Copyright

by

Hai Zhou

1999

SIGNAL INTEGRITY AND LOW POWER ISSUES IN DEEP SUB-MICRON VLSI DESIGN

by

Hai Zhou, B.E., M.E.

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

May 1999

SIGNAL INTEGRITY AND LOW POWER ISSUES IN DEEP SUB-MICRON VLSI DESIGN

Approved by
Dissertation Committee:

To Ivana and Bozhao

Acknowledgments

I am greatly indebted to my thesis advisor Professor Martin D. F. Wong for introducing me to the area of VLSI design automation. It is to his constant motivation and encouragement that I owe this work. It was indeed very satisfying for me to work with a supervisor who shared my love for mathematical and algorithmic problems.

I would like to thank the members of my thesis committee, Dr. Charles Alpert, Professors Adnan Aziz, E. Allen Emerson, and C. Greg Plaxton, for their interest and feedback on my work. Special thanks go to Professor Adnan Aziz, who broadened my interests in design automation to include logic synthesis and formal verification, and who even provided me a one-year research assistantship in his laboratory to do research in these areas.

I thank all the members of the CAD group, specifically Hung-Ming Chen, Chris C. N. Chu, Youxin Gao, James K.-K. Lee, Huiqun Liu, Arthur W.-K. Mak, and Evan F. Y. Young, for many providing a forum to present and discuss my work. The unfailing efficiency of Gloria Ramirez at the department's Graduate Office kept the tangle of red tape manageable.

I thank all my friends for the fun time I had in Austin, socially and academically. In particular I would like to thank Xingang Guo, who has been my classmate for 10 years and who was always the first person I thought of whenever I needed help. It was also fun solving puzzles and discussing research (his and mine) with him.

It is with the risk of inadvertently leaving out someone important that I have mentioned specific names in the above. I acknowledge their help, and pray that they forgive my lapse.

Credit also goes to my parents for their confidence in me over my long years of education. Their patience and encouragement have always been my greatest assets.

Last, but not the least, I thank Bozhao, my wife, for her love, sacrifice, and support, without which this dissertation could not have been written.

HAI ZHOU

The University of Texas at Austin
May 1999

SIGNAL INTEGRITY AND LOW POWER ISSUES IN DEEP SUB-MICRON VLSI DESIGN

Publication No. ____________________

Hai Zhou, Ph.D.
The University of Texas at Austin, 1999

Supervisor: Martin D. F. Wong

With VLSI fabrication entering the deep sub-micron (DSM) era, devices are scaled down to smaller sizes, clocks are running at higher frequencies, and more functions are integrated into one chip. As a result, coupling effect between neighboring wires is increasing, and signal switching on one wire produces noise on the other wire. This phenomenon is called *crosstalk*. Another result of increasing frequency and density is the increasing power dissipation.

In this dissertation, we studied several major CAD problems for crosstalk and power reduction, namely crosstalk-driven routing, low power gate decomposition, low power floorplanning, and multi-objective buffered maze routing.

In crosstalk-driven routing, we first consider river routing where only one layer is available. We establish a relationship between crosstalk reduction and space allocation. Based on this relationship, a polynomial time network-flow based algorithm is designed to construct an optimal solution. We then consider crosstalk control in maze routing which is a general routing technique. We give the complexity result and design an algorithm which is based on Lagrangian relaxation. We

also consider crosstalk reduction in global routing. To enable crosstalk estimation, simple layer and track assignments are included in the routing. A multi-stage global router is designed to construct a crosstalk-feasible solution in such a model.

For power reduction, we first consider low power gate decomposition where a multiple input gate is decomposed into a tree of two-input gates. In the case of AND/OR gate, we first prove many properties for an optimal decomposition. Based on these properties, we design an exact algorithm which is the fastest in the literature and a heuristic algorithm which outperforms known heuristics. In the case of XOR gate, we give polynomial algorithms to construct optimal decompositions. We then consider the problem of cell selection in a slicing floorplan to minimize power dissipation under area constraint. We characterize the complexity of the problem and give pseudo-polynomial time algorithms for it. We also consider the problem of simultaneous routing and buffer insertion which forbids buffer insertions over macro blocks. We design algorithms for this problem where the objectives include power, delay, and congestion.

Contents

Chapter 1

Introduction

The driving force behind the rapid growth of the VLSI technology has been the constant reduction of the feature size of VLSI devices (i.e., the minimum transistor size). The feature size decreased from about $1\mu m$ in 1990 to $0.18\mu m$ today. The prediction is that it will be reduced to about $0.10\mu m$ in 2006 [5].

