NEW ROUTING STRATEGIES FOR VLSI

by

SHINICHIRO HARUYAMA, B.S., M.S.

DISSERTATION

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

DOCTOR OF PHILOSOPHY

THE UNIVERSITY OF TEXAS AT AUSTIN

December, 1990

Dedicated to my father Masaya Haruyama and my mother Eiko Haruyama

VITA

Shinichiro Haruyama was born in Kobe, Japan on December 4, 1957, the son of Masaya Haruyama and Eiko Haruyama. After completing his work at Koyo Gakuin High School, Hyogo-ken, Japan, in 1976, he entered Tokyo University in Tokyo, Japan. He received the degree of Bachelor of Science in Physics from Tokyo University in March 1981.

He moved to the U.S.A. in September 1981, and received the degree of the Master of Engineering Science from the Graduate School of the University of California at Berkeley in 1983. In January 1984, he entered the Graduate School of the Department of Computer Sciences of the University of Texas at Austin.

Permanent address: 772-11 Nakano,
Yamaguchi-cho,
Nishinomiya-shi, Hyogo-ken
651-14 Japan

This dissertation was typeset[1] with LaTeX by the author.

[1] LaTeX document preparation system was developed by Leslie Lamport as a special version of Donald Knuth's TeX program for computer typesetting. TeX is a trademark of the American Mathematical Society. The LaTeX macro package for The University of Texas at Austin dissertation format was written by Khe-Sing The.

Acknowledgments

I would like to express my sincere appreciation to my co-advisers Dr. Donald S. Fussell and Dr. D. F. Wong. They have given me constructive criticism about my work and expert knowledge about VLSI CAD. Weekly meetings with them kept me on track.

I am also deeply grateful to Dr. Miroslaw Malek, Dr. Daniel P. Miranker of the University of Texas at Austin, and Dr. Robert J. Smith, II of MCC for their serving as committee members.

My sincere thanks also go to a good friend of mine, Mr. Kazuhiko Nishi, President of ASCII Corporation, Tokyo, who decided to offer a scholarship all through my Ph.D. years. Without ASCII's financial help, this doctoral research would not have been possible.

Finally, I am deeply indebted to members of my family who were constantly giving me encouragement and support : my father Masaya Haruyama, my mother Eiko Haruyama and my grandmother Kaoru Hidaka.

SHINICHIRO HARUYAMA

The University of Texas at Austin

December, 1990

Abstract

This dissertation describes two methods of routing for VLSI layout : one method for channel routing and another method for power wire routing.

Our two-layer channel router is designed to find solutions which minimize both wiring area and number of vias simultaneously. Our method, called topological channel routing, analyzes the topological relationship of wires before the wires are mapped onto the channel. A unique layout design rule called an interleaving mesh is used. The interleaving mesh prohibits long wires on one layer from overlapping with wires on the other layer, thus has smaller cross talks of signals because of smaller capacitive couplings between those wires on different layers. Experimental results show that the algorithm generates very good solutions. For example, we have obtained a height of 41 for the famous Deutsch's Difficult Example without any parallel overlaps of wires and simultaneously a via count of 186, which is one of the best results ever reported in the literature.

Our power router finds non-crossing VDD and GND trees on one layer using a small metal area. The solution is obtained under the constraints of metal migration and voltage drop. Experimental results show that the power wire area is considerably smaller than a previously developed method for single-layer routing.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

This dissertation presents new algorithms for routing the wires of integrated circuit layouts : a new channel routing algorithm and a new power routing algorithm. Routing is an important part of the chip design process, but it is only part of the circuit layout step. Layout, in turn, can be done only after chip designers specify what a circuit is supposed to do. So we begin the introduction chapter by describing process of how a chip is designed from concept to product. We show some design styles that are currently widely used. We then discuss methods used for circuit layout, i.e., placement and routing. In particular, we focus on signal wire routing and power wire routing.

1.1 Integrated Circuit Design

Rapid progress in intergrated circuit fabrication technology has enabled the manufacturing of chips with more than one million devices. It is now impossible to design integrated circuits of this complexity without the aid of computers. Integrated circuits are designed by going through the following steps : architecture design, circuit design, circuit layout, and testing. Computers are used in each step.

In the architecture design step, we define how the chip system behaves, how the system is partitioned into units, how and what information each unit communicates with others, and how and what information the system communicates with outside world. Defining the architecture is the first step in creating a new system, and it is a step primarily requiring human creativity. So, this step depends less on computer aids than the following steps. Silicon compilers [22] (ideally) accept behavioral descriptions and produce chip layouts.

In the circuit design step, logic circuits are specified that perform the desired architecture. The logic circuits consist not only of such simple components as NAND gates and NOR gates, but also large modules such as RAM, ROM, and PLA. The circuit specification also includes interconnection information among these components. Construction of a circuit from a behavioral specification can be done by computers almost automatically.

In the circuit layout step, circuits are mapped to the two-dimensional surface of a chip. The mapping process includes floorplanning, placement of components, and routing of wires between components and power wires. Depending on how urgently chips need to be made, and how much performance is needed, designers can choose from several layout design style options for

mapping circuits. There are three major design styles : gate array design style, standard cell design style, and macrocell design style. If quick turn-around time is required, the gate array design style is the best choice. If high performance is needed, the macrocell design style is preferred, though the design time is much longer than that of the gate array design style. Standard cell design is between them in terms of design time and quality. It is also used to design modules in macrocell design style. The circuit layout of gate array designs and standard cell designs can now be done completely by computers. But the layout of macrocell designs still need human interaction to achieve high performance.

In the testing step, the chip is tested to see if it performs correctly according to the architecture design specification. A computer generates test patterns, and feeds them to the chip. The output is then compared with the ideal output of a chip without faults. Even with automatic test pattern generation, it is impossible to achieve 100% fault coverage, though the percentage is usually close to 100. We will describe the layout step in more detail next.

1.2 Integrated Circuit Layout

Integrated circuit layout is the process of mapping a logic circuit to the two-dimensional surface of a chip. The circuits to be mapped are not only computing elements such as logic gates, but also interconnecting wires.

The main objectives of integrated circuit layout are the minimization of circuit area and the optimization of circuit performance, i.e., maximization of speed. If there are N flaws per unit area, and the area of a chip is A, the probability that a chip is defect-free is e^{-NA}. (See [28]) Because of this exponential probability, even a small percentage of area reduction improves the production yield of chips significantly. The maximization of speed, another

objective, can be achieved by various methods such as logic minimization and minimization of wire length. Minimization of wire length is effective because a longer wire has a larger capacitance that takes longer time to be charged up and down. Minimization of wire length is partly achieved by the first objective of area minimization because shorter wires are needed to run across smaller area.

Statistics [47] of 1985 show that as much as 40% of chip area was used for inter-module signal wires, and as much as 20% was used for power wires (See Table 1.1 for ratios of chip area). As the number of logic gates increases with the advance of fabrication technologies, the number of wires to connect them increases even more rapidly. As a result the percentage of the area for

Chip components	%
Modules	30
Inter-module Wires	40
Power Wires	20
Peripheral (Bonding Pads)	10

Table 1.1: Percentage of Area used by Modules, Wires, and Bonding Pads

wires is becoming larger as compared with that of the area for logic gates. To achieve the above objectives when the percentage of area occupancy by wires is increasing, minimization of chip area, especially minimization of wire area, is becoming more important than ever before.

1.3 Design Styles

There are several popular design styles used today : gate array design, standard cell design, and macrocell design. A gate array design (See Figure 1.1) uses horizontal arrays of gate cells separated by channels. The cells are usually

of the same height and the same width. All cells are pre-fabricated, and only metal wires to connect these cells are user-defined. Only the masks for metal layers need to be designed on demand for a particular function. Thus the time to design a chip is significantly less than for fully customized designs. Usually it is not possible to use all of the transistors because the number of transistors a user needs to realize the circuit must be smaller than the fixed number of transistors of a given gate array chip. Since the area for routing wires is fixed, failure of routing sometimes occurs when a routing algorithm cannot find a wire layout that can fit in the given area reserved for wires.

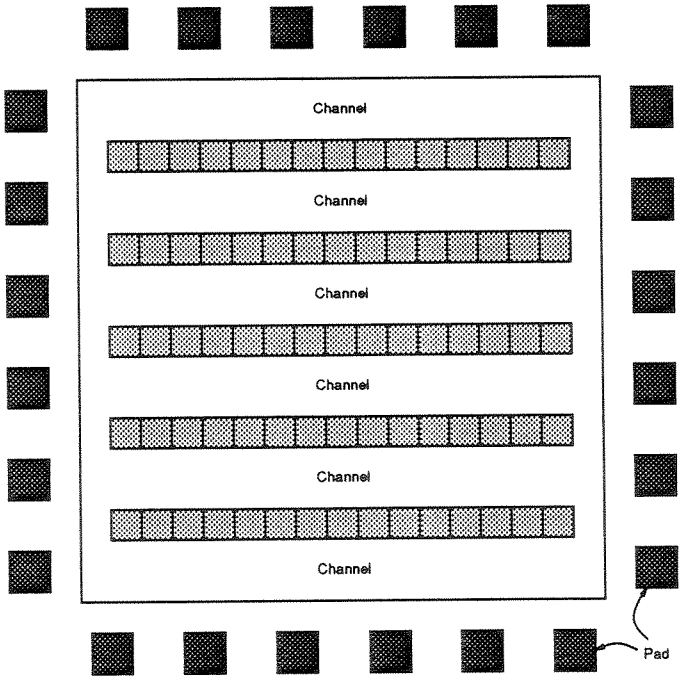


Figure 1.1: Gate Array Design Style

The standard cell design style has the same arrangement of cells as the gate array design style. However, in a standard cell design, cells are not prefabricated on a chip, so a user has the freedom to place cells at any location as long as they form a horizontal row. The cells of a standard cell design can

be of variable width, but are of the same height, so that if they are placed on a row, the top and the bottom of the row form straight lines. The horizontal channels between horizontal cell rows are used for laying out wires. The height of each channel is defined by the user. As a result, users do not have to worry about routing failures due to wire congestion in a channel. Thus, it is always possible to achieve a 100% routing completion rate.

A macrocell design (See Figure 1.2) is the most dense and the highest performing design. A macrocell design chip usually consists of rectilinear modules called macrocells which may have a fixed shape or a flexible shape with certain constraints (e.g. aspect ratio constraints). Macrocells include RAMs, ROMs, PLAs, ALUs, and even standard-cell blocks. These macrocells are placed inside a near-square area, and they are interconnected by wires using space between the macrocells.

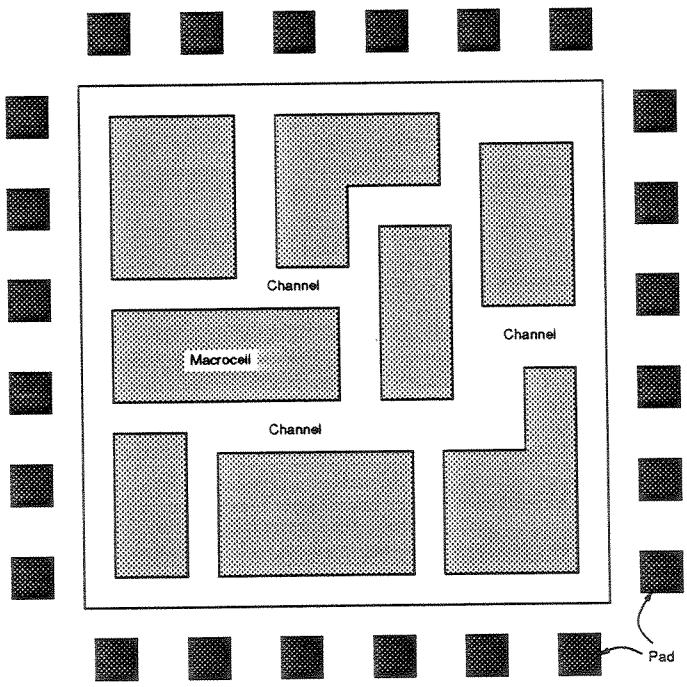


Figure 1.2: Macrocell Design Style

1.4 Placement and Routing

The problem of circuit layout is so difficult that there exists no known polynomial time algorithm to obtain an optimal solution. Usually a divide-and-conquer method is used to tackle this difficult problem. The layout problem is usually divided into two sub-problems : placement and routing. Placement is the problem of placing all the components on a chip with the objective of minimum area or minimum wire congestion. Routing is the problem of laying out all the wires that interconnect components. The routing problem is further divided into global routing and detailed routing. Global routing roughly determines which regions each wire passes through, and detailed routing determines the precise location of each wire.

1.4.1 Placement

In the placement step, the positions of cells are determined. The simplest and most intuitive way of placing cells is by initial placement followed by iterative improvement. Examples of initial placement algorithms are the cluster growth algorithm [13] and the force-directed placement algorithm [32]. In the cluster growth algorithm, a cell is selected which has the largest connectivity to already-placed cells. The cell is placed at a location that minimizes the net length. This continues until all the cells are placed. In the force-directed placement algorithm, it is assumed that there are attractive forces between components sharing a common signal, and repulsive forces between components having no signals in common. Simultaneous equations specifying the equilibrium conditions of this physical system are solved to find the placement of all the components. After initial placement, an iterative improvement procedure is used to improve the placement. One example of iterative improvement

is pairwise interchange. Pairwise interchange iteratively picks a pair of cells and checks if the total wire length becomes shorter when the cells are swapped. If so, the cells are actually swapped. All the combinations of pairs are tried in each iteration.

The above algorithms have the objective of minimizing total layout area and total wire length. This objective, as it turns out, is not a good one. It often causes serious routing congestion at the center of a chip and sometimes causes the failure of routing. This congestion problem may be solved by expanding the positions of cells, but that jeopardizes the initial effort to minimize the area. In 1977, Breuer [4] proposed a new placement algorithm called the min-cut algorithm, which solved this wire congestion problem at the time of placement. The min-cut algorithm divides a circuit approximately in half such that the number of nets connecting the two sets is minimal. An efficient graph partitioning algorithm by Kernighan and Lin [24] is used to partition the circuit. The layout area is then partitioned into two parts by either a vertical or horizontal cut line, and the two sub-circuits are placed at these two parts of the chip. This min-cut procedure is recursively applied by alternatively cutting the layout area by vertical and horizontal cut lines. The algorithm thus finds a placement with minimal congestion, instead of minimal total area or total wire length.

In 1983, Kirkpatrick, Gelatt, and Vecchi [25] proposed a new placement algorithm based on the technique of simulated annealing. Simulated annealing can be used not only for the placement problem, but also for various other combinatorial optimization problems. In a conventional iterative improvement algorithm, a local adjustment is done if it decreases the cost function. This often ends up at a local minimum, and once it is stuck at the local

minimum, it can never escape from it. Simulated annealing, however, allows an uphill movement, enabling the escape from the local minimum. The probability of uphill movements is controlled throughout the optimization process. Since simulated annealing is capable of minimizing a cost function with many degrees of freedom, the function can be a combination of total area, total wire length, wire congestion, or weighted critical net value, thus taking into consideration many complex placement situations. Even though simulated annealing takes a lot of computation time, it generally achieves very good placement results. Various applications of simulated annealing are described in a book by Wong, Leong, and Liu [49].

The above placement algorithms are used for gate array designs, standard cell designs, and macrocell designs [40]. The placement of a gate array design concerns the assignment of gates to cells, because the actual positions of cells of a gate array are already defined. Since the area of the chip cannot be changed, the main objective of gate array placement is to be able to interconnect every signal net within the area. The secondary objective is to maximize the chip performance. A standard cell design, on the other hand, does not have any pre-fabricated gates. Positions of all the gates are defined by a user. So the objective of standard cell placement is to minimize the area and maximize the chip performance. Similarly, macrocell placement has the same objectives, because the positions of macrocells are not pre-defined.

1.4.2 Routing of Signal Wires

After all the components are placed, the routing step finds a layout of wires to interconnect these components. The routing usually consists of two steps : global routing and detailed routing. Global routing determines which

channels each wire goes through, and detailed routing determines the path of every wire precisely in each channel. This approach is effective because it decomposes the routing problem into a number of smaller problems that are easier to solve.

There are two main types of detailed routers : maze routers and line routers. A maze router uses a breadth-first search to route the nets one at a time. The advantage of a maze router is that it always finds a connection if the solution exists. The disadvantage is that if many nets are routed by a maze router, the result depends on the order in which the nets are routed.

A line router finds a path between two points by constructing a sequence of line segments from each point. A connection path is found when the two sequences intersect. A line router is faster than a maze router, but has the disadvantage that it does not guarantee finding a path even if it exists. There is also the same disadvantage as a maze router that the result of a line router depends on the order in which the nets are routed since nets are routed one at a time.

The above detailed routers are general-purpose routers that can be used for connecting pins in any situation. However, if we know what type of detailed routing problem must be solved, we can use problem-specific detailed routers which perform better than general-purpose routers. There are two main types of problem-specific routers : channel routers and switchbox routers.

A channel router generates a routing pattern for a channel with pins on the top and the bottom of the channel as shown in Figure 1.3. The pin positions on both sides of the channel are usually fixed. The shape of a channel is usually a rectangle. The left and right sides are open, and wires are allowed to enter from the left and the right sides. The height of a channel

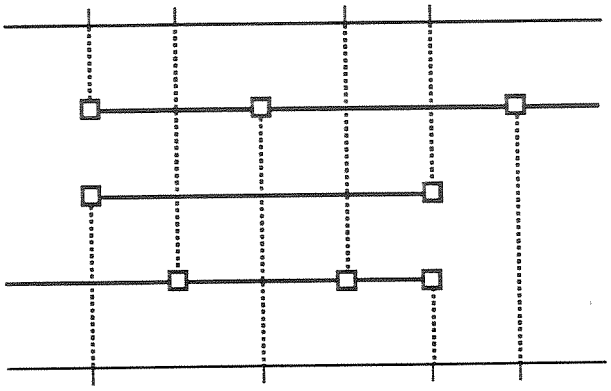


Figure 1.3: Example of Channel Routing

in a gate array design is fixed, so there exist cases where a channel router cannot find a solution. However, the height of a channel in a standard cell design or a macrocell design can be changed. So it is always possible to find a channel routing solution for a standard cell design or a macrocell design by adjusting the channel height. Since the nets are not routed one at a time, but all at once, there is no problem of the order in which nets are routed. The objectives of a channel router are minimizing the channel area, minimizing the number of vias (connecting holes between two layers), and minimizing the wire length. Minimizing the channel area improves the yield of a chip as described in Section 1.2. Minimizing the number of vias also improves the yield and the reliability. Wire length minimization helps shorten the delay time of signals through wires, which improves the speed of a chip. Channel routers are so powerful that they are used in most computer-aided design systems.

A switchbox is a rectangular routing region with pins on all four sides. A switchbox router routes wires in the switchbox whose shape is fixed. See Figure 1.4 for an example of switchbox routing. The shape is fixed, so that it is not guaranteed that the routing will always succeed. The main objective of a switchbox router is a 100% routing completion. Additional objectives are

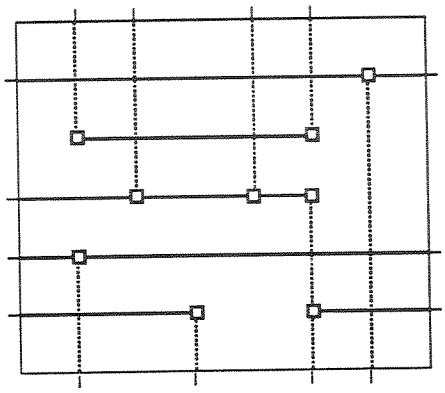


Figure 1.4: Example of Switchbox Routing

minimizing the number of vias and the wire length. Since the dimensions of a switchbox are fixed, the objectives do not include minimizing the switchbox area.

1.4.3 Routing of Power Wires

Since transistors cannot function without a power supply, all the cells need VDD and GND wires. For gate array designs and standard cell designs, cells are placed to form horizontal rows. So VDD and GND power connections can be realized by running two horizontal wires through the row.

For a macrocell design, all the macrocells need to be powered. VDD and GND wires must reach every macrocell. VDD wires and GND wires usually form a tree whose root is a power pad and whose branches reach macrocells. If two layers are available for power supply, one layer is used for laying out the VDD wire tree and another layer for the GND wire tree. However, if only one layer is available, a VDD tree and a GND tree must be laid out on one layer without any overlap. Vias are avoided because of their high impedance. Compared with signal wire routing, there are additional constraints to be considered for power wire routing : metal migration and voltage drop.

Metal migration is a phenomenon where metal destroys itself if an excessive current is passed though the wire. Metal migration can be prevented by having wide enough metal wires. Voltage drops must be kept small, because a large voltage drop between a pad and cells decreases switching speeds and noise margins. The voltage drop can also be reduced by having wide metal wires.

1.5 Outline of Dissertation

In Chapter 2, we show previous approaches to the channel routing and power routing problems. In Chapters 3 and 4, we describe our new channel router called topological channel router, followed by its experimental results in Chapter 5. In Chapter 6, we describe our new area-efficient power router.

Chapter 2

Previous Work

2.1 Previous Work on Channel Routing

In this section, we describe previous algorithms for channel routing for VLSI. The following algorithms all assume that there are two wiring layers available. One layer is used for vertical wire segments and another layer is used for horizontal wire segments. Vias are used to connect vertical wire segments to horizontal wire segments.

2.1.1 Left-Edge Channel Router by Hashimoto and Stevens

The left-edge algorithm by Hashimoto and Stevens [17] was the first attempt to find a channel routing solution. Each net is considered as an interval. Nets are processed in increasing order of the lower bounds of their intervals. A net is assigned to the first track if its lower bound is greater than the largest upper bound of nets previously assigned to this track. When no more nets can be assigned to the first track, the process is repeated for the second track, and then for subsequent tracks until no nets remain unassigned.

This algorithm is very simple but cannot solve the channel routing problem shown in the left half of Figure 2.1, where a signal 1 pin is above a signal 2 pin at one column and a signal 2 pin is above a signal 1 pin at another column. This is because the algorithm assumes that each net cannot have more than one horizontal segment. This case in Figure 2.1 has a cycle in a vertical constraint graph, which is defined as follows : A vertical constraint graph is

a directed graph whose vertices represent nets and whose edges represent the relative positions of the horizontal wires of a net. When we look at one column of a channel, if there is a pin which belongs to a net n_1 on the top of a channel, and a pin which belongs to a net n_2 at the bottom of a channel on the same column, a horizontal wire of n_1 must be placed above a horizontal wire of n_2. So there is a directed edge from V_1 (which corresponds to net n_1) to V_2 (which corresponds to net n_2) in the vertical constraint graph.

This method produces a minimum track solution when there is no vertical constraint, i.e., the vertical constraint graph is null. However, if there are vertical constraints, the solution is often much worse than the optimum, and it is very dependent on the order in which nets are chosen.

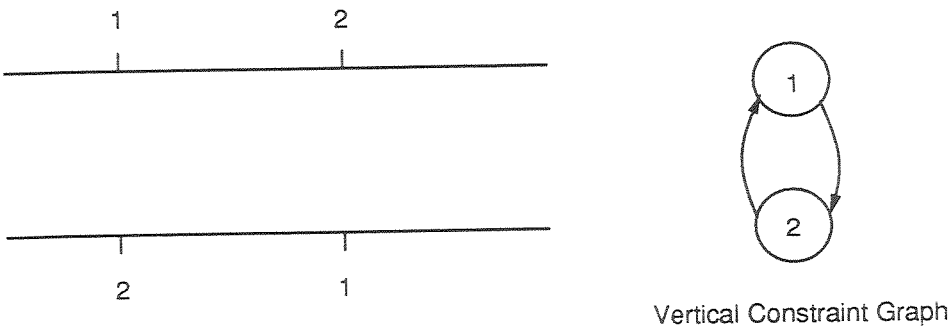


Figure 2.1: Channel Routing Problem whose Vertical Constraint Graph has a Cycle

2.1.2 Dogleg Channel Router by Deutsch

The Dogleg Channel Router by Deutsch [11] allows a net to be assigned to more than one track. The algorithm divides a net into subnets at terminal positions. This enabled the routing of a special problem shown in Figure 2.2. Even though there is a cycle in the vertical constraint graph in the original input, the cycle can be broken by splitting the signal 2 into 2 and

2'. The dogleg routing solution for this problem is shown at the right of Figure 2.2. Not only can the dogleg channel router solve some problems which the left-edge algorithm cannot solve, but it also produces better results than the left-edge algorithm. However, the dogleg channel router still cannot solve the problem shown in Figure 2.1. Even if a dogleg is used by dividing a net into subnets at terminal positions, this special channel routing problem cannot be solved because there are only two terminal positions for each net. The dogleg channel router still uses the left edge algorithm, so the dogleg solution is also very dependent on the order in which nets are chosen.

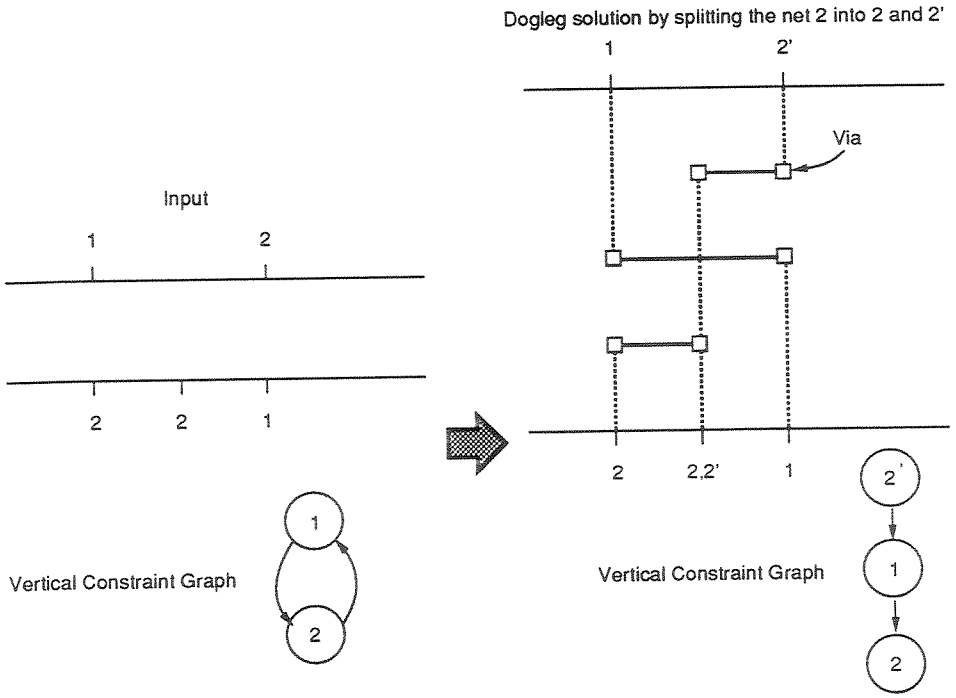


Figure 2.2: Dogleg Solution

2.1.3 Net-Merging Channel Router by Yoshimura and Kuh

The Net-Merging Channel Router by Yoshimura and Kuh [50] was developed in 1982. Since it is better to assign as many nets to one track as

possible, the net-merging channel router optimizes the track assignment before nets are actually assigned to tracks. They called the operation of assigning nets to the same track "merging nets". Since a vertex in a vertical constraint graph corresponds to a net, the net merging can be done by merging two vertices of the graph. The algorithm first builds a vertical constraint graph from the input. Merging of vertices is done so as to reduce the length of the longest path in the vertical constraint graph. At the end of the algorithm, each merged vertex represents one wiring track of the channel, i.e., all the nets in the vertex occupy the same track. Experiments show that the net-merging channel router gives better results than the left-edge channel router or the dogleg channel router. The advantage of the net-merging channel router is that it does not depend on the order in which nets are routed. However, the net-merging channel router also cannot solve the problem shown in Figure 2.1.

2.1.4 Greedy Channel Router by Rivest and Fiduccia

In 1982, Rivest and Fiduccia proposed a greedy channel router [33]. Their method scans a channel from left to right and routes one column at a time. Several heuristics are used for routing each column. The greedy channel router is the first that was able to find a solution for the channel routing problem of Figure 2.1, as shown in Figure 2.3.

The router is simple and easy to implement. However, it has the disadvantage that the router attempts to optimize routing at each column in a greedy manner, consequently creating situations where decisions made early may eventually increase the height of the channel.

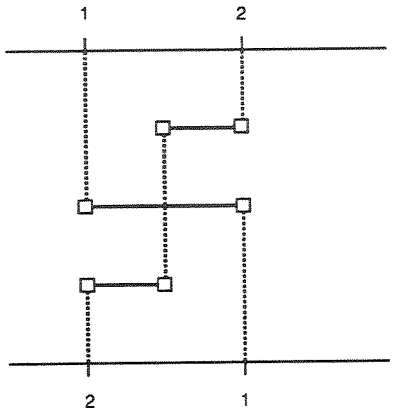


Figure 2.3: Solution for a Channel Routing Problem whose Vertical Constraint Graph has a Cycle

2.1.5 Hierarchical Channel Router by Burstein and Pelavin

The Hierarchical Channel Router was developed by Burstein and Pelavin [5] in 1983. It is based on reduction of the problem to the case of a $(2 \times n)$ grid and on consistent utilization of a "divide and conquer" approach. The channel is treated as an $(m \times n)$ grid. By grouping vertical grids, an $(m \times n)$ grid can be treated as a $(2 \times n)$ grid, i.e., two horizontal strips. An algorithm to find the minimum length Steiner Tree for a $(2 \times n)$ grid is known [1]. After the tree is obtained, each of the horizontal strips is partitioned into two $(2 \times n)$ subproblems. This is done recursively until a single cell resolution is reached.

The Hierarchical Channel Router was the first channel router that was able to find a 19 track solution for Deutsch's difficult example [11]. The Dogleg Channel Router produced a solution with 21 tracks, The Net Merging Channel Router produced a solution with 20 tracks, and the Greedy Channel Router also produced a solution with 20 tracks.

2.2 Previous Work on Power Routing

As explained in the introduction, for gate array designs and standard cell designs, power wires run through horizontal gate cells, so there is no need to use a sophisticated power routing algorithm. A typical power wire pattern is shown in Figure 2.4.

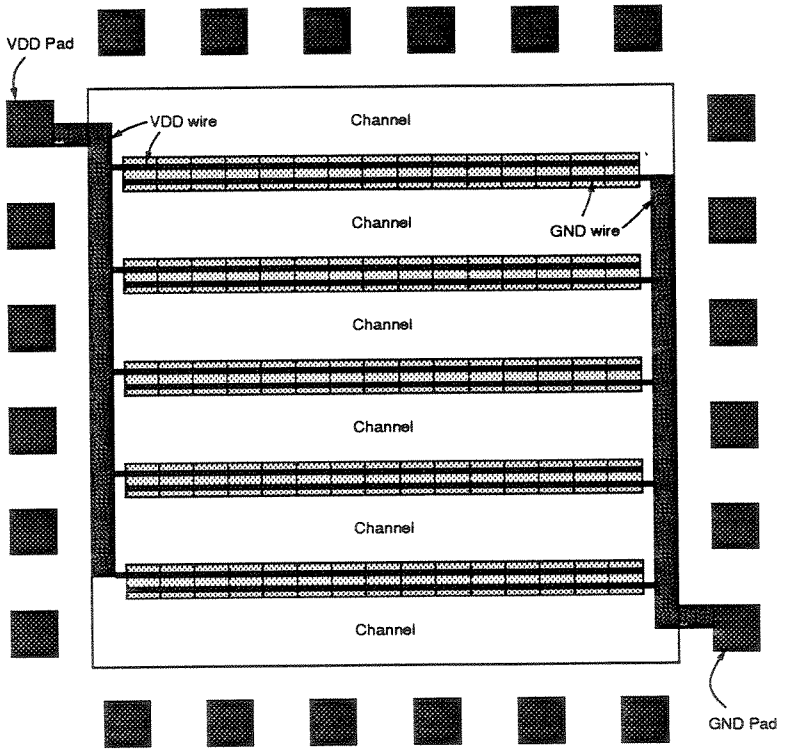


Figure 2.4: Typical Power Routing for a Gate Array Design Style or a Standard Cell Design Style

For a macrocell design, we must find a power wire connection to all the macrocells. The macrocells are arbitrarily placed on a chip. So there is a need for a more sophisticated power wire router to connect all the macrocells to VDD and GND pads. All the previous algorithms of power routing described below are concerned with macrocell design.

2.2.1 Left-Right Power Routing by Rothermel and Mlynski

In left-right power routing by Rothermel and Mlynski [34], one tree extends from the left edge of the chip and the other from the right edge. This strategy, however, imposes the restriction that the VDD and GND pads have to be on opposite sides of the chip. The power trees whose edge widths are zero are formed without considering wire width first, and the edges of the trees are thickened later based on their electrical current requirements. This method has the disadvantage that even if trees can be constructed, there is a case where the thickening fails because there may not be enough routing space between macrocells.

2.2.2 Top-left Bottom-right Power Routing by Lie and Horng

In top-left bottom-right power routing by Lie and Horng [26], a restriction on the positions of pins is imposed such that the VDD pin of each macrocell must be either on the top or left side, and the GND pin on the bottom or right side as shown in Figure 2.5. This restriction makes the VDD and GND tree routing simple. However, because of this restriction, macrocells with pins at arbitrary locations cannot be used. This method has a fixed pattern of wires so that even if power trees can be constructed, the thickening of power wires may fail because of insufficient routing area.

2.2.3 Scan-line Power Routing by Xiong and Kuh

Other approaches avoid such pin and pad placement restrictions. In scan-line power routing by Xiong and Kuh [51], as a scan line sweeps across a layout, the topology of two trees is determined simultaneously. This is an efficient algorithm since the two-dimensional layout problem is solved by suc-

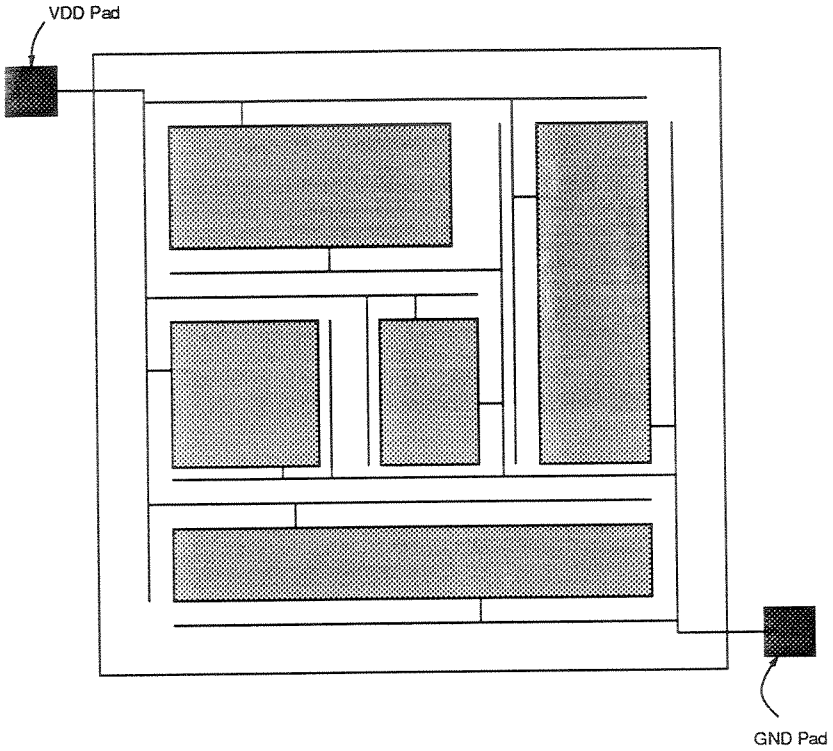


Figure 2.5: Top-left Bottom-right Power Routing by Lie and Horng

cessively solving one-dimensional problems on each scan line. This algorithm does not, however, consider power wire width, i.e., it is only concerned with the routing completion of two trees with zero width on one layer. Even though the algorithm imposes no restriction on pin and pad restrictions, the algorithm does not try to optimize routing parameters such as total wire length or area.

