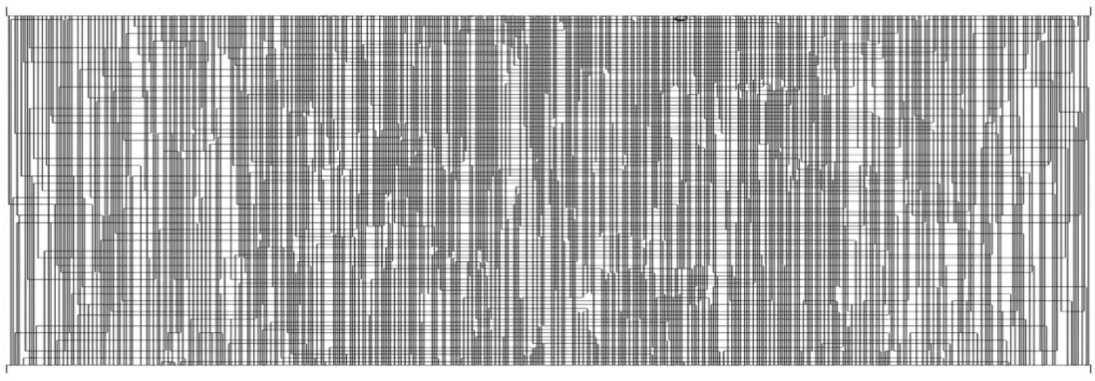
Figure 6.31: (a) The initial crosstalk of a general channel instance comprising 220 nets (track number 87 and channel length 480) is 18897 units after execution of *Track_Assignment_Heuristic*. **(b)** Crosstalk after execution of algorithm *Track_Change_General* is 17141 units. **(c)** Crosstalk after execution of algorithm *Net_Change* is 14933 units only.



(a)



(b)



(c)

Figure 6.32: (a) The initial crosstalk of a general channel instance comprising 250 nets (track number 97 and channel length 516) is 21377 units after execution of *Track_Assignment_Heuristic*. **(b)** Crosstalk after execution of algorithm *Track_Change_General* is 18205 units. **(c)** Crosstalk after execution of algorithm *Net_Change* is 16292 units only.

6.4 Summary

In this chapter, we have made some experimentation in two phases; in the first phase, our objective is to generate desired random channel instances of both types, simple and general, and in the second phase, we have shown hardcopy routing solutions for some of the selected instances, where for each channel three routing solutions are there with their associated amount of crosstalk. Experimental results have been included in tables; some are in respective chapters, either Chapter 4 or Chapter 5, and the remaining are in this chapter. Crosstalk is measured after computation of initial routing solutions, and then subsequently for the solutions obtained after execution of algorithms *Track_Change* and *Net_Change*.

Parenthetically, in this thesis, we have randomly generated as many as 4800 each of simple channel specifications and general channel specifications, ranging the number of nets 10 through 15000 in 24 sets of channels, where each set comprises 200 instances. Out of all these instances, in this chapter, we have depicted merely 15 smaller sample simple channel instances (only 0.3125% of total generated instances of this type) and no more than 19 smaller sample general channel instances (only 0.3958% of total generated instances of this kind).

Also, in this chapter, we have included minimized crosstalk channel routing solutions for some initially computed two-layer channel routing solutions. For simple channel instances, we have executed *MCC1* [49, 52, 53] to obtain the initial density routing solutions, and for general channel instances, we have executed *TAH* [49, 51] to acquire the initial routing solutions of minimum possible area. After obtaining each such initial channel routing solution, we have carried out respective algorithms *Track_Change* and *Net_Change* one after the other; hardcopy routing solutions of just 18 simple channel instances (only 0.2045% of total generated instances of this type) and no more than 11 general channel instances (only 0.2750% of total generated instances of this kind) have been included in this chapter. Computed results are included in tables, for a large number of randomly generated channel instances. Graphs are also drawn for showing a reduction in crosstalk after execution of associated algorithm for minimizing crosstalk; these show the deviation in percentage reduction in crosstalk as the number of nets increases.

Chapter: 7

Parallel Algorithms for Computing Two-Layer Reduced Crosstalk Channel Routing

7.1 Overview

It has already been mentioned several times in this thesis that based on the advancement of fabrication technology, devices and interconnection wire segments are placed as close as possible, and circuits operate at higher frequencies. These result in crosstalk between interconnecting wire segments. As the work on routing channels with reduced crosstalk is a very important area of current research [19, 21, 25, 27, 30, 39, 47, 49, 75, 87, 92, 93], we have already studied extensively the crosstalk minimization in two-layer channel routing for VLSI circuit synthesis. In Chapter 3, we have proved a host of crosstalk minimization problems NP-hard. Subsequently, sequential crosstalk minimization algorithms have been developed in Chapter 4 both for the simple as well as the general channel instances.

Since the problem of minimizing crosstalk is NP-complete in channel routing, heuristics have been developed for reducing crosstalk, and the heuristics designed are essentially sequential in nature. In this chapter, we study the problem for obtaining efficient parallel algorithms. We present two such parallel heuristic algorithms for computing reduced crosstalk routing solutions.

In Chapter 4, we have developed several sequential algorithms for reducing crosstalk in two-layer channel routing. The algorithms devised, in this chapter, too start from a given two-layer feasible routing solution of a channel whose crosstalk is assumed as the initial amount of crosstalk of the said channel. Our proposed heuristics are much better in computational complexity than their sequential counterparts invented in Chapter 4.

7.2 A Revisit to the Channel Routing Problem

In VLSI physical design it is required to realize a specified interconnection among different modules using minimum possible area. This is known as the routing problem. One of the essential types of routing strategies is *channel routing* [49, 81, 96]. A *net* is a set of terminals that need to be electrically connected together, and the terminals of the same net are assigned the same number. Different numbers (as diverse symbols) signify different nets that must not ne shorted; terminals not to be connected are assigned the number zero.

Like earlier chapters, in this chapter too we consider the *reserved two-layer Manhattan routing model*, where one layer is reserved for horizontal wire segments and the other layer is reserved for vertical wire segments. The connection between a horizontal and a vertical wire segment of the same net in adjacent layers is achieved by means of a *via* (a contact along the third dimension).

In this chapter, we represent horizontal constraints using the complement of the horizontal constraint graph (HCG). We call the complement of the HCG, $HC = (V, E)$, the *horizontal non-constraint graph* (*HNCG*) and denote it by $HNC = (V,E')$, where V is the set of vertices corresponding to the intervals, and $E' = \{\{v_i, v_j\} \mid \{v_i, v_j\} \notin E\}$. The notation of the HNCG was introduced in [49, 51, 52, 53, 57] to represent horizontal constraints. Note that a clique of the HNCG corresponds to a set of non-overlapping intervals that may safely be assigned to the same track in a routing solution.

We may recall that the *channel routing problem* (*CRP*) is the problem of assigning the horizontal wire segments of a given set of nets to tracks obeying the constraints present in a channel so that the number of tracks required (and hence the channel area) is minimized. We say that a routing solution is *feasible* if all constraints are satisfied and the nets can be assigned to the channel without any conflict.

Like most of the earlier chapters, in this chapter too, we consider the crosstalk minimization problem as performance driven channel routing. As fabrication technology advances, devices and interconnection wires are being placed in closer proximity and circuits are being operated at higher frequencies. This results in crosstalk between wire segments. Crosstalk between wire segments is proportional to the coupling capacitance, which is, in turn, proportional to the coupling length (the total length of the overlap between wires). Crosstalk is also proportional to the frequency of operation and inversely proportional to the separation between wires. Therefore, it is important that these factors be considered in the design of channel routing algorithms. The aim should be to avoid long overlapping wire segments and/or the wire segments that lie close to each other on the same layer [25, 27].

It is desirable to design channel routing algorithms that consider the factor of minimizing crosstalk. The main objective in performance driven routing is to reduce signal delays due to crosstalk. Note that the crosstalk minimization problem in the reserved two-layer Manhattan routing model is NP-hard, even for the channels without any vertical constraints; all these results have been established in Chapter 3 of this thesis. Since minimizing crosstalk is NP-hard, polynomial time heuristics have been devised for reducing crosstalk that are included in Chapter 4. All these ideas prior to this chapter, which are introduced as heuristics, are essentially sequential algorithms. In this chapter, we have developed two fast parallel heuristics to compute reduced crosstalk routing solutions, in the reserved two-layer Manhattan routing model, for simple channel instances that are free from any vertical constraint. Our proposed algorithms have been developed based on the same notion that has been utilized while devising the algorithms presented in Chapter 4. However, the algorithms proposed in this chapter are much better from the point of view of computational complexity than the sequential ones.