Scaling down of VLSI fabrication into the deep sub-micron (DSM) dimensions has strong impact on the VLSI technology in several ways. First, the device density on integrated circuits grows quadratically with the rate of decrease in the feature size. Also, there is a trend in increasing chip die sizes. As a combined result, the total number of transistors on a single chip has increased from less than $500,000$ in 1985 to 21 million today and will reach 200 million in 2006 [5]. Second, when wires are scaled down, in order to keep resistance small, the heights are usually scaled at much lower rate. This gives wires of high aspect ratio. Furthermore, to keep up communication among these huge numbers of transistors, more interconnect layers are generally used. All these make the coupling capacitance become the dominant part of capacitance on a wire. Finally, feature size reduction also makes the transistors switch at faster rate. This, together with the insatiable market demand for higher performance cost-effective products, makes chips run at progressively higher frequencies.

All these bring great promise of "system-on-a-chip," but also introduce many new issues in the design process. Signal integrity and power dissipation are two important issues among them.

1.1 Signal integrity

Crosstalk is the phenomenon of noise induced on one wire by a signal switching on a neighboring wire. It is a result of mutual capacitances and inductances between neighboring wires. The closer the wires, the higher they are from the ground plane, and the longer a distance they neighbor each other, the larger the amount of coupling is. Crosstalk is also proportional to the rate of change of the voltage waveforms.

In static logic, crosstalk on coupling wires could increase delay when signals are switching in opposite directions. But when signals are switching in the same direction, it helps to reduce delay. This means crosstalk will make delay more unpredictable. In dynamic logic, crosstalk can even make nodes mistakenly discharged and thus produce logic errors. Therefore, it is very important to control crosstalk.

There are many ways to control crosstalk in the design process. Since crosstalk is proportional to the rate of change of the voltage waveforms, which in turn are controlled by drive strength and load capacitance, interconnect optimization techniques such as gate sizing and buffer insertion can be used to reduce crosstalk. But since crosstalk is heavily dependent on the couplings between wires, which in turn are decided by the physical positions of wires, routing is therefore a more effective stage to control crosstalk. In this dissertation, we mainly focus on how to control crosstalk during routing.

1.2 Power dissipation

In the past, the major concerns of VLSI design were performance, area and reliability; power considerations were of only secondary importance. However, this has begun to change recently, and power is being given comparable weight to speed and area. Two factors mainly contribute to this trend. On one hand is the remarkable success and growth of portable applications which demand high-speed computation and complex functionality with low power consumption. Without low-power design techniques, current and future portable devices will suffer from either a very short battery life or a very heavy battery pack. On the other hand, power dissipation in high-end products rises with increasing integration density and circuit speed. The resulting heat will increase the packaging and cooling costs and shorten the life of ICs. Therefore, low-power techniques are also needed.

Power dissipation in digital CMOS VLSI circuits is caused by the following four sources: leakage current, standby current, short-circuit current, and capacitance current. The leakage current is primarily determined by the fabrication technology and consists of two components: reverse bias current in the parasitic diodes formed between source and drain diffusions and the bulk region in a MOS transistor, and the sub-threshold current that arises from the inversion charge that exists at the gate voltages below the threshold voltage. The standby current is the DC current drawn continuously from V_{dd} to ground. The short-circuit current is due to the DC path between the supply rails during output transitions. The capacitance current is the one which flows to charge and discharge capacitive loads during logic changes.

The sum of leakage and standby dissipations is called *static power dissipation* and the sum of the other two is called *dynamic power dissipation*. The dominant source of power dissipation in CMOS circuits is the charging and discharging of the node capacitances (also referred to as the capacitive power dissipation) and is given by:

$$P = 0.5V_{dd}^2 f_{clk} C_L E_{sw}$$

where C_L is the physical capacitance at the output of the node, V_{dd} is the supply voltage, E_{sw} (referred to as the *switching activity*) is the average number of output transitions per clock cycle, and f_{clk} is the clock frequency.

As we can see from the above formula, V_{dd} and f_{clk} are fixed at the beginning of the design process, but C_L and E_{sw} can be controlled during design. Generally, switching activity E_{sw} is connected with the functionality of the node and is usually controlled in logic synthesis. On the other hand, C_L is generally controlled in physical design. In this dissertation, both the power reductions by switching activity minimization and by load capacitance minimization are considered.