2.2.4 Hamiltonian Power Routing by Moulton

Some existing algorithms try to minimize the total wire length with no restrictions on pad or pin positions. In Hamiltonian power routing by Moulton [29], a graph representing the placement is constructed, and a Hamiltonian cycle in the graph separating the power and ground pins is selected. VDD

pins and a VDD pad are connected inside the cycle to form a VDD tree. The Hamiltonian cycle is deleted and GND pins and a GND pad are connected to form a GND tree, thus avoiding a crossing of trees. The channel width is increased when there is insufficient space to place the power wires.

2.2.5 Minimum Wire Length Power Routing by Russell

The minimum wire length power routing by Russell [36] also has no restrictions on pad or pin positions. A routing graph is constructed from input representing the macrocell placements, and nets are routed using a shortest path algorithm, which tries to minimize the wire length of multiple trees. However, the widths of wires are not taken into consideration. This algorithm can handle more than two power trees. When a wire blocks another wire to be connected to a macrocell, the wire is removed and later re-routed. Even with the rip-up and re-route method, there are cases when multiple (more than two) trees cannot be topologically realized on one layer. In those cases, the tree jumps to another layer and comes back to the original layer in order to jump blocking wires.

2.2.6 Minimal Area Power Routing by Chowdhury

In minimal area power routing by Chowdhury [9], an algorithm which attempts to find VDD and GND trees with minimal area under constraints such as metal migration and voltage drop is presented. However, the topology of the layout is assumed to be given or manually constructed, so that the problem of finding non-crossing trees on one layer is avoided. It was found in an experiment that minimal length power trees may not lead to minimum power routing area, i.e., even if the total wire length of two different solutions of VDD and GND

trees are the same, the areas may be different.

Chapter 3

Topological Channel Routing

3.1 Introduction

The objective of a channel router is to lay out all the nets in the smallest possible area using a small number of vias. Most channel routers, including those in Section 2.1, tend to use a large number of vias because they put all horizontal wires on one layer and all vertical wires on the other layer, and horizontal wires and vertical wires are connected by vias. This is called the reserved layer model. By allowing both horizontal and vertical wires on each layer, we may be able to obtain results with smaller numbers of vias and smaller area. Reducing the number of vias is important, because it helps increasing the yield and the reliability, and improving the performance of a chip.

There are two ways to reduce the number of vias : constrained via minimization (CVM) and unconstrained via minimization (UVM). CVM is done as a post-processing step after a reserved layer channel router completes routing, i.e., given the location of wire segments and vias, find a layer assignment for the wire segments using a minimum number of vias. See Figure 3.1 for an example of CVM. CVM does not change the topology of the given layout. Instead, only the layer assignment of wires is re-defined. After CVM is performed, the reserved layer model becomes a non-reserved layer model, because each layer allows to have both vertical and horizontal wire segments. CVM was first proposed by Hashimoto and Stevens [17]. In 1983, Chen, Kajitani, and Chan showed that the CVM problem can be solved in polynomial time

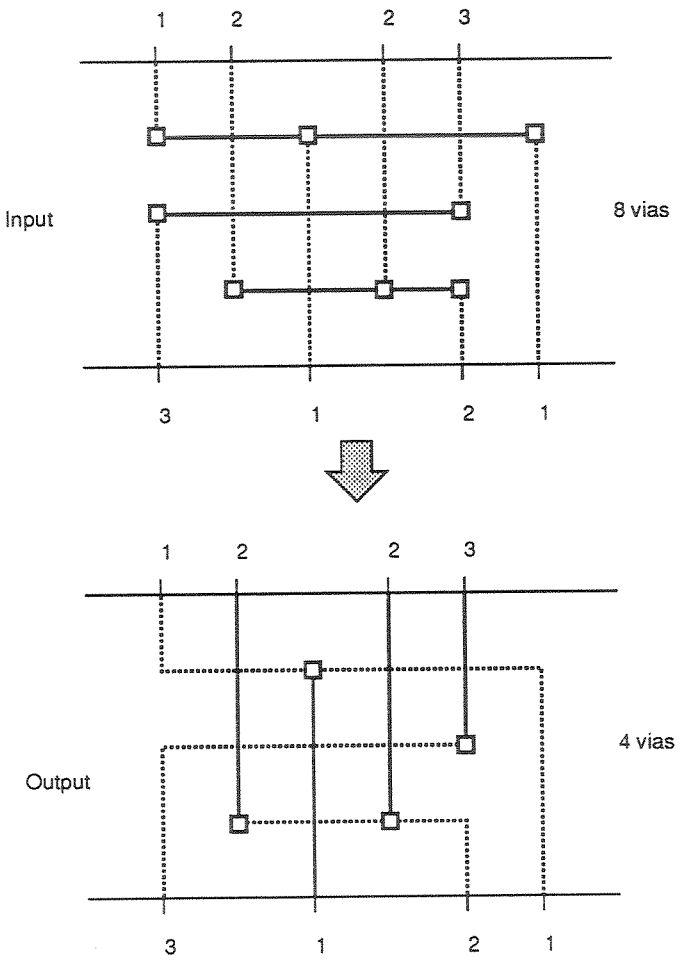


Figure 3.1: Example of Constrained Via Minimization

under certain constraints [6]. Since any good reserved layer channel router can be used before CVM post processing, CVM is useful in reducing the number of vias after a compact routing solution is obtained.

Unconstrained via minimization (UVM), on the other hand, determines routing topology as well as layer assignments of wires. Figure 3.2 is an example of UVM. After the topology is determined, geometric mapping is needed to get an actual routing solution. UVM was first introduced by Hsu [18] [19] [20]. It determines how each net crosses others topologically without im-

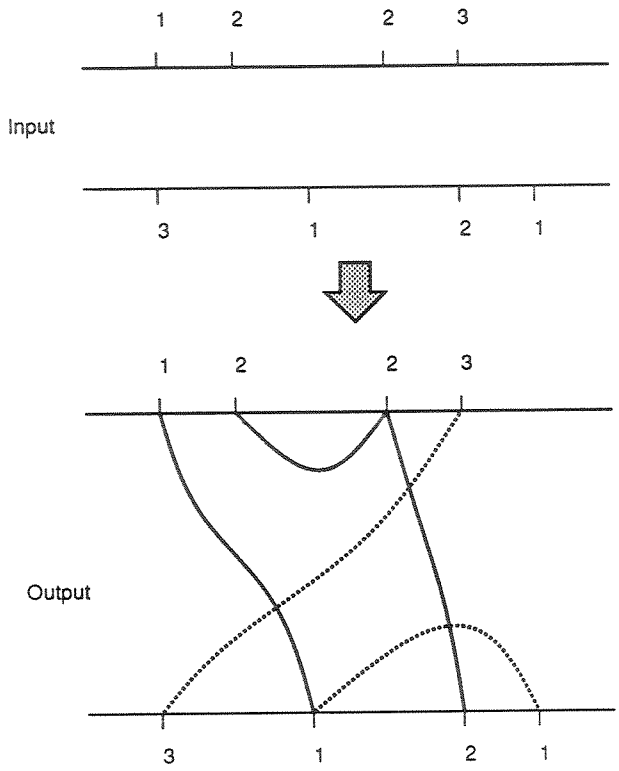
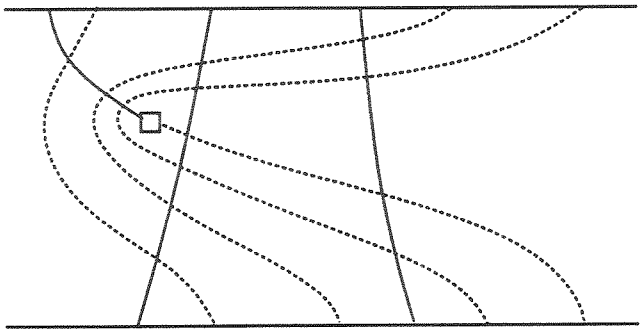


Figure 3.2: Example of Unconstrained Via Minimization

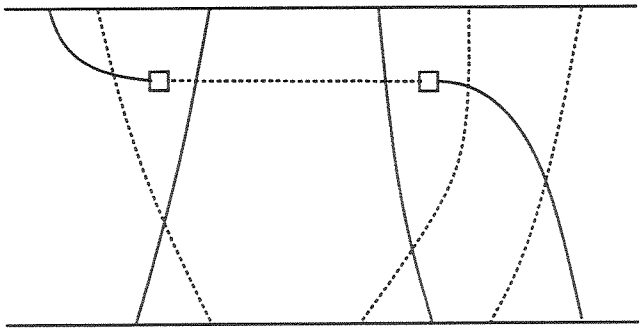
posing any constraints on the physical wire positions. UVM has more degrees of freedom than CVM in terms of minimizing vias because the physical locations of wires are not pre-determined in UVM. However, this freedom makes UVM more difficult. Hsu [18] conjectured that UVM was NP-hard, which was later proved by Sarrafzadeh and Lee [38].

Historically, the main purpose of UVM is to minimize the number of vias. Even though such methods produce results using very small numbers of vias, nets are frequently forced to meander around vias [27], which results in an increase in area (See Figure 3.3).

We propose a new channel routing algorithm [15] which minimizes both the area and the number of vias by using a new layout model in which both



Nets meander around a via if the number of vias is minimized



A less meandering solution

Figure 3.3: Meandering of Nets

directions of wire segments are allowed to be placed on both layers. Our unique layout model utilizes the channel area more efficiently than the traditional layout model while avoiding the cross talk problem. Our algorithm attempts to minimize the number of vias while avoiding the meandering of nets.

3.2 Layout Model

We assume that two layers are available for routing, and design rules are the same for both layers. Both vertical and the horizontal wire segments are allowed on both layers. However, the parallel overlapping of long wire segments on different layers is undesirable because the high capacitive coupling associated with the long wire segments cause a cross talk. The solution we have adopted is an interleaving wire model as shown in Figure 3.4.

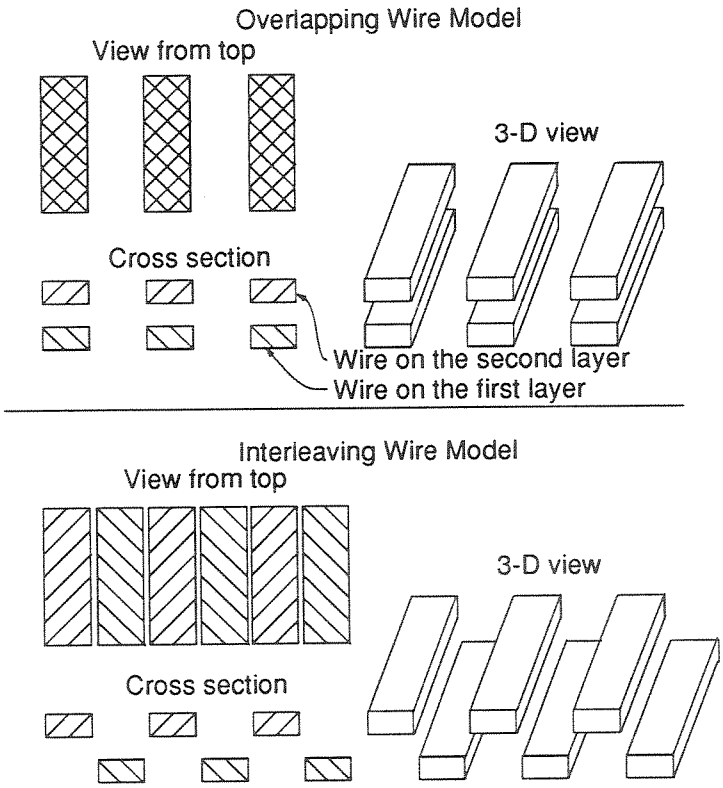


Figure 3.4: Layout Models

We begin by calculating the capacitance between a wire on the first layer and a wire on the second layer when the wire width is 2 μ, the wire thickness is 1 μ, the wire separation on the same layer is 2 μ, and the thickness of the insulator ($\epsilon = 3.9$) between the two layers is 1 μ. See Figure 3.5 for the

calculation conditions. The capacitance C_{12} between a wire on the first layer

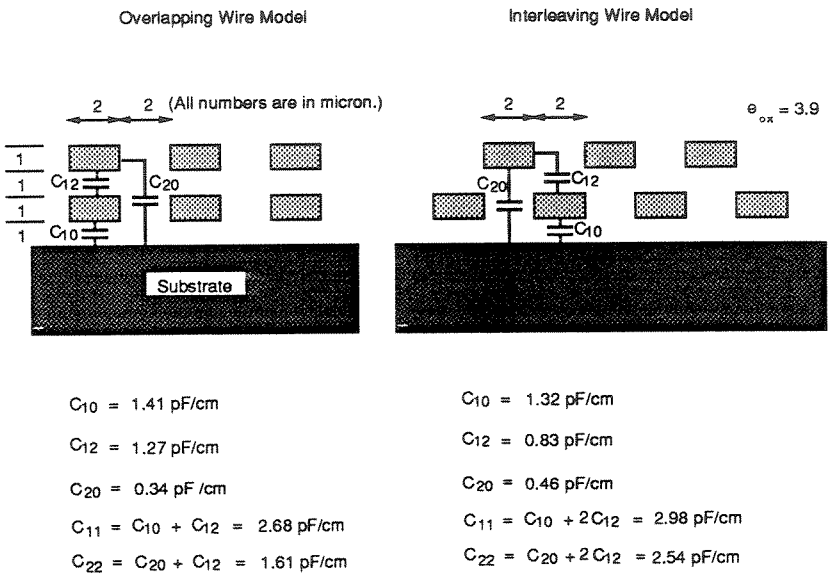


Figure 3.5: Capacitive Couplings Between Wires

and a wire on the second layer causes cross talk between them. The calculated capacitance C_{12} in the interleaving wire model is 0.83 pF/cm, whereas C_{12} in the overlapping wire model is 1.27 pF/cm. Thus the interleaving wire model has a 35% smaller capacitive coupling.

The interleaving wire model is used for both the vertical and the horizontal wire segments. By using solid lines to represent the allowed locations of wire segments on the first layer and dotted lines to represent the allowed locations of wire segments on the second layer, the channel can be represented by a mesh called an interleaving mesh as shown in Figure 3.6. The only overlap between the two wires on the interleaving wire model is at the 90 degree crossing of two wires on different layers. Since vertical wire segments are allowed on both layers, we assume that all terminals can be reached from both layers.

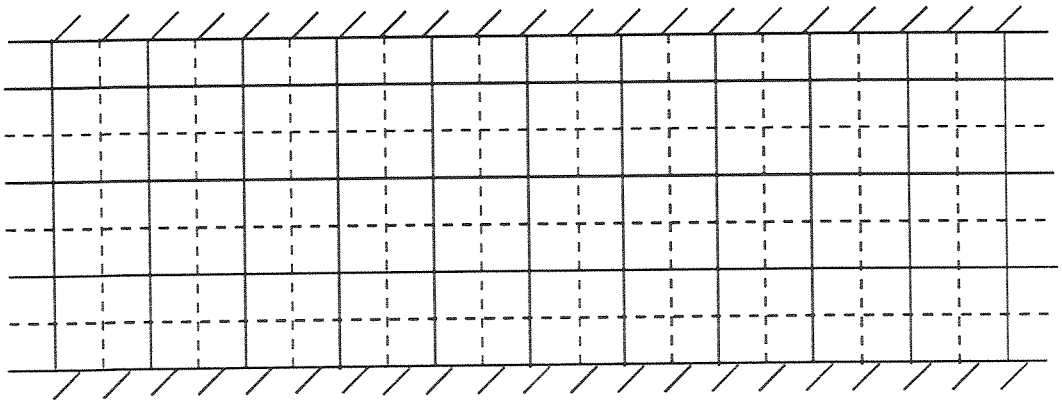


Figure 3.6: Interleaving Mesh

3.3 Topological Channel Routing Algorithm

We now present the basic ideas of our algorithm (See Figure 3.7). Our five-stage algorithm consists of decomposition of multi-terminal nets, assignment of nets to the two layers, construction of a topological graph, mapping of the topological graph onto the channel, and channel compaction.

First, the multi-terminal nets are decomposed into two-terminal nets. Then, to have as many nets without vias as possible, the two-terminal nets are classified into three types: nets without vias on the first layer, nets without vias on the second layer, and nets with vias. In the third stage, it is determined how each net runs through the others. This stage is accomplished by constructing a graph called a topological graph whose vertices explicitly represent terminals, vias, or crossings of the nets. The fourth stage determines the physical locations of the nets by mapping the topological graph onto the interleaving mesh. Finally, in the last stage, compaction of the channel takes place. In the following subsection, details of each stage are described.

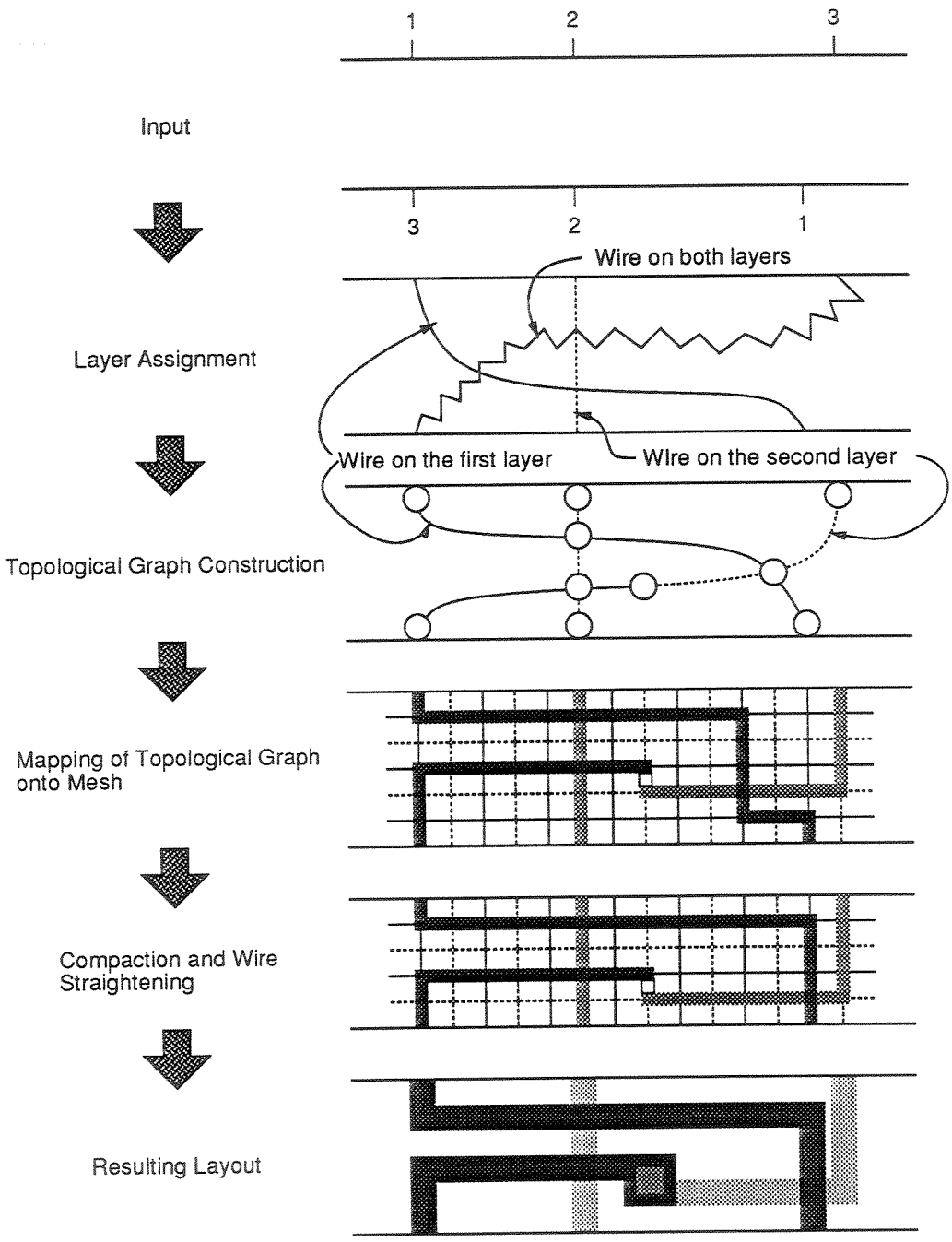


Figure 3.7: Overview of Topological Channel Routing Algorithm

3.4 Details of Algorithm

3.4.1 Decomposition of Multi-Terminal Nets

A net must be able to have a connection to any number of terminals. The algorithm used in the next subsection requires all nets to be two-terminal nets. Therefore, multi-terminal nets have to be decomposed into two-terminal nets. If a net has t terminals ($t > 2$), it is decomposed into $(t-1)$ two-terminal nets. We use two methods for decomposition. One method of decomposition is to form a net with two terminals whose horizontal separation is small. The other method is to find a minimum spanning tree of $(t-1)$ edges. Manhattan distances are used between pins on the top wall and pins on the bottom wall. Since the height of a channel is not fixed at this stage, we must predict the height before nets are routed. The estimated height H_{est} is obtained by the following equation :

$$H_{est} = d(w+s) + s,$$

where d is the channel density, which is the maximum number of nets that cross a vertical cut line at any position in a channel, w is the wire width, and s is the wire spacing. Note that H_{est} is actually the minimum height achievable by a reserved layer channel router.

Both methods are incorporated in our current implementation. From now on, without loss of generality, we will assume that all nets have only two terminals.

3.4.2 Assignment of Nets to Layers

It is clear that a set of non-intersecting nets can be placed on one layer without using any vias. No vias are needed for two such sets, because

one set can be placed on the first layer and the other set on the second layer. Vias are needed only by those nets which are not in those two sets. Therefore, it is desirable to assign as many nets without using any vias to the two layers as possible. Our algorithm for assigning nets to the two layers is based on the following algorithm which finds a maximum set of non-intersecting nets.

A graph is constructed from the intersection information of nets, where each vertex i represents net i and there is an edge between vertex i and vertex j if and only if net i and net j intersect. The constructed graph is a circle graph. The problem of finding a maximum set of non-intersecting nets is equivalent to the problem of finding a maximum independent set of a circle graph, where an independent set of a graph is defined as a subset of its vertices of which no two are joined by an edge, and a maximum independent set of a graph is defined as an independent set of a graph whose cardinality is maximum among different independent sets of a given graph. It is known that the maximum independent set problem for an arbitrary graph is NP-hard [23]. Fortunately an $O(n^2)$ time dynamic programming algorithm by Supowit [45] is known for circle graphs where n is the number of vertices (= the number of nets). The details of the algorithm will be described below.

The input contains a set W of T two-terminal nets. See Figure 3.8. Let the pin numbers be $0, 1, ..., 2T-1$ in clockwise order ($T = 8$ in the case of Figure 3.8). Let $n_{p,q}$ denote a net which connects pin p and pin q and $p < q$.

We construct a graph in which each vertex $v_{p,q}$ corresponds to a net $n_{p,q}$, and there is an edge in E for each pair of vertices whose corresponding nets intersect. Thus, $G = (V, E)$ is a circle graph. Let $G_{i,j}$ denote the subgraph of G, with vertices $V_{i,j} = \{v_{p,q} : i \leq p, q \leq j\}$ and edges $E_{i,j} = \{(v_1, v_2) \in E : v_1, v_2 \in V_{i,j}\}$.

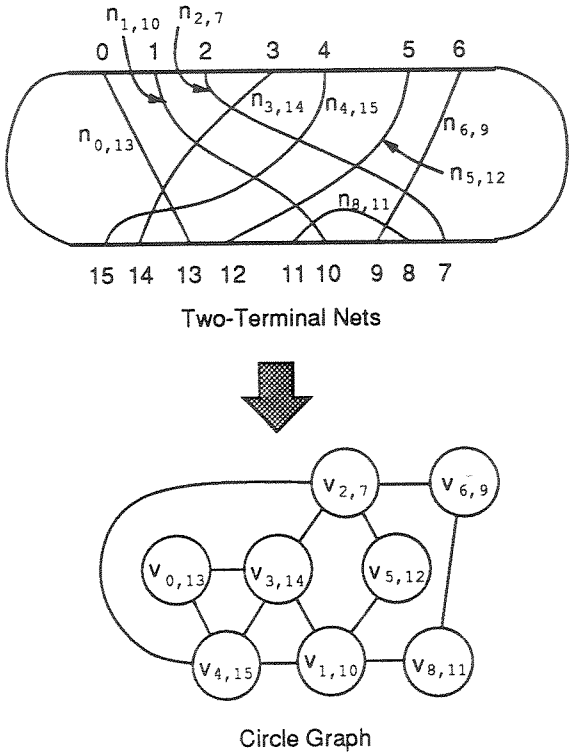


Figure 3.8: Construction of Circle Graph

Let $MIS(i,j)$ denote a maximum independent set of $G_{i,j}$. $MIS(i,j)$ can be calculated using a dynamic programming method as described below. Let k be a pin number such that $n_{j,k} \in W$ or $n_{k,j} \in W$. There are two cases to consider (See Figure 3.9).

If k is not in the range $[i, j-1]$ (Case 1 in Figure 3.9), then $G_{i,j} = G_{i,j-1}$, because a net $n_{j,k}$ (or $n_{k,j}$) is not a member vertex of $G_{i,j}$. So, $MIS(i,j) = MIS(i,j-1)$.

If k is in the range $[i, j-1]$ (Case 2 in Figure 3.9), then there are two sub-cases to consider. One sub-case is that $v_{k,j}$ is a member of $MIS(i,j)$. If so, $MIS(i,j)$ contains no vertices $v_{p,q}$ such that $p \in [i, k-1]$ and $q \in [k+1, j-1]$.

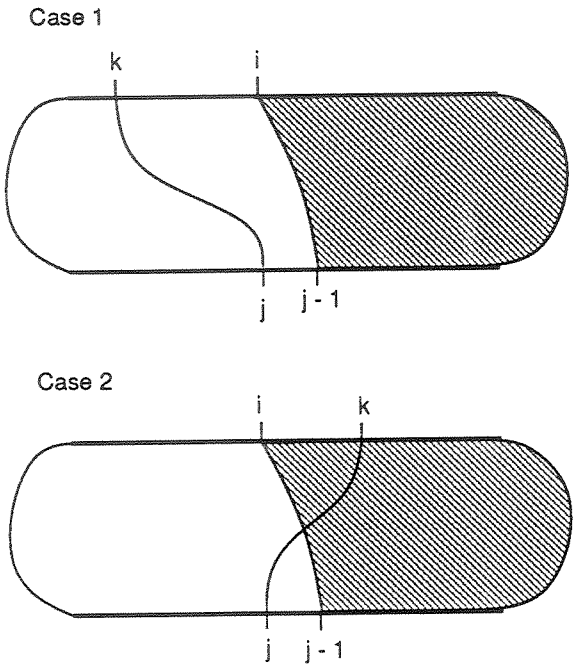


Figure 3.9: Two Cases of the Positions of pin k

So in this sub-case,

$$MIS(i,j) = MIS(i,k-1) \cup \{v_{k,j}\} \cup MIS(k+1,j-1).$$

The other sub-case is that $v_{k,j}$ is not a member of $MIS(i,j)$. In this sub-case,

$$MIS(i,j) = MIS(i,j-1).$$

So, $MIS(i,j)$ is the larger of $MIS(i,k-1) \cup \{v_{k,j}\} \cup MIS(k+1,j-1)$ and $MIS(i,j-1)$.

For both Case 1 and Case 2, the calculation of $MIS(i,j)$ needs only the information of $MIS(i,k-1)$, $MIS(k+1,j-1)$, and $MIS(i,j-1)$, where $i \leq k \leq j-1$. Thus $MIS(0,2T-1)$ can be obtained by computing and storing the results of

$$MIS(0,1),$$

$MIS(0,2), MIS(1,2),$

$MIS(0,3), MIS(1,3), MIS(2,3),$

$MIS(0,4), MIS(1,4), MIS(2,4), MIS(3,4),$

...,

$MIS(0,2T-2), MIS(1,2T-2), ..., MIS(2T-3,2T-2),$

$MIS(0,2T-1)$ in this order.

A maximum independent set of G is $MIS(0,2T-1)$, because $G = G_{0,2T-1}$.

Our algorithm for assigning nets to the two layers is as follows. A set N_1 of non-intersecting nets is first selected by the maximum independent set algorithm and placed on the first layer. The same algorithm of finding non-intersecting nets is then applied to the remaining nets to find a set N_2 of non-intersecting nets to be placed on the second layer. The remaining nets N_{12} have to use vias because each net in N_{12} intersects both nets in N_1 and nets in N_2.

3.4.3 Construction of Topological Graph

Since the layer assignment step only assigns the nets to the two layers, it is not yet decided how each net runs through other nets. Precisely how each net runs through the others is determined by constructing a graph called a topological graph whose vertices explicitly represent pins, vias, or crossings of nets, and where there is an edge between two vertices if and only if they are connected by a net (See Figure 3.7). We call a vertex that represents a pin, a via, and a crossing of two wires on different layers, "a pin vertex", "a via vertex", and "a cross vertex" respectively. See Figure 3.10 for the three types

of vertices. The degree of a pin vertex, a via vertex, and a cross vertex is 1,

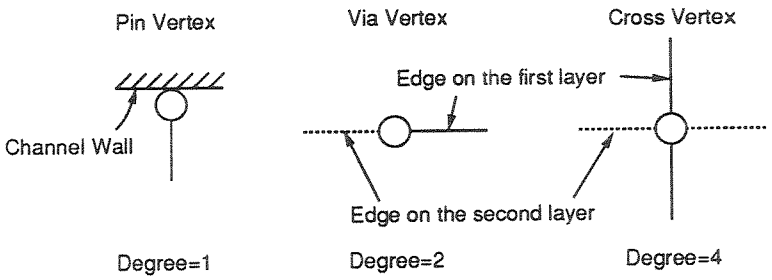


Figure 3.10: Three Vertex Types of a Topological Graph

2, and 4 respectively. There are two types of edges : an edge on the first layer (solid line in the topological graph of Figure 3.10) and an edge on the second layer (dotted line in the topological graph of Figure 3.10). After the graph is mapped to a channel, the solid edge corresponds to a wire on the first layer, and the dotted edge corresponds to a wire on the second layer.

In the topological graph construction, each net is topologically connected (i.e., each net finds which nets to cross). In [27], N_1, N_{12}, and N_2 are topologically connected in this order. This order ensures that each net in N_{12} uses only one via. However, by minimizing the number of vias, the approach in [27] requires many nets to meander around a via as shown in Figure 3.3. Each meandering net uses two or more tracks, which results in a larger channel area.

Instead of using only one via per net, our algorithm finds a less meandering solution in which each net in N_{12} may use more than one via. The less meandering solution for each net is obtained by topologically connecting the net in such a way to minimize the total number of crossings with other nets. In our method, nets of N_1, N_2, and N_{12} are routed in this order. If a net in N_{12} has more than one solution with the same number of crossings, the one with the minimum number of vias is chosen among them.

3.4.3.1 Algorithm for Finding a Path with a Minimum Number of Crossings

We want to construct a topological graph, which will later be mapped to a channel. When we search for a path of a net from one pin to the other pin with the minimum number of crossings with other nets, we need a data structure which is suitable for counting the number of crossings. For example, when we want to find a path from Start_pin to End_pin as shown at the top of Figure 3.11, it would be convenient if there were a graph in which each vertex represents a region and there is an edge between two vertices if and only if corresponding regions are adjacent as shown at the bottom of Figure 3.11. Then we can easily find a path of length 2 from *Start_region_vertex*

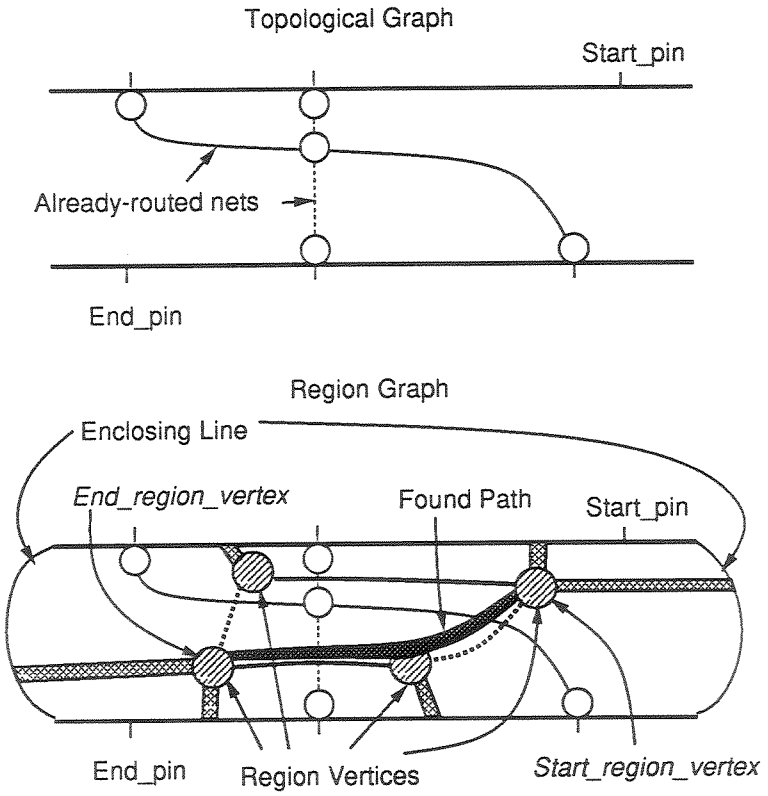


Figure 3.11: A Region Graph is Needed To Find a Path in a Topological Graph

to *End_region_vertex*, meaning that the path from Start_pin to End_pin crosses two nets. More formally a region graph is defined as follows. A vertex (called a region vertex) corresponds to a region surrounded by edges of a topological graph, a part of a wall, and enclosing line(s). An enclosing line is a line that connects the left end of the top wall to the left end of the bottom wall, and a line that connects the right end of the top wall to the right end of the bottom wall as shown in Figure 3.11. An edge between two region vertices of a region graph exists if two corresponding regions touch each other (separated by an edge of a topological graph). There are three types of region edges as shown in Figure 3.12. If an edge of a topological graph is on the first layer, the

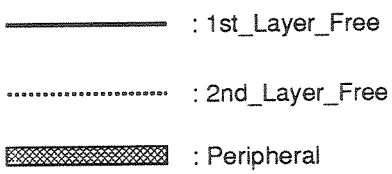


Figure 3.12: Three Types of Region Edges

corresponding region edge is called "2nd_Layer_Free", because a wire on the second layer can cross a wire on the first layer. The "2nd_Layer_Free" region edge is represented by a thick dotted line. Similarly, if a separating edge of a topological graph is on the second layer, the corresponding region edge is called "1st_Layer_Free", and it is represented by a thick solid line. An edge also exists between a region vertex and a wall and between a region vertex and an enclosing line. We call this type of edge "Peripheral" and it is represented by a wide band. Obviously, a topological graph and its region graph are both planar graphs and dual to each other, except that there is no "infinite vertex" in a region graph that corresponds to an infinite region, and there is no edge of a topological graph that corresponds to a Peripheral region edge.