We know that parallel processing is an efficient style of information processing that emphasizes concurrent execution of independent events in the computing process. The parallel events may appear in the resources at the same time instance, and pipelined events may occur in overlapped time spans. These concurrent events are attainable in a computer system at various processing levels.

Parallel processing and distributed processing are closely related. In some cases, we use certain distributed technique to achieve parallelism. As data communication technology advances, the distance between parallel and distributed processing become small. So in recent time, we may see distributed processing as a form of parallel processing in a special environment [3, 37, 68]. We know that most of the problems belonging to VLSI physical design process are typically NP-hard [49, 81], and subsequent heuristics developed to execute these problems take significant amount of time sequentially. Thus, developing parallel heuristics might be a novel way out to resolve the problems. In this chapter, we consider the problem of crosstalk minimization in two-layer channel routing for the instances without any vertical constraint and develop parallel heuristics to resolve them. To design parallel algorithms for computing reduced crosstalk routing solutions, we consider the sequential algorithms for simple channel instances developed in Chapter 4 and make them parallel. Analytical results in terms of computational complexity of the algorithms developed in this chapter are excellent.

In the following section, we discuss the crosstalk minimization problem in the context of area minimization problem in channel routing. Subsequently, we propose two parallel heuristics for reducing crosstalk and analyze their complexity issues.

7.3 Area and Crosstalk Minimization in Channel Routing

As the channel routing problem (CRP) of area minimization is an NP-complete problem [41, 49, 80, 86], several (polynomial time) heuristics have been proposed for solving the problem [10, 12, 15, 33, 49, 51, 54, 71, 73, 80, 96]. We know that the CRP of area minimization is polynomial time computable if the instances are free from any vertical constraints and we are interested only in resolving horizontal constraints in the two-layer VH routing model [32, 49, 52, 53].

Since the problem of minimizing area for the instances of routing channels without any vertical constraint is polynomial time solvable (using exactly d_{max} tracks), such instances are defined as *simple channel instances*. Hashimoto and Stevens proposed a scheme for solving this problem [32], and according to Schaper, it can be implemented in $O(n (\log n + d_{max}))$ time, where d_{max} is the channel density and n is the number of nets belonging to the channel [80]. Later on, Pal *et al.* developed and analyzed two different algorithms *MCC1* and *MCC2* [49, 52, 53], based on the Hashimoto and Stevens's scheme. The first algorithm *MCC1* uses a graph theoretic approach and runs in $O(n + e)$ time, where n is the number of nets and e is the size of the HNCG of the given simple channel. The second algorithm *MCC2* is achieved using a balanced binary search tree data structure that runs in time $O(n \log n)$, where n is the number of nets. The details of the algorithms are available in [49]. Though a routing solution of only d_{max} tracks is guaranteed for a simple channel specification in polynomial time in the stated routing model, it may not be a *good* routing solution from the resulting crosstalk point of view.

We have already observed in Figure 4.1, the presence of crosstalk between nets (or intervals) assigned to different tracks in a two-layer channel without any vertical constraint. Note that if two intervals do not overlap, there is no horizontal constraint between the nets. That is, if there is a horizontal constraint between a pair of nets, there is a possibility of having *accountable* crosstalk between them if the nets are assigned to adjacent tracks. We compute crosstalk in terms of the number of units a pair of nets overlaps on adjacent tracks in a feasible routing solution, as we defined and measured it earlier too.

In this context, we like to remember *VHP*, the crosstalk minimization problem in (two-terminal no-dogleg) two-layer VH channel routing (given an a priori partition of nets), posed in Chapter 3. Subsequently, we keep in mind *VHS*, the crosstalk minimization problem in (two-terminal no-dogleg) two-layer VH channel routing for simple instances of channel specifications that has also been posed in the same chapter.

Note that the problem of area minimization for a simple channel instance is polynomial time solvable, and there exist several algorithms to compute a routing solution for such an instance using exactly density number of tracks [32, 49, 52, 53]. However, the noticeable feature is that the crosstalk minimization problem in two-layer VH channel routing for such instances of channel specifications is NP-hard; see Chapter 3 of this thesis. Thus, we like to point out the following: The crosstalk minimization problem is not only important from its practical outlook of computing high performance routing solutions, it is equally motivating to observe the same as a combinatorial optimization problem.

In the next section, we present two parallel algorithms to compute reduced crosstalk routing solutions for existing routing solutions of minimum area for simple instances of CRP.

7.4 Parallel Algorithms for Minimizing Crosstalk

It has been proved in Chapter 3 that the crosstalk minimization problem in two-layer channel routing is NP-hard even for the simple instances of channel specifications. Observe that, for any feasible two-layer VH routing solution *S*, we can compute another routing solution *S** with the total amount of crosstalk equals to zero. Suppose we have a two-layer feasible routing solution *S* of *t* tracks. In computing *S**, we merely introduce t–1 blank tracks into the routing solution *S*, where between each pair of adjacent tracks in *S* a blank track is placed in. As a result we must not have any crosstalk in *S**, following the process of measuring crosstalk we have made and the geometry of the routing model we have assumed. Thus, *S** is obtained as a valid routing solution of nearly 2t tracks without any crosstalk in it.

Here the main thing we like to emphasize is the following. If we provide sufficient space between the wires assigned to adjacent tracks (and layers), the amount of crosstalk will eventually be reduced. However, we all know that the area minimization problem is the most important cost optimization problem in VLSI physical design. Therefore, we must not encourage in computing such a routing solution *S** that takes almost twice the area of *S*. So it is a trade-off between routing area and the resulting crosstalk in routing a channel. That is why, instead of computing *S**, we start with *S* of *t* tracks, compute another feasible routing solution *S′* of the same *t* tracks with reduced total crosstalk, as we did in devising sequential crosstalk minimization algorithms in Chapter 4. To do that, we consider the routing solutions *S*, that are computed using *MCC1* for the simple instances of channel specifications [49, 52, 53], as the area minimization problem of two-layer channel routing is polynomial time computable; however, the subsequent crosstalk minimization problem is NP-hard even for such instances of routing channels.

7.4.1 Algorithm 1: *Parallel Track Interchange*

The *Parallel Track Interchange* algorithm is naturally evolved from the theory of reducing crosstalk (see Figure 4.1). The algorithm starts with a *t*-track two-layer feasible routing solution *S* that is computed using *MCC1* [49, 52, 53] for a simple instance of the channel specification and computes another *t*-track two-layer feasible routing solution *S′*with a reduced total crosstalk. In the algorithm, we first compute in parallel the effective spans of intervals of all the tracks in *S*. The *effective span of intervals* of track *i* is obtained by adding the actual spans of intervals of all the nets assigned to track *i* in *S*. Then, we sort the tracks in descending order according to their effective spans of intervals.

In the proposed algorithm, our intention is to sandwich the track (comprising net(s)) with the minimum effective span of intervals into the tracks (comprising nets) with the maximum and the next to maximum (or the second maximum) effective spans of intervals. Then we sandwich the track (comprising net(s)) with the next to minimum (or the second minimum) effective span of intervals into the tracks (comprising nets) with the second and the third maximum effective spans of intervals, and so forth and so on. The flanked assignment of a track (comprising net(s)) with a less effective span of intervals by a pair of tracks (comprising nets) with more effective spans of intervals is absolutely motivated by the geometry of the channel and the initial routing solution given as input to execute the algorithm. In other words, in order to compute S', we reassign the tracks of intervals from the computed sorted sequence as the following. Suppose $\Pi = \{\Pi_1, \Pi_2, \Pi_3, \ldots, \Pi_{t-2}, \Pi_{t-1}, \Pi_t\}$ is the sorted sequence of tracks in descending order in their effective spans of intervals. Here our desired sequence of effective spans of intervals is $\Pi' = \{\Pi_1, \Pi_t, \Pi_2, \Pi_{t-1}, \Pi_3, \Pi_{t-2}, \ldots\}$ to assign the nets from top to bottom in t different tracks, and hence the resulting solution S' is obtained.

One more clarification in our *Parallel Track Interchange* algorithm is required when we have two or more tracks with the same effective span of intervals. In this case in sorting those tracks of the same effective span of intervals, we compute the total span of intervals of each such track. The *total span of intervals* of the nets assigned to a track in S is the separation of columns between the starting column of the first net and the terminating column of the last net, i.e. the span of the track used. Here we sort such tracks with the same effective span of intervals in ascending order based on their total spans of intervals in computing Π, as stated above. This is done in ascending order being motivated that the nets belonging to a track with more total span of intervals are more distributed over the track, and its reassignment to a track will eventually result in reducing long overlapping (i.e. crosstalk) between the nets in this track and the nets assigned to its adjacent track(s). If there are two or more tracks with the same total span of intervals, we sort them arbitrarily. Hence, Π is computed from the given routing solution S, Π' is computed from Π as stated above, and following the sequence of tracks in Π' we reassign the nets to tracks from top to bottom of the channel, and a routing solution S' with reduced total crosstalk is obtained. Each of these steps is computed in parallel. This completes the presentation of the parallel heuristic algorithm *Parallel Track Interchange*. We use *EREW PRAM* model for the implementation of these steps of the algorithm. Details of computational complexity are presented next.