1.3 Dissertation overview

We give an overview of the dissertation as follows.

In Chapter 2, we solve river routing with crosstalk constraints. Given the positions of the pins in a single-layer routing channel and the maximum tolerable crosstalk between each pair of neighboring nets, we give a polynomial time algorithm to decide whether there is a feasible river routing solution and produce one with minimum crosstalk when it is feasible.

In Chapter 3, we consider maze routing with crosstalk constraints. As we know, maze routing is a robust and general approach which can be used in many situations including multi-layer interconnections. After formulating the crosstalk-constrained maze routing problem, we show it is equivalent to the multi-constrained shortest path problem and is strongly NP-complete. Based on the Lagrangian relaxation technique, an effective heuristic algorithm is designed to solve the problem.

In Chapter 4, we deal with the global routing with crosstalk constraints. Global routing has more flexibility to change wires thus more flexibility to control crosstalk. Since it is not easy to accurately estimate crosstalk in global routing, we extend global routing to include layer and track assignments to facilitate crosstalk estimation. Based on the estimation, we present a global routing algorithm based on a new Steiner tree formulation and the Lagrangian relaxation technique. We also give theoretical results on the complexity of the problem.

In Chapter 5, we consider low power AND/OR gate decomposition. Gate decomposition is a step before technology mapping. How decomposition is done can have a significant impact on the power dissipation of the final implementation. In the literature, only heuristic algorithms are given for the low power AND/OR gate decomposition problem. In that chapter, we prove many properties an optimal decomposition tree must have. Based on these optimality properties, we design an efficient exact algorithm to solve the decomposition problem. Moreover, the exact algorithm can be easily modified to a heuristic algorithm which performs much better than the known heuristics.

In Chapter 6, we solve the low power XOR gate decomposition problem. It has two contributions. The first is an optimal algorithm for dynamic logic. Based on optimality properties we proved for this problem, we design an algorithm to construct an optimal decomposition tree. The time complexity of the algorithm is $O(n \log n)$. The second contribution is an optimal algorithm for low power XOR gate decomposition in static logic. We find that the only algorithm in the literature, which requires both polarities available for the inputs, is **not** optimal. Based on the properties of decompositions in static logic, we design an $O(n \log n)$ time optimal algorithm which does not even have the polarity requirement.

In Chapter 7, we deal with the floorplan power and area optimization. Floorplan optimization is an important problem in floorplanning which asks to choose an implementation for each block such that some objectives are optimized under a given topology. Traditionally, only geometrical dimensions are considered to minimize the total area. Since different implementations may have different power consumptions, besides the area, the total power consumption also needs to be minimized. In [8], Chao and Wong formulated a slicing floorplan power and area optimization problem and proposed a dynamic programming approach for it. Although they showed the algorithm uses exponential storages (therefore, has exponential running time), the complexity of the problem is not known. In that chapter, we first characterize the complexity of the problem. We show that it is NP-hard even when the number of values in height or in width is upper bounded. Besides their algorithm, we also give another pseudo-polynomial time algorithm whose time does not depend on the values of power dissipation.

In Chapter 8, we consider simultaneous routing and buffer insertion. During the routing of global interconnects, macro blocks form useful routing regions which allow wires to go through but forbid buffers to be inserted. They give restrictions on buffer locations. We take these buffer location restrictions into consideration and solve the simultaneous maze routing and buffer insertion problem. Given a block placement defining buffer location restrictions and a pair of pins (a source and a sink), we give a polynomial time exact algorithm to find a buffered path from the source to the sink with minimum Elmore delay. We also extend the algorithm to consider power and congestion minimization.

Chapter 2

River Routing with Crosstalk Constraints

2.1 Introduction

With VLSI fabrication entering the deep sub-micron era, devices and interconnection wires are being placed at an ever increasing proximity. Reduction in the interconnection and transistor switching delays results in faster signal transition times. All these factors increase the coupling effect (inductive and capacitive) between interconnection wires. Increased coupling effect not only increases signal delays, but also decreases signal integrity due to transmission line behavior. This phenomenon is called *crosstalk* [6].

In the literature, previous works on the crosstalk avoidance in detailed routing fall into two main categories. In the first category, the gridless routing model is used and spacings between wires are adjusted to reduce crosstalk [11, 9]. In the second category, the gridded routing model is used; works in this category focus on channel routing [19, 26, 55] and switchbox routing [20]. The general approach taken in most previous works belongs to post-routing optimization. That is, a routing solution is first determined by a conventional routing algorithm, then algorithms are designed to modify the routing solution to reduce crosstalk.