Before any nets are routed, the region graph consists of only one region vertex and Peripheral region edges that connect the region vertex to a wall, as shown in Figure 3.13. The wall is considered as a sequence of wall

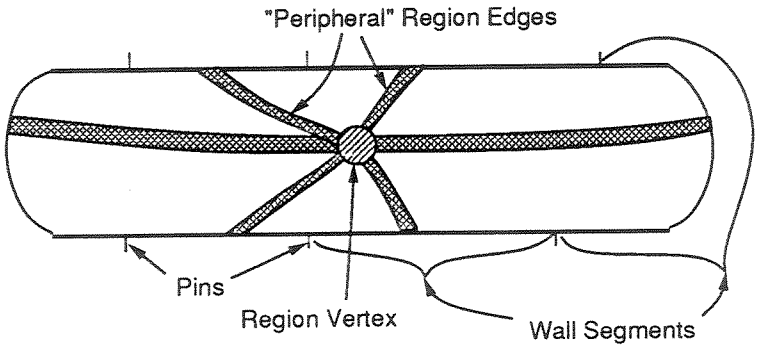


Figure 3.13: An Initial Region Graph before any Nets are Routed

segments. There is a Peripheral region edge to each wall segment. There is a pin between two adjacent wall segments. These Peripheral region edges are not essential in finding a path with minimum crossings, but are only needed to simplify the description of our algorithm.

The region graph is used for finding a path with a minimum number of crossings. Actually when nets are being connected, a topological graph is never constructed, but a region graph is constructed. After all the nets are connected, a topological graph is constructed using the information in a region graph. The topological graph must be constructed because that is the graph to be mapped to a channel.

When we want to find a connection from Start_pin to End_pin in Figure 3.11, we first search for *Start_region_vertex* and *End_region_vertex* which touch Start_pin and End_pin respectively. Note that a shortest path from *Start_region_vertex* to *End_region_vertex* corresponds to a connection from Start_pin to End_pin with minimum crossings. If there are many paths of

the same shortest length, we want to choose the one among them which uses the minimum number of vias. We developed a modified shortest path algorithm [10] to handle this problem, and the algorithm is described as follows.

Each region vertex has three parameters needed for a breadth first search : d, v_1, and v_2. "d" is the distance from a *Start_region_vertex*. "v_1" is the number of vias needed to reach the first layer of the region vertex from a *Start_region_vertex*. "v_2" similarly is the number of vias needed to reach the second layer of the region vertex from a *Start_region_vertex*. Note that if v_1 vias are needed to reach the first layer of a certain region vertex, reaching the second layer of the same region vertex can be achieved by just jumping the layer through a via. So, v_2 is either $v_1 + 1$, v_1, or $v_1 - 1$.

These parameters of all the region vertices are initially set to ∞ except the *Start_region_vertex* whose parameters are set to 0 (Figure 3.14). The

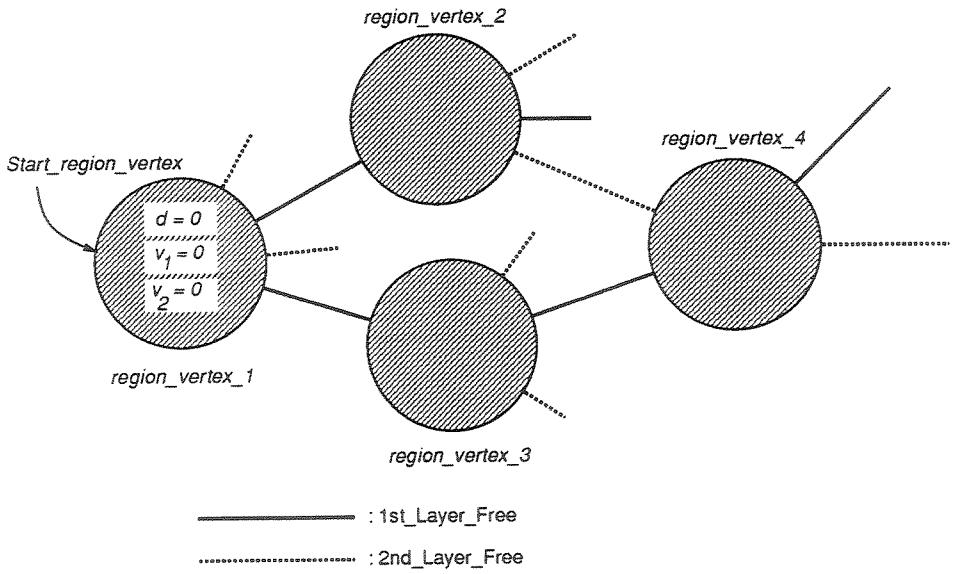


Figure 3.14: Start of Algorithm for Finding a Path with a Minimum Number of Crossings

breadth first search starts from the *Start_region_vertex*. For each region vertex

of the same distance, all the adjacent region vertices are checked. v_1 and v_2 are determined as follows. If the adjacent region vertex is connected by a region edge of 1st_Layer_Free, this means that a path on the first layer from the region vertex to its adjacent vertex can go through the region edge freely without using any via. So $adj\_vertex.v_1$ is set to min ($region\_vertex.v_1$, $adj\_vertex.v_1$). Thus, $adj\_vertex.v_1$ stays the same if $region\_vertex.v_1 > adj\_vertex.v_1$, which means that there is already a path to this adjacent vertex found with a smaller number of vias. The second layer of this adjacent region vertex can be reached just by jumping the layers as described above. So, $adj\_vertex.v_2$ is set to min ($region\_vertex.v_1 + 1$, $adj\_vertex.v_2$). A similar procedure is applied when the adjacent region vertex is connected by a region edge of 2nd_Layer_Free.

Figure 3.15 shows the parameters of region vertices after region vertices of distance $d = 1$ are visited. Figure 3.16 shows a snapshot when

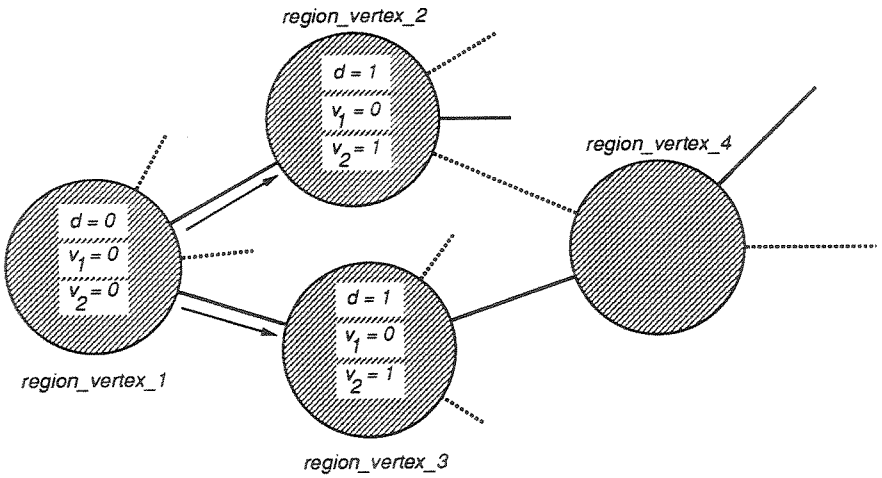


Figure 3.15: Snapshot after Region Vertices of $d = 1$ are Visited

$region\_vertex\_4$ is visited from $region\_vertex\_2$, but not from $region\_vertex\_3$ yet.

v_2 of $region\_vertex\_4$ is set to 1, which is the same as v_2 of $region\_vertex\_2$

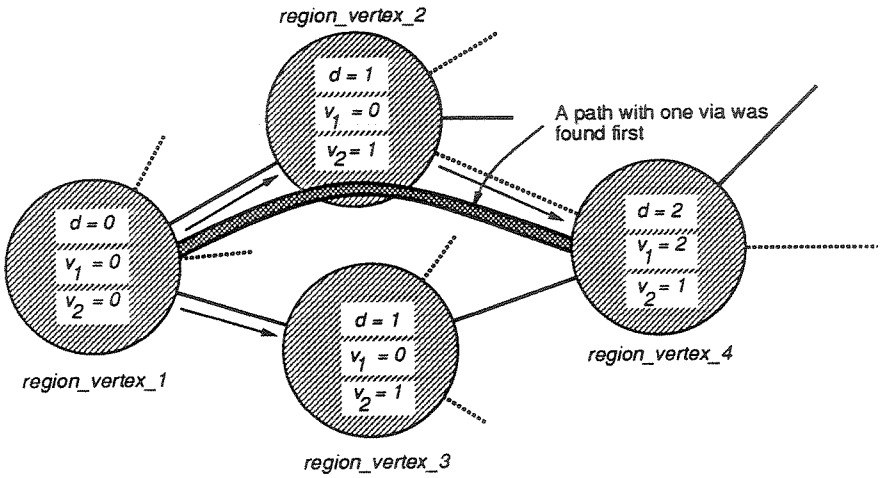


Figure 3.16: Snapshot when *region_vertex_4* is Visited from *region_vertex_2*

because *region_vertex_2* and *region_vertex_4* are connected by a region edge of 2nd_Layer_Free. v_1 of *region_vertex_4* is set to $1+1=2$. Thus a path of distance $d=2$ from *region_vertex_1* (= *Start_region_vertex*) to *region_vertex_4* using one via is found at this time. Figure 3.17 shows a snapshot when *region_vertex_4* is visited from *region_vertex_3*. A path from *region_vertex_3*

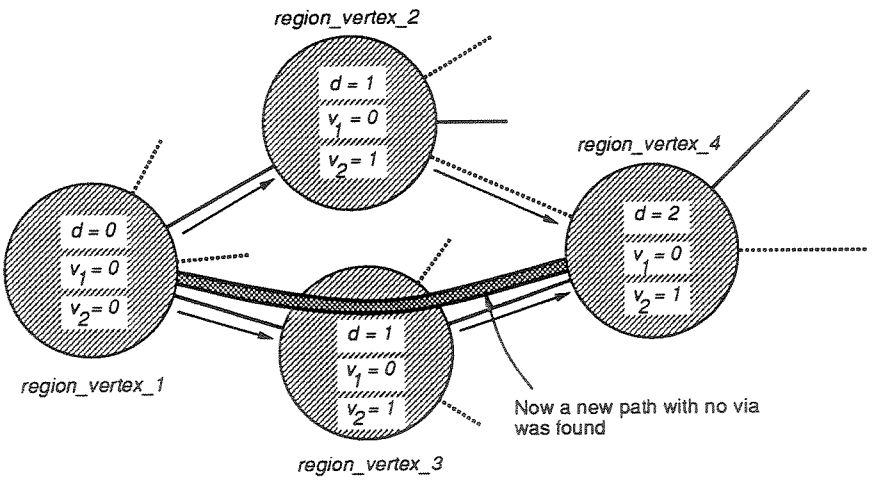


Figure 3.17: Snapshot when *region_vertex_4* is Visited from *region_vertex_3*

can reach *region_vertex_4* without any via through a region edge of

1st_Layer_Free. So, v_1 of *region_vertex_4* is set to v_1 of *region_vertex_3*, which is 0. Now a new path of the same distance $d = 2$ with no via through *region_vertex_3* replaces an old path with one via through *region_vertex_2*. This breadth first search continues until it reaches an *End_region_vertex*.

After an *End_region_vertex* is reached, the region graph is traced back from the *End_region_vertex* to the *Start_region_vertex*. The trace back procedure checks the region vertices on the shortest path and determines on which layer the path runs, based on the information of the parameters of v_1 and v_2. Each region vertex must keep pointers of which vertex and edge were the previous vertex and edge on the shortest path. So parameters of prev_vertex_on_1st_layer and prev_vertex_on_2nd_layer are kept in each region vertex. If there were no such parameters, it would be impossible to trace back from *region_vertex_4* in Figure 3.17, because, from *region_vertex_4*, only *region_vertex_2* and *region_vertex_3* can be seen, and the trace back procedure would not be able to see that a path through *region_vertex_2* needs to use a via. Parameters of prev_edge_on_1st_layer and prev_edge_on_2nd_layer are also kept at each region vertex, because there is sometimes more than one region edge between two region vertices, and it cannot tell which edge is on the shortest path without this information.

Pseudo-code to find a shortest path with minimum number of vias is shown in Figure 3.18, and pseudo-code of a procedure to set parameters of adjacent region vertices is shown in Figure 3.19.

```
procedure find_shortest_path_with_minimum_number_of_vias

(* reset region_vertex parameters *)
for all region_vertex
begin
   region_vertex.d = ∞;
   region_vertex.v1 = ∞;
   region_vertex.v2 = ∞;
end
(* set Start_region_vertex parameters *)
Start_region_vertex.d = 0;
Start_region_vertex.v1 = 0;
Start_region_vertex.v2 = 0;
(* start breadth first search from Start_region_vertex *)
current_distance = 0;
while End_region_vertex.d ≠ current_distance
begin
   for all region_vertex whose d is current_distance
   begin
      set_parameters_of_adjacent_vertices; (* see Figure 3.19 *)
   end
   current_distance = current_distance + 1;
end
trace_back_the_shortest_path;
```

Figure 3.18: Pseudo-code to Find a Shortest Path with Minimum Number of Vias

```
procedure set_parameters_of_adjacent_vertices

for all the adj_vertex which are adjacent to region_vertex
begin
  if adj_vertex.d ≥ current_distance +1
  begin
    adj_vertex.d = current_distance +1;
    if type of region_edge to adj_vertex is 1st_Layer_Free
    begin
      if adj_vertex.v1 > region_vertex.v1
      begin
        adj_vertex.v1 = region_vertex.v1;
        adj_vertex.v2 = min(region_vertex.v1+1, adj_vertex.v2);
        adj_vertex.prev_vertex_on_1st_layer = region_vertex;
        adj_vertex.prev_edge_on_1st_layer = region_edge;
      end
    end
    else if type of region_edge to adj_vertex is 2nd_Layer_Free
    begin
      if adj_vertex.v2 > region_vertex.v2
      begin
        adj_vertex.v2 = region_vertex.v2;
        adj_vertex.v1 = min(region_vertex.v2+1, adj_vertex.v1);
        adj_vertex.prev_vertex_on_2nd_layer = region_vertex;
        adj_vertex.prev_edge_on_2nd_layer = region_edge;
      end
    end
  end
end
```

Figure 3.19: Pseudo-code of a Procedure to Set Parameters of Adjacent Region Vertices

3.4.3.2 Algorithm for Updating the Region Graph after a Path is Found

After the shortest path with minimum number of vias is found, the path must be added. Since we are adding this path to a region graph, and not a topological graph, the addition can be achieved by splitting each region vertex along the path and connecting the split region vertices by region edges.

When an initial net is routed, we need to find a shortest path in the region graph shown in Figure 3.13. However, since there is only one region vertex in the graph, the *Start_region_vertex* and the *End_region_vertex* are the same. The shortest path has length 0 starting and ending at the same region vertex. This region vertex is simply split into two, and they are connected by one region edge as shown in Figure 3.20. In this case, the wire to connect

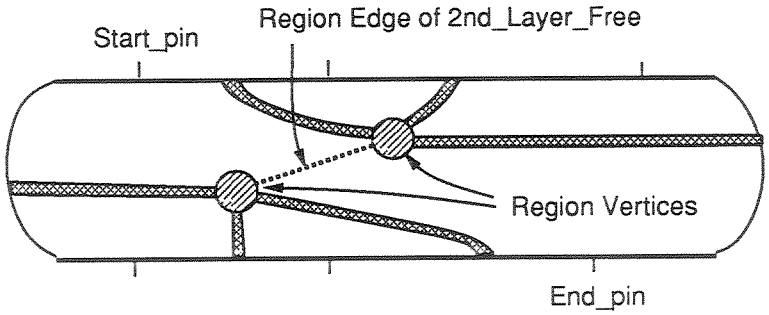


Figure 3.20: Region Graph after the First Net is Routed

Start_pin and End_pin is on the first layer, so that the type of the region edge connecting the split region vertices is 2nd_Layer_Free.

When the shortest path has non-zero length, region vertices along the shortest path must be split because the region is cut by a path. The split regions are connected by a region edge of either type 1st_Layer_Free or type 2nd_Layer_Free. If it is cut by a path on the first layer, the type of the region

edge must be 2nd_Layer_Free, and if it is cut by a path on the second layer, the type of the region edge must be 1st_Layer_Free. Figure 3.21 shows how region vertices are split in a region graph and its corresponding topological graph. This is the case when the shortest path is on the first layer and has no vias.

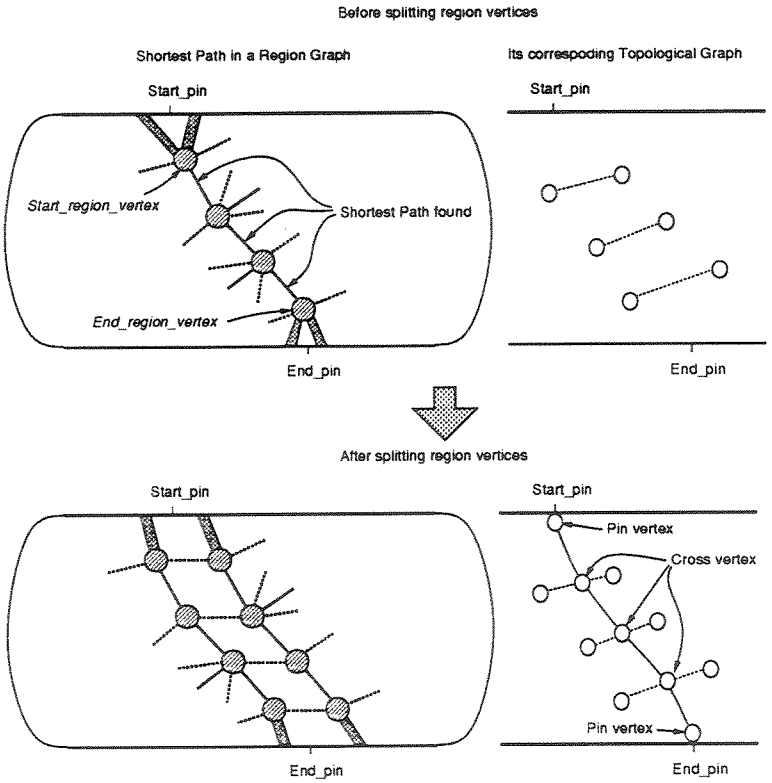


Figure 3.21: Splitting of Region Vertices with No Vias

If the shortest path has vias, extra edges are needed to connect split region vertices, as shown in Figure 3.22. When region vertices are being split along the shortest path, a via is needed when the type of region edges changes from 1st_Layer_Free to 2nd_Layer_Free or from 2nd_Layer_Free to 1st_Layer_Free. The split vertices are connected by two region edges, one of which is type 1st_Layer_Free and the other is type 2nd_Layer_Free. A via vertex in a topological graph corresponds to the area between these two region edges as

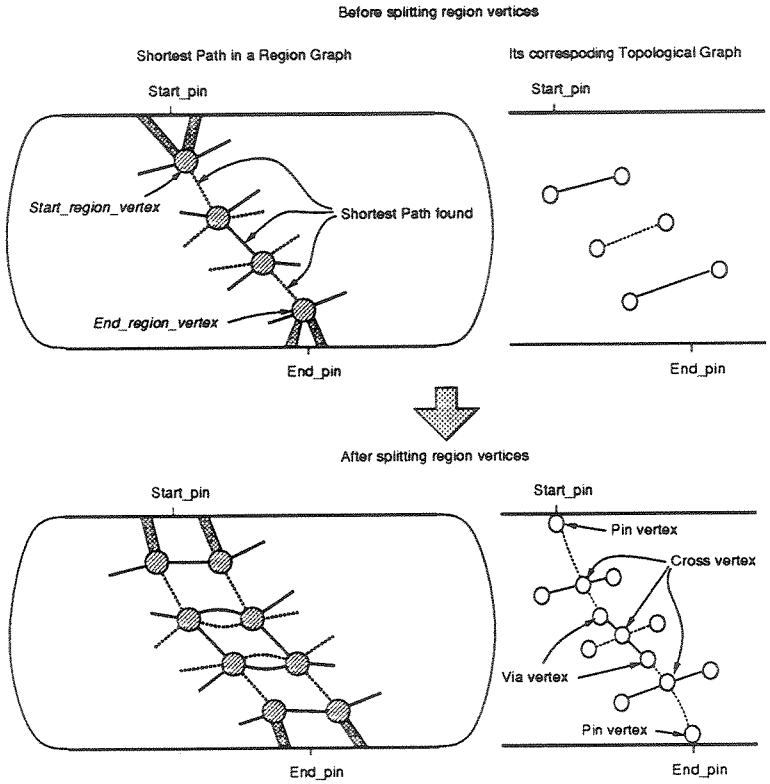


Figure 3.22: Splitting of Region Vertices with Vias

shown in the bottom right part of Figure 3.22. Note that region edges incident on the same region vertex are, if looked at clockwise from the region vertex, of alternating types of 1st_Layer_Free and 2nd_Layer_Free. In other words, a region edge of 1st_Layer_Free is between two region edges of 2nd_Layer_Free. No two region vertices of the same type are next to each other. This is because two adjacent edges incident on the same cross vertex (or a via vertex) in a topological graph are on different layers, thus the corresponding region edges should be of different types. So, when new edges are connected to the split region vertices, the edges must be connected so that the alternating types are preserved.

After all the nets are routed, a topological graph is constructed from

the region graph. An edge on the first layer of the topological graph corresponds to a region edge of type 2nd_Layer_Free. Similarly an edge on the second layer of the topological graph corresponds to a region edge of type 1st_Layer_Free. A vertex of the topological graph corresponds to a region (or a face) of the region graph.

3.4.3.3 Time Complexity of Topological Graph Construction

The time complexity to construct a topological graph is $O(n^3)$ for the following reason.

We first derive an upper bound of the number of vertices of a topological graph when k nets ($1 \leq k \leq n$) are routed. Each net in a topological graph has 2 pin vertices, $O(k)$ via vertices, and $O(k)$ cross vertices. So, the total number of vertices of a topological graph, which is the sum of pin vertices, via vertices, and cross vertices, is $k(2 + O(k) + O(k)) = O(k^2)$. Since each vertex of a topological graph is shared by at most four regions, the number of region vertices of a region graph is $v = 4O(k^2) = O(k^2)$.

A breadth first search algorithm [41] takes $O(v + e)$ time to find a path for each net, where e is the number of edges in the region graph with v vertices. The region graph is a planar graph, so $e = O(n)$. Thus $O(v + e) = O(v) = O(k^2)$. So the time complexity of finding n paths to construct a region graph is

$$\sum_{k=1}^{n} O(v) = \sum_{k=1}^{n} O(k^2) = O(n^3).$$

3.4.4 Mapping of Topological Graph onto Channel

3.4.4.1 Greedy Graph Mapping Algorithm

The constructed topological graph represents only the crossing relationship among the nets and does not specify the physical position of each net on the channel. At this point, only the positions of terminals are fixed. To determine the physical locations of the nets, the topological graph has to be mapped onto the channel.

A greedy algorithm is used to map the topological graph onto the

mesh. The basic idea of the greedy mapping is as follows. A vertical scan line starts from the leftmost pin. As the scan line sweeps across the mesh, the topological graph is mapped onto the mesh from left to right. Appropriate vertices in the topological graph are mapped onto the mesh as soon as possible. If the mesh does not have enough space for more vertices to be placed, a new track is inserted. After all vertices that can be mapped at the current scan line have been processed, the scan line moves to the right column of the interleaving mesh. The scan line continues sweeping across the mesh until the entire graph is mapped onto the mesh.

The greedy algorithm dynamically assigns each vertex of the topological graph to one of four states: "Non-Active Floating", "Active Floating", "Active Fixed", and "Non-Active Fixed", as shown in Figure 3.23. An "Ac-

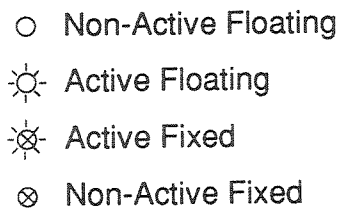


Figure 3.23: States of Vertices of Topological Graph

tive" vertex means that the vertex is ready to be processed. A "Non-Active" vertex means that the vertex is not ready to be processed or finished after it is processed. A "Floating" vertex means that the vertex has not been mapped onto the mesh yet, and a "Fixed" vertex means that the vertex is fixed in the mesh. Vertices with degrees 1, 2, and 4 are called a pin vertex, a via vertex, and a cross vertex, respectively. A pin vertex corresponds to a terminal pin, a via vertex corresponds to a via, and a cross vertex corresponds to the crossing of two different nets. Note that pin vertices are already "Fixed" even before the greedy algorithm starts, while via vertices and cross vertices are "Floating"

initially.

The state transition rules are different for three different vertex types. State transition diagrams for different types of vertex are shown in Figures 3.24, 3.25, and 3.26.

The states of pin vertices change from "Non-Active Fixed", to "Active Fixed", and to "Non-Active Fixed" (Figure 3.24). As the scan line sweeps

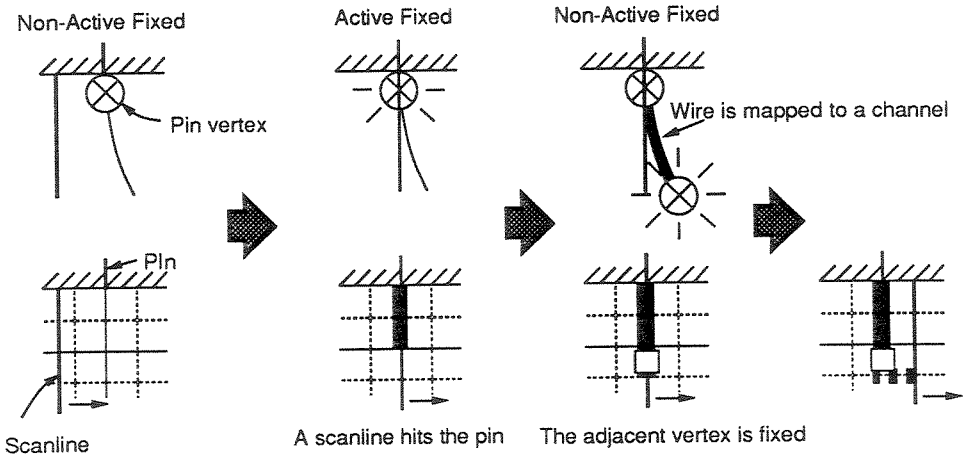


Figure 3.24: State Transition of a Pin Vertex

across the channel, it hits pin positions, at which time the corresponding pin vertices become "Active Fixed". At the same time, a vertical wire is extended from the pin to the nearest available horizontal wiring track on the same layer. The state becomes "Non-Active Fixed" when its adjacent vertex is fixed. When an edge on the first layer is still not mapped, it is represented by a thin solid line, and when the edge is mapped, it becomes thick solid line as shown in Figure 3.24. Similarly, when an edge on the second layer is still not mapped, it is represented by a thin dotted line, and when the edge is mapped, it becomes thick dotted line as shown in Figure 3.25.

The states of via vertices and cross vertices of the topological graph

change from "Non-Active Floating", to "Active Floating", to "Active Fixed", and finally to "Non-Active Fixed" (Figure 3.25 and Figure 3.26). A "Non-

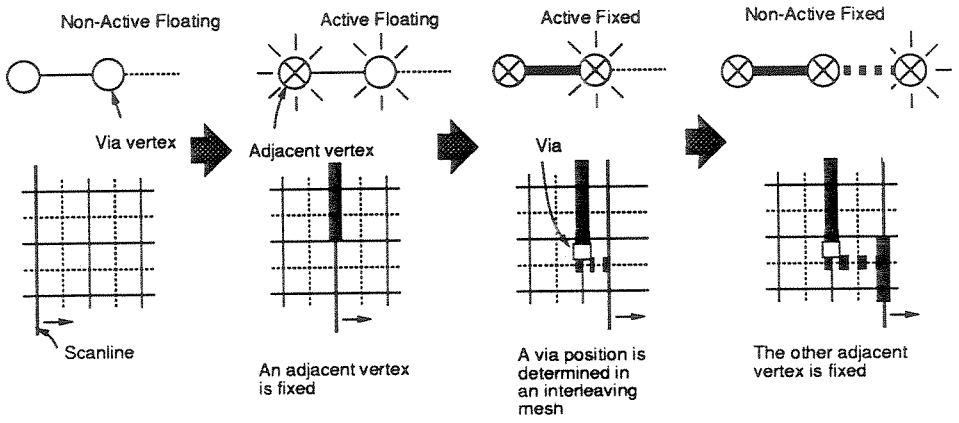


Figure 3.25: State Transition of a Via Vertex

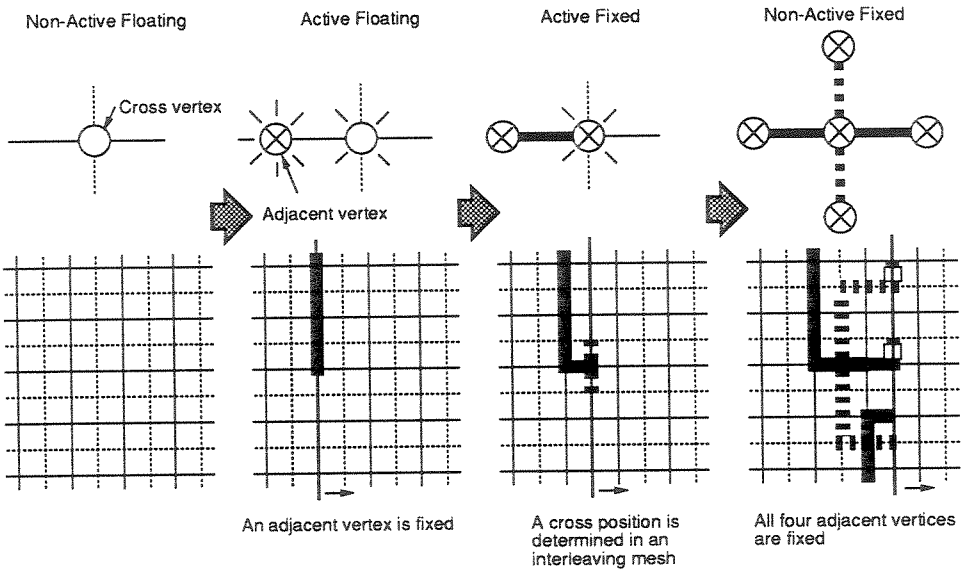


Figure 3.26: State Transition of a Cross Vertex

Active Floating" vertex becomes "Active Floating" when one of its adjacent vertices becomes fixed. This means that the vertex is ready to be mapped onto the mesh only after one of its adjacent vertices is mapped. A floating vertex is

mapped onto the mesh as soon as possible to become an "Active Fixed" vertex. After a via is placed in the mesh at the end of one edge which was already laid out on the mesh, the other edge incident on the same via vertex begins to be laid out on the mesh. The wire corresponding to the other edge is extended to the right if necessary as the scan line moves right until the adjacent vertex is mapped onto the mesh. Similarly, after a cross vertex is placed in the mesh at the end of one edge which was already laid out on the mesh, all the other three edges incident on the same cross vertex begin to be laid out on the mesh. The wires corresponding to these three edges are extended to the right if necessary as the scan line moves right until the adjacent vertices are mapped onto the mesh. After all adjacent vertices are "Fixed" on the mesh, the vertex becomes "Non-Active Fixed".

It is clear that a planar topological graph can be always mapped to a mesh by this algorithm based on the following arguments. All the pin vertices can become "Active Fixed" when they are hit by a scan line. The pin vertices can return to "Non-Active Fixed" when their adjacent "Floating" vertices (let's call the set of vertices adjacent to the pin vertices "group 1 vertices") are fixed, which is guaranteed to happen because "group 1 vertices" can always be mapped onto a mesh by creating a new track whenever needed. Because all the "group 1 vertices" can become "Fixed", all the vertices that are adjacent to "group 1 vertices" and that are not pin vertices (henceforth called "group 2 vertices") can become "Active Floating". The "Active Floating" group 2 vertices in turn can always become "Active Fixed" (or mapped onto a mesh) by creating a new track whenever needed. This argument (which is like peeling off layers of an onion) continues for vertices of group 1, 2, 3, ..., n, until all the vertices are fixed. Thus, all the vertices are guaranteed to be mapped onto

a mesh. A simple example in Figure 3.27 shows how the topological graph is mapped onto the mesh.

The greedy mapping algorithm takes $O(wn^2)$ time, where w is the width of a channel. For each column where a scan line is sweeping, all the vertices of the topological graph are checked to see if they can be mapped to a mesh. The number of vertices of a topological graph for n nets is $O(n^2)$ as explained above. Since the scan line sweeps for $O(w)$ columns, the time complexity is $O(wn^2)$.

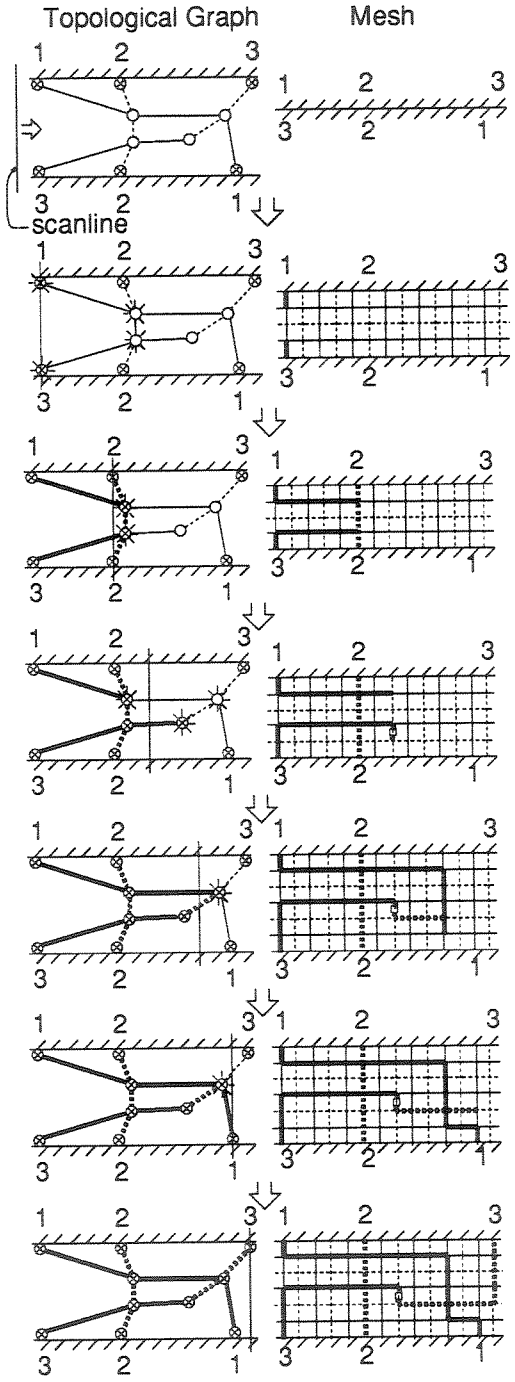


Figure 3.27: Example of Greedy Mapping Algorithm

3.4.4.2 Special Treatment for Multi-Terminal Nets

Multi-terminal nets are decomposed into two-terminal nets before a topological graph is constructed. When a scan line hits a pin, a wire is extended from a pin into a channel.