7.4.1.1 Computational Complexity of Algorithm *Parallel Track Interchange*

Now we analyze the time complexity of the algorithm *Parallel Track Interchange*. In order to do that, we consider an *Exclusive Read Exclusive Write* (*EREW*) *Parallel Random Access Machine* (*PRAM*), where a control unit issues an instruction to be executed simultaneously by all processors on their respective data. In our algorithm we primarily perform three sorts of computation, as follows: (1) Trackwise computation of effective span of intervals (and total span of intervals, whenever required), (2) Trackwise sorting of nets based on their effective spans of intervals (and total spans of intervals, whenever required), and (3) Trackwise reassignment of nets to tracks in computing S'. Initially, we keep all the information (i.e. starting column position, terminating column position, span of the interval, etc.) related to each of the n nets belonging to a simple channel specification in the shared memory of the PRAM.

We assign one processor to each track in the given routing solution S of a channel. Let P_{ij} be the i-th processor assigned for the j-th track in the initial solution, S. An *Exclusive Read* (*ER*) instruction is executed by all processors, where processors gain access to share memory for the purpose of reading the horizontal span(s) of the net(s) associated to tracks in a one-to-one fashion. Thus, when this instruction is executed, p processors simultaneously read the contents of p distinct memory locations such that each of the p processors involved reads from exactly one memory location and each of the p memory locations involved is read by exactly one processor.

For a given j, we compute the sum of all values read by P_{ij}. Observe that this sum of values read by P_{ij}, for all i, can be computed in parallel. Furthermore, each sum can be computed in parallel for all j. As sum of m numbers can be computed in O(log m) time using O(m) processors on an EREW PRAM machine, the Step (1) of computing the effective span of nets/intervals (and total span of intervals, whenever required) in all tracks can be computed in O(log n) time using O(n) processors on an EREW PRAM, where n is the number of nets belonging to the channel; however, the best value of n (the number of processors involved) is d_{max} for a density routing solution S of a simple channel instance. This completes the Step (1) of the parallel algorithm.

Step (2) of the parallel algorithm can be thought of sorting d_{max} elements [6]; i-th element is the effective span of the net(s)/interval(s) in the i-th track in S. Now, n elements can be sorted in O(log n) time using O(n) processors on a *Concurrent Read Exclusive Write* (*CREW*) *Parallel Random Access Machine* (*PRAM*) machine [38]. Note that we need to sort d_{max} elements here, and $d_{max} \leq n$. Therefore, using O(n)

processors, Step (2) can be implemented in O(log n) time on a CREW PRAM machine. Since a *concurrent read* (*CR*) instruction by n processors can be simulated on an EREW PRAM machine in O(log n) time using the same number of processors, this step of the algorithm can be implemented in O($\log^2 n$) time using O(n) processors on an EREW PRAM.

Now we reassign the nets based on the sorted sequence obtained in the previous step, and compute S'. A more succinct formulation of this part of the algorithm is given next as algorithm *PRAM_Reassignment*.

Algorithm: *PRAM_Reassignment*

```
For all $i = 1$ to $d_{max}$ do in parallel
    If $i \leq \lceil d_{max}/2 \rceil$,
    then $i \leftarrow 2i - 1$
    Else $i \leftarrow 2(d_{max} - i + 1)$
    End if
End for
```

In algorithm *PRAM_Reassignment,* the statement

"***For*** all $i = 1$ to d_{max} **do in parallel**"

means that all processors P_i, $1 \leq i \leq d_{max}$, perform the step simultaneously. Obviously, this PRAM step takes O(1) time. Note that no concurrent access is required. Therefore, the algorithm can be implemented on an EREW PRAM only. Thus, the trackwise reassignment of nets takes constant time using an EREW PRAM machine using d_{max} processors.

Now we compute the overall computational complexity of the algorithm *Parallel Track Interchange*. All steps can be performed in parallel in O(log n) time using O(n) processors on a CREW PRAM machine. If an EREW PRAM machine is used, all steps can be performed in parallel in O($\log^2 n$) time using O(n) processors. We summarize the correctness and the computational complexity of the algorithm *Parallel Track Interchange* in the following theorem.

Theorem 7.1: *The algorithm Parallel Track Interchange computes a two-layer VH routing solution with reduced total crosstalk on a PRAM model for simple channel* *instances. The time and processor complexity of this algorithm are O(log n) and O(n), respectively, if a CREW PRAM machine is used. Instead, if an EREW PRAM machine is used, the time and processor complexities become O($\log^2$ n) and O(n), respectively.*

7.4.2 Algorithm 2: *Parallel Net Change*

The heuristic, *Parallel Track Interchange*, presented in the previous section, is simple but efficient enough to reduce substantial amount of crosstalk over a given routing solution where the nets are free from vertical constraints, merely by reassigning the nets trackwise. An attempt has been made to reduce the crosstalk further using heuristic *Parallel Net Change*.

Observe that, if two nets are (i) horizontally constrained to each other, (ii) the interchange does not introduce any horizontal constraint violation due to overlapping with some other nets, and (iii) the resulting crosstalk after swapping the nets is reduced, then these two nets can be interchanged to reduce crosstalk further. Incidentally, we do not talk about vertical constraint violation as the instances under consideration are only simple channel specifications. Nevertheless, this is not at all a clear-cut task, since we do not know a priori the sequence of pairs of nets to be exchanged so that a maximum amount of crosstalk is reduced. Furthermore, a particular net can be swapped O(d_{max}) times (where d_{max} is O(n)) over the tracks without giving any remarkable gain in the overall crosstalk, and that might make the problem of minimizing crosstalk drastically cost expensive. As a consequence, in this algorithm, without swapping a pair of nets we do exchange a net with a blank space in some other track if this substitution reduces the overall crosstalk. For some net x, if several such swapping is possible, we perform the exchange that results in the maximum reduction in crosstalk.

We define a net, in a given routing solution S of d_{max} tracks, that could be swapped with a blank space in some other track as an *interchangeable net*; otherwise, as a *non-interchangeable net*. Note that each non-interchangeable net p_i, $1 \le i \le n$, assigned to a track s, $1 \le s \le d_{max}$, in S, must also be assigned to track s in S'. However, an interchangeable net q_i, $1 \le i \le n$, assigned to a track t, $1 \le t \le d_{max}$, in S, may be assigned to a possibly different track t' in S'. It might so happen that an interchangeable net cannot be swapped though there is a blank space in some other track in *S*, as there is no gain in crosstalk out of that exchange.

Theoretically, there can be O(*n*) such interchangeable nets in *S* for a channel comprising *n* nets in total, though it is unlikely as the channel instances are simple and the solutions under consideration are all density routing solutions. Thus, as a matter of fact, only a few interchangeable nets are there in such an initial routing solution *S*, which is assumed as the preferred routing solution (with mostly reduced crosstalk) computed after algorithm *Parallel Track Interchange*. This claim could be verified based on the reduction in crosstalk after algorithm *Net_Change* over the amount of crosstalk obtained after algorithm *Track_Change_Simple*, included in Table 6.1 and visualized in Figure 6.1, which is just about 1% reduction or even less.

Furthermore, if among *n* nets in total, the channel density d_{max} is O(*n*), only then a bulk of the remaining nets may be interchangeable nets as they do not belong to density columns (in *S*). However, in such a situation, most of such nets are not required to be shifted if their adjacent tracks are blank (that do not render any crosstalk) in the given routing solution *S*. Thus, even for an interchangeable net, searching for a necessary span of blank space in some other track could often be a superfluous task. In accordance with the above, the steps of the algorithm *Parallel Net Change* state below how the heuristic works and searches for a blank space for an interchangeable net q_i in some other track in order to compute S'.

Algorithm: *Parallel Net Change*

Input: A density routing solution, *S*, of a simple channel instance.