However, there is a case when several two-terminal nets share the same pin. In this case, wires must be extended for all the two-terminal nets. Figure 3.28 shows how this is done. When several nets share the same pin, the

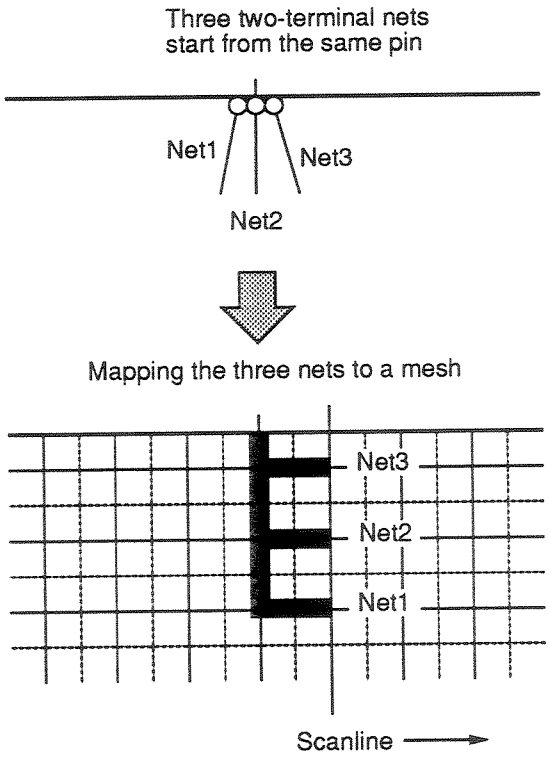


Figure 3.28: Mapping of Multi-Terminal Nets Sharing the Same Pin

longest wire is extended from the leftmost pin, and the shortest wire is extended from the rightmost pin because the scan line is sweeping the channel from left to right. As a result, the topological relationship of the nets is preserved.

3.4.4.3 Mesh Data Structure

The interleaving mesh can be represented by a two-dimensional array. However, during the greedy mapping algorithm, new tracks are inserted any time there is such a need. Consequently, we need a data structure for the mesh which is suitable for quick insertion of tracks.

One dimensional array of Index[y] of pointers to the tracks of Mesh[x,y] is maintained as shown in Figure 3.29. Each index contains a track number. When rows of Mesh[x, y] are rearranged, Mesh[x, Index[y]] forms a valid layout without design-rule violations.

Before the greedy mapping starts, Index[y] contains nothing, and the height of the mesh is 0. As the mapping proceeds, the tracks are inserted to a mesh. If a track is inserted at y_{new} of Mesh[x, y], all the tracks from $y_{new} + 1$ through y need to be shifted up, which takes time. Instead, a new track is added on top of Mesh[x, y], thus eliminating a step of shifting up many rows of Mesh[x, y]. Only numbers in Index[y] need to be shifted up, and the new item in Index[y] points to a newly added track on the top of the channel.

In Figure 3.29, two new tracks are needed between tracks 1 and 2. Mesh[x, 7] and Mesh[x, 8] are used as new tracks, but Mesh[x, 0] through Mesh[x, 6] are not changed at all. Items in Index[2] through Index[6] are shifted up by 2, and a new Index[2] and Index[3] becomes 7 and 8, respectively. After this track insertion, wires are vertically extended, and the scan line moves one unit to the right as shown in Figure 3.30.

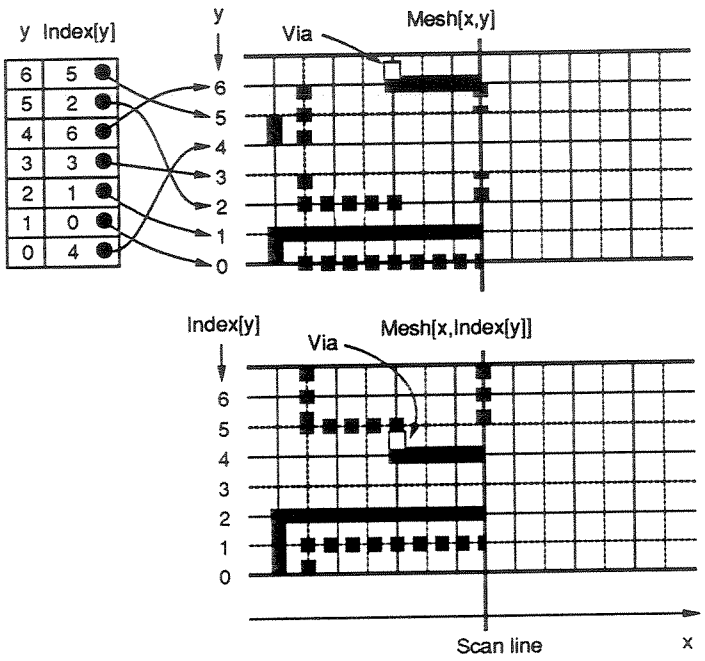


Figure 3.29: Mesh[x,y] and Mesh[x,Index[y]] Before New Tracks are Inserted

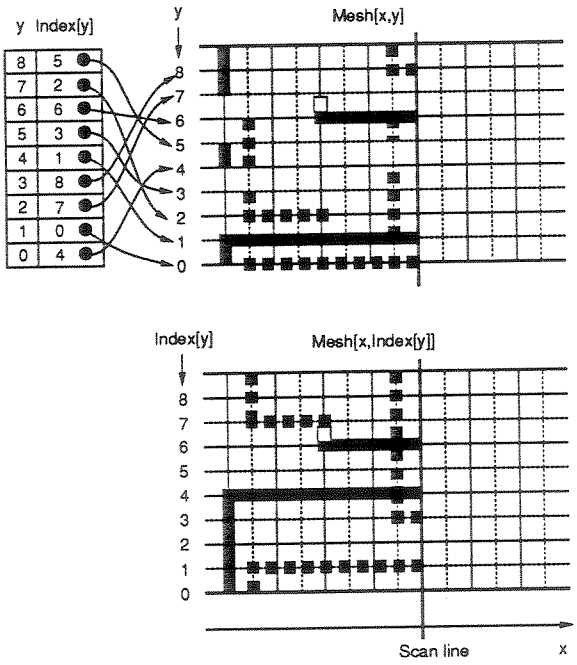


Figure 3.30: Mesh[x,y] and Mesh[x,Index[y]] After New Tracks are Inserted

3.4.5 Compaction

During the greedy mapping of the topological graph, new tracks are inserted into the interleaving mesh whenever they are needed. Since tracks are very sparsely occupied by wires, such unused portions of the mesh need to be removed by compaction. We have developed a sophisticated two-dimensional compacter. First, we describe previous approaches to the compaction problem, and then we present our new two-dimensional compacter.

3.4.5.1 Previous Approaches

Compaction is a process which minimizes area or wire length. It is needed to remove wasted space between components and between wires. Ideally compaction should be done in two dimensions. The simplest way to compact a layout in two dimensions is to repeat applying a one-dimensional compacter in one direction and applying the same algorithm in the other direction alternatively. This method is very efficient and needs only a one-dimensional compacter. Compacting a layout in both directions simultaneously, however, would result in better compaction results, because compacting in one direction without considering the other direction may block further compaction in the other direction. Since the problem of finding an optimum result of two-dimensional compaction is NP-hard [39], heuristics have been proposed. We first describe several one-dimensional compaction algorithms, followed by a two-dimensional compaction algorithm.

The shear-line method [3] compacts area one-dimensionally by removing excess space. A band of excess space with the same width is removed. This is the first compaction method and it is intuitively easy to understand. However, the computational cost of this method is very high, so that its use is

limited to small size layouts.

Another method of one-dimensional compaction called the critical path method uses a graph to find the most compact solution [8] [21]. Each vertex of the graph represents an object in the layout, and each directed edge represents the minimum geometric design rule spacing. After the graph is constructed, a longest path, i.e., a critical path, is found for each object, which determines its location. The computation time is faster than the shear-line method and hence the critical path method can be used for large layouts.

A virtual grid one-dimensional compaction method [48] places objects on a virtual grid. Compaction is done by determining the location of each virtual grid line. Since only the adjacent grid line needs to be checked to determine the location of the current virtual grid, the computation is very fast. The speed up is obtained at the cost of a slightly larger compaction area.

In 1986, a new two-dimensional compaction algorithm called zone refining compaction [42] [43] [44] was developed which simulates a zone refining process to purify crystal ingots. In the zone refining process, an ingot is slowly pulled through a heater that locally heats the crystal to melting temperature. The melted location is called a molten zone. After it has passed through the heater, the crystal recrystalizes, resulting in a lower concentration of impurities. This is because the impurities are formed into an atomic lattice at a lower rate than crystal atoms. This phenomenon "shakes" impurities out of the crystal lattice.

In zone refining compaction, elements are taken off row by row (or melted in physics terms) from a cluster at the top, moved across a zone, and reassembled (or recrystalized) at the lowest possible location on another cluster at the bottom. A simple zone refining compaction applied to bin packing is

shown in Figure 3.31. The impurities in the compaction are the unnecessary

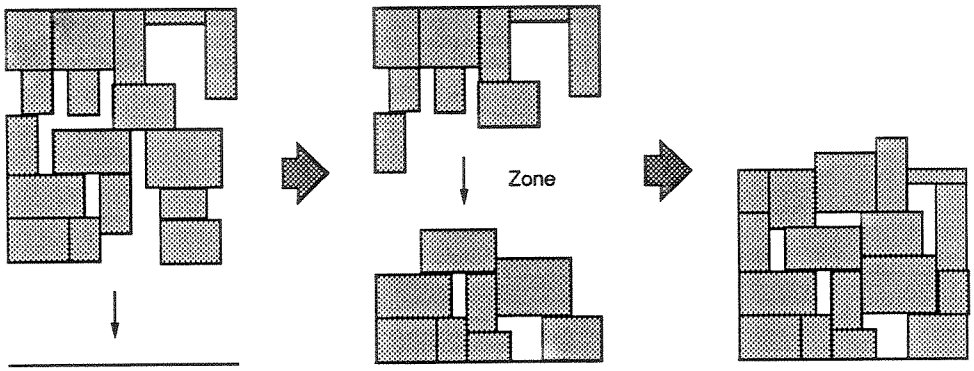


Figure 3.31: Example of Zone Refining Compaction

empty spaces between elements. When the elements are deposited onto the bottom, both coordinates of the elements can be changed. Thus, zone refining compaction combines a one-dimensional compaction procedure with sophisticated lateral movements of elements. The zone refining compaction can be repeated by shaking elements up and down, or left and right, or both. When compacting an integrated circuit layout, circuit elements are interconnected by wires that limit the degree to which each element can move.

3.4.5.2 Our Compaction Method

Our compaction method is similar to the zone refining technique described in the previous section, with a modification called horizontal make-space. The channel is horizontally scanned from bottom to top, and when vias and wires are hit, they are moved down. We now define solid elements and flexible elements: a solid element is defined as an object that can move in the channel without changing its shape, and a flexible element is defined as an object that can move in the channel and that can change its shape. Vias are treated as solid elements and wires are treated as flexible elements. In

our compaction step, a via is taken from one cluster at the top as shown in Figure 3.32 a and Figure 3.32 b. The lowest position for a via is then found

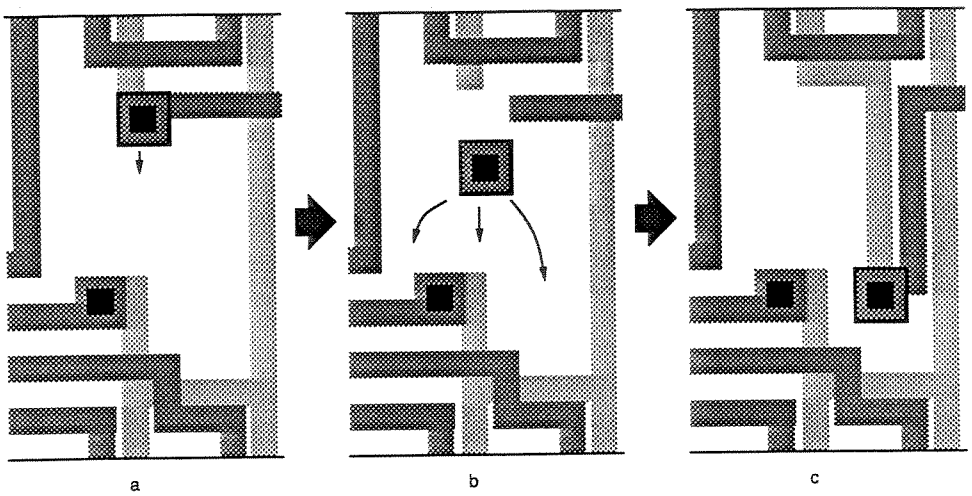


Figure 3.32: A Via is Moved Down to the Lowest Position

by horizontally moving a via and searching for the lowest bottom. Note that a via has elements on both layers, so design rule checking must be done on both layers. The via then moves to the location, and wires connected to the via are also pulled down as shown in Figure 3.32 c. Since vias are solid elements, they never change their shapes.

Wires are simply pushed straight down as shown in Figure 3.33. If the contour of the cluster at the bottom is not flat, the wire, which is a flexible element, is bent to fit the contour as shown in Figure 3.34.

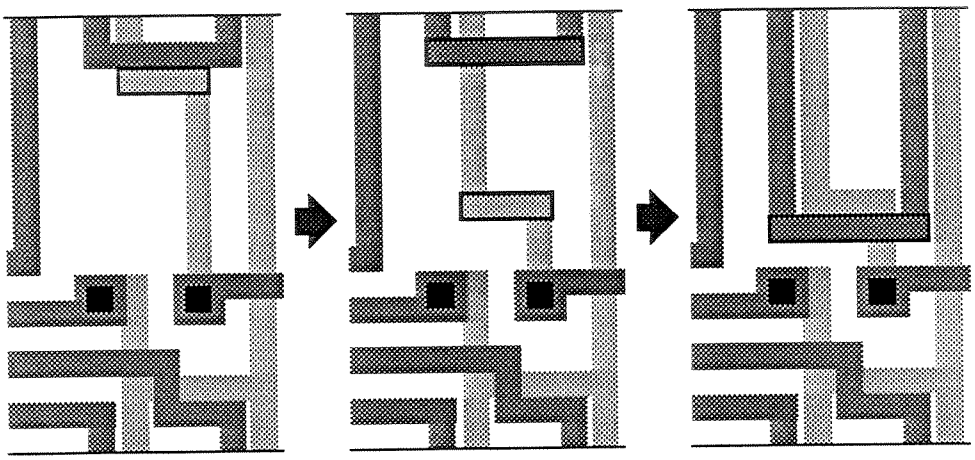


Figure 3.33: A Wire is Moved Down to the Lowest Position

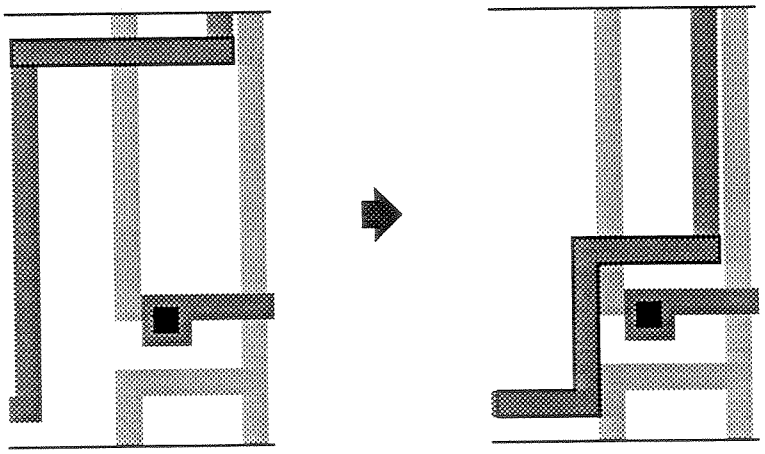


Figure 3.34: A Wire is Bent to Fit a Contour

3.4.5.3 Making Space for Better Shaking Results

When a via is pushed down, it is moved horizontally in the zone to find the lowest position to be pushed to. In the channel routing problem, however, there are many vertical wires that limit the horizontal movement of a via. This limits the search area for the lowest position.

We have developed a new method called horizontal make-space to deal with this problem. The horizontal make-space procedure creates space by horizontally moving wires and vias away from the via to be moved down. As shown in the top of Figure 3.35, vertical wires on the first layer severely limit the horizontal movement of the via to be pushed down. We call the via a target via. For freer target via movement, wire and vias are pushed away from the target via as shown in the second row of Figure 3.35. Three cases are tried : the first case is that only wires on the first layer are pushed away, as shown in the left drawing, the second case is that only wires on the second layer are pushed away, as shown in the middle drawing, and the third case is that wires on both layers and vias are pushed away, as shown in the right drawing. After wires (and vias) are pushed away, the target via looks for the lowest position in each case. The case in which the lowest position is obtained is chosen and kept. The other two cases are discarded. In Figure 3.35, the case when wires on the first layer are pushed away produced the lowest target via position. Note that pushing wires on both layers and vias does not always make better results than the other two cases. This is because the pushed horizontal wires on the first layer or on the second layer needlessly prevent the target via from finding a lower position.

When wires and vias are pushed away from the target via, we must set a limit on how far they are pushed away, otherwise they are pushed away to

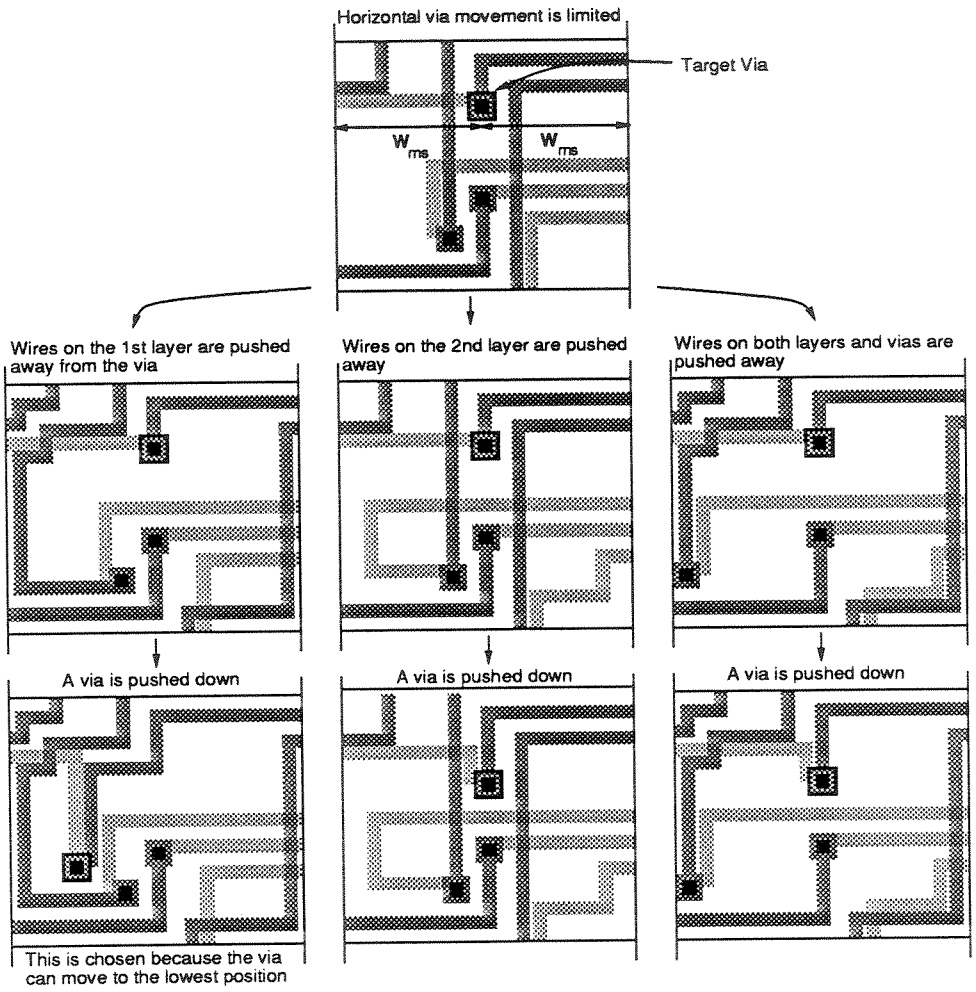


Figure 3.35: Making Space for Better Shaking Results

infinity. The horizontal width limits to the left and to the right are W_{ms} each as shown in the top of Figure 3.35. We might think that the larger W_{ms} is, the better result we can obtain. It turns out, however, that if W_{ms} is too large, many wires meander around the target via, resulting in an increase in height. Obviously if W_{ms} is 0, the result is worse than when W_{ms} is a positive number. So we did an experiment to find the best W_{ms} value. The result is shown in Figure 3.36. Four examples from [50] are used. The experiment confirms that if excessive space is made, the resulting height becomes larger. In example 3a, for

example, the best W_{ms} value, which yields the smallest height, ranges between 10 and 26. In this case, heights happen to be all 32 when W_{ms} is between 10 and 26. Since larger space requires more computation time, it is best to choose $W_{ms} = 10$, which is the minimum W_{ms} that yields the best height. Similarly, the best W_{ms} is 10, 18, and 24 for examples 3b, 3c, and 4b, respectively.

After the make-space procedure, wires around the target via meander. These are straightened before the next push down of vias and wires takes place.

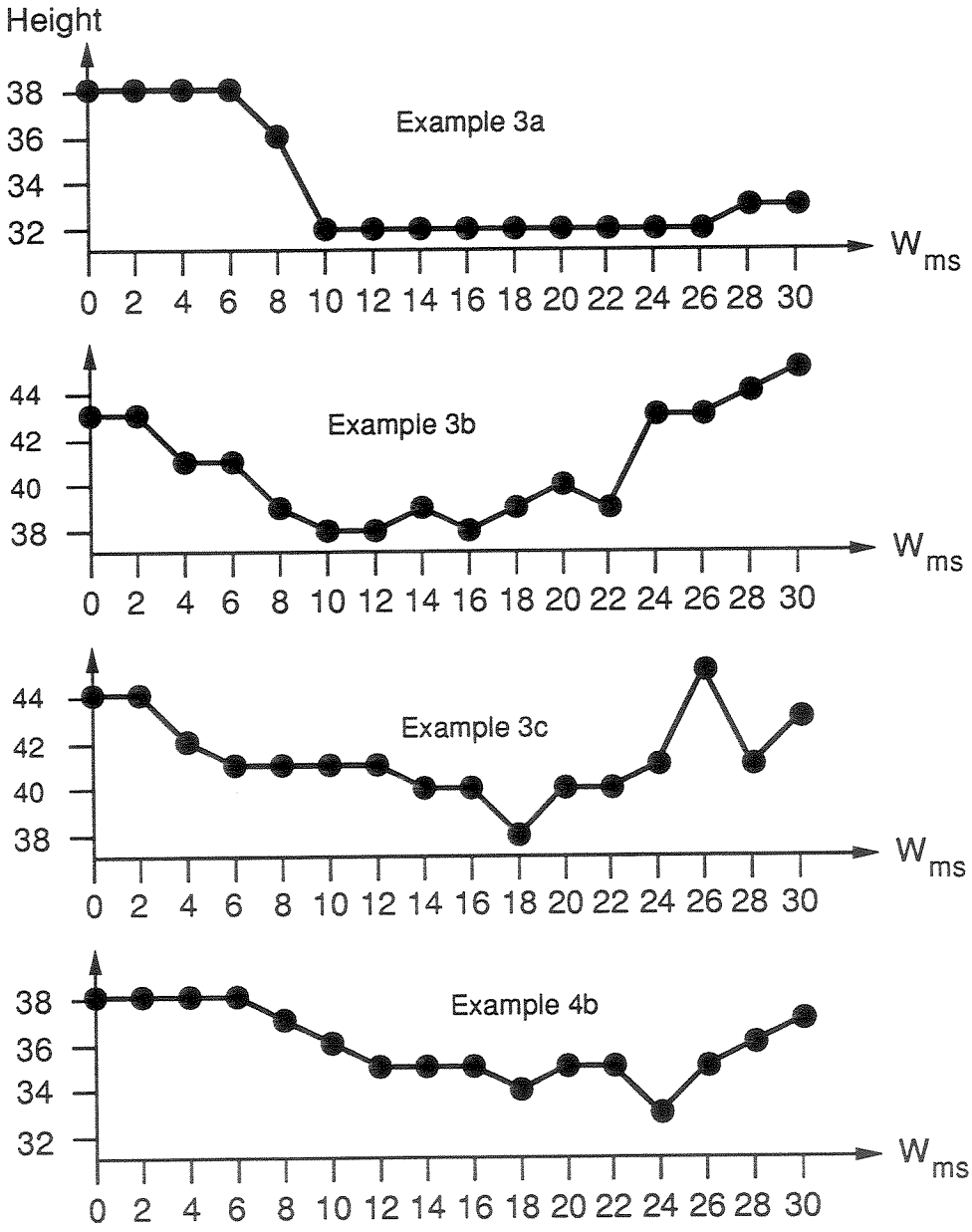


Figure 3.36: Channel Height v.s. Make-Space Width

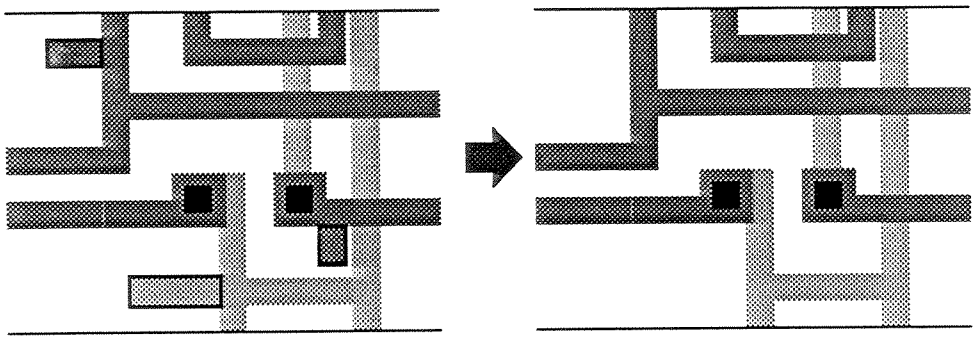


Figure 3.38: Pruning an Unnecessary Wire

- Cutting Unnecessary Loops

  Sometimes unnecessary loops appear after the shake as shown in Figure 3.39. These loops are removed by traversing each net from a pin to

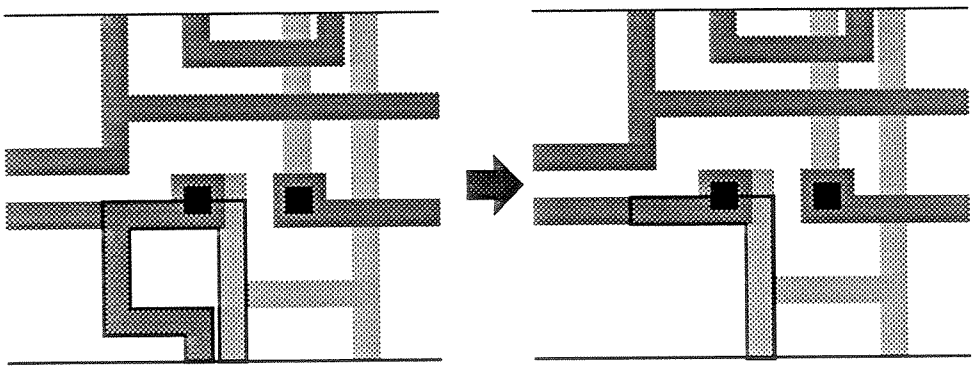


Figure 3.39: Cutting an Unnecessary Loop

identify them, and then cutting them so that the net becomes a tree with no loops. The loops must be carefully cut, because if carelessly cut, the net becomes multiple trees.

Chapter 4

Improvements to Topological Channel Routing

Our method of topological channel routing described in the previous chapter determines a topological relationship between the nets first, and then maps the nets to an actual routing region. Although this method is very effective in producing compact routing solutions that use a small number of vias, it can be improved to obtain more compact solutions by using geometric information when we construct a topological solution. In Figure 4.1, for example, even though both topological solutions A and B can be constructed by the algorithm in the previous chapter, layout A, which is obtained by mapping the topological solution A to a channel, has a height of only 6, whereas layout B has a height of 8. The algorithm in the previous chapter cannot find any distinction between the two topological solutions before they are mapped to a channel.

In this chapter, we present an improved topological channel routing algorithm which takes geometric information into consideration when a topological solution is obtained [16].

4.1 Topological Routing using Geometric Information

There have been many attempts to compact a channel to get a result with a small area [12] [35] [7]. These techniques achieve compact results by starting with a reserved layer router and then compacting the result. On the other hand, topological routers which do not observe the reserved layer rules

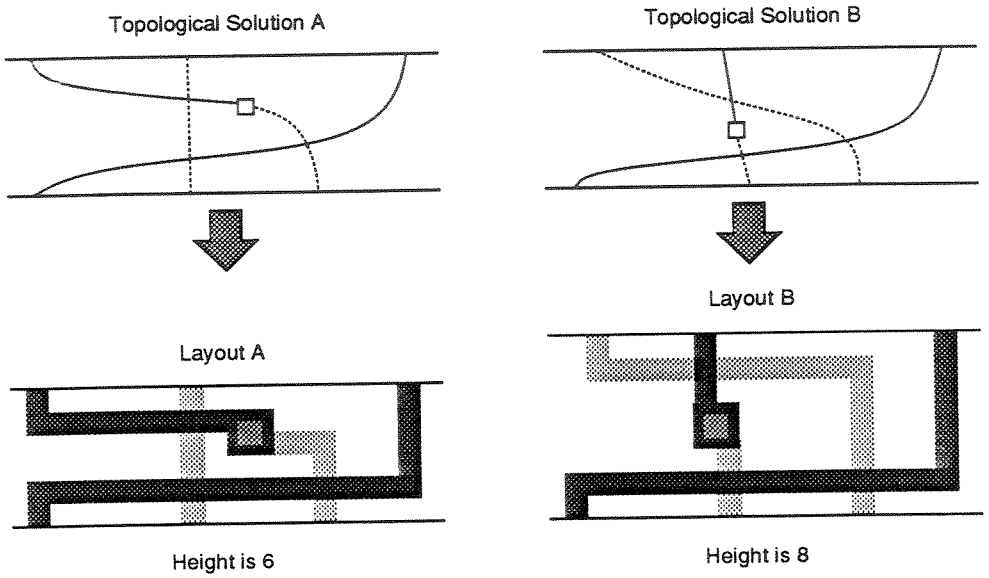


Figure 4.1: Two Different Topological Solutions and Their Mappings to a Channel

can find solutions with small numbers of vias, but have to work harder to produce compact results.

We incorporate some of the geometric constraints of reserved layer routing into a topological router in an attempt to take advantage of the best features of both methods, i.e., producing results with a small area and a small number of vias.

4.2 Assignment of Nets to Two Layers

In the reserved layer model, all horizontal wires are placed on the first layer and all vertical wires are placed on the second layer. If a horizontally short net (i.e. a net whose distance along a channel is short) connects a pin on the top wall to a pin on the bottom wall, we call it a near-vertical net.

Nets are assigned to layers in a way similar to that used in the reserved layer model. Near-vertical nets are assigned to the second layer using no vias

where possible, and long horizontal nets use two vias and contain a horizontal run on the first layer and two vertical runs on the second layer. So, near-vertical nets form the set N_2 (the set of nets on the second layer), long horizontal nets form the set N_{12} (the set of nets on both layers), and the set N_1 (the set of nets on the first layer) is ϕ.

Now, we discuss the selection ordering of nets without vias to be placed on the second layer. First, near-vertical nets which do not cross each other are chosen and assigned to the second layer (See Figure 4.2). It is best to assign as many near-vertical nets to the second layer as possible, i.e., to maximize the cardinality of N_2 for near-vertical nets, because horizontal wires on the first layer contribute to the increase of channel height, while near-vertical wires on the second layer do not. Let d_H be the horizontal distance between two pins of a near-vertical net. The sequence of choosing near-vertical nets is as follows. First, all the near-vertical nets with $d_H = 0$ (i.e. truly vertical nets) are chosen. It is obvious that no truly vertical nets cross each other. Then near-vertical nets with $d_H = 1$ are chosen if they do not cross already chosen nets. We continue this process with $d_H = 2, 3, 4$, and so on, until no more near-vertical nets which do not cross other nets are found. Note that this algorithm does not find a maximum set N_2 of non-intersecting nets on the second layer (as we did in Section 3.4.2), because near-vertical nets are preferred to horizontally-long nets.

Notice that there are still more nets that can be assigned to the second layer, as shown in step 2 of Figure 4.2. These are nets which connect two pins on one wall (top or bottom) of the channel.

All remaining nets can be routed using exactly two vias per net. Each remaining net connects two pins first using the second layer, then switching to

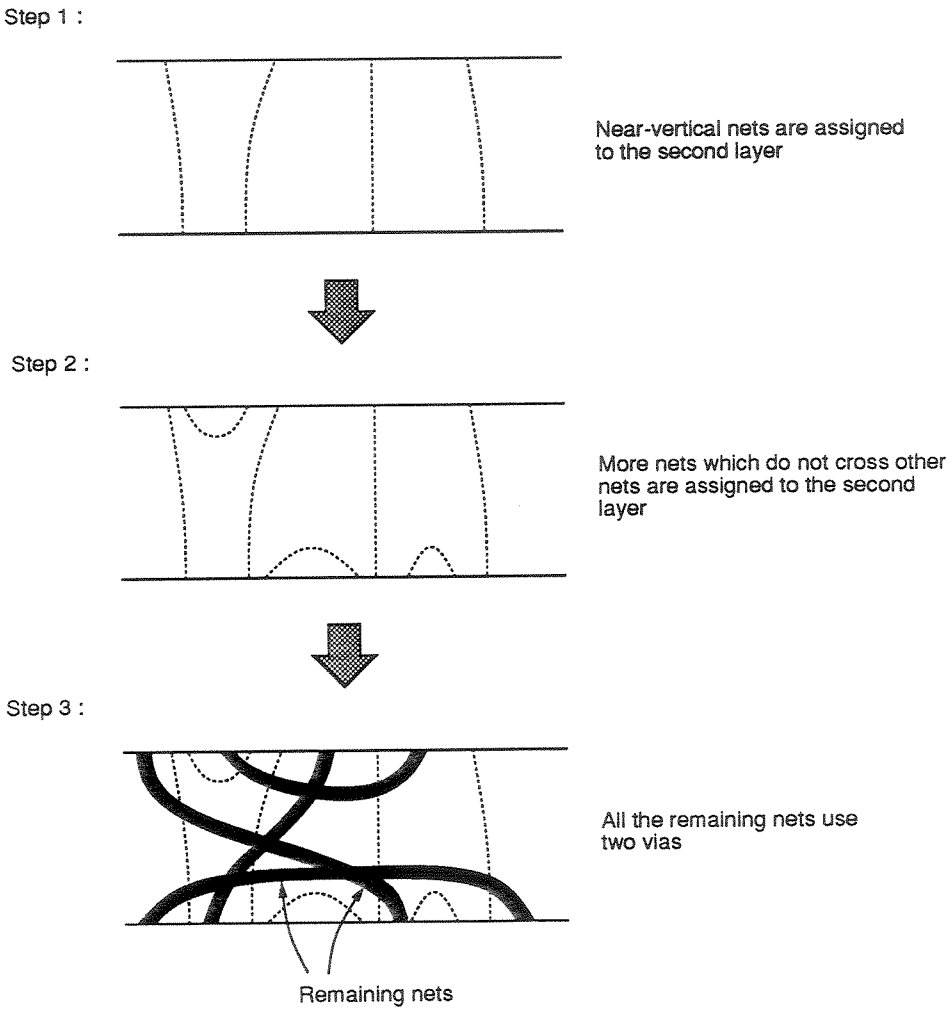


Figure 4.2: Layer Assignment

the first layer, and then switching back to the second layer. These nets with two vias cross each other. They also cross nets assigned to the second layer, but they can be routed without failure by jumping to the other layer using vias.

4.3 Improved Topological Graph Construction

After the layer assignment, topological solutions are obtained by constructing topological graphs. The algorithm to construct a topological graph is slightly different from that described in Section 3.4.3.

In the algorithm described in Section 3.4.3, if a region edge type is 1st_Layer_Free, the search path on the first layer can go along the edge, i.e., a path on the first layer can cross a wire on the second layer, and similarly a path on the second layer can cross a wire on the first layer. However, as shown below, when geometric constraints are imposed, a search path cannot go along some region edges. From now on, we explain the algorithm using the topological graph instead of the region graph because it makes the algorithm description easier to understand. When we say that an edge of a topological graph cannot be crossed, it actually means that an edge of a region graph cannot be used for a search path.

All the nets with two vias are topologically routed before nets without vias are routed on the second layer. We first discuss how each net with two vias is routed. When a net with two vias is routed, a path starts on the second layer, uses a wire segment on the second layer, jumps to the first layer through a via, uses a wire segment on the first layer, jumps back to the second layer through a via, and reaches the other pin again using a wire segment on the second layer. Each wire segment consists of one or more edges of the topological graph. Before discussing the rules to prohibit the crossing of edges of a topological graph, we define some values associated with the edges and the wire segments. (See an example in Figure 4.3).

- A "position x" of a wire segment on the second layer is defined as the position of a pin to which the wire segment is connected.

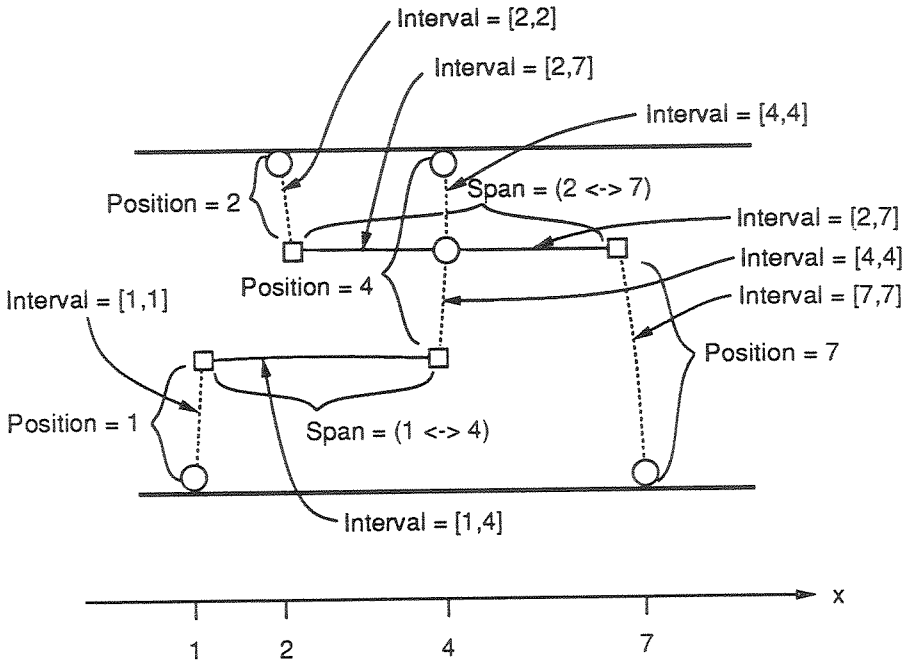


Figure 4.3: Example of Position, Span, and Interval

- A "span $(x_{min} \leftrightarrow x_{max})$" of a wire segment on the first layer is defined as the smaller and the larger position x of two wire segments on the second layer to which this wire segment on the first layer is connected.

- Each edge of the topological graph comprising a portion of a wire segment with the span $(x_{min} \leftrightarrow x_{max})$ on the first layer has an interval $[x_{min}, x_{max}]$.

- Each edge of the topological graph comprising a portion of a wire segment with the position x on the second layer has an interval $[\ x, x\ ]$.

Note that even though the words "position", "span", and "interval" are used, wire segments and edges are not fixed to any physical location (i.e., floating), because they are not mapped to a channel yet.

The following three rules are used when a breadth first search algorithm expands search paths from Pin A toward Pin B for a net with two vias

(See Figures 4.4, 4.5, and 4.6).

Note that these rules are only for nets with two vias, and are ignored for nets without vias on the second layer. Consider any path P expanded by the breadth first search, and assume without loss of generality that $x_A \leq x_B$. A similar rule is applied when $x_A \geq x_B$.

Rule 1 (Figure 4.4) : A search path P from Pin A at position x_A to Pin B at position x_B cannot cross any edge with interval $[x, x]$ on the second layer unless $x_A \leq x \leq x_B$.

When a search path encounters an edge on the second layer with the interval of $[x, x]$, the program checks if $x_A \leq x \leq x_B$, and if so, the path on the first layer can cross this edge on the second layer. This rule prevents unnecessary meandering of the net from Pin A to Pin B as shown at the bottom of the figure. Via 1 adjacent to Pin A is not actually needed when this connection from A to B is made. However, each net must use two vias so that nets which are topologically routed later never fail to find a path.

Rule 2 (Figure 4.5) : A search path P cannot cross any edge on the first layer with interval $[x_{min}, x_{max}]$ if $x_A < x_{min}$ and $x_{max} < x_B$, i.e., if the interval of the edge is inside the range of Pin A and Pin B.

This prevents the unnecessary bending of a wire segment on the second layer as shown at the bottom of the figure. The bending sometimes consumes precious space in the channel, so it is better to have straight vertical wires on the second layer.

Rule 3 (Figure 4.6) : If $x_A < x_B$ and a search path P has crossed an edge on the second layer with interval $[x_{prev}, x_{prev}]$, then P cannot cross any edge on

the second layer with interval $[x_{next}, x_{next}]$ where $x_{next} < x_{prev}$.

If $x_A < x_B$ and the path on the first layer crosses edges on the second layer with the interval of $[x_1, x_1]$, $[x_2, x_2]$, $[x_3, x_3]$,... $[x_k, x_k]$ in this order, x_i $(1 \leq i \leq k)$ must increase monotonically, i.e., $x_1 \leq x_2 \leq ... \leq x_k$. So the crossing of an edge that causes non-monotonic increase must not be allowed.

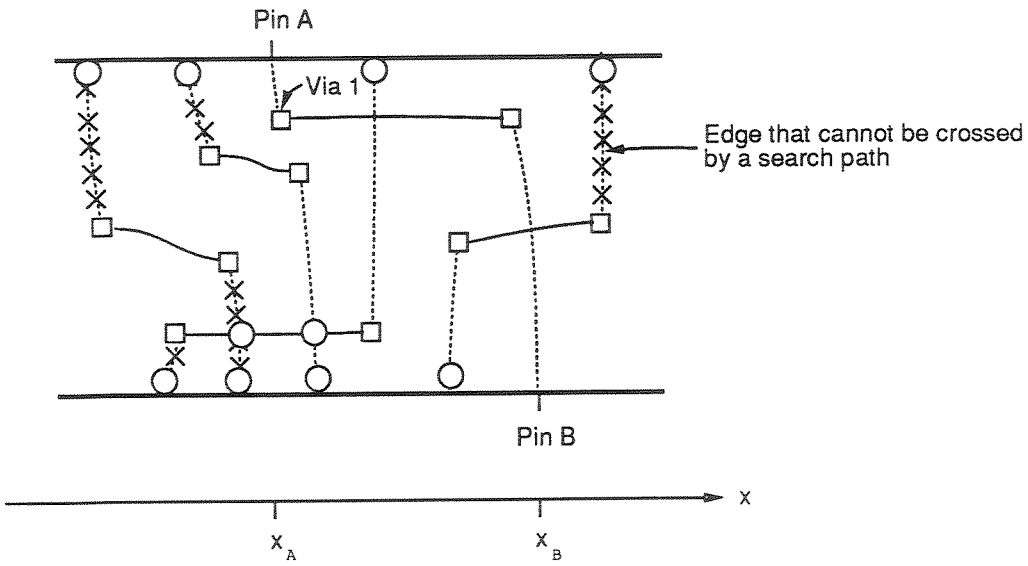
When breadth first search is in progress, each region vertex remembers a previous region edge which contains the shortest path from Pin A to this region vertex. So it is easy to recall the value of x_{prev}. This rule prevents horizontal u-turns in the path, which obviously increase the area as shown at the bottom of the figure.

After the path is found, it is added to the topological graph by splitting the edges along the path. If the interval of an edge is $[x_1, x_2]$, both of the split edges have the same interval $[x_1, x_2]$. The newly added edges on the second layer along the found path are assigned an interval of $[x, x]$, where x is the position of a wire segment of the edges on the second layer, and the newly added edges on the first layer along the found path are assigned an interval of $[x_{min}, x_{max}]$, where a wire segment of the edges on the first layer has a span of $(x_{min} \leftrightarrow x_{max})$.

As described before, the topological channel routing algorithm actually uses a region graph instead of a topological graph when paths are being added, and after the region graph is complete, the topological graph is constructed from it. Special care is needed when a path from Start_pin to End_pin uses two vias even though the path does not need any vias as shown in Figure 4.7. These needless vias must be used in order to enable routing of nets

which will be later routed without vias. The path splits a region into two new regions just like the original algorithm, but three region edges are needed to connect the two region vertices. The two of the three region edges are 1st_Layer_Free, and the other one is 2nd_Layer_Free in this case. The three edges are needed because three new edges of a topological graph are created.

After all the nets with two vias are routed, all the nets assigned to the second layer are topologically routed without vias. When nets assigned to the second layer are routed, the above rules are ignored, i.e., any edges are allowed to cross except the obvious rule that the search path on the first layer should not cross edges on the first layer and similarly the search path on the second layer should not cross edges on the second layer.



This prevents the meandering as shown below

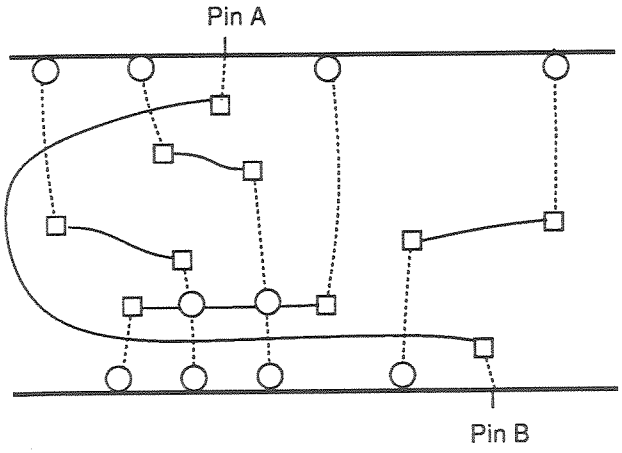
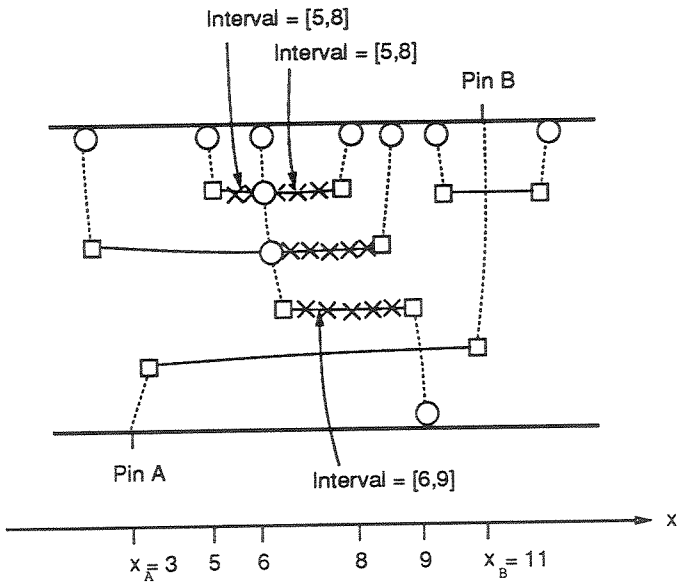


Figure 4.4: Rule 1



This prevents an unnecessary bending of a wire segment on the second layer as shown below

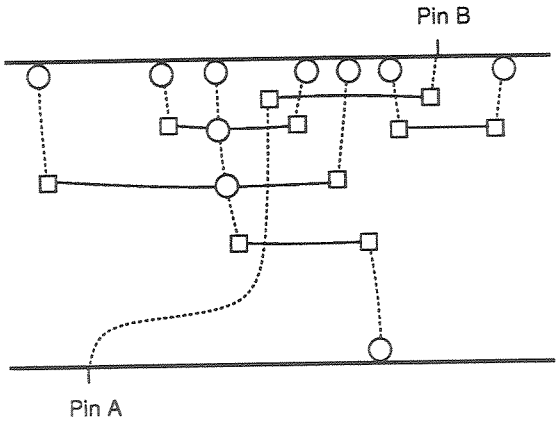
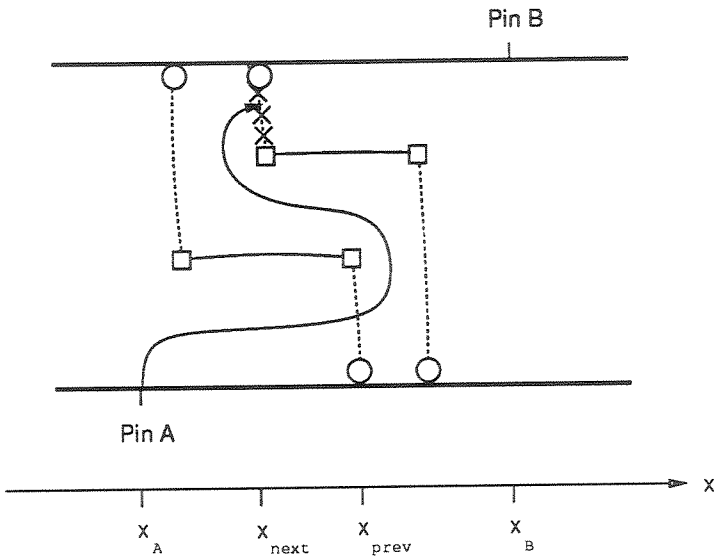


Figure 4.5: Rule 2



This prevents the U-turn of the path as shown below.

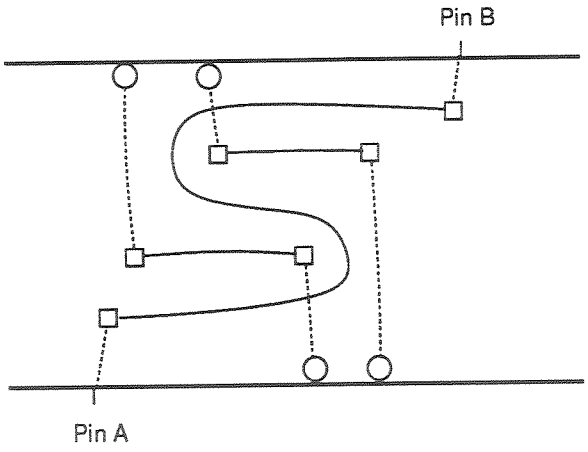


Figure 4.6: Rule 3

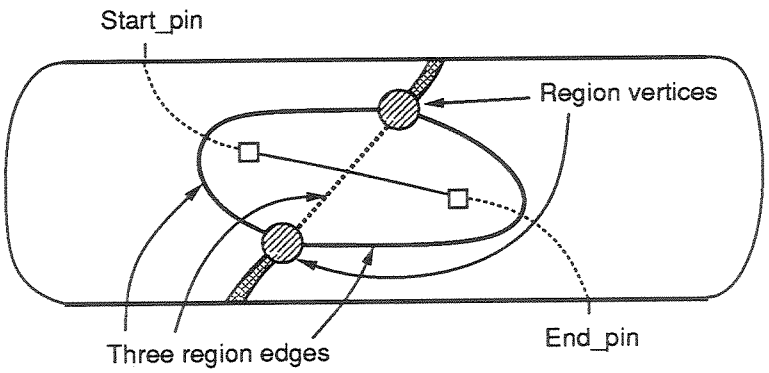


Figure 4.7: Region Graph after a Net with Two Vias is Routed

4.4 Via Reduction

After a topology of the nets is determined, we want to reduce the number of vias. The main objective of the channel router is to obtain a result with a small area, and the next objective is to do it with a minimum number of vias. So even after the topology is determined, we must be careful not to over-minimize the number of vias, which might result in an increase in area.

As described in Section 3.1, there are basically two approaches to reduce the number of vias: Constrained Via Minimization (CVM) and Unconstrained Via Minimization (UVM).

Since the topology of the nets is determined when the topological graph is constructed, we do not want to change the topology, i.e., we cannot use UVM techniques at this stage. Instead, we could reduce the number of vias by switching the layer assignment of edges of the topological graph without changing the topology of nets. This idea is similar to CVM methods, except that the traditional CVM is applied to a physical wire layout which is already placed in a channel (See Figure 3.1 for an example of CVM). A method similar to this idea which performs CVM to a topological graph without physical wire and via location constraints was developed by Naclerio, Masuda, and Nakajima [30]. Their method can obtain the minimum number of vias of a given topology in $O(n^3)$ time, where n is the number of nets.

In the improved topological graph construction, however, we determine the topology of nets intelligently using geometric information. If we perform CVM on the topological graph, all the geometric information used to construct it would be lost. So we will do local layer assignment modifications of the topological graph. If the modifications are done locally, the loss of the geometric information contained in the topological graph will be minimal.

We perform two types of local layer assignment modifications : one is local layer assignment modification at pin vertices and the other is local layer assignment modification at other vertices.

4.4.1 Local Layer Assignment Modification at Pin Vertices

After the topology of all the nets is determined, vias adjacent to a pin can be easily removed by changing the layer of the pin connection as shown in Figure 4.8. If p pins of the same signal must be connected, $(p-1)$ two-terminal

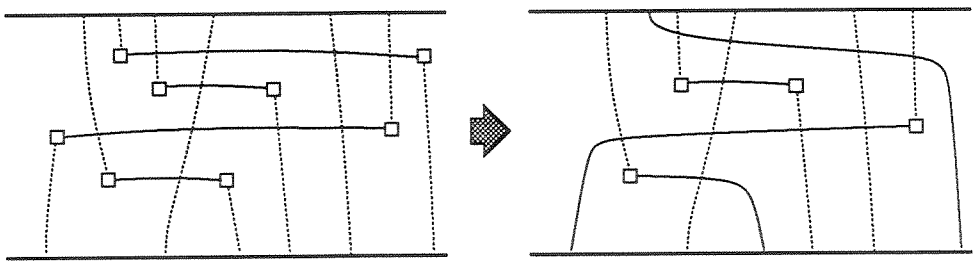


Figure 4.8: Via Reduction at Pins

nets are needed. In the worst case, $2(p-1)$ vias are needed for this signal if all the $(p-1)$ two-terminal nets use two vias. On the other hand, in a reserved layer channel router, only p vias are needed. However, this worst case seldom happens, because many nets among the $(p-1)$ nets are assigned to the second layer without vias, vias adjacent to pin vertices are thus removed, and some vias at other vertices are also removed as described next.

4.4.2 Local Layer Assignment Modification at Other Vertices

Vias can be reduced not only at pin vertices but also at other vertices. Suppose a cluster of v_{cross} cross vertices are connected in sequence. See Figure 4.9 where $v_{cross} = 3$. There are $2v_{cross} + 2$ vertices connected to the cluster. If there are more than $v_{cross} + 1$ vias among the $2v_{cross} + 2$ vertices, we

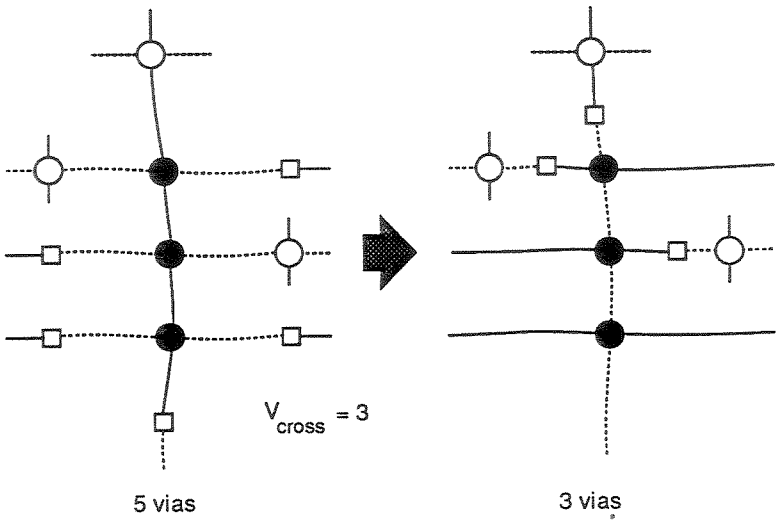


Figure 4.9: Via Reduction around Cross Vertices

can reduce the number of vias by changing the layer assignment of the edges incident on vertices of the cluster as shown in the right part of Figure 4.9. In other words, if there are v_{via} vias ($v_{via} > v_{cross} + 1$), and $2v_{cross} + 2 - v_{via}$ cross vertices that are adjacent to the cluster, the number of vias can be reduced to $2v_{cross} + 2 - v_{via}$, which is smaller than v_{via}.

We start with $v_{cross} = 1$ and scan the topological graph to find such a situation. If it is found, the switching of the layer assignment takes place. We continue the operation with $v_{cross} = 2$, 3, 4, and so on. We usually cut off at $v_{cross} = 6$.

4.5 Time Complexity

The following steps are different from those in the original topological channel routing algorithm described in Section 3.1 : the net assignment step, the topological graph construction step, and the via reduction step. We show the time complexity of these steps.

- **Assignment of Nets to Layers :**

  Near-vertical nets are assigned to the second layer. The shorter the the horizontal distance d_H of a net is, the earlier it is assigned to the second layer. So n nets need to be sorted by d_H, which takes $O(nlog(n))$ time. Checking if a chosen net does not cross k already-assigned nets takes time $O(k)$. So, it takes $\sum_{k=1}^{n} O(k) = O(n^2)$ time for checking n nets. More nets can be assigned to the second layer as shown in the step 2 of Figure 4.2, which can be done in $O(n^2)$ time also. The remaining nets are assigned to the first layer. So, the total time complexity of the net assignment step is $O(nlog(n)) + O(n^2) + O(n^2) = O(n^2)$.

- **Construction of Topological Graph :**

  When a topological graph is constructed, a region graph is constructed. The improved algorithm imposes some rules so that some region edges of the region graph cannot be used when a shortest path algorithm is performed. Since the rule checking is done using parameters stored in the region edge, it can be done in $O(1)$ time. So, the time complexity to construct a topological graph is the same as the one in the original algorithm, $O(n^3)$.

- **Via Reduction :**

  Two types of local layer assignment modifications are done to reduce the number of vias : one is local layer assignment modification at pin vertices, and the other is local layer assignment modification at other vertices.

  The local layer assignment modification at pin vertices can be done in $O(n)$ time, because there are $2n$ pin vertices and each pin vertex can be processed in $O(1)$ time. The local layer assignment modification at other

vertices is done by scanning clusters of c vertices of a topological graph, where c is smaller than a given constant. The number of vertices of a topological graph for n nets is $O(n^2)$, so this scanning takes $O(n^2)$ time. So, the total time complexity of via reduction is $O(n) + O(n^2) = O(n^2)$.

Chapter 5

Experimental Results on Topological Channel Routing

After a topological solution is obtained, it is mapped to an interleaving mesh. We used the same design rules as those in [12] : path width = 1.0, feature separation = 1.0, and size of contact = 2.0 x 2.0. The space between two adjacent pins is 4.0.

5.1 Results on Channel Height

Experiments have been performed using examples 3a, 3b, 3c, 4b, and 5 in Yoshimura and Kuh's paper [50], and Deutsch's Difficult Example [11]. Table 5.1 shows the channel heights for those examples. Maximum densities and minimum heights achievable by reserved layer routers are also shown. When maximum density is d, d wires have to cross at certain column of a channel on one layer if a reserved layer channel routing is performed. Since wire width is 1.0 and wire separation is 1.0, the minimum height by reserved layer routers is $2d + 1$. Note that for examples 4b and 5, our topological channel router was able to obtain heights smaller than the minimum heights by reserved layer routers, because it allows horizontal wires on both layers.

Compaction (or shaking) is very effective in obtaining above channel heights. Figure 5.1 shows channel height results after 20 shakes for examples 3a, 3b, 3c, 4b, and 5, and after 32 shakes for Deutsch's Difficult Example.

Examples	Maximum Density (= d)	Minimum Height of Reserved Layer Router (= 2d+1)	Height by Our Topological Channel Router
Example 3a	15	31	31
Example 3b	17	35	38
Example 3c	18	37	37
Example 4b	17	35	33
Example 5	20	41	33
Deutsch's Difficult Example	19	39	43

Table 5.1: Channel Height by Topological Channel Router

The height when the number of shakes is 0 is the one obtained by a simple one-dimensional compaction without any horizontal movement of vias. This one-dimensional compaction result is used as input for the shaking procedure that follows. The experiments show that the first shake reduces the height drastically, because a two-dimensional compaction is performed after the one-dimensional compaction. During the shaking, a result with the smallest height is saved until an even smaller result is obtained. The smallest height is usually obtained after 3 to 7 shakes. The exceptions are 15 shakes for example 3b and 26 shakes for Deutsch's Difficult Example. It is hard to predict how many shakes are needed beforehand. But obviously only one shake is not enough. At least several shakes are needed to obtain a height close to the best obtainable heights. If enough computation time is available, it is best to perform as many shakes as the time permits.

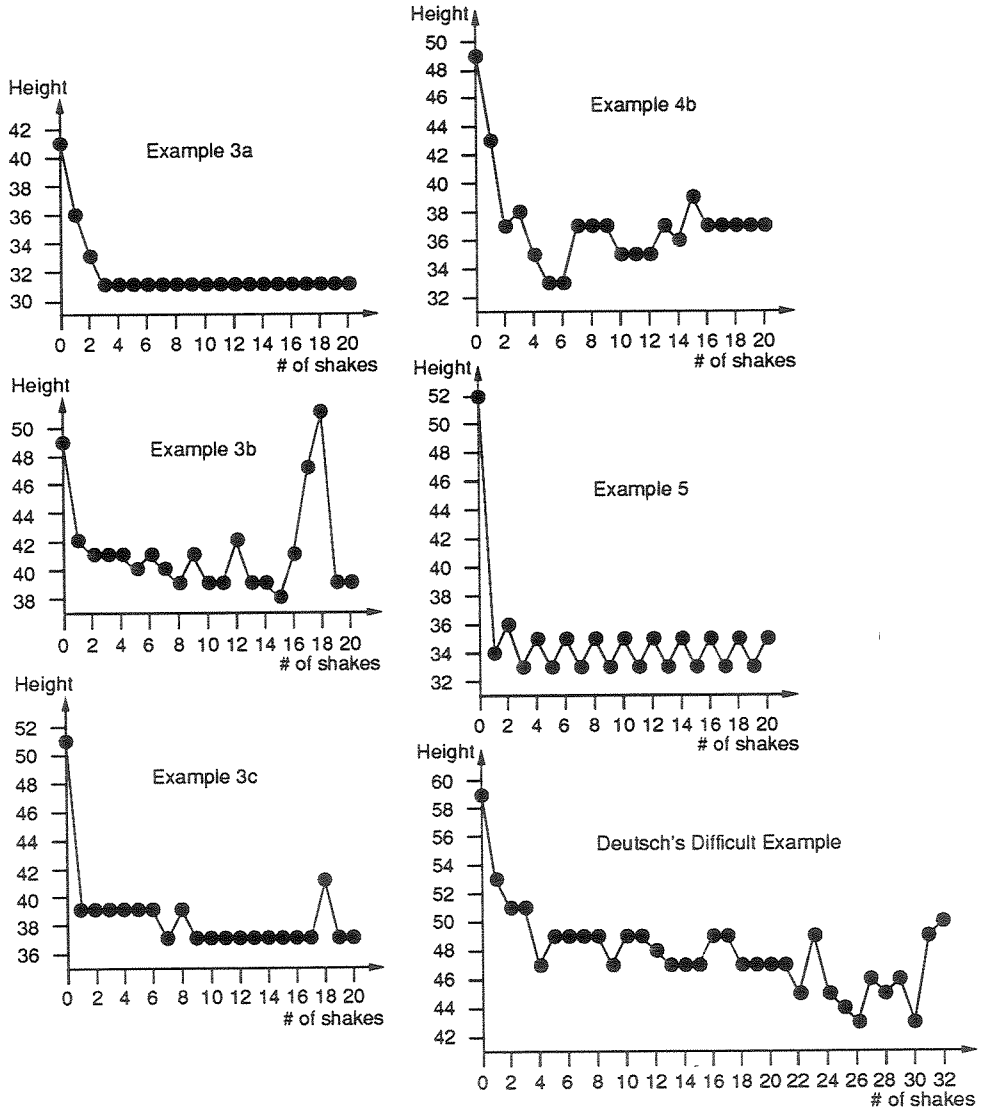


Figure 5.1: Channel Height after n shakes ($0 \leq n \leq 32$)

5.2 Results on the Number of Vias

Table 5.2 shows a comparison of the numbers of vias used by four different methods. Both the table and its graph are shown. The left-most column shows the numbers of vias in results on Examples 3a, 3b, 3c, 4b, and 5 in the original paper by Yoshimura and Kuh [50], and three different results on Deutsch's Difficult Example by Burstein and Pelavin [5] and by Yoshimura and Kuh [50]. The second column shows the numbers of vias by the optimum CVM (Constrained Via Minimization) algorithm of Chen, Kajitani, and Chan [6]. The CVM algorithm finds the minimum number of vias but does not change the topology of the layout. The third column shows the numbers of vias by an algorithm "Via Minimization by Layout Modification" by The, Wong, and Cong [46]. It can obtain a result with smaller a number of vias than the CVM algorithm because the topology of layouts is locally modified. The right-most column shows the numbers of vias by our method. On the average, our results have numbers of vias that are 39% smaller than those in [5] [50], 25% smaller than those by the optimum CVM, and 17% smaller than those by The, Wong, and Cong. The number of vias in the original papers are much larger than the rest, obviously because they use a reserved layer model in which a via is always needed when a wire makes a 90 degree turn. Both CVM and the algorithm by The, Wong, and Cong tried to minimize the number of vias after the layout is obtained. Ours achieved a smaller number of vias than the above two algorithms because it minimizes the number of vias in a topological graph before it is mapped onto a channel.

Examples		The Number of Vias in the Original Papers	Optimum CVM	Via Minimization by Local Modification by The, Wong and Cong	Our Method
Example 3a		91	72	66	42
Example 3b		107	91	78	69
Example 3c		125	109	103	83
Example 4b		179	—	—	86
Example 5		150	114	105	84
Deutsch's Difficult Example	Tracks=19 by Burstein and Pelavin	354	263	233	186
	Tracks=20 by Yoshimura and Kuh	308	240	218	
	Tracks=28 by Yoshimura and Kuh	290	234	207	

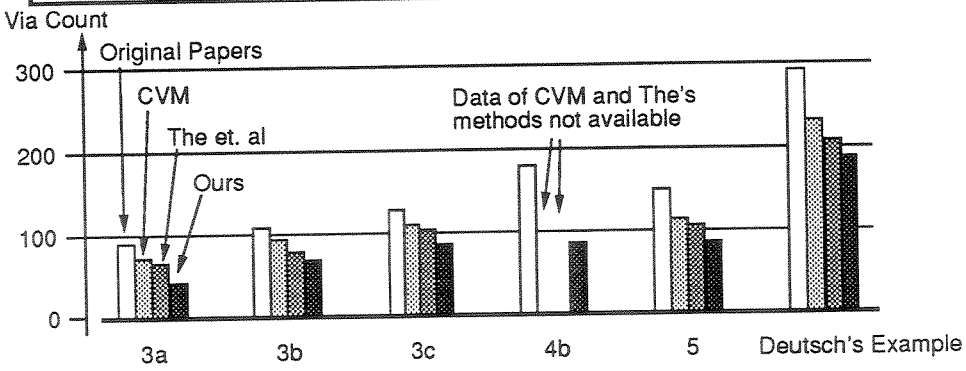


Table 5.2: Comparison of the Number of Vias by Different Algorithms

5.3 Comparison of Topological Channel Router with and without Geometric Information

Table 5.3 shows the comparison of the topological channel router without geometric information described in chapter 3 and the topological channel router with geometric information described in chapter 4 using examples 3a, 3b, 3c, 4b, 5 in Yoshimura and Kuh's paper [50] and Deutsch's Difficult Example [11]. All the results except Example 5 obtained by the topological channel router with geometric information have smaller (or equal) heights than those by the topological channel router without geometric information. In most cases, the topological channel router with geometric information produced results with smaller numbers of vias. In Example 4b, the topological channel router with geometric information obtained the solution of height 33 and via count 101 as shown in Table 5.3. However the same algorithm also obtained the solution of height 36 and via count 86. The smaller height of 33 was obtained at the cost of the larger number of vias.

We obtained the height of 43 [1] for Deutsch's Difficult Example [11] without any parallel overlaps of wires, which is one of the best results ever reported. The number of vias for Deutsch's Difficult Example in our result is 186, which is much smaller than all the other channel routers. Comparison of Deutsch's Difficult Example by different algorithms is shown in Section 5.5. Note that all the compacted results on this example previously reported including [12] [35] [7] have vertical and/or horizontal overlaps of wires.