Output: Another density routing solution, S', of *S* with reduced crosstalk.

```
Begin
Step 1: O(n) processors are employed to identify the interchangeable nets in S, where
        processor $P_i$ is deployed for net $n_i$, $1 \le i \le n$, all in parallel.
Step 2: For each interchangeable net $n_i$, $P_i$ computes the amount of crosstalk in S
        involving $n_i$; all processors do the same in parallel.
Step 3: If the amount of crosstalk for an interchangeable net $n_i$ in S is zero, then $n_i$ is
        kept in its own track in computing $S'$.
        Else
```

```
$P_i$ searches for a track (with necessary blank span) where $n_i$ could be reallocated in order to compute $S'$, all in parallel.
Step 3.1: If only one such a track, $t$, is found for only one interchangeable net $n_i$, where $n_i$ could be reallocated (means the crosstalk for $n_i$ in its new track is reduced in comparison to that in $S$), then $n_i$ is shifted to track $t$; otherwise, $n_i$ is kept in its earlier track in $S$, all in parallel.
    Else
    Step 3.1.1: If two or more such (blank) spaces in different tracks are found for only one interchangeable net $n_i$, where $n_i$ could be reallocated (means the crosstalk for $n_i$ in two or more of its new tracks is reduced in comparison to that in $S$), then $n_i$ is shifted to a new track where it is best fitted (means the reduction in crosstalk is maximum); otherwise, $n_i$ is kept in its earlier track in $S$, all in parallel.
        End if
    End if
Step 3.2: If two or more overlapped interchangeable nets are found assignable to just one new track $t$, where each of them could be reallocated in isolation, then the best such net is shifted to track $t$ (whose reallocation mostly reduces the crosstalk in computing $S'$); otherwise, all such nets are kept in their earlier tracks in $S$, all in parallel.
    Else
    Step 3.2.1: If $p \geq 2$ overlapped interchangeable nets are found assignable to a set of $q \geq p$ new adjacent tracks, where these could be reallocated, then we call Parallel Track Interchange, only on this set of interchangeable nets for their reallocation to the targeted set of blank tracks in order to compute $S'$. If such reallocation results more crosstalk, two tasks are performed as follows, all in parallel. (1) The alternate even numbered reallocated net(s) is (are) kept in the new track(s) and the alternate odd numbered reallocated net(s) is (are) revert back to its (their) earlier track(s) in $S$ (keeping these allocated tracks blank). (2) The alternate odd numbered reallocated net(s) is
```

```
                (are) kept in the new track(s) and the alternate even numbered
                reallocated net(s) is (are) revert back to its (their) earlier
                track(s) in $S$ (keeping these allocated tracks blank). Now,
                among the (three) routing solutions, $S$ and the two computed
                after (1) and (2) above, the one with the minimum crosstalk is
                regarded as the preferred minimum crosstalk routing solution
                $S'$.
                End if
            End if
End
```

It is straightforward to observe that each interchangeable net does not contain any density column. In other words, the zonal density [49, 51] of each interchangeable net is less than d_{max} in S, as there is at least one blank track spanned by the net where it could be swapped and S is a density (or d_{max}-track) routing solution. The *zonal density* of net i in S is the maximum density of the columns within the span/interval of net i.

Now, if the interchangeable nets are pairwise nonoverlapping, each of them could be moved to the blank places in the new tracks, in order to compute S', using an EREW PRAM machine. If there are two or more overlapped interchangeable nets (as has been included in Step 3.2) that could be shifted to the blank places in a new track, then we need to shift the net that reduces the crosstalk by maximum amount. For this reason, we use a priority CRCW PRAM model [3].

Here, we like to mention one more point in performing two additional tasks (1) and (2), as has been included in Step 3.2.1, where we observe that after reallocation of intervals the computed routing solution S' might contain more crosstalk in comparison to that in S. Then we like to revert back a subset of intervals/nets to their earlier tracks tentatively keeping an alternate/non-adjacent half of them to the newly allocated tracks. This certainly enhances the scope of reduction in crosstalk further as concentrated intervals (for their consecutive presence over the tracks in computing S') are now distributed over the (blank) tracks in alternation (in order to compute S'). However, if the resulting crosstalk in S' is found to be more than that in S, we accept S as the final solution with no reduction in crosstalk.

On the other hand, to satisfy the case $2 \le p \le q$, where p is the number of overlapped interchangeable nets (which is equal or less) and q is the new set of adjacent blank tracks (which is equal or more), the algorithm *Parallel Net Change* can successively be executed for a constant number of times, if there is a further improved routing solution in terms of minimizing crosstalk. Thus, we presume that the newly computed routing solution S' as S to start for computing even a probable better routing solution for subsequent interchangeable nets (or zones of columns) exposed in S'. If S' is superior than S in terms of minimizing crosstalk (i.e. the amount of crosstalk in S' is less than that in S), we accept it; otherwise, the algorithm terminates without any further iterations. We now analyze the computational complexity of the algorithm.

7.4.2.1 Computational Complexity of Algorithm *Parallel Net Change*

In the heuristic *Parallel Net Change*, we assign a processor to each of the (interchangeable) nets in S. Further, we assume that blank spaces in the present solution are stored in the global (shared) memory. In addition, for each processor P_i, $1 \le i \le n$, we have the interval information of blank spaces in all the tracks in S in the form of sequential search of the blank spaces available in other tracks in S. Note that the data structure contains at most $O(d_{max})$ information (of blank spaces) for each interchangeable net. Moreover, for each interchangeable net, the initial crosstalk due to a net is measured in Step 2 in constant time as all processors do it in parallel. However, the interchangeable net identification (in Step 1) and the track where it could possibly be reallocated (in Step 3) can be performed in time $O(d_{max})$, as S is a d_{max}-track routing solution.

The problem of reallocation of an interchangeable net to a blank space in other track becomes a bit tricky, when there are more interchangeable nets and less blank spaces. Here the question of selecting a subset of interchangeable nets arises that should get more priority for their transfer. This task is performed by executing a *Concurrent Write* (*CW*) to the memory corresponding to blank spaces. As we need the maximum reduction in crosstalk, we assume that an interchangeable net (or the corresponding processor) succeeds in writing its value. This means that we need a priority CRCW PRAM for implementation of our algorithm. Therefore, we conclude the following.

Theorem 7.2: *The algorithm Parallel Net Change computes a d_{max}-track two-layer VH routing solution with reduced total crosstalk on a priority CRCW PRAM from a d_{max}-track two-layer VH routing solution of a simple channel instance by reassigning nets to some other tracks of the given routing solution. The time and processor complexity of this algorithm are $O(d_{max})$ and $O(n)$, respectively.*

Corollary 7.1: *As the sequential algorithm for the problem presented in Section 4.3.2.3 takes $O(nd_{max})$ time, this Parallel Net Change algorithm is cost optimal.*

Corollary 7.2: *As a concurrent read/write instruction by n processors on a CRCW PRAM can be simulated on an EREW PRAM in O(log n) time, Parallel Net Change algorithm can be simulated on an EREW PRAM in $O(d_{max} \log d_{max})$ time using O(n) processors.*

7.5 Summary

In this chapter, we have developed two parallel heuristics for the problem of computing reduced crosstalk routing solutions in two-layer VH channel routing for the simple instances of channel specifications. The first algorithm *Parallel Track Interchange* runs in time O(log n) on a CREW PRAM using O(n) processors; whereas the second algorithm *Parallel Net Change* runs in O(d_{max}) time using O(n) processors using a priority CRCW PRAM. The second parallel algorithm is cost optimal. On an EREW PRAM, they can be implemented respectively in time O($\log^2 n$) and O(d_{max} log d_{max}) using the same number of processors. The sequential counterparts of these algorithms take time O(d_{max} log d_{max}), and O(nd_{max}), respectively. Both parallel algorithms are substantially faster than their sequential counterparts.

Here we like to point out a few possible extensions and open problems as mentioned below. (i) It may be investigated to design similar algorithms for the general instances of channel specifications with multi-terminal nets where both the horizontal as well as vertical constraints are present in two-layer channel routing. (ii) A generalized version of algorithm *Parallel Net Change* may produce better routing solutions in terms of reduction in crosstalk when two overlapping nets interchange their tracks. (iii) Researchers may also be interested in computing much reduced crosstalk routing solutions in the expense of negligibly more channel area. (iv) Instead of starting from a given routing solution researchers may also compute *good* routing solutions directly optimizing crosstalk and some other cost optimization factor(s) of CRP. (v) Minimized crosstalk routing solutions in the case of three- and multi-layer channel routing might draw the current interest of research. Also, doglegging may be introduced in all these cases.

Chapter: 8

Conclusion

8.1 Contribution of the Thesis

VLSI fabrication technologies now allow multiple layers of interconnection in integrated circuits. Even then, several works in two-layer channel routing are still due for their execution and experimentation for high performance challenges. Theoretical issues are also getting importance from the viewpoint of combinatorial optimization.

The thesis starts with an introductory chapter where the channel routing problem (CRP) along with its inherent constraints and different routing models have been described. The interesting and important matters related to CRP have also been elaborated in the same chapter. These include usual as well as several high performance issues involving CRP.