[1]The algorithm produced a solution with height 43, which was easily modified by hand to produce the solution of height 41 shown in Section 5.6

Examples		Our Experimental Results: Topological Router without Geometric Information	Our Experimental Results: Topological Router with Geometric Information
Example 3a	Height	31	31
	Via count	46	42
Example 3b	Height	38	38
	Via count	87	69
Example 3c	Height	38	37
	Via count	94	83
Example 4b	Height	36	33
	Via count	86	101
Example 5	Height	33	35
	Via count	94	84
Deutsch's Difficult Example	Height	45	43
	Via count	222	186

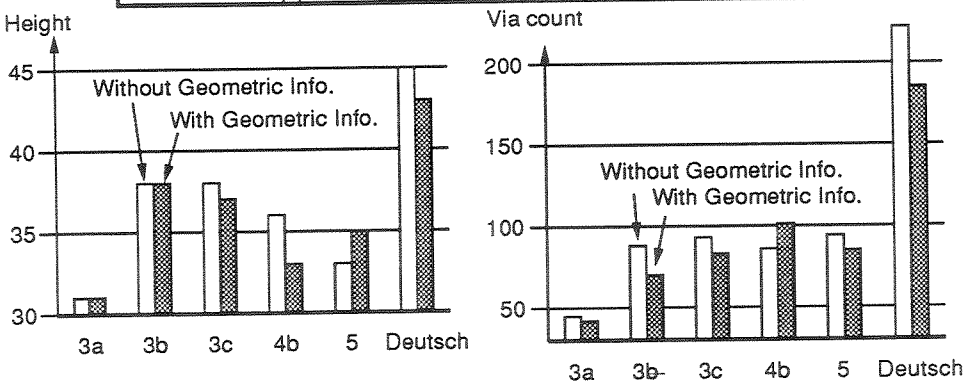


Table 5.3: Comparison of Topological Router without Geometric Information and Topological Router with Geometric Information

5.4 Effect of Pin Pitch Expansion on Channel Height

In our channel model, both vertical and horizontal wires are allowed to be placed on both layers. If horizontal wires are placed on both layers, the height of a channel will be lower. However, vias in a channel often prevent the horizontal wires on different layers from being placed in interleaving way. The horizontal wire interleaving by two layers will work better if there is more horizontal space available, because vias can be pushed to a lower position. If we increase the pin pitch, i.e. spacing between two adjacent pins, we can have more horizontal space. So we did an experiment on the effect of pin pitch expansion on channel height. The pin pitch expansion ratio (PE) is a scale up ratio of pin spacing. For example, if PE is 2, the spacing between two adjacent pins is twice as large as when PE is 1 as shown in Figure 5.2. Six inputs 1, 2, 3, 4, 5, 6 are used for testing. Pin signal numbers are randomly generated. The maximum densities (d) of inputs 1, 2, 3, 4, 5, 6 are 10, 10, 10, 19, 20, 20, respectively. If the wire width is 1 and the wire spacing is 1, the minimum height of a reserved layer router is $2d + 1$, which is 21, 21, 21, 39, 41, 41 for inputs 1, 2, 3, 4, 5, 6, respectively. The number of pins on each side of the wall of inputs 1, 2, 3, 4, 5, 6 are 20, 26, 31, 39, 49, 53, respectively. Our topological channel router allows horizontal wires on both layers, so theoretically it can achieve height which is smaller than the minimum height of a reserved layer router. Figure 5.3 shows that when PE is 1, it is hard to reach this minimum height. However, the experiment shows that, as PE increases, the height decreases and approaches the minimum, and in many cases the height is smaller than the minimum.

We also compared the height by the channel router without geometric information with the height by the channel router with geometric information. The height by a channel router without geometric information is drawn in

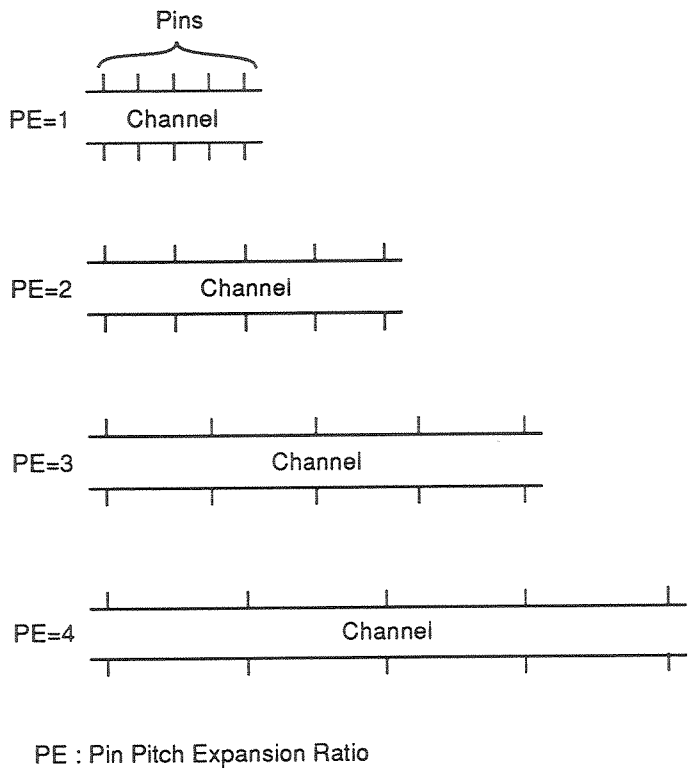


Figure 5.2: Expansion of Pin Pitch

solid lines, and the height by a channel router with geometric information is drawn in dotted lines in Figure 5.3. When PE is 1, vias must be crammed into a horizontally short channel. If geometric information is used, the wires are arranged in a more controlled manner. As a result, the height of a channel by the channel router with geometric information is smaller than the height of a channel by the channel router without geometric information in most cases except the case of input 4. However, as PE increases, there is enough space in the channel so that vias do not collide with each other. So, when the pin pitch is large, a channel router without geometric information is able to produce results with about the same height as a channel router with geometric information.

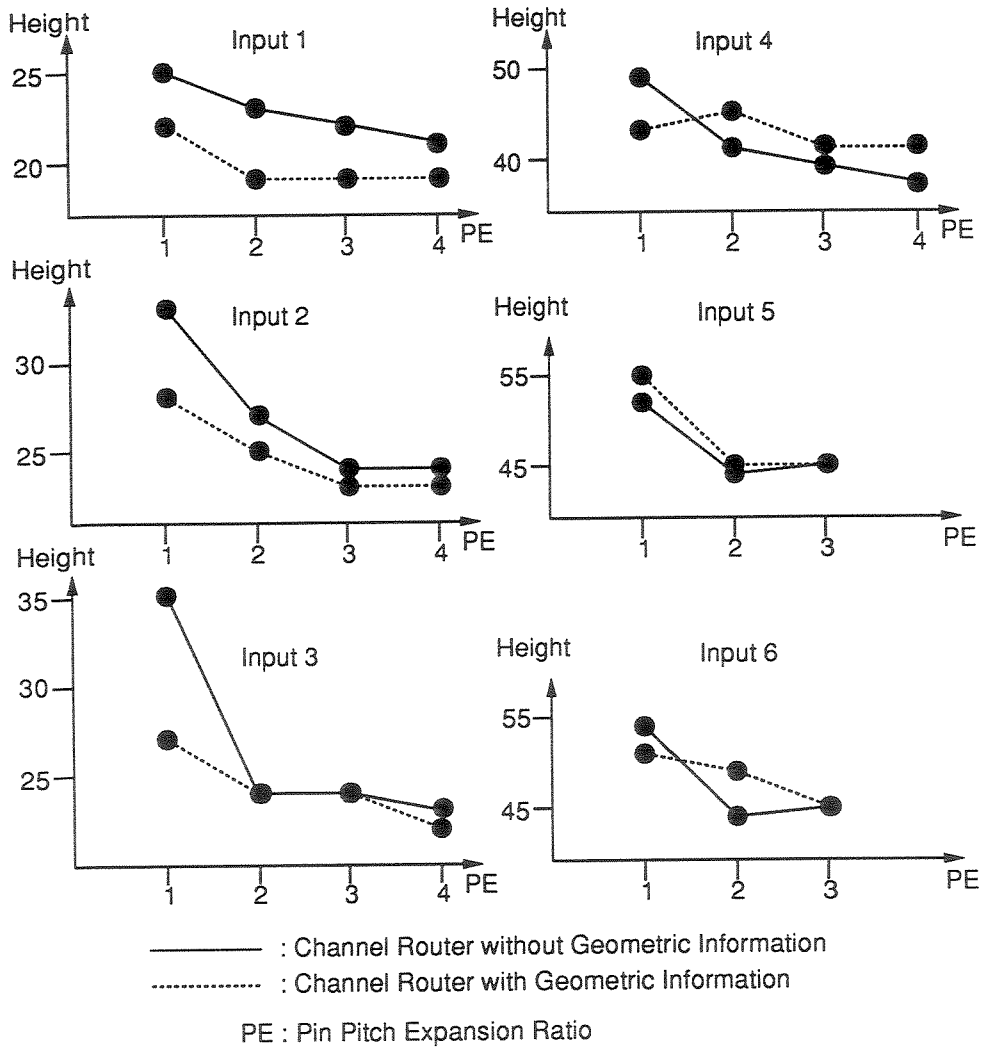


Figure 5.3: Effect of Pin Pitch Expansion on Channel Height

5.5 History of Deutsch's Difficult Example

Figure 5.4 shows the history of the channel height and the number of vias by different algorithms on Deutsch's Difficult Example. Each point is the height and the number of vias obtained by an algorithm whose inventors are listed on the right side. For details of these algorithms, refer to following papers. A : Deutsch [11], B : Yoshimura and Kuh [50], B : Rivest and Fiduccia [33], C : Burstein and Pelavin [5], D : Sangiovanni-Vincentelli, Santomauro, and Reed [37], E : Deutsch [12], F : Royle, Palczewski, VerHeyen, Naccache, and Soukup [35], G : Cheng and Deutsch [7], and H : Ours [15] [16]. The channel height has been decreasing steadily because that is the main objective of a channel router. However, the numbers of vias by some older channel routers are smaller than those by some newer channel routers. This is because they usually obtained results with small height as a main objective, and then some tried to minimize the number of vias as a secondary objective, while others did not try to reduce the number of vias at all because they stuck with the reserved layer model. Our algorithm produced the result with both the smallest height (= 41) and the smallest number of vias (= 186) partly due to the new topological channel routing with geometric information that minimizes the number of vias even before wires are laid out on a channel, and partly due to a sophisticated compaction algorithm.

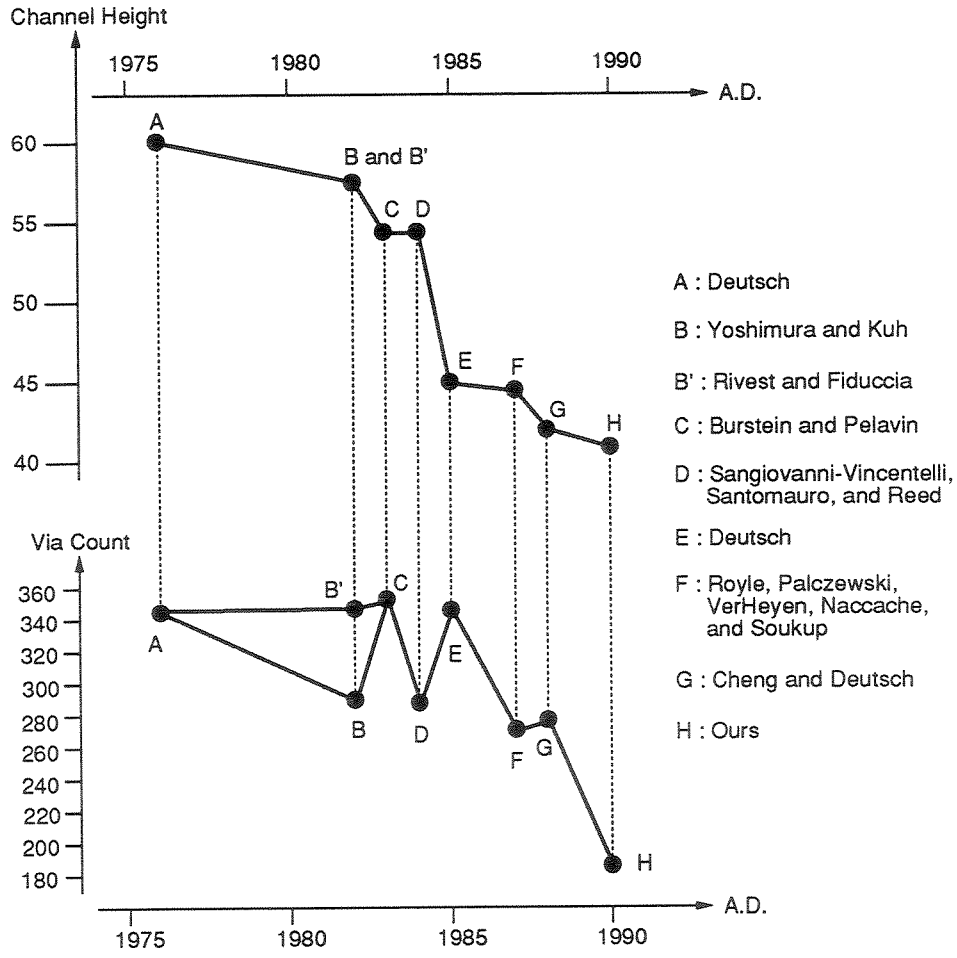


Figure 5.4: History of Channel Height and the Number of Vias by Different Channel Routers on Deutsch's Difficult Example since 1976

5.6 Layout Results

The best results obtained by our topological channel router are shown here. Deutsch's Difficult Example and Five examples from Yoshimura and Kuh's paper [50] are used as inputs.

5.6.1 Deutsch's Difficult Example

The result of Deutsch's Difficult Example has height of 41 and via count of 186.

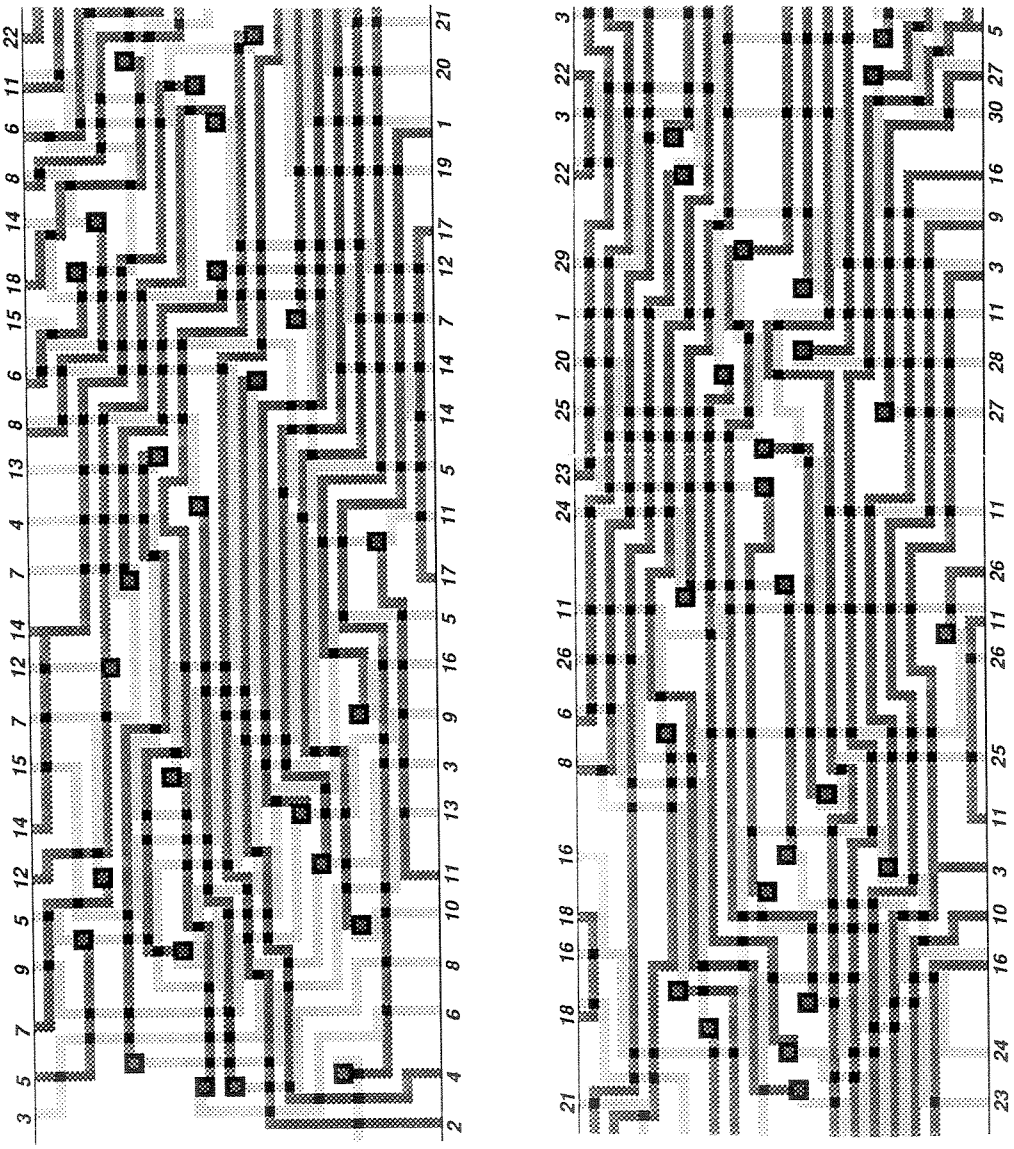


Figure 5.5: Layout Result of Deutsch's Difficult Example

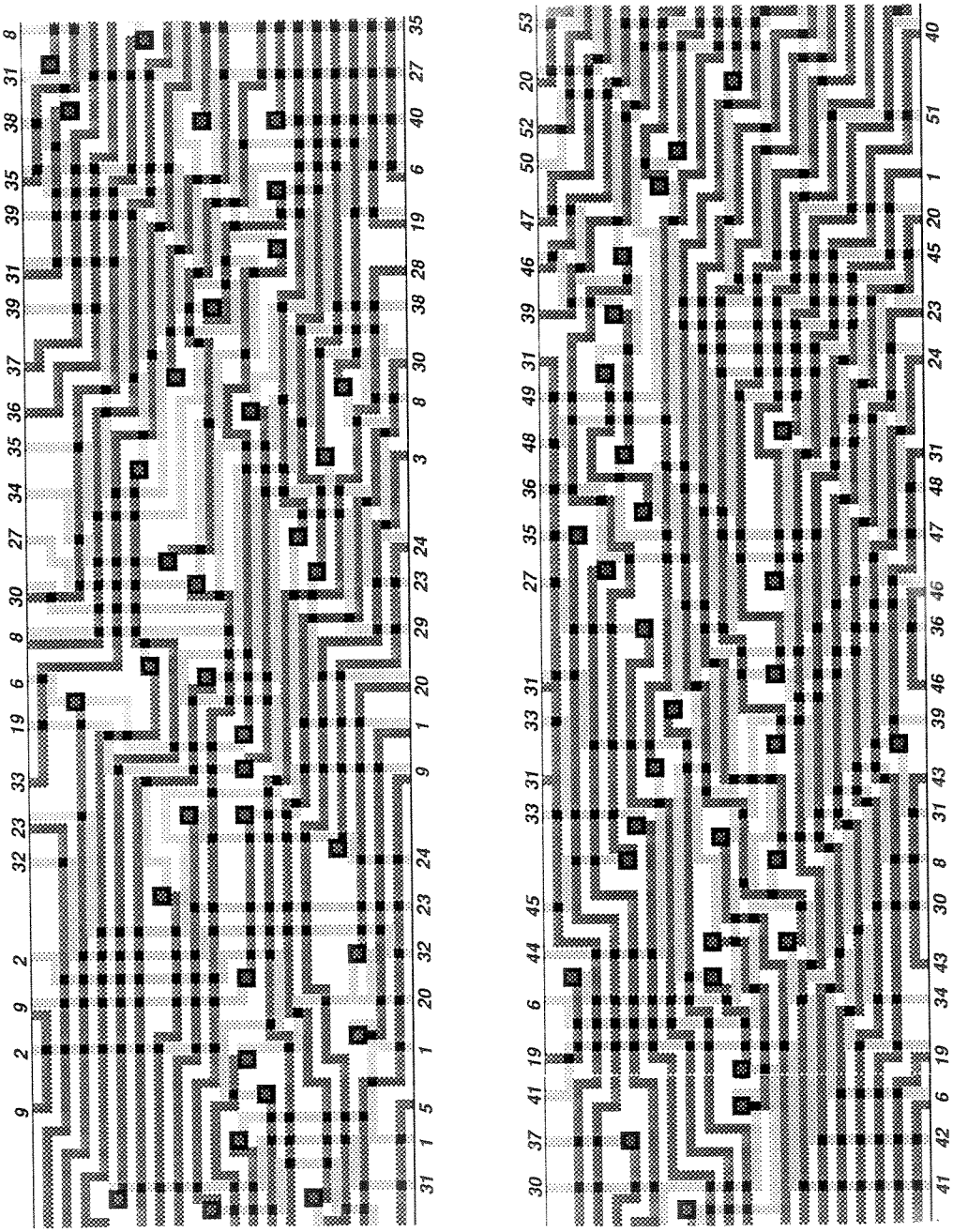


Figure 5.5 (continued)

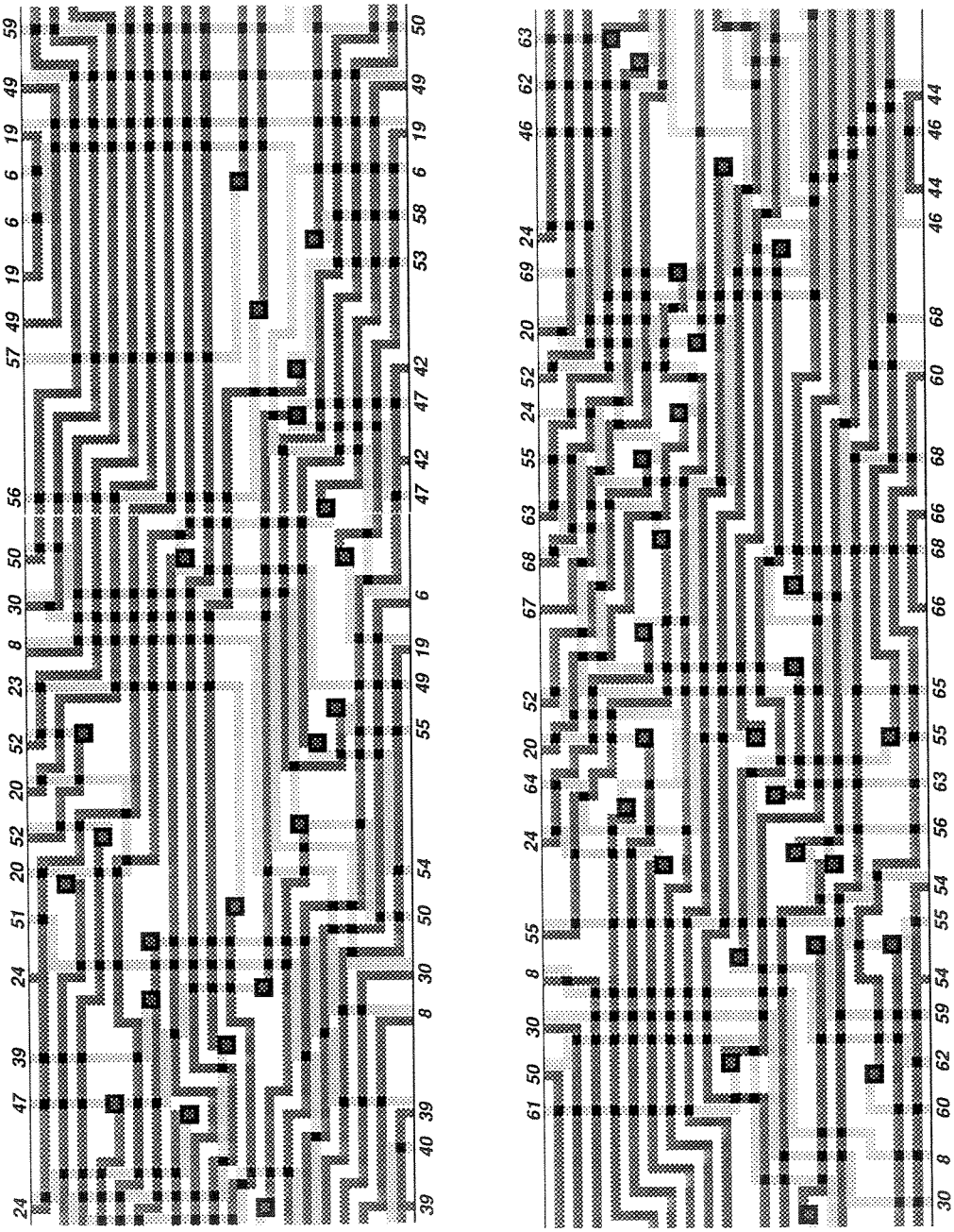


Figure 5.5 (continued)

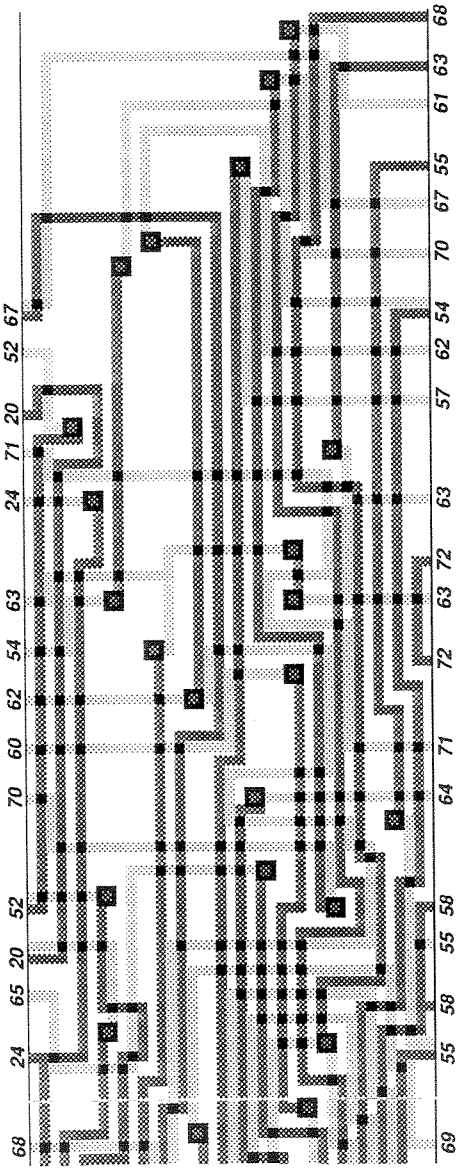


Figure 5.5 (continued)

5.6.2 Yoshimura and Kuh's Example 3a

The result of Yoshimura and Kuh's Example 3a has height of 31 and via count of 42.

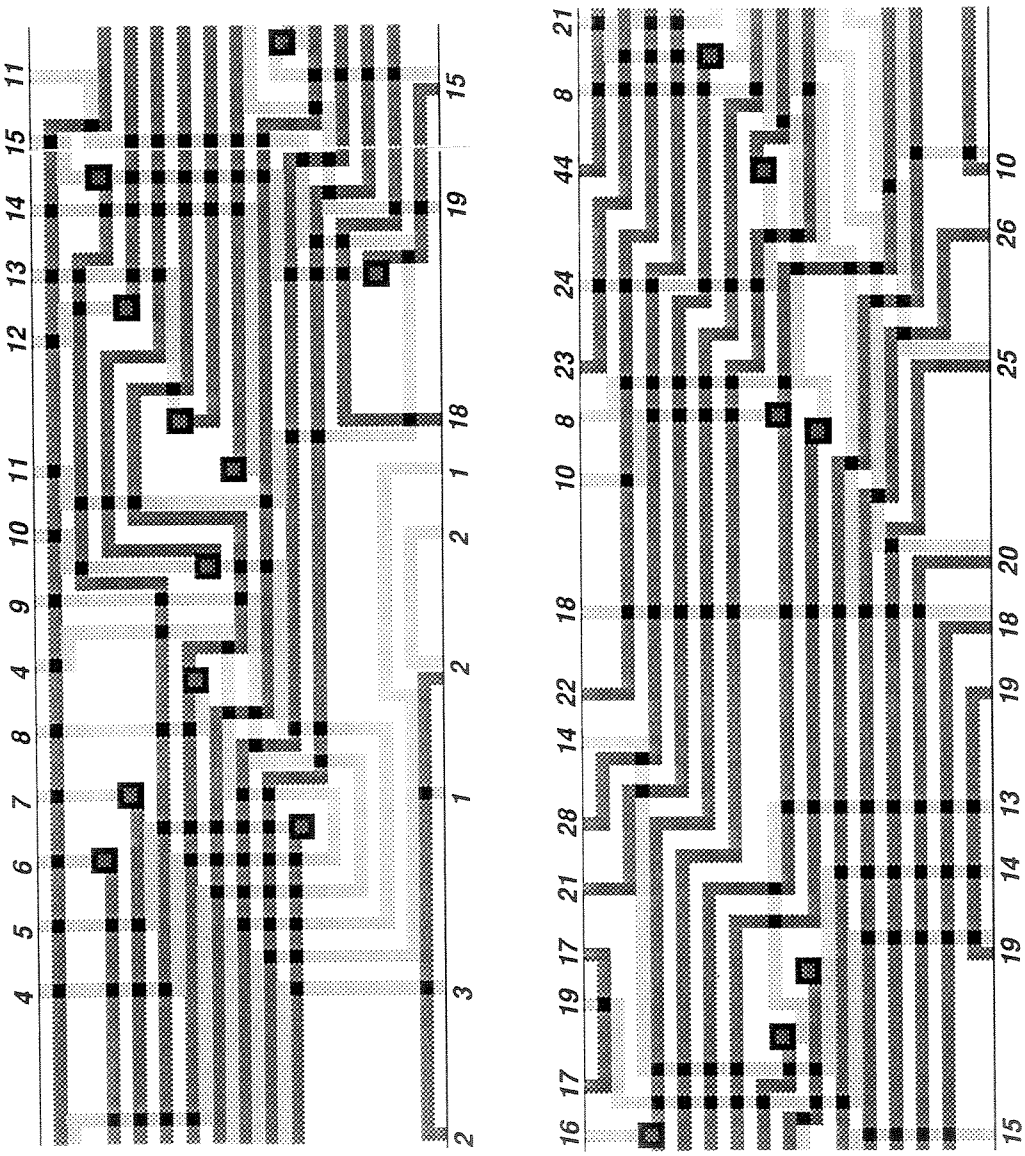


Figure 5.6: Layout Result of Yoshimura and Kuh's Example 3a

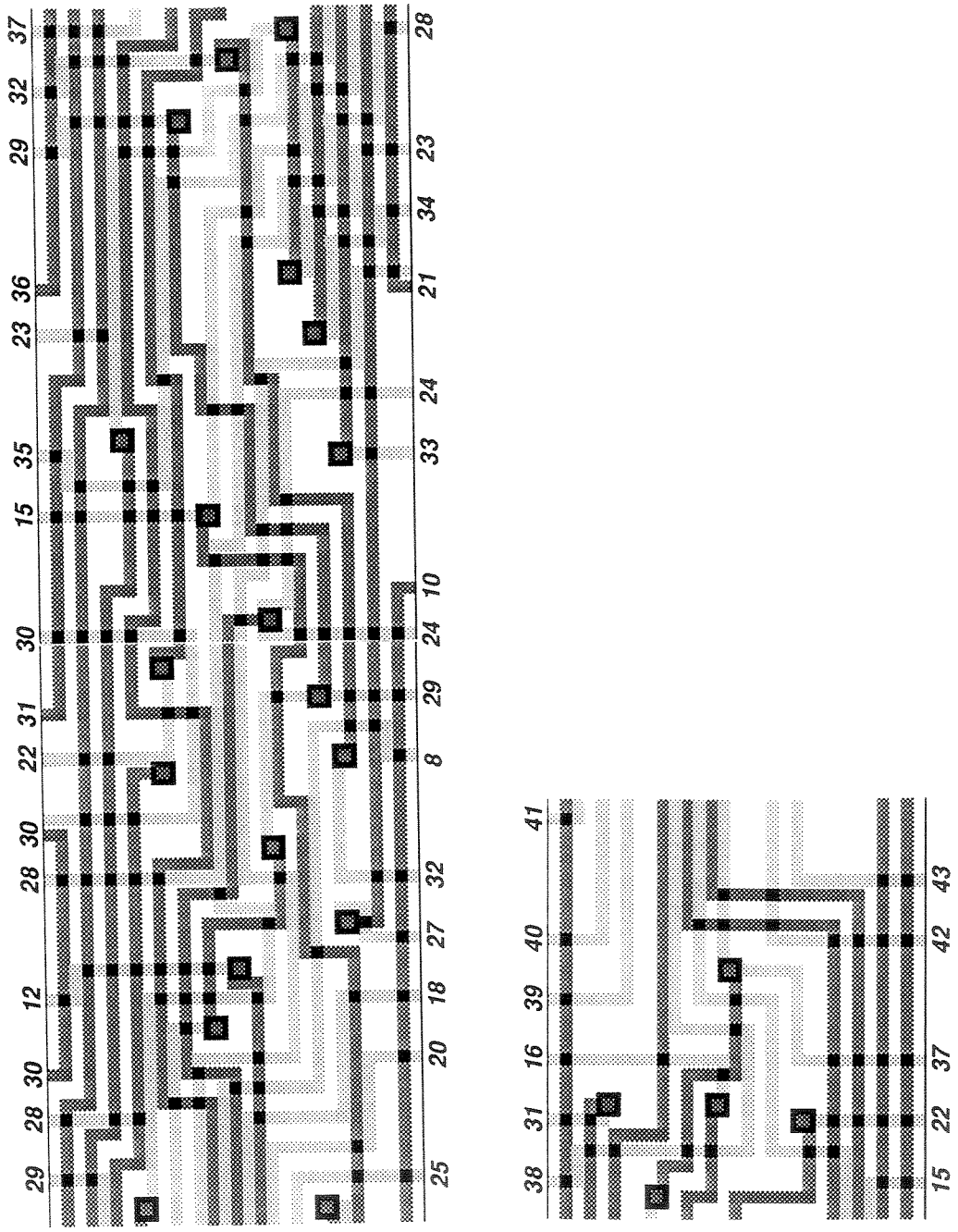


Figure 5.6 (continued)

5.6.3 Yoshimura and Kuh's Example 3b

The result of Yoshimura and Kuh's Example 3b has height of 38 and via count of 69.