Chapter 2 is the one, which is committed to almost all allied concerns relating to electrical hazards, rather crosstalk, in several associated domains of research fields in VLSI including network, fabrication technology, communication (or signal transition), and so forth and so on. In the same chapter, before the literature survey, we have briefly discussed on the theories of NP-completeness and NP-hardness and mentioned a class of allied problems.

As have already been cited several times, the thesis primarily focuses on the crosstalk minimization problem in two-layer channel routing. Undoubtedly, it is an important problem whose fair consideration makes a routing solution satisfactory from its high performance stand. As a consequence, there were some theoretical apprehensions. This thesis has resolved most of them. Specifically, in Chapter 3 we have considered the issues on the hardness of crosstalk minimization in two-layer channel routing including the simplest of all crosstalk minimization problems. In this chapter, *we have proved that the crosstalk minimization problem in the reserved two-layer Manhattan routing model is NP-hard for the simple and the general instances of channel specifications with partitioning of nets so that the nets in a class of a given partition are assigned to the same track.*

In the same chapter, we have also investigated upon the simple as well as the general instances of channel specifications with only two-terminal nets, but without any imposed partition of (non-overlapping) nets to tracks. *This, being a more general case, is also NP-complete. Moreover, we have introduced the problem of minimizing bottleneck crosstalk in the reserved no-dogleg two-layer channel routing model*. *We have proved that the problem is NP-hard too*.

On completion of the proof of NP-completeness of the crosstalk minimization problem, one should look for the next best possible option. This is nothing but to look for an approximation algorithm to solve the problem. We have further proved in this chapter that this is not possible. *We have proved that if P ≠ NP, it is impossible to design an approximation algorithm for the (crosstalk minimization) problem even in no-dogleg two-layer channel routing*. These remain so even if doglegging is allowed.

The only option that remains with us to tackle the problem is to design efficient heuristics for the problem. In Chapter 4, *we have developed two efficient heuristics for simple instances of channel specifications*. *They have been found to produce optimal / near optimal routing solutions in most of the cases*. *Afterward, the heuristics have also been generalized to compute an optimal / near optimal crosstalk routing solution of a general instance of two-layer channel routing in a novel manner*. *The performance of our algorithms is encouraging enough for most of the existing benchmark channels*. *Moreover, the experimental results obtained on the execution of the heuristics show a lot of improvement over the existing routing solutions of reduced area*.

We know that the efficiency of a heuristic is well established when it is executed for a variety of a large number of randomly generated similar (channel) instances, and the final upshot is computed making an average on all of them. Incidentally, the simple channel instances are hardly available in the literature. Also, a very few general channel instances, commonly known as *benchmark channel specifications*, are available in the literature. For these reasons, *we have developed algorithms for generating random channel instances in Chapter 5, for the purpose of running the heuristics developed in Chapter 4*.

Simple channel instances that are (randomly) generated in Chapter 5 are all containing only two-terminal nets; simple channel instances with multi-terminal nets *can also be created. General channel instances that are also randomly generated in this chapter contain two- as well as multi-terminal nets*. Furthermore, the devised algorithms can create channel instances of any number of nets containing any number of terminals per net.

Chapter 6 of this thesis is based on experimental results of most of the implementations made in this thesis, principally for all the algorithms designed in Chapters 4 and 5. *The first pair of algorithms for reducing crosstalk devised in Chapter 4 has also been right away executed in the same chapter for a smaller number of randomly generated simple channel instances and also for a set of only 14 existing benchmark (general) channel specifications*. All these results have also been included there in Chapter 4. However, in Chapter 5, *a second pair of algorithms has been developed, and these have been formulated for generating random channel instances*. Precisely, in Chapter 5, *at once we generate two sets of channel specifications at random, simple as well as general, each containing 4800 instances in total, for 24 sets of a given number of nets ranging from 10 through 15000*.

On the other hand, in Chapter 6, we first randomly generate as many as 8800 simple channel instances containing a number of nets ranging from 10 to 1000 for further experimentation of reducing crosstalk. Likewise, herein, we also have produced 4000 general channel instances containing a number of nets ranging from 20 to 2000. For a particular net number, exactly 200 instances have been produced. We observed that for a maximum number of nets as mentioned above, the instances, on an average, get saturated in terms of reducing crosstalk when these are allowed to go through the relevant *Track_Change* and *Net_Change* algorithms (devised in Chapter 4).

Essentially, Chapter 6 contains only a number of small simple as well as general channel instances and shows only a little number of hardcopy routing solutions for inclusion in the thesis. For each example channel with reasonably shorter in length and lesser number of nets, three routing solutions have been presented: (a) The first one displays the minimum area routing solution against after a standard existing routing algorithm, (b) the second one depicts the significantly reduced crosstalk routing solution after execution of the first crosstalk reduction algorithm, *Track_Change*, and (c) the third one includes the mostly reduced crosstalk routing solution obtained after execution of algorithm *Net_Change*. Graphs have also been drawn for showing a reduction in crosstalk after execution of each associated algorithm for minimizing crosstalk; these demonstrate the deviation in percentage reduction in crosstalk as the number of nets increases. *The reduction in crosstalk in each case is extremely significant*.

Chapter 7 is the last contributory chapter of the thesis in which *we have developed two simple, efficient parallel heuristics for computing reduced crosstalk routing solutions for simple instances of channel specifications. The parallel algorithms presented herein demonstrate that the algorithms devised in Chapter 4 can eventually be parallelized to get efficient parallel algorithms*.

8.2 Open Problems and Future Scopes

Now at the end of the thesis, we would like to draw attention to some possible open problems that future researchers may consider as their field of work. Although several works have been accomplished and included in this thesis, plenty of other allied tasks is still there to do. Crosstalk may be minimized by sacrificing a tolerable limit of more area. We observed in Chapter 3 that when t–1 blank tracks (i.e. the tracks containing no interval of any net) are introduced into a t-track two-layer feasible routing solution such that a blank track is inserted in between every pair of consecutive tracks of nets, the resulting routing solution may not have any crosstalk (as the gap between two adjacent tracks containing nets now is sufficiently large). However, this routing solution uses almost twice the area of the initial routing solution, whereas our prime interest of some VLSI chip design is to compute a routing solution of as minimum area as possible. Thus, such a routing solution is, in general, not acceptable. Nevertheless, this tradeoff between area and crosstalk in computing different two-layer channel routing solutions can be exercised and experimented scrupulously.

As a part of work, in this thesis, we have designed heuristics for reducing crosstalk in a given (two-layer) routing solution of the minimum area; however, it is often enviable if the crosstalk is straightway reduced starting from a given channel instance wherein the area is also minimized as much as possible. If we start with a routing solution, often that forces to assign a pair of nets to be placed on the same track that, in effect, may render more crosstalk. As such, for any given channel instance, either simple or general, neither we can guarantee an optimal crosstalk two-layer channel routing solution, nor we can compute a near-optimal one.

Moreover, instead of considering crosstalk along the length of the channel (i.e. horizontal crosstalk) crosstalk along the height of the channel (i.e. vertical crosstalk) may also play an important role, where we may assume it as well, as an issue of optimization, or simultaneous consideration along with horizontal crosstalk. Crosstalk along with congestion of interconnecting wires over a region of the chip floor, and thus hot spot formation may attract scholars in future, and their synchronized optimization might enhance the performance of a routing solution.

The problem of crosstalk minimization in three-layer and multi-layer channel routing is still open in any routing model. Usually, there are two reserved three-layer channel routing models: VHV and HVH. Among several other models, reserved multi-layer channel routing models may include V_iH_i, $V_{i+1}H_i$, and V_iH_{i+1}, where vertical and horizontal layers of interconnect alternate within a channel (along the third dimension). Though we firmly believe that the problems are hard to solve, yet the nature of the problems of crosstalk minimization in the three- and multi-layer channel routing is still undiscovered.

At the same time, the issue of bottleneck crosstalk minimization is also open in the aforementioned three- and multi-layer channel routing models. We have only guessed the hardness of the problems; no necessary proofs have yet been established along with devising efficient heuristics. In the two-layer VH routing model, the trivial lower bound (l) on the number of tracks is $max(d_{max}, v_{max})$, where d_{max} is the channel density and v_{max} is the length of the longest path in the vertical constraint graph (VCG), which is acyclic [49, 96]. A non-trivial lower bound on the number of tracks (in the two-layer VH routing model), for a general channel instance that does not contain any cyclic VCG, has also been computed in [62]. We may note that this non-trivial lower bound is never worse than the aforesaid trivial lower bound, l, in the stated routing model. For a simple channel instance, however, in the two-layer VH routing model, the supposed lower bound (l) is simply d_{max}, as $d_{max} \geq v_{max} = 1$, which is same for the three-layer VHV routing model for any channel, simple or general, or any channel that contains a cycle in its VCG.