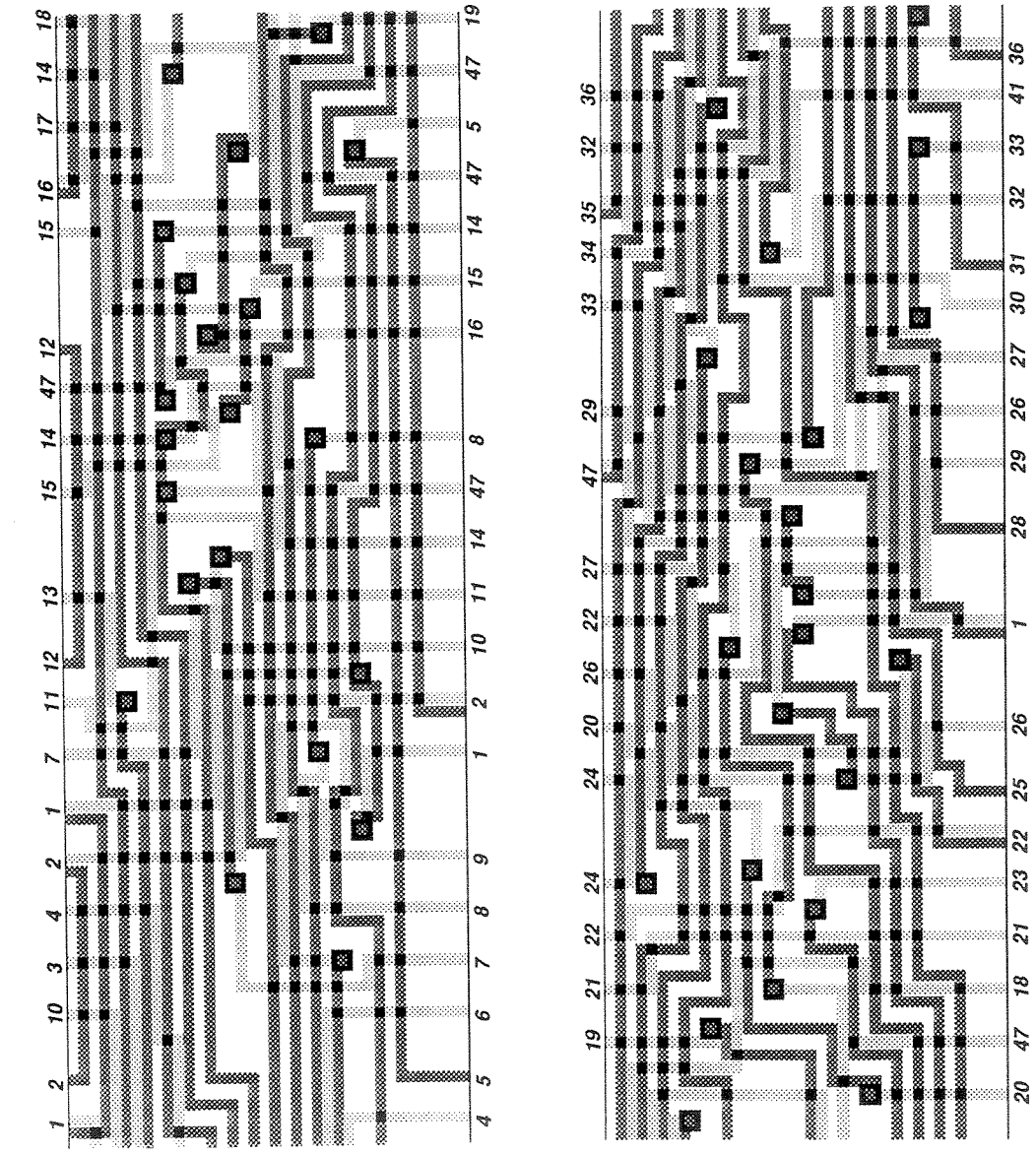


Figure 5.7: Layout Result of Yoshimura and Kuh's Example 3b

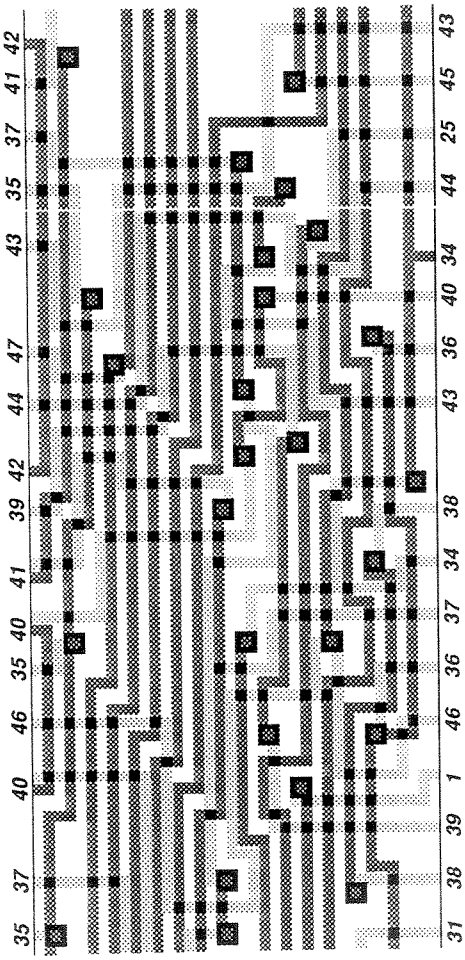


Figure 5.7 (continued)

5.6.4 Yoshimura and Kuh's Example 3c

The result of Yoshimura and Kuh's Example 3c has height of 37 and via count of 83.

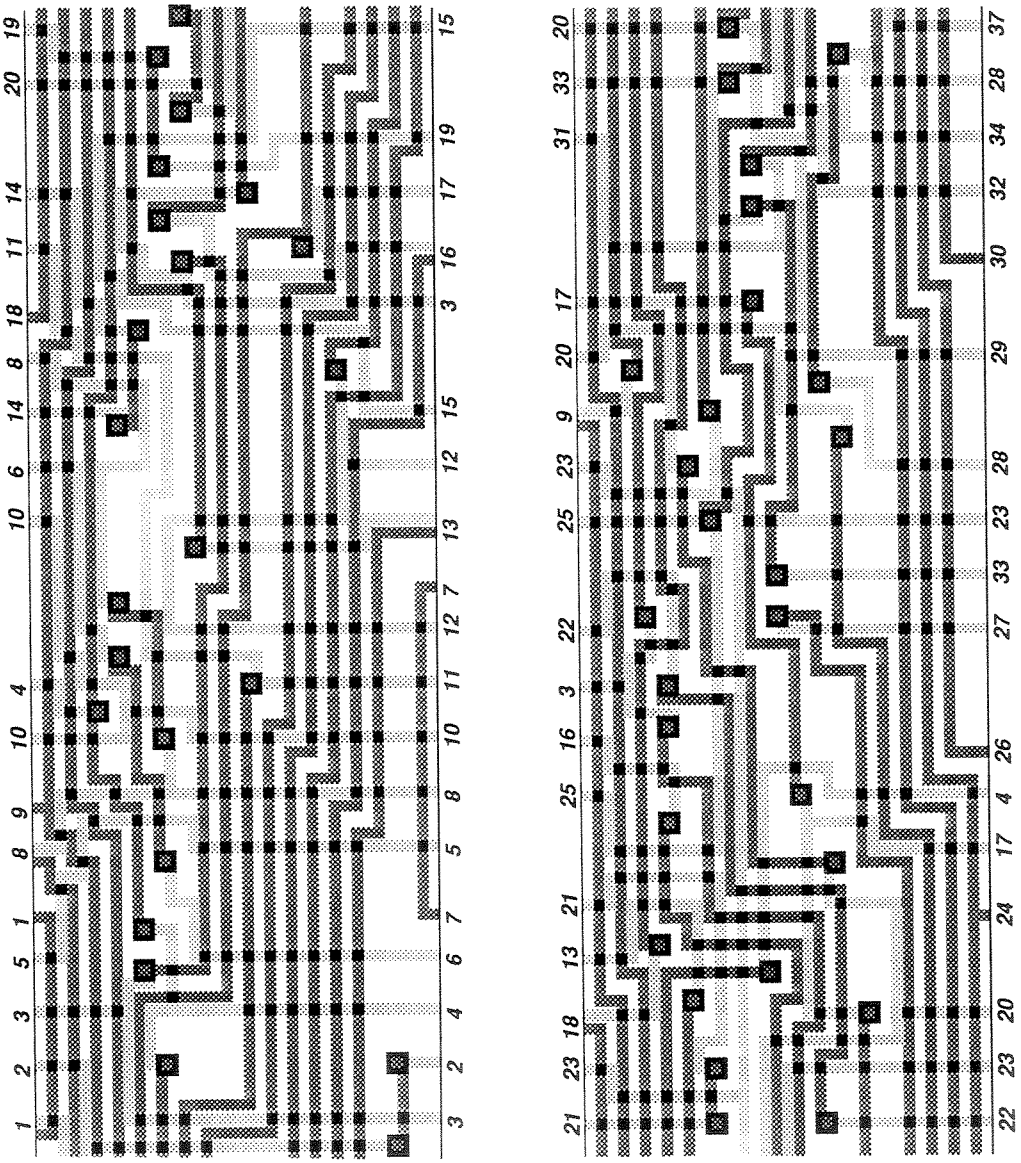


Figure 5.8: Layout Result of Yoshimura and Kuh's Example 3c

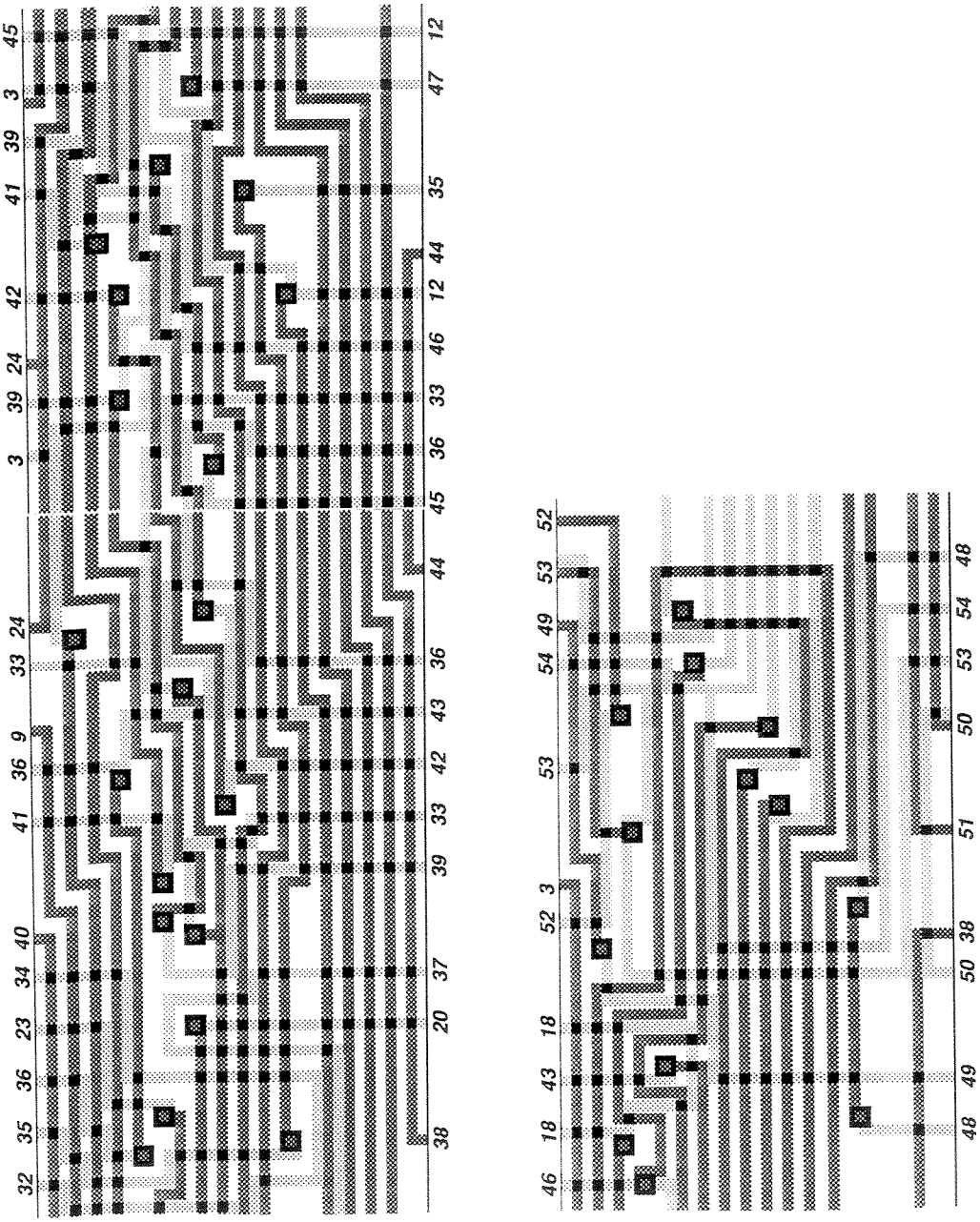


Figure 5.8 (continued)

5.6.5 Yoshimura and Kuh's Example 4b

The result of Yoshimura and Kuh's Example 4b has height of 33 and via count of 101.

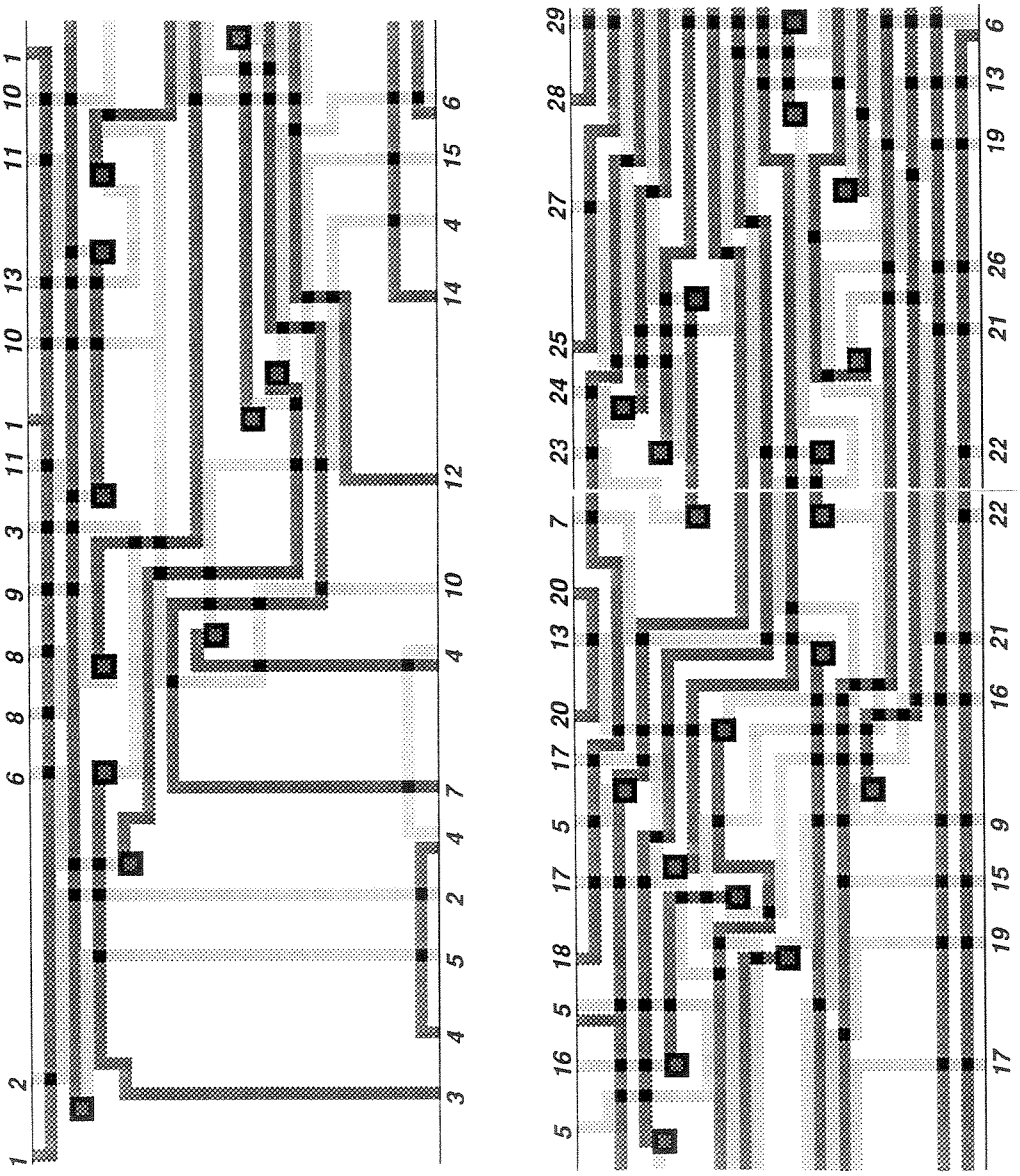


Figure 5.9: Layout Result of Yoshimura and Kuh's Example 4b

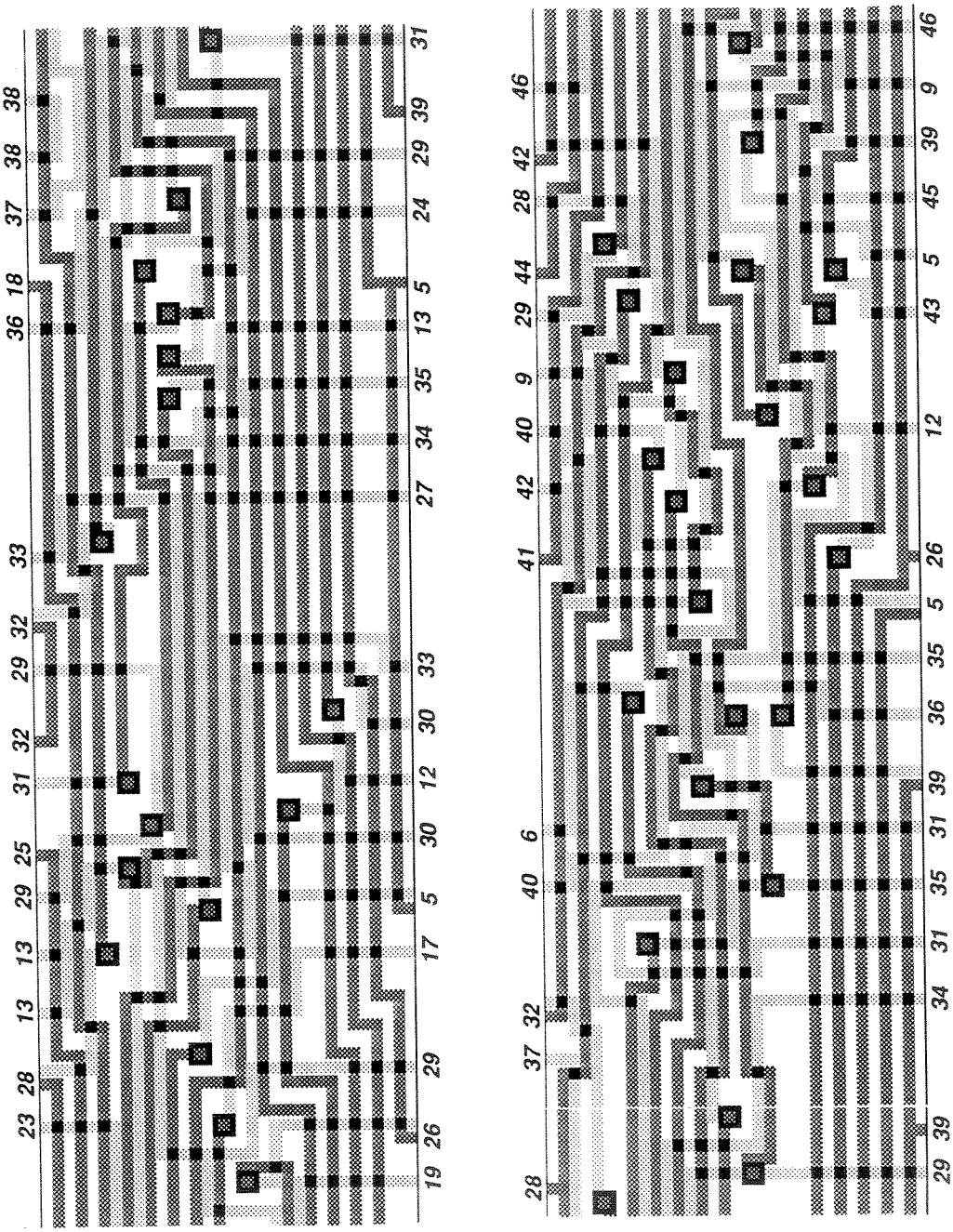


Figure 5.9 (continued)

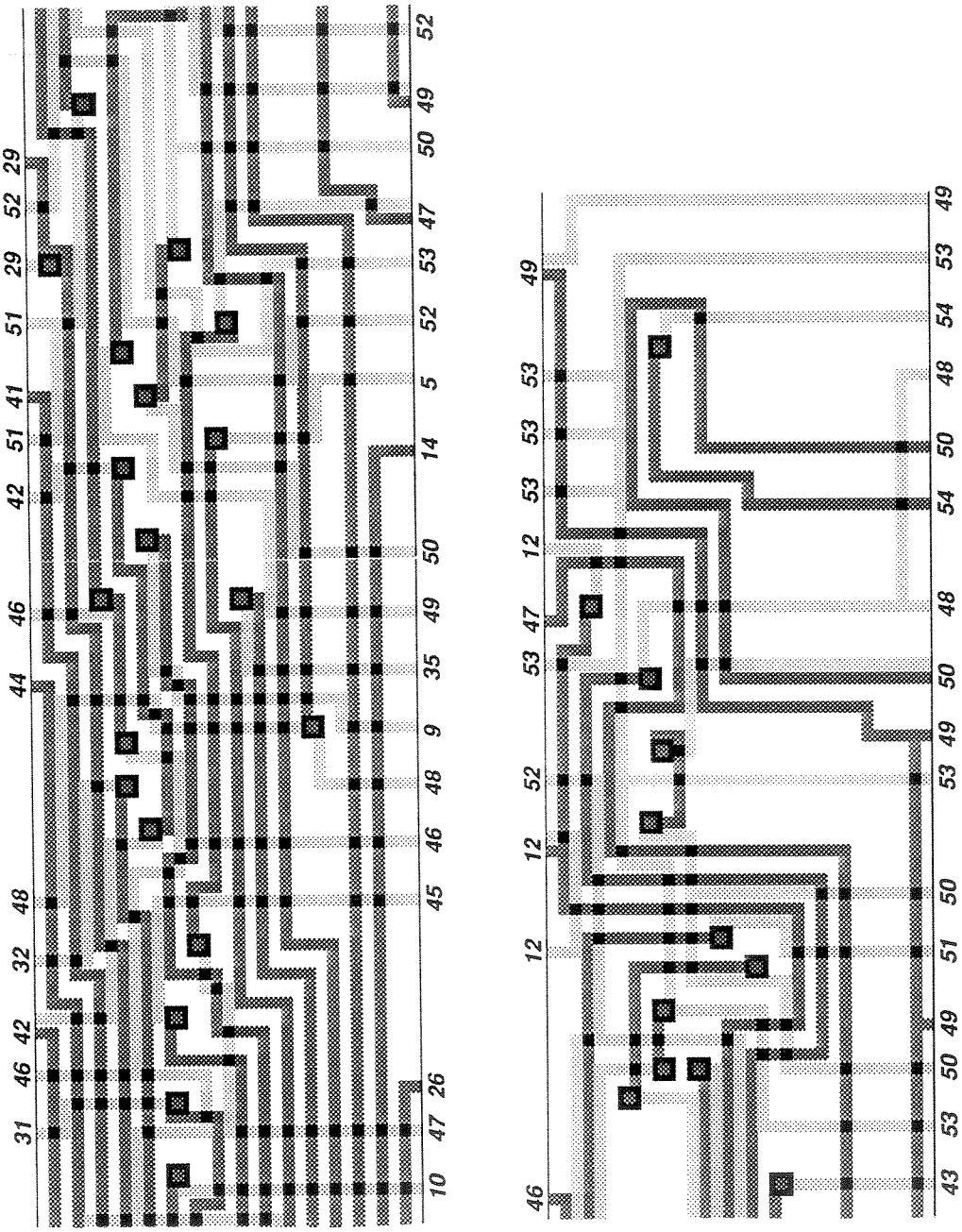


Figure 5.9 (continued)

5.6.6 Yoshimura and Kuh's Example 5

The result of Yoshimura and Kuh's Example 5 has height of 33 and via count of 94.

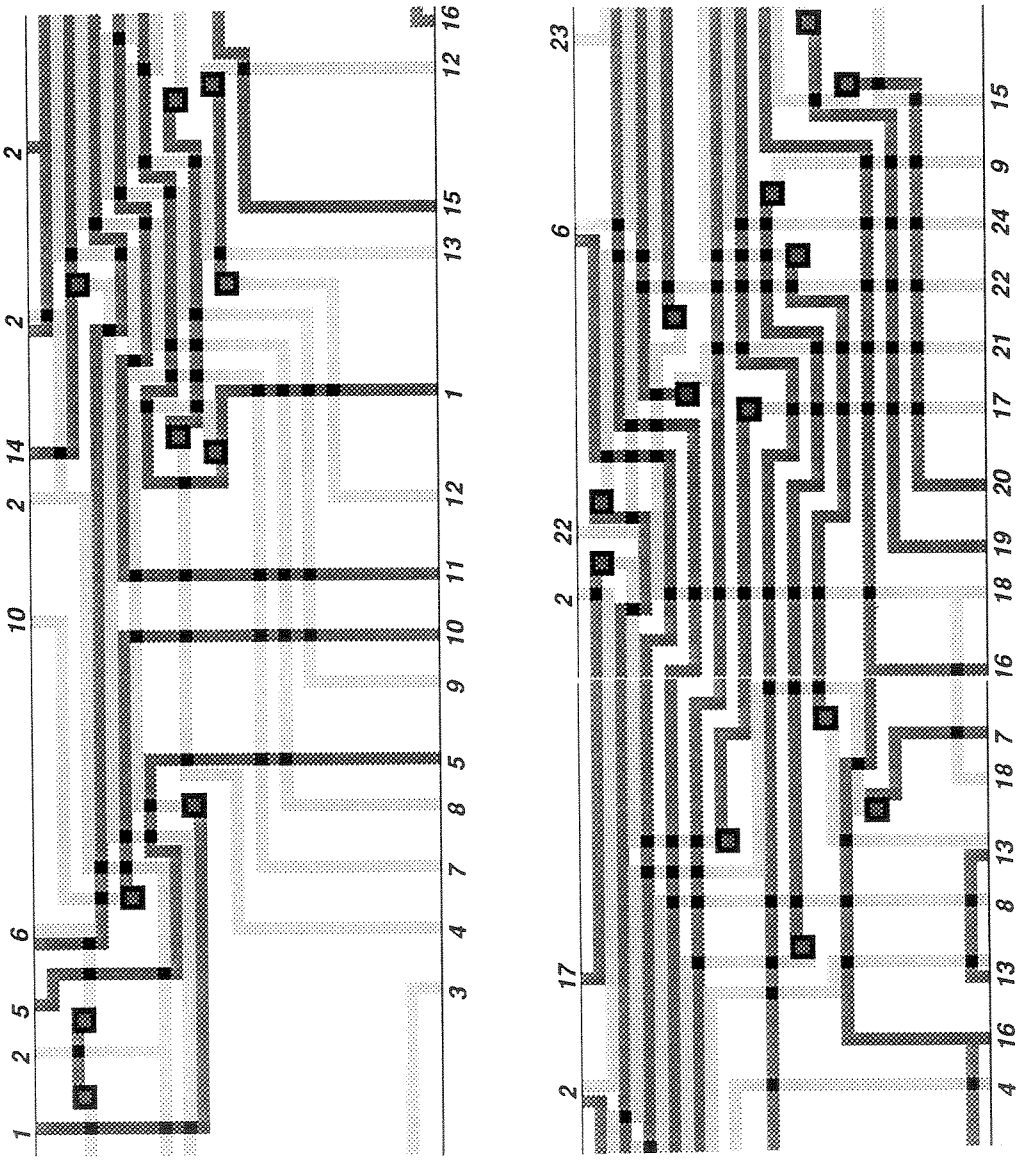


Figure 5.10: Layout Result of Yoshimura and Kuh's Example 5

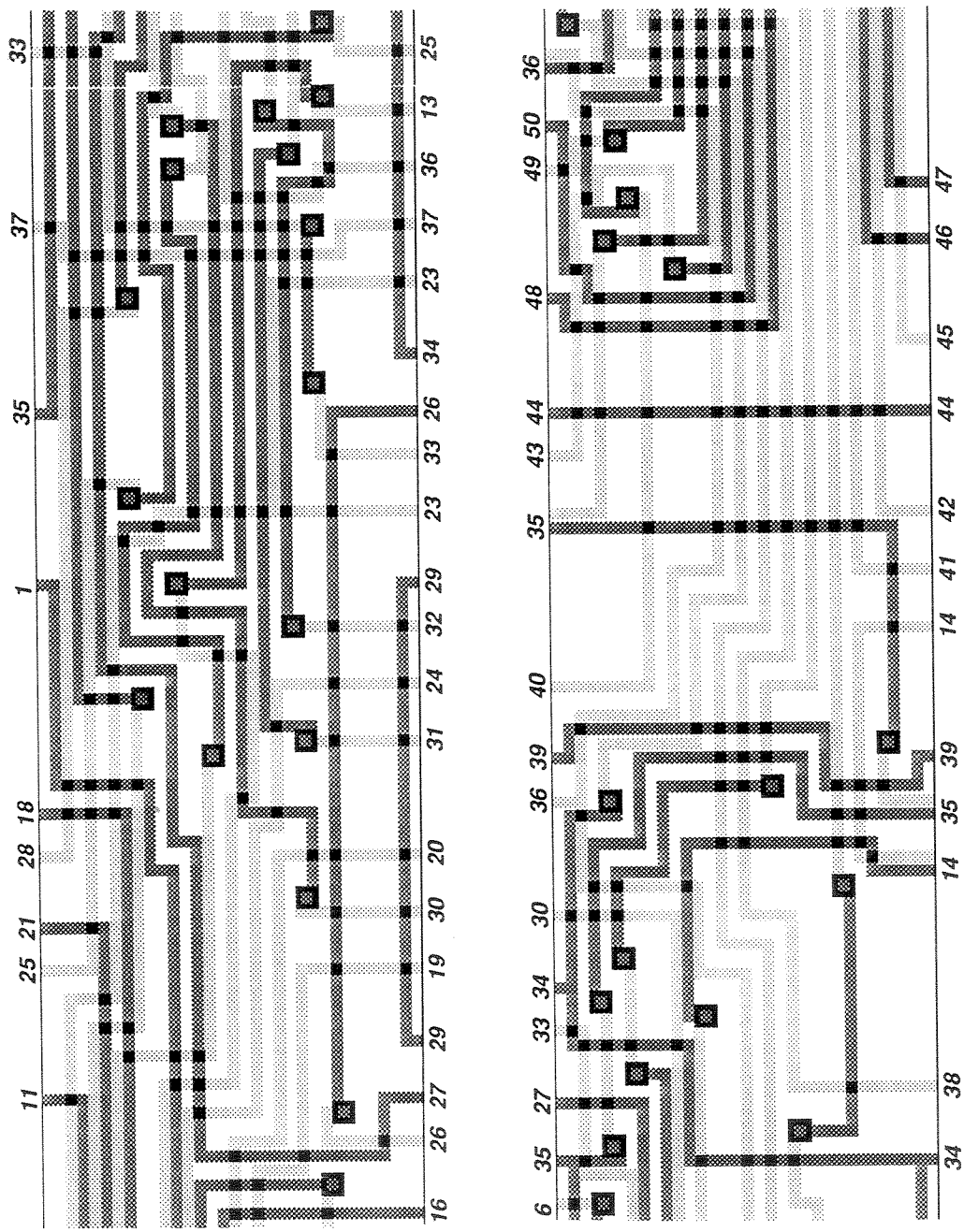


Figure 5.10 (continued)

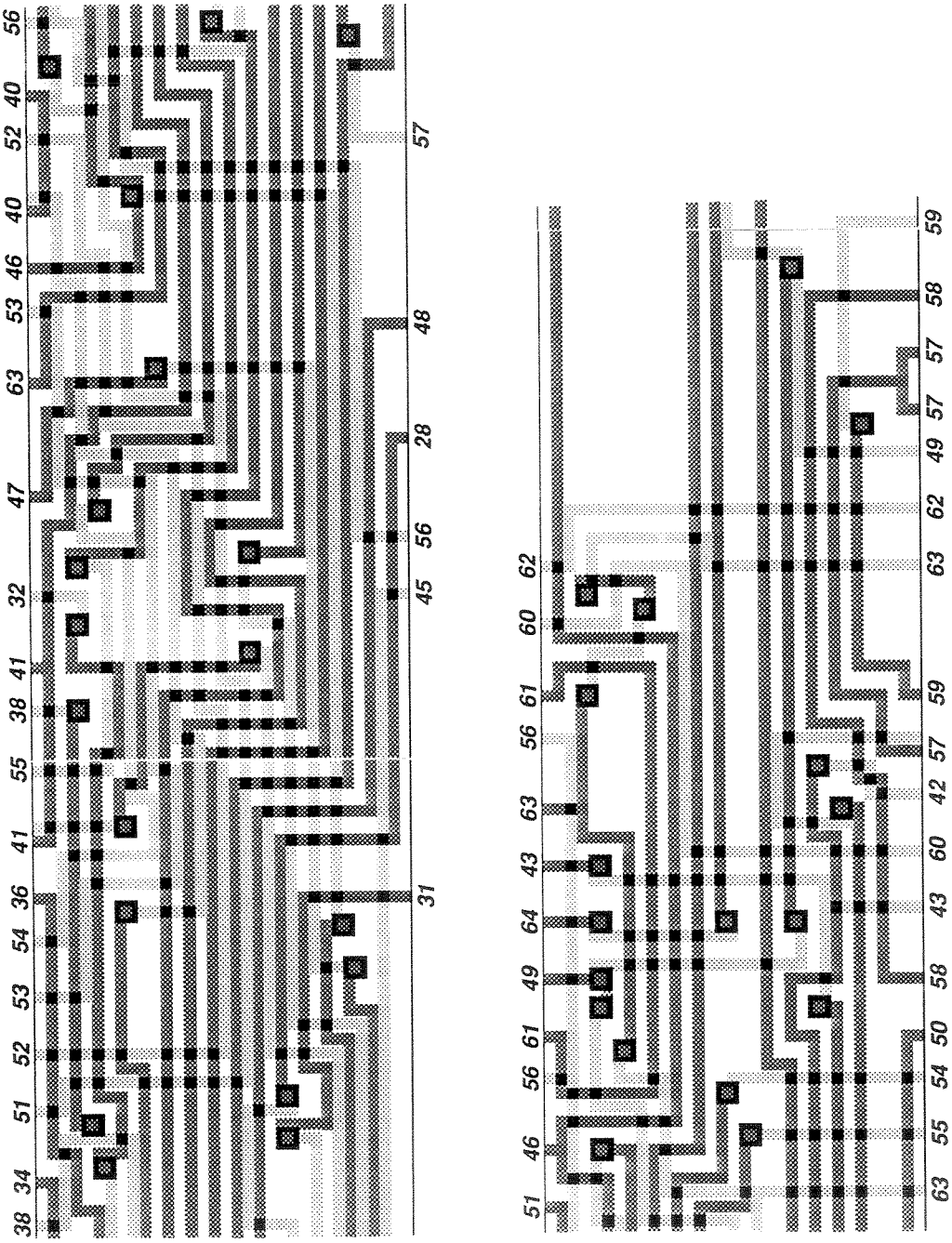


Figure 5.10 (continued)

Chapter 6

Area-Efficient Power Routing

6.1 Introduction

We have developed a new graph algorithm to route non-crossing VDD and GND trees on one layer VLSI designs [14]. The algorithm attempts to minimize the metal area of the trees under metal migration and voltage drop constraints. Experimental results show that the power wire area is considerably smaller than a previously developed method for single-layer routing.

Routing VDD and GND nets on one metal layer for arbitrarily-shaped macrocells needs special consideration because the nets are not allowed to cross each other and the width of the wire is dependent on the current flow.

In routing signal nets, minimizing the wire length is an appropriate goal, because shorter wires have shorter signal delays. In case of VDD and GND nets, however, it is not important to minimize the total wire length, since no signals run through them. Instead, wire area should be minimized, so that either the total chip area can be reduced, or the free area can be used for other purposes such as signal wire routing, depending on placement model.

This paper describes a method for routing non-crossing VDD and GND trees on one layer which tries to minimize the chip area devoted to power routing under metal migration and voltage drop constraints. The metal migration has to be prevented by having wide enough metal wires. Besides, the voltage drop has to be kept small, because a large voltage drop between a pad and a macrocell decreases switching speed and noise margin. The algorithm

also takes the width of channels into consideration, so that if a channel is too congested to allow a wire to go through, the wire avoids the congested channel and chooses other channels.

6.2 Model

We deal with VLSI designs in which macrocells are rectangles with no restriction on their size. We assume that all macrocells are already placed, and the maximum power consumption and allowed voltage drop of each macrocell are given. VDD and GND pins can be placed anywhere on the periphery of a macrocell. Positions of VDD and GND pads from which power is supplied are defined on the periphery of the chip. Chips are allowed to have any number of VDD pads and GND pads.