In the case of three-layer HVH routing model, the aforementioned lower bound (l) is $max(\lceil d_{max}/2 \rceil, v_{max})$, where the VCG should also be free from any cyclic vertical constraint. Accordingly, different lower bounds (l) on the number of tracks are also there for different multi-layer channel routing models [49]. Now the question is whether a channel instance has a routing solution in an assumed routing model whose bottleneck crosstalk is no more than $p \geq 1$, for any given integer p, where the number of tracks required is at most $q \geq 0$ more than the optimal? Evidently, a lower bound (l) on the number of tracks required in a stated routing model is either less than or equal to the optimal number of tracks required for a given channel. Consequently, devising desired routing solutions where bottleneck crosstalk is no more than p, and the number of tracks required is no more than q above the optimal number of tracks could be a task for future researchers.

We would also like to point out a few more possible extensions of the algorithms developed in Chapter 7 of this thesis as follows. The algorithms included in the said chapter have been devised for simple channel instances only; these may also be worthy in modifying the algorithm for general instances of channel specifications with multi-terminal nets. A comprehensive version of algorithm *Parallel Net Change* may produce much-reduced crosstalk routing solutions when two overlapping nets (or a group of overlapping nets) exchange their tracks pairwise (in a sequence). Computation of minimized crosstalk routing solutions in the case of three- and multi-layer channel routing in different routing models may draw the interest of future researchers. Also, doglegging may be introduced in all these cases.

On a broader scale, there are several opportunities for future research. For example, routing congestion is a work in VLSI circuit synthesis that estimates and optimizes delay as well as hot spots present in a circuit [79]. Delay and power related issues have been considered and statistically analyzed and attempted to optimize for VLSI in [84]. Power consumption is a burning issue that has also been acknowledged by Sherwani in his renowned book entitled ‘Algorithms for VLSI Physical Design Automation’ along with multi-chip module [81].

It is unlikely for high speed systems to achieve very low power while enhancing system performance under the current trends for MOS technologies [95]. High performance circuits usually consume significant amounts of power due to increase in frequency, bandwidth, and system integration, and this consumed power leads to higher heat dissipation and in turn to higher working temperature(s). This not only affects circuit performance directly, by slowing down the CMOS transistors on ICs but also reduces the reliability. A circuit with considerable power consumption requires extra cost to remove heat at the packaging level, and therefore, the reduction of power dissipation is needed at the chip design stage. In general, it is advisable to have an even temperature distribution for temperature sensitive circuits [95].

The thermal management of microprocessors has become an increasing challenge in recent years because of localized high flux hot spots which cannot be effectively removed by conventional cooling techniques. The work of Wang and Bar-Cohen [90] describes the use of the silicon chip itself as a thermoelectric cooler to suppress the hot spot temperature. As semiconductor-based technology has rapidly developed, producing ever smaller and faster silicon-chip based computer processors, effective cooling of these chips has remained an unsolved issue. As a consequence, researchers have started developing ways to cool hot spots using tiny on-chip silicon *microcoolers* [90, 91].

A three-dimensional analytical thermal model of the silicon chip, including localized thermoelectric cooling, thermoelectric heating, silicon Joule heating, hot spot heating, background heating, and conductive/convective cooling on the back of the silicon chip, has been developed and used to predict the on-chip hot spot cooling performance [90]. This work also investigates the effects of hot spot size, hot spot heat flux, silicon chip thickness, microcooler size, the doping concentration in the silicon, and parasitic Joule heating from electric contact resistances on the cooling of on-chip hot spots [90].

Hot spots can severely degrade the performance and reliability of a microprocessor. However, cooling methods addressing the entire chip can often cause unnecessary over-cooling, as well as raise the cost, weight, and volume of the cooling solution [90].

Building on prior analytical work, Bar-Cohen and Wang, both mechanical engineers, developed a three-dimensional mathematical model of the thermal behaviour of a silicon chip using computer software. The model accounts for all aspects of heating and cooling on the chip, including localized cooling, hot spot heating, background heating from nearby circuitry, and conductive/convective cooling through the back of the chip.

The model predicts that when an electric current is applied to a region of *highly doped* silicon (silicon with a high level of added impurities) on the back of a chip, a cool region is created on the chip. If the cool region is located opposite a microprocessor hot spot, it absorbs heat and lowers the hot spot temperature. This localized cooling phenomenon occurs via the thermoelectric effect – the use of electrical energy to transfer heat against the natural hot-to-cold thermal gradient. The silicon and the metal lead that brings electric current to the back of the chip have very different thermoelectric sensitivities. As a result, a cooling effect occurs at the contact-cap and cap-silicon junctions and heat is pulled out of the hot spot [90, 91].

Similar microcooling systems have been proposed, such as thin-film thermoelectric coolers (TFTECs) that consist of two layered ultra-thin semiconductor lattices, such as silicon-germanium on top of silicon. Like the silicon microcoolers, TFTECs are positioned on the back of the silicon chip to pull away heat. Among their advantages are compactness and fast cooling response. One main disadvantage, however, is that for TFTECs, a thermal *interface* resistance is present between the chip and the thin film, reducing the cooling effect [91].

Whatever may be the situation, to achieve high performance routing/design in VLSI, the factors to be given careful consideration are power dissipation density caused by the distribution of components and connecting wires over the chip floor. These factors often lead to congestion and the subsequent formation of hot spots. To minimize all these, the power supplied to the circuit is to be diminished, and this encourages research in the field of low power VLSI design [7, 17, 74, 94, 95]. Based on congestion of wire segments that are placed closer to each other over a local routing region crosstalk comes into existence [9, 25, 26, 27, 30, 39, 42, 45, 46, 75, 87, 89, 93]. Crosstalk is intensified if an aggressor net is placed around other nets. Moreover, the amount and direction of current flowing through the different wires located in the region often give rise to electrical hazards like crosstalk, hot spot formation, and eventually, delay in propagating electrical signals.

Bibliography

[1] A Brief Look at Semiconductor Technology, 2014.http://www.cis.poly.edu /cs2204/silicon.pdf.

[2] http://www.itrs.net/ITRS%201999-2014%20Mtgs,%20Presentations%20&%20-Links/2013ITRS/2013TableSummaries/2013ORTC_SummaryTable.pdf.

[3] Akl S. G., *Parallel Computation: Models and Methods*, Prentice Hall, Upper Saddle River, New Jersey, USA, 1997.

[4] Anand Kumar A., *Fundamentals of Digital Circuits*, Second Edition, Prentice-Hall of India Learning Pvt. Ltd., New Delhi, 2009.

[5] Banerjee S., M. T. Dey, and S. Dutta, Difficult Channel Generation using Genetic Algorithm, *International Journal of Artificial Intelligence and Applications*, vol. 1, no. 4, pp. 145-157, 2010.

[6] Batcher K. E., Sorting Networks and Their Applications, *Proceedings of the AFIPS Spring Joint Computer Conference*, vol. 32, AFIPS Press, Reston, VA, pp 307-314, 1968. Reprinted in: C. L. Wu and T. S. Feng (Eds.): *Interconnection Networks for Parallel and Distributed Processing*, IEEE Computer Society, pp 576-583, 1984.

[7] Bellaouar A. and M. Elmasry, *Low-Power Digital VLSI Design: Circuits and Systems*, Springer Science + Business Media, LLC, New York, 1995.

[8] Berge C. and V. Chvatal (Editors), *Topics on Perfect Graphs*, Elsevier Science Ltd., North-Holland Mathematics Studies, 1984.

[9] Bianco A., D. Cuda, M. Garrich, G. G. Castillo, V. Martina, and F. Neri, Crosstalk Minimization in Microring-based Wavelength Routing Matrices, *Proceedings of the IEEE International Conference on Global Telecommunications Conference* (*GLOBECOM 2011*), pp. 1-5, 2011.

[10] Burstein M. and R. Pelavin, Hierarchical Channel Router, *INTEGRATION: The VLSI Journal*, vol. 1, pp. 21-38, 1983.

[11] Chao H.-Y. and M. P. Harper, A Difficult Channel Routing Generator, *ECE Technical Reports* (*TR-EE 95-1*): Paper 108, Electrical and Computer

Engineering, Purdue University, Purdue e-Pubs: http://docs.lib.purdue.edu/ecetr /108, 1995.

[12] Chen Y. K. and M. L. Liu, Three-Layer Channel Routing, *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol.3, no. 2, pp. 156-163, 1984.

[13] Cormen T. H., C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, Third Edition, Prentice-Hall of India Learning Pvt. Ltd., New Delhi, 2010.

[14] Dasgupta S. and A. Bulusu, FinFET Device Circuit Co-Design: Issues and Challenges: A Tutorial, *Presented in the 28th International Conference on VLSI Design and the 14th International Conference on Embedded Systems*, Bangalore, India, Jan 3-7, 2015.