6.3 Algorithm Overview

The objective is to route VDD and GND nets on one layer without crossing with minimum area under metal migration and voltage drop constraints. We assume that these nets are trees whose roots are pads and whose leaves are macrocell pins. We will give an overview of the algorithm here and describe the details of routing without crossing in the next section.

From input as described in the previous section (top of Figure 6.1), we first construct a channel intersection graph (middle of Figure 6.1). Vertices are placed at the corners of every macrocell, and vertical and horizontal edges connect these vertices. In cases where a vertex has degree 4, it is split into two separate vertices of degree 3 connected by an edge of 0 length. This simplifies the process of creating an auxiliary graph described in the next section. Next, pins and pads are connected to the nearest edges by adding new edges and new

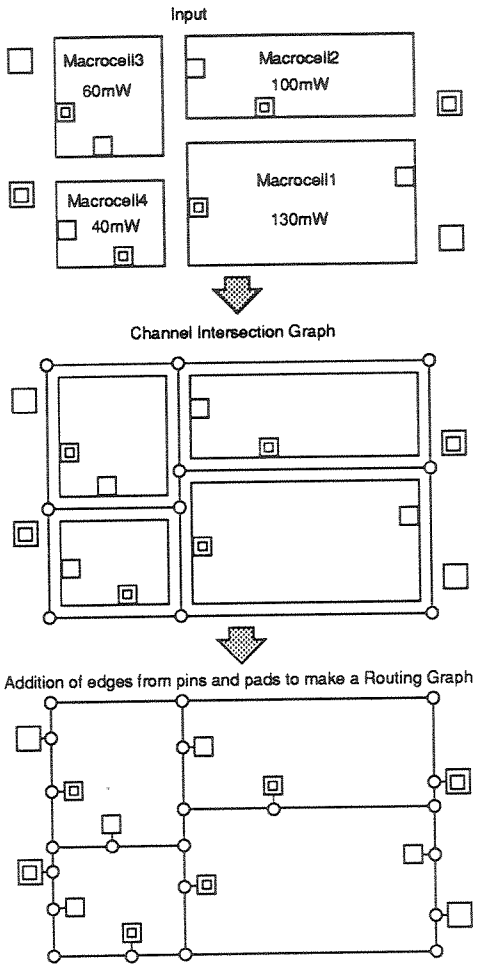


Figure 6.1: Creation of a Routing Graph

vertices (bottom of Figure 6.1). We call the result a routing graph. There are 6 types of vertex as shown in Figure 6.2. All vertices for pins and pads have degree 1, while other vertices have degree either 2 or 3.

Each edge is assigned two weights representing its length and its capacity. The length is used for calculating path lengths, and the capacity is used for deciding whether an edge has enough room for a wire to go through. We assume that any number of VDD and GND wires can go through an edge. For this purpose, a linked list is adopted as the data structure representing an

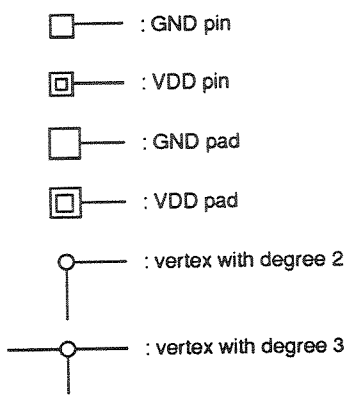


Figure 6.2: 6 Vertex Types

edge. A horizontal edge list is ordered from the bottom wire to the top wire, and a vertical edge list from the left wire to the right wire.

After the construction of the graph, we start growing VDD and GND trees by connecting macrocells one by one to trees under construction. Macrocells are sorted by their power consumption. First, the pins of the most power-consuming macrocell are connected to the pads. Subsequent macrocells are routed in decreasing order of power consumption. This is a greedy approach based on the notion that it is better for more power-consuming macrocells to have shorter paths, since more power-consuming macrocells need wider wires in order to be supplied with more current. At earlier stages in the routing, paths can generally be shorter, since they are blocked by fewer wires that are already routed. This results in a smaller area occupied by a wire. Pins of the second macrocell (and later macrocells considered) are connected to non-root vertices of the net (or possibly to an unconnected pad when there is more than one VDD or GND pad), making the constructed net a tree whose leaves are pins of macrocells and whose root is a pad. When there is more than one VDD or GND pad, the algorithm may create a forest of multiple trees. The wire area becomes even smaller than when there is only one VDD pad and one GND pad

because the search can find a shorter path to a power source. This multiple pad method eases the current load of each pad.

When macrocell M is routed, it makes a difference whether a VDD pin of M is routed first or a GND pin of M is routed first, because if the GND wire is routed first, VDD search may be blocked by the GND wire, or vice versa. So both are tried and the routing with shorter wire length is adopted.

6.4 Routing without Crossing

When each macrocell pin is connected to a tree, we have to find a non-crossing shortest path from the pin to a vertex in the existing tree. A modified version of a standard shortest path algorithm [10] is used for this purpose. For each macrocell, a VDD pin and a GND pin are connected separately to their respective trees. We will describe the routing of the VDD wire of each macrocell. Routing of the GND wire is similar.

The shortest path algorithm finds a shortest path from one vertex to another of a given graph. However, in the power routing problem, VDD and GND nets have to be routed, and there should be no crossings of nets. To deal with this, an auxiliary graph is constructed each time a new macrocell pin is to be routed. The auxiliary graph shows the paths in the routing graph which are not blocked by already-routed wires. The auxiliary graph is created based on rules shown in Figure 6.3. If there are no wires on the edge of a routing

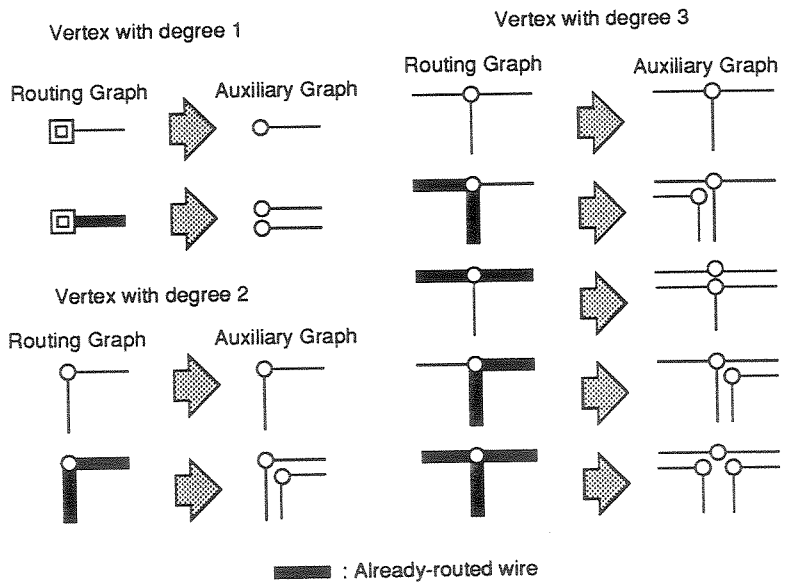


Figure 6.3: Creation of Auxiliary Graph from Routing Graph

graph, only one corresponding edge of an auxiliary graph is created. However, if there are already routed wires on the edge of a routing graph, two edges of an auxiliary graph are created: one edge running along one side of the wires, and the other running along the other side. An example of an auxiliary graph is shown in Figure 6.4.

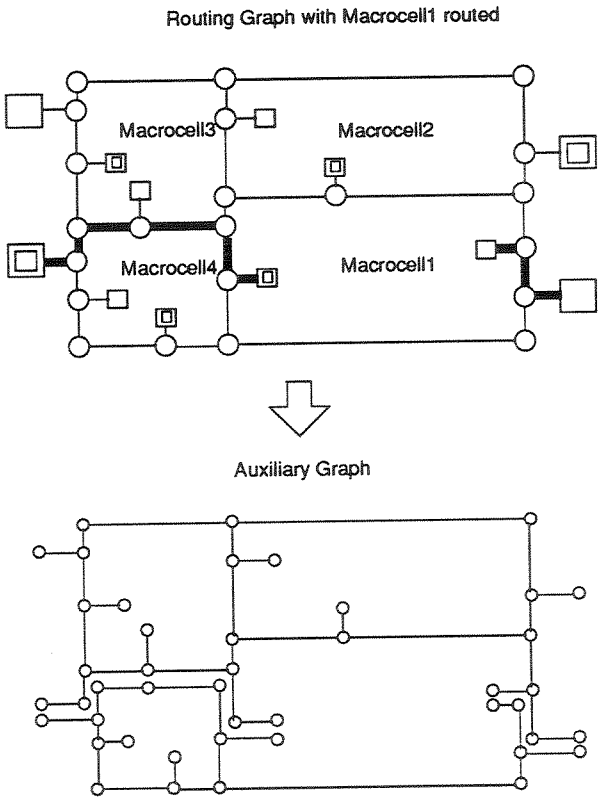


Figure 6.4: Example of an Auxiliary Graph

If this auxiliary graph is used, it is guaranteed that a VDD path never crosses GND trees, because no edges of the auxiliary graph cross the already-routed wires.

The shortest path algorithm is performed on this auxiliary graph. The VDD search starts at a VDD pin of a macrocell and terminates when it encounters a VDD pad or a vertex which is part of a VDD tree.

Each time a path visits a vertex, it has to go through an edge with a specified capacity and length. The capacity is the width of the free area inside a channel available for routing. When a path goes through an edge, the capacity of the edge is decremented by the width of the path. Initially the width of the path is calculated based on a metal migration constraint. Since the search starts from a macrocell whose maximum power consumption is P_{max}, the maximum current I_{max} is

$$I_{max} = \frac{P_{max}}{V_{vdd}}(mA),$$

where V_{vdd} is a power supply voltage. If an allowed current density to prevent metal migration is D $(\frac{mA}{\mu})$, the width $W_{migration}$ is

$$W_{migration} = \frac{I_{max}}{D} = \frac{P_{max}}{D \cdot V_{vdd}}(\mu).$$

If (the edge capacity - $W_{migration}$) becomes negative, the edge capacity is insufficient for routing the path, so the path through this edge is abandoned and the search continues through other edges.

When a VDD path reaches a vertex X of a VDD tree, a path length L from a VDD pin of the macrocell to a VDD pad is obtained. A voltage drop V_{drop} from the pad to the pin can be calculated as follows.

$$V_{drop} = I_{max} \cdot \rho \cdot \frac{L}{W_{drop}} = \frac{P_{max}}{V_{vdd}} \cdot \rho \cdot \frac{L}{W_{drop}},$$

where ρ and W_{drop} are sheet resistance$(\frac{\Omega}{\square})$ and wire width which causes V_{drop} voltage drop, respectively. V_{drop} must be smaller than an allowed voltage drop $V_{allowed}$ for the macrocell ($V_{drop} \leq V_{allowed}$). So

$$W_{drop} \geq \frac{\rho \cdot P_{max} \cdot L}{V_{vdd} \cdot V_{allowed}},$$

and the minimum W_{drop} is

$$MinW_{drop} = \frac{\rho \cdot P_{max} \cdot L}{V_{vdd} \cdot V_{allowed}}.$$

If $W_{migration}$ is smaller than $MinW_{drop}$, the width of wires between the VDD pin and the vertex X has to be increased to $MinW_{drop}$. If this thickening fails due to a too narrow channel, the search starts again from the VDD pin with width $MinW_{drop}$. The width of wires between the vertex X and the VDD pad also has to be increased by $max(W_{migration}, MinW_{drop})$. There are cases when this thickening fails, or the search never reaches a VDD pad or a tree owing to a too narrow channel. This is because macrocells are not properly placed. The only solution, then, is to try again with a different macrocell placement with wider channels. The overall routing algorithm is shown in pseudocode in Figure 6.5, and an algorithm to find a shortest path is shown in Figure 6.6.

```
procedure power_route

begin
  sort modules by their maximum power consumption
  number modules such that Power(module_1) >
              Power(module_2) >...> Power(module_n)
  for i = 1 to n
  begin
    find_shortest_path(VDD pin of module_i, W_migration)
              to VDD tree (or to VDD pad)
    (* path_VDD is the found path *)
    find_shortest_path(GND pin of module_i, W_migration)
              to GND tree (or to GND pad)
    (* path_GND is the found path *)
    clear path_GND and path_VDD
    find_shortest_path(GND pin of module_i, W_migration)
              to GND tree (or to GND pad)
    (* path'_GND is the found path *)
    find_shortest_path(VDD pin of module_i, W_migration)
              to VDD tree (or to VDD pad)
    (* path'_VDD is the found path *)
    if path_GND and path_VDD have shorter wire length
                 than path'_GND and path'_VDD
    then
      adopt path_VDD and path_GND
    else
      adopt path'_VDD and path'_GND
  end
end
```

Figure 6.5: Overall Power Routing Algorithm

```
procedure find_shortest_path(V,W)

begin
  create Auxiliary Graph from Routing Graph based
    on rules in Figure 6.3
  (* each vertex of Auxiliary Graph has a label which is either
    "permanent" or "temporary" and a weight called "distance" *)
  set the label of a vertex (of Auxiliary Graph that corresponds to
    V of Routing Graph) to "permanent" and "distance" to 0
  set labels of all other vertices of Auxiliary Graph to "temporary"
    and "distance" to ∞
  repeat
    find a "temporary" vertex_i whose "distance" is smallest
    change the label of vertex_i to "permanent"
    for all adjacent vertex_j of vertex_i begin
      if connection from vertex_i to vertex_j has
      enough room for the path of width W to go through then
        "distance" of vertex_j ≡ min("distance" of vertex_j,
          ("distance" of vertex_i + length between vertex_i and vertex_j))
    end
  until (it finds a pad or the vertex of a tree)
    or (a vertex whose label is "temporary" cannot be found)
  if it found a pad or the vertex X of a tree begin
    if MinW_drop > W begin
      thicken wires from V to X by (MinW_drop − W)
      if the thickening fails then find_shortest_path(V,MinW_drop)
    end else begin
      thicken wires from X to a pad by W
      if the thickening succeeds then the path is recorded
      else try with wider channel placement
    end
  end else try with wider channel placement
end
```

Figure 6.6: Algorithm to Find a Shortest Path from a Macrocell to a Power Tree

6.5 Time Complexity and Experimental Results

Since the shortest path algorithm is performed on an auxiliary graph which is a planar graph [2], the time complexity for finding a path from each pin is $O(m)$, where m is the number of vertices of the auxiliary graph. Each macrocell has two pins, each of which has at most two vertices of auxiliary graph. Each macrocell also has four corners and two vertices connecting pins to edges, each of which has at most three vertices of auxiliary graph. So one macrocell has at most $2 * 2 + (4 + 2) * 3 = 22$ vertices. So $m \leq 22n$, where n is the number of macrocells, and the time complexity for each macrocell is $O(n)$. Since there are n macrocells to be connected, the total time complexity is $O(n^2)$. Experimental results of routing time for 10 to 100 macrocells are shown in Figure 6.7. The run time was measured about $n^{2.05}$, which is close to $O(n^2)$.

It would be best to implement all the existing algorithms to compare their performance in terms of wire area. However, it is impossible to do it because the model is different in each algorithm. So only one different method was compared with ours. This is a method tried in [36], which sorts pins so that a pin which is farthest from a pad is connected first. This "farthest pin first" approach is based on a guess that the farthest pin is the hardest to connect, so that it is better to route it before many wires block it. Experimental results (Figure 6.8), however, show that the area obtained by "the farthest pin first" approach is about 20% larger than that of "the most power-consuming macrocell first" approach, showing that our method is clearly superior in terms of wire area. An example of 15 macrocells is shown in Figure 6.9.

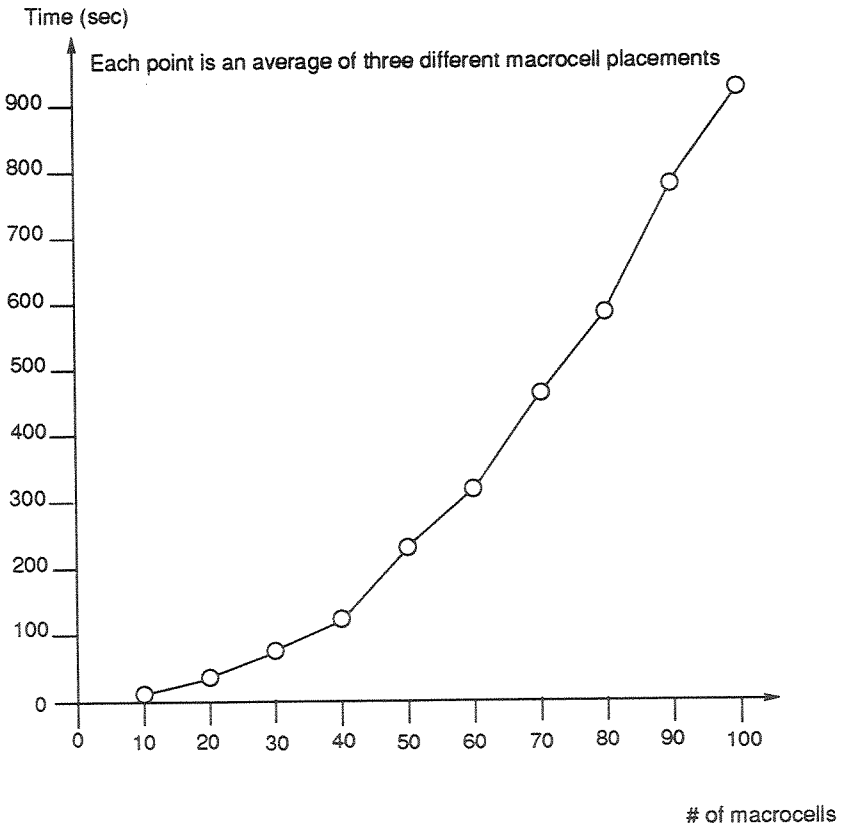


Figure 6.7: Run Time

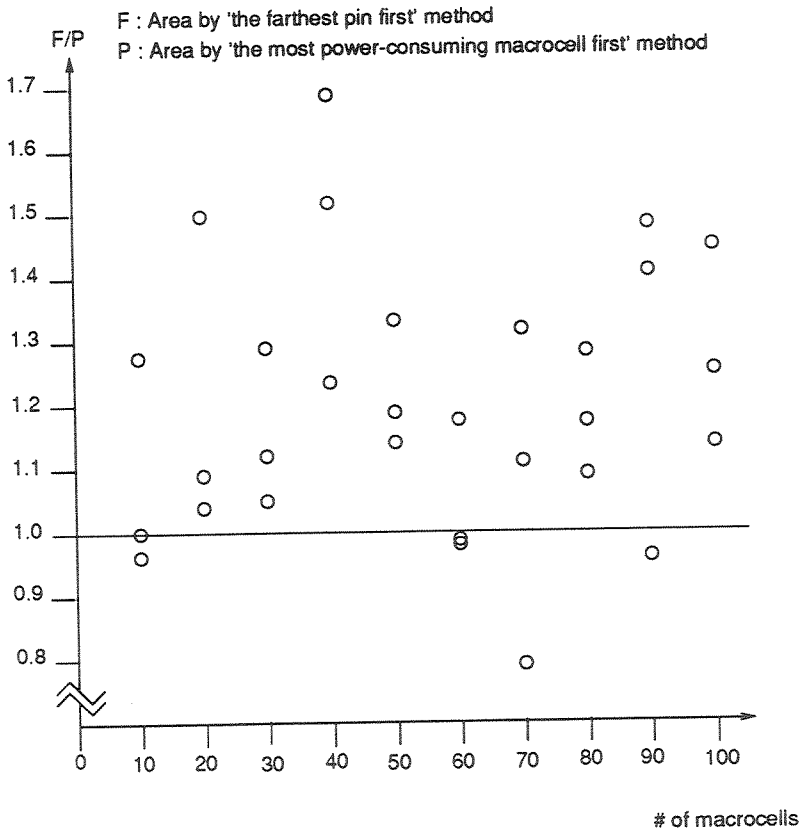


Figure 6.8: Area by "the Farthest Pin First" Method is about 20 Percent Larger than Area by "the Most Power-consuming Macrocell First" Method

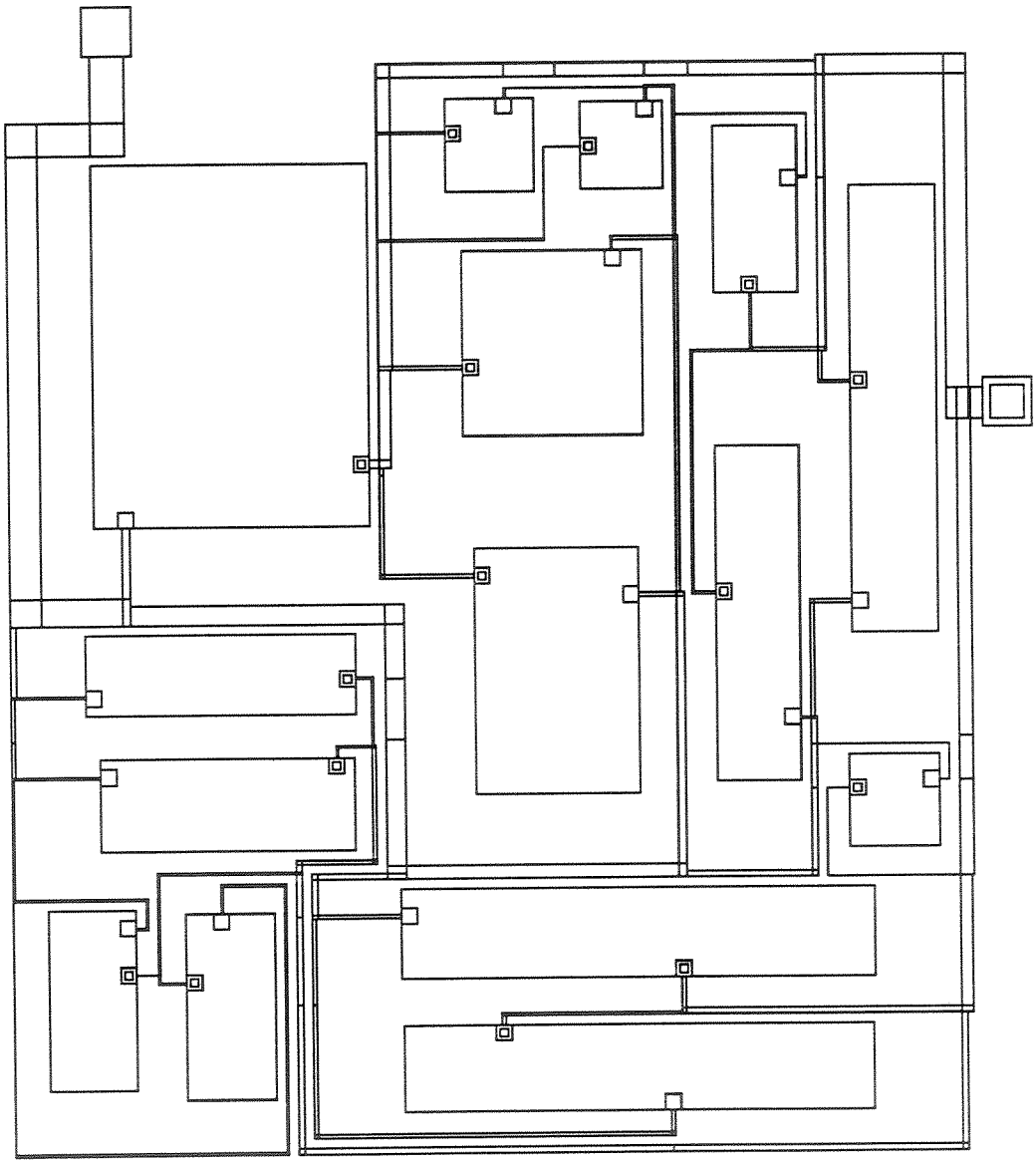


Figure 6.9: Example of Power Routing for 15 Macrocells

Chapter 7

Conclusion

We have discussed new methods of channel routing and power routing for VLSI layout.

Our topological channel router finds solutions with small height and small number of vias. The channel router first determines the topological relationship among nets before they are mapped onto a channel. A topological graph which contains the information of the topological relationship among nets is then mapped to a channel. The channel is compacted by a compaction method called shaking with horizontal make-space. Wires in the channel are placed using an interleaving model, in which there is no parallel overlap of wires, thus reducing the cross talk between signals.

The analysis of the topological relationship among nets has the advantage that a solution with a small number of vias can be found. However, the topology is determined without using geometric information. We modified the topological channel router to take the geometric information into consideration. As a result, we were able to obtain results with even further reduction of height and via count.

As new fabrication technologies enable the use of more than two layers for metal wires, topological analysis will be ever more important to achieve high chip performance. The analysis of topological relationships among nets before mapping them is a promising approach not only for channel routing as discussed in this dissertation, but also for switchbox routing [31].

The second router is concerned with power router for VLSI. It finds non-crossing VDD and GND trees on one layer using small metal area under the constraints of metal migration and voltage drop. Experimental results show that the power wire area is considerably smaller than a previously developed method for single-layer routing. This algorithm has several advantages over previous approaches. There is no restriction on pin and pad positions, so that macrocells with arbitrary pin positions can be handled with arbitrarily positioned pads. More than one VDD pad and more than one GND pad are allowed, resulting in a forest of multiple trees, which eases the current load of each pad and also results in even smaller wire area than just one VDD pad and one GND pad.

Finding a small area power wire routing solution by hand will be virtually impossible as the number of macrocells becomes large. As modular design approaches become more common in the future, there will be a greater demand to automatically route power wires for hundreds of macrocells. This area-efficient routing will be useful in such a case, and it will become especially important to minimize the area when wide wires need to be used to support large current requirements.

BIBLIOGRAPHY

[1] A. Aho, M. R. Garey, and F. K. Hwang, "Rectilinear Steiner Trees : Efficient Special Case Algorithms", *Networks*, Vol. 7, pp. 37-58, 1977.

[2] Ravindra K. Ahuja, Kurt Mehlhorn, James B. Orlin, and Robert E. Tarjan, "Faster Algorithms for the Shortest Path Problem", *Journal of the Association for Computing Machinery*, Vol. 37, No. 2, pp. 213-223, April 1990. pp. 213-223, .

[3] Sheldon B. Akers, James M. Geyer, and Donald L. Roberts, "IC Mask Layout with a Single Conductor Layer", *Proceedings of 7th Design Automation Workshop*, pp. 7-16, 1970.

[4] M. Breuer, "Min-Cut Placement", *Journal of Design Automation and Fault Tolerant Computing*, pp. 343-362, 1977.

[5] Michael Burstein, Richard Pelavin, "Hierarchical Channel Router", *Proceedings of 20th Design Automation Conference*, pp. 591-597, 1983.

[6] Ruen-Wu Chen, Yoji Kajitani, and Shu-Park Chan, "A Graph-Theoretic Via Minimization Algorithm for Two-Layer Printed Circuit Boards", *IEEE Transactions on Circuits and Systems*, Vol. CAS-30, No. 5, pp. 284-299, 1983.

[7] Chung-Kuan Cheng and David N. Deutsch, "Improved Channel Routing by Via Minimization and Shifting", *Proceedings of 25th Design Automation Conference*, pp. 677-680, 1988.

[8] Y. E. Cho, A. J. Korenjak, and D. E. Stockton, "FLOSS : An Approach to Automated Layout for High Volume Designs", *Proceedings of 14th Design Automation Conference*, pp. 138-141, 1977.

[9] S. Chowdhury, "An Automated Design of Minimum-Area IC Power/Ground Nets", *Proceedings of 24th Design Automation Conference*, pp. 223-229, 1987.

[10] Narsingh Deo, *A Graph Theory with Applications to Engineering and Computer Science*, Prentice-Hall, Inc., pp. 290-297, 1974.

[11] David N. Deutsch, "A 'Dogleg' Channel Router", *Proceedings of 13th Design Automation Conference*, pp. 425-433, 1976.

[12] David N. Deutsch, "Compacted Channel Routing", *Proceedings of IEEE International Conference on Computer Aided Design*, pp. 223-225, 1985.

[13] Maurice Hanan, Peter K. Wolff, Sr., and Barbara J. Agule, "Some Experimental Results on Placement Techniques", *Proceedings of 13th Design Automation Conference*, pp. 214-224, 1976.

[14] Shinichiro Haruyama and Don Fussell, "A New Area-efficient Power Routing Algorithm for VLSI Layout", *Proceedings of IEEE International Conference on Computer Aided Design*, pp. 38-41, 1987.

[15] Shinichiro Haruyama, D. F. Wong, and Don Fussell, "Topological Channel Routing", *Proceedings of IEEE International Conference on Computer Aided Design*, pp. 406-409, 1988.

[16] Shinichiro Haruyama, D. F. Wong, and Don Fussell, "Topological Routing Using Geometric Information", To Appear in *Proceedings of IEEE International Conference on Computer Aided Design*, 1990.

[17] Akihiro Hashimoto, James Stevens, "Wire Routing by Optimizing Channel Assignment within Large Apertures", *Proceedings of 8th Design Automation Workshop*, pp. 155-169, 1971.

[18] Chi-Ping Hsu, "Minimum Via Topological Routing", *IEEE Transactions on Computer-Aided Design*, Vol. CAD-2, pp. 235-246, 1983.

[19] Chi-Ping Hsu, "Minimum Via Two-layer Two-dimensional Routing", *Proceedings of IEEE International Conference on Computer Aided Design*, pp. 119-120, 1983.

[20] Chi-Ping Hsu, *Signal Routing in Integrated Circuit Layout*, UMI Research Press, 1986.

[21] Min-Yu Hsueh, "Symbolic Layout and Compaction of Integrated Circuits", Ph.D. Dissertation, University of California, Berkeley, California, 1980.

[22] David Lawrence Johannsen, "Silicon Compilation", Ph.D. Dissertation, Technical Report #4530, California Institute of Technology, Pasadena, California, 1981.

[23] R. M. Karp, "Reducibility among combinatorial problems", *Complexity of Computer Computations*, R. E. Miller and J. W. Thatcher, Eds., Plenum Press, New York, pp. 85-103, 1972.

[24] B. Kernighan and S. Lin, "An Efficient Procedure for Partitioning Graphs", *Bell System Technical Journal*, pp. 291-307, 1970.

[25] S. Kirkpatrick, C. Gelatt, and M. Vecchi, "Optimization by Simulated Annealing", *Science*, Vol. 220, No. 4598, pp. 671-680, May 13, 1983.

[26] Margaret Lie and Chi-Song Horng, "A Bus Router for IC Layout", *Proceedings of 19th Design Automation Conference*, pp. 129-132, 1982.

[27] Malgorzata Marek-Sadowska, "An Unconstrained Topological Via Minimization Problem for Two-Layer Routing", *IEEE Transactions on Computer-Aided Design*, Vol. CAD-3, No. 3, pp. 184-190, July 1984.

[28] C. A. Mead and L. A. Conway , *Introduction to VLSI Systems*, Addison Wesley, 1980.

[29] Andrew S. Moulton, "Laying the Power and Ground Wires on a VLSI Chip", *Proceedings of 20th Design Automation Conference*, pp. 754-755, 1983.

[30] N. J. Naclerio, S. Masuda, and K. Nakajima, "Via Minimization for Gridless Layouts", *Proceedings of 24th Design Automation Conference*, pp. 159-164, 1987.

[31] R. Pyke, "A Gridless Switchbox Router", *Proceedings of Custom Integrated Circuits Conference*, pp. 629-632, 1987.

[32] Neil R. Quinn, Jr. and Melvin A. Breuer , "A Forced Directed Component Placement Procedure for Printed Circuit Boards", *IEEE Transactions on Circuits and Systems*, Vol. CAS-26, No. 6, pp. 377-388, 1979.

[33] Ronald L. Rivest, Charles M. Fiduccia, "A 'Greedy' Channel Router", *Proceedings of 19th Design Automation Conference*, pp. 418-424, 1982.

[34] H-J. Rothermel and D. A. Mlynski, "Computation of Power Supply Nets in VLSI Layout", *Proceedings of 18th Design Automation Conference*, pp. 37-42, 1981.

[35] J. Royle, M. Palczewski, H. VerHeyen, N. Naccache, and J. Soukup, "Geometrical Compaction in One Dimension for Channel Routing", *Proceedings of 24th Design Automation Conference*, pp. 140-145, 1987.

[36] David W. Russell, "Hierarchical Routing of Single Layer Metal Trees in Compiled VLSI", *IEEE International Conference on Computer-Aided Design*, pp. 270-272, 1985.

[37] A. Sangiovanni-Vincentelli, M. Santomauro, and J. Reed, "A New Gridless Channel Router : Yet Another Channel Router the Second (YACR-II)", *Proceedings of IEEE International Conference on Computer Aided Design*, pp. 72-75, 1984.

[38] Majid Sarrafzadeh and D. T. Lee , "A New Approach to Topological Via Minimization", *IEEE Transactions on Computer-Aided Design*, Vol. CAD-8, No. 8, pp. 890-900, August 1989.

[39] Sarma Sastry and Alice Parker, "The Complexity of Two-dimensional Compaction of VLSI Layouts", *Proceedings of IEEE International Conference on Circuits and Computers*, pp. 402-406, 1982.

[40] Carl Sechen and Algerto Sangiovanni-Vincentelli, "The TimberWolf Placement and Routing Package", *Proceedings of Custom Integrated Circuits Conference*, pp. 522-527, 1984.

[41] Robert Sedgewick, *Algorithms*, Addison Wesley, 1983.

[42] Hyunchul Shin, Alberto L. Sangiovanni-Vincentelli, and Carlo H. Séquin, "Two-Dimensional Compaction by Zone Refining", *Proceedings of 23rd Design Automation Conference*, pp. 115-122, 1986.

[43] Hyunchul Shin, Alberto L. Sangiovanni-Vincentelli, and Carlo H. Séquin, "Two-Dimensional Module Compactor Based on 'Zone-Refining' ", *Proceedings of International Conference on Computer Design*, pp. 201-208, 1987.

[44] Hyunchul Shin, Alberto L. Sangiovanni-Vincentelli, and Carlo H. Séquin, " 'Zone-Refining' Techniques for IC Layout Compaction", *IEEE Transactions on Computer-Aided Design*, Vol. CAD-9, No. 2, pp. 167-179, February 1990.

[45] Kenneth J. Supowit, "Finding a Maximum Planar Subset of a Set of Nets in a Channel", *IEEE Transactions on Computer-Aided Design*, Vol. CAD-6, No. 1, pp. 93-94, January 1987.

[46] Khe-Sing The, D. F. Wong, and Jinsheng Cong, "Via Minimization by Layout Modification", *Proceedings of 26th Design Automation Conference*, pp. 799-802, 1989.

[47] Makoto Watanabe, Kunihiro Asada, Kenji Kaji, and Tatsuo Ohtsuki, *VLSI Design I : Circuits and Layout, Iwanami Koza Micro Electronics*, Iwanami Shoten, 1985.

[48] N. Weste, "Virtual Grid Symbolic Layout", *Proceedings of 18th Design Automation Conference*, pp. 225-233, 1981.

[49] D. F. Wong, H. W. Leong, and C. L. Liu, *Simulated Annealing for VLSI Design*, Kluwer Academic Publishers, 1988.

[50] Takeshi Yoshimura, Ernest S. Kuh, "Efficient Algorithms for Channel Routing", *IEEE Transactions on Computer-Aided Design*, Vol. CAD-1, pp. 25-35, January 1982.

[51] Xiao-Ming Xiong and Ernest S. Kuh, "The Scan Line Approach to Power and Ground Routing", *IEEE International Conference on Computer-Aided Design*, pp. 6-9, 1986.