[15] Deutsch D. N., A Dogleg Channel Router, *Proceedings of the 13th ACM/IEEE Design Automation Conference*, pp. 425-433, 1976.

[16] Eckhoff. J., Extremal Interval Graphs, *Journal of Graph Theory*, vol. 17, no. 1, pp. 117-127, 1993.

[17] EE Herald, *Low Power VLSI Chip Design: Circuit Design Techniques*, available at: http://www.eeherald.com/section/design-guide/Low-Power-VLSI-Design.html.

[18] Emanuel B., S. Wimer, and G. Wolansky, Using Well-Solvable Quadratic Assignment Problems for VLSI Interconnect Applications, *Discrete Applied Mathematics*, vol. 160, no. 4, pp. 525-535, 2012.

[19] Ernesto Rayas-Sánchez, J., A Frequency-Domain Approach to Interconnect Crosstalk Simulation and Minimization, *Microelectronics Reliability*, vol. 44, no. 4, pp. 673-681, 2004.

[20] Eustace R. A., *Intra-Region Routing*, Ph.D. Thesis, Department of Computer Science, University of Central Florida, Orlando, Aug. 1984.

[21] Fan C.-P. and C.-H. Fang, Efficient RC Low-Power Bus Encoding Methods for Crosstalk Reduction, *INTEGRATION: the VLSI Journal*, vol. 44, no. 1, pp. 75-86, 2011.

[22] Fishburn P. C., *Interval Orders and Interval Graphs: A Study of Partially Ordered Sets*, New York: Wiley, 1985.

[23] Floyd T. L., *Digital Fundamentals*, Eleventh Edition, Pearson Education India (Prentice-Hall), 2014.

[24] Frenzel L. E., *Crash Course in Digital Technology*, Second Edition, Newnes, 1998.

[25] Gao T. and C. L. Liu, Minimum Crosstalk Channel Routing, *Proceedings of the IEEE International Conference on Computer-Aided Design*, pp. 692-696, 1993.

[26] Gao T. and C. L. Liu, Minimum Crosstalk Switchbox Routing, *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, IEEE Computer Society Press, pp. 610-615, 1994.

[27] Gao T. and C. L. Liu. Minimum Crosstalk Channel Routing, *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 15, no. 5, pp. 465-474, 1996.

[28] Garey M. R. and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, Freeman, New York, 1979.

[29] Golumbic M. C., *Algorithmic Graph Theory and Perfect Graphs*, Academic Press, New York, 1980.

[30] Gupta U. and N. Ranganathan, A Utilitarian Approach to Variation Aware Delay, Power, and Crosstalk Noise Optimization, *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 19, no. 9, pp. 1723-1726, 2011.

[31] Hamachi G. T. and J. K. Ousterhout, A Switchbox Router with Obstacle Avoidance, *Proceedings of the 21st ACM/IEEE Design Automation Conference*, pp. 173-179, 1984.

[32] Hashimoto A. and J. Stevens, Wire Routing by Optimizing Channel Assignment within Large Apertures, *Proceedings of the 8th ACM Design Automation Workshop*, pp. 155-169, 1971.

[33] Ho T.-T., S. S. Iyengar, and S.-Q. Zheng, A General Greedy Channel Routing Algorithm, *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 10, no. 2, pp. 204-211, 1991.

[34] Hougardy S., Classes of Perfect Graphs, *Discrete Mathematics*, vol. 306, pp. 2529-2571, 2006.

[35] Hsu C.-C., M. P.-H. Lin, and Y.-T. Chang, Crosstalk-Aware Multi-Bit Flip-Flop Generation for Power Optimization, *INTEGRATION: the VLSI Journal*, vol. 48, pp. 146-157, 2015.

[36] Hsu W.-L. and T.-H. Ma, Fast and Simple Algorithms for Recognizing Chordal Comparability Graphs and Interval Graphs, *SIAM Journal on Computing*, vol. 28, no. 3, pp. 1004-1020, 1999.

[37] Hwang K., *Advanced Computer Architecture: Parallelism, Scalability, Programmability*, McGraw-Hill, Inc., New York, 1993.

[38] JáJá J., *An Introduction to Parallel Algorithms*, vol. 17. Reading: Addison-Wesley, 1992.

[39] Jhang, K.-S., S. Ha, and C. S. Jhon, Simulated Annealing Approach to Crosstalk Minimization in Gridded Channel Routing, *VLSI Design*, vol. 7, no. 1, pp. 85-95, 1998.

[40] Joobbani R., *An Artificial Intelligence Approach to VLSI Routing*, Kluwer Academic Publishers, Boston, USA, 1986.

[41] LaPaugh A. S., *Algorithms for Integrated Circuit Layout: An Analytic Approach*, Ph.D. Thesis, Laboratory for Computer Science, Massachusetts Institute of Technology, Cambridge, 1980.

[42] Liberali V., R. Rossi, and G. Torelli, Crosstalk Effects in Mixed-Signal ICs in Deep Submicron Digital CMOS Technology, *Microelectronics Journal*, vol. 31, no. 11, pp. 893-904, 2000.

[43] Lincoln B., *Introduction to Digital Electronics*, First Edition, Pearson Education India, 2014.

[44] Maheswari, M. and G. Seetharaman, Multi-Bit Random and Burst Error Correction Code with Crosstalk Avoidance for Reliable on Chip Interconnection Links, *Microprocessors and Microsystems*, vol. 37, no. 4, pp. 420-429, 2013.

[45] Manem H. and G. S. Rose, A Crosstalk Minimization Technique for Sublithographic Programmable Logic Arrays, *Proceedings of the Ninth IEEE*

International Conference on Nanotechnology (*IEEE NANO 2009*), pp. 218-221, 2009.

[46] Moghaddam S. A. and N. Masoumi, Analysis and Simulation of a Novel Gradually Low-K Dielectric Structure for Crosstalk Reduction in VLSI, *Microelectronics Journal*, vol. 39, no. 12, pp. 1751-1760, 2008.

[47] Moiseev K., S. Wimer, and A. Kolodny, Timing-Constrained Power Minimization in VLSI Circuits by Simultaneous Multilayer Wire Spacing, *INTEGRATION: the VLSI Journal*, vol. 48, pp. 116-128, 2015.

[48] Mukherjee A., *Introduction to nMOS and CMOS VLSI Systems Design*, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, USA, 1986.

[49] Pal R. K., *Multi-Layer Channel Routing: Complexity and Algorithms*, Narosa Publishing House, New Delhi (Also published from CRC Press, Boca Raton, USA and Alpha Science International Ltd., UK), 2000.

[50] Pal R. K., Absolute Area Approximation in Channel Routing is NP-Hard, *Journal of Informatics and Mathematical Sciences*, vol. 1, nos. 2-3, pp. 121-137, 2009.

[51] Pal R. K., A. K. Datta, S. P. Pal, M. M. Das, and A. Pal, A General Graph Theoretic Framework for Multi-Layer Channel Routing, *Proceedings of the Eighth VSI/IEEE International Conference on VLSI Design*, pp. 202-207, Jan. 4-7, 1995.

[52] Pal R. K., A. K. Datta, S. P. Pal, and A. Pal, Resolving Horizontal Constraints and Minimizing Net Wire Length for VHV Channel Routing, *Technical Report, no.: TR/IIT/CSE/92/01*, Department of Computer Science and Engineering, IIT, Kharagpur (1992).

[53] Pal R. K., A. K. Datta, S. P. Pal, and A. Pal, Resolving Horizontal Constraints and Minimizing Net Wire Length for Multi-Layer Channel Routing, *Proceedings of the IEEE Region 10's Eighth Annual International Conference on Computer, Communication, Control and Engineering* (*TENCON 1993*), vol. 1, pp. 569-573, 1993.

[54] Pal R. K. and A. Pal, An Efficient Graph-Theoretic Algorithm for Three-Layer Channel Routing, *Proceedings of the Fifth IEEE International Conference on VLSI Design*, pp. 259-262, 1992.

[55] Pal R. K., S. P. Pal, A. K. Datta, and A. Pal, NP-Completeness of Multi-Layer No-Dogleg Channel Routing, and an Efficient Heuristic, *Proceedings of the Sixth VSI/IEEE International Conference on VLSI Design*, pp. 80-83, 1993.

[56] Pal R. K., S. P. Pal, and A. Pal, On the Computational Complexity of Multi-Layer Channel Routing, *Technical Report no.: TR/IIT/CSE/92/02*, Dept. of Computer Sc. & Engg., IIT, Kharagpur, 1992.

[57] Pal R. K., S. P. Pal, and A. Pal, Wire Length Minimization in Multi-Layer Channel Routing: Complexity Results, and Efficient Algorithms, *Technical Report no.: TR/IIT/CSE/93/07*, Dept. of Computer Sc. & Engg., IIT, Kharagpur, 1993.

[58] Pal R. K., S. P. Pal, and A. Pal, On the Computational Complexity of Area, and Wire Length Minimization in Multi-Layer Channel Routing, *Proceedings of the Third National Seminar on Theoretical Computer Science* (*NSTCS 1993*), pp. 103-119, 1993.

[59] Pal R. K., S. P. Pal, and A. Pal, Minimizing Net Wire Length in Multi-Layer Channel Routing, *Proceedings of the CSA Silver Jubilee Workshop on Computing, and Intelligent Systems*, pp. 171-188, 1993.

[60] Pal R. K., S. P. Pal, and A. Pal, Absolute Approximation for Channel Routing is NP-Hard, *Proceedings of the Fourth National Seminar on Theoretical Computer Science* (*NSTCS 1994*), pp. 28-39, 1994.

[61] Pal R. K., S. P. Pal, and A. Pal, On the Computational Complexity of Approximate Area Minimization in VLSI Design, *Proceedings of the International Conference on Computer Systems and Education* (*ICCSE 1994*), pp. 378-380, 1994.

[62] Pal R. K., S. P. Pal, and A. Pal, An Algorithm for Finding a Non-Trivial Lower Bound for Channel Routing, *INTEGRATION: the VLSI Journal*, vol. 25, no. 1, pp. 71-84, 1998.

[63] Pal R. K., S. P. Pal, A. Pal, and A. K. Dutta, NP-Completeness of Multi-Layer no-Dogleg Channel Routing and an Efficient Heuristic, *Proceedings of the Sixth IEEE International Conference on VLSI Design*, pp. 80-83, 1993.

[64] Pal R. K. and S. Sen Sarma, Wire Length Minimization in Routing and Performance Enhancement in VLSI Design: A Tutorial, *Presented in the Fourth IEEE VLSI Design and Test Workshops 2000* (*IEEE VDAT 2000*); *VLSI Design and Test: Milestones and Challenges*, Edited by C. P. Ravikumar, pp. 70-71, Phoenix Publishing House Pvt. Ltd., New Delhi, 2000.

[65] Pande P. P., A. Ganguly, H. Zhu, and C. Grecu, Energy Reduction through Crosstalk Avoidance Coding in Networks on Chip, *Journal of Systems Architecture*, vol. 54, no. 3, pp. 441-451, 2008.

[66] Papadimitriou C. H., *Computational Complexity*, Addison-Wesley Publishing Co., Reading, Massachusetts, 1995.

[67] Patooghy A., S. G. Miremadi, and H. Tabkhi, A Reliable and Power Efficient Flow-Control Method to Eliminate Crosstalk Faults in Network-on-Chips, *Microprocessors and Microsystems*, vol. 35, no. 8, pp. 766-778, 2011.

[68] Quinn M. J., *Parallel Computing: Theory and Practice*, Second Edition, McGraw-Hill, Inc., New York, USA, 1994.

[69] Rajaraman D. and V. Rajaraman, *Computer Primer*, Second Edition, Prentice-Hall of India Learning Pvt. Ltd., New Delhi, 2012.

[70] Ramirez Alfonsin J. L. and B. A. Reed (Editors), *Perfect Graphs*, John Wiley and Sons Ltd., Chichester, 2001.

[71] Reed J., A. Sangiovanni-Vincentelli, and M. Santomauro, A New Symbolic Channel Router: YACR2, *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 4, no. 3, pp. 208-219, 1985.

[72] Reis J. D., M. V. Drummond, A. L. Teixeira, R. N. Nogueira, P. Monteiro, S. Shinada, N. Wada, and G. M. Beleffi, Experimental Demonstration of a Nonlinear Effects Crosstalk Minimization Algorithm, *Proceedings of the National Fiber Optic Engineers Conference*, p. JThA32, Optical Society of America, 2010.

[73] Rivest R. L. and C. M. Fiduccia, A ‘Greedy’ Channel Router, *Proceedings of the 19th ACM/IEEE Design Automation Conference*, pp. 418-424, 1982.

[74] Roy K. and S. Prasad, *Low Power CMOS VLSI Circuit Design*, John Wiley and Sons, Inc., New York, 2000.

[75] Saini S. and S. B. Mandalika, A New Bus Coding Technique to Minimize Crosstalk in VLSI Bus, *Proceedings of the Third IEEE International Conference on Electronics and Computer Technology* (*ICECT 2011*), vol. 1, pp. 424-428, 2011.

[76] Sankaran H. and S. Katkoori, Simultaneous Scheduling, Allocation, Binding, Re-Ordering, and Encoding for Crosstalk Pattern Minimization During High-Level Synthesis, *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 19, no. 2, pp. 217-226, 2011.

[77] Sarrafzadeh M. and C. K. Wong, *An Introduction to VLSI Physical Design*, The McGraw-Hill Companies, Inc., New York, USA, 1996.

[78] Sathish A. and M. Madhavi Latha, Data Encoding Technique for Crosstalk Delay Reduction on Fault Tolerant Data-Bus in DSM Technology, *Procedia Engineering*, vol. 38, pp. 2967-2972, 2012.

[79] Saxena P., R. S. Shelar, and S. Sapatnekar, Routing Congestion in VLSI Circuits: Estimation and Optimization, Springer, New York, USA, 2007.

[80] Schaper G. A., *Multi-Layer Channel Routing*, Ph.D. Thesis, Department of Computer Science, University of Central Florida, Orlando, 1989.

[81] Sherwani N. A., *Algorithms for VLSI Physical Design Automation*, Kluwer Academic Publishers, Boston, 1993.

[82] Smith L., *An Analysis of Area Routing*, Ph.D. Thesis, Stanford University, California, 1983.

[83] Soukup J., Fast Maze Router, *Proceedings of the 15th ACM/IEEE Design Automation Conference*, pp. 100-102, 1978.

[84] Srivastava A., D. Sylvester, and D. Blaauw, Statistical Analysis and Optimization for VLSI: Timing and Power, Springer, New York, USA, 2005.

[85] Supowit K. J., A Minimum Impact Routing Algorithm, *Proceedings of the 19th ACM/IEEE Design Automation Conference*, pp. 104-112, 1982.

[86] Szymanski T. G., Dogleg Channel Routing is NP-Complete, *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 4, no. 1, pp. 31-41, 1985.

[87] Terapasirdsin A. and N. Wattanapongsakorn, Crosstalk Minimization in VLSI Design using Signal Transition Avoidance, *Proceedings of the IEEE International Symposium on Communications and Information Technologies* (*ISCIT 2010*), pp. 911-915, 2010.

[88] Velivala S., Verification Challenges at Advanced nodes: A Tutorial, *Presented in the 28th International Conference on VLSI Design and the 14th International Conference on Embedded Systems*, Bangalore, India, Jan 3-7, 2015.

[89] Verma S. K. and B. K. Kaushik, Crosstalk and Power Reduction using Bus Encoding in RC Coupled VLSI Interconnects, *Proceedings of the Third IEEE International Conference on Emerging Trends in Engineering and Technology* (*ICETET 2010*), pp. 735-740, 2010.

[90] Wang P. and A. Bar-Cohen, On-chip Hot Spot Cooling using Silicon Thermoelectric Microcoolers, *Journal of Applied Physics*, AIP Publishing, vol. 102, issue 3, 2007.

[91] Wang P., A. Bar-Cohen, B. Yang, G. L. Solbrekken, and A. Shakouri, Analytical Modeling of Silicon Thermoelectric Microcooler, *Journal of Applied Physics*, vol. 100, 014501, 2006.

[92] Wang Y., S. Yang, X. Li, Y. Cao, and H. Yang, Research on Orientation between Dual Stripline for Minimizing Crosstalk in Integrated High-Density Circuits, *Proceedings of the Second IEEE International Conference on Computer Engineering and Technology* (*ICCET 2010*), vol. 6, pp. V6-221, 2010.

[93] Xu J., X. Hong, T. Jing, L. Zhang, and J. Gu, A Coupling and Crosstalk-Considered Timing-Driven Global Routing Algorithm for High-Performance Circuit Design, *INTEGRATION: The VLSI Journal*, vol. 39, no. 4, pp. 457-473, 2006.

[94] Yeap G. and A. Wild, Introduction to Low-Power VLSI Design, *International Journal of High Speed Electronics and Systems*, vol. 7, Issue 2, 1996.

[95] Yeap G. K. and F. N. Najm (Editors), Low Power VLSI Design and Technology, World Scientific, Singapore, 1996.

[96] Yoshimura T. and E. S. Kuh, Efficient Algorithms for Channel Routing, *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 1, no. 1, pp. 25-35, 1982.