In this chapter, we consider the river routing problem with crosstalk constraints. Although river routing has the best developed theory among all detailed routing problems [30], there is no previous work with consideration of crosstalk avoidance. Contrary to works in [11, 19, 20, 9, 55], we consider crosstalk among wires in a global way during the routing process and develop a novel routing algorithm. Given the pin positions and the maximum tolerable crosstalk between each pair of neighboring nets, for any fixed channel width, the algorithm can decide whether there is a feasible routing solution. Furthermore, if there is one, the algorithm can give a routing solution with minimum crosstalk at the same time complexity.

The rest of the chapter is organized as follows. Section 2.2 formally defines the crosstalk-constrained river routing problem. The optimal algorithm is presented in Section 2.3. In Section 2.4 we present a post-processing heuristic to reduce the number of bends in the solution. Finally, experimental results and conclusions are presented in Section 2.5 and Section 2.6.

2.2 Problem formulation

In a river routing problem, we are given a single-layer rectangular routing channel with pins located at the top and the bottom of the channel. Let $t_1, t_2, \ldots, t_n$ be the pins at the top boundary of the channel (from left to right) and $b_1, b_2, \ldots, b_n$ be the pins at the bottom boundary of the channel (from left to right). We assume that there is a grid super-imposed over the channel and that all pins are on grid points. For $1 \le i \le n$, *net* i is defined by (t_i, b_i). A *route* of net i is a simple path in the grid graph connecting t_i and b_i. We also assume all routes to be *monotonic*, that is, starting from one pin, a route can only proceed in at most two directions.

All previous works on river routing used this assumption, since it does not hurt routability [30], and at the same time, make each route to be the shortest one. A *routing solution* is a set of routes, one for each net, such that no route shares a grid point. A route is *legal* if it exists in at least one routing solution.

Generally speaking, crosstalk between two parallel straight wires is proportional to the coupling capacitance between them, which in turn is proportional to their coupling length and inversely proportional to their separating distance. Since the coupling capacitance between two wires decreases rapidly as the distance between them increase, it is reasonable to assume that crosstalk only exists between wires in adjacent rows or columns. Without loss of generality, we can set the proportion constant to 1 and treat the adjacent length between two wires as their crosstalk. Crosstalk between two routes is then the total number of their adjacent grid edges. Note that this is also the crosstalk model used in [19, 20]. In any routing solution, let R_i denote the crosstalk between net i and net $i+1$, for $1 \leq i \leq n-1$. For example, Figure 2.1 shows a river routing solution and crosstalk between each pair of routes. Here, $R_1 = 2+2 = 4, R_2 = 3, R_3 = 2, R_4 = 1+1 = 2$, and $R_5 = 2$.

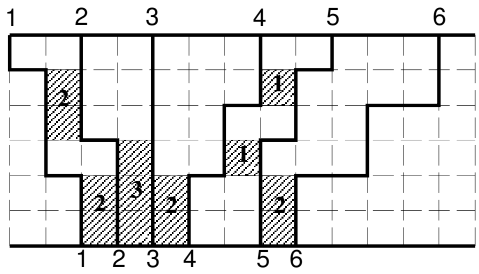

Figure 2.1: River routing solution and crosstalk

For each pair of neighboring nets i and $i+1$, where $1 \leq i \leq n-1$, a constant C_i is given as the maximum tolerable crosstalk between them. A routing solution such that $R_i \leq C_i$, for $1 \leq i \leq n-1$, is called a *feasible* routing solution. A route is *feasible* if it exists in at least one feasible routing solution. We also call $C_i - R_i$, for all $1 \le i \le n-1$, crosstalk *slacks* and define *min-slack* to be $\min_{1 \le i \le n-1}(C_i - R_i)$. The problem we address in this chapter can be stated as follows:

Crosstalk-Constrained River Routing (CCRR) Problem: Given a channel of width W, a set of n nets (t_1, b_1), (t_2, b_2), $\ldots$, (t_n, b_n), and a set of crosstalk constraints C_1, C_2, $\ldots$, C_{n-1}, determine whether there is a feasible routing solution. If there is one, give a routing solution with maximum min-slack.

2.3 Optimal algorithm

In a river routing problem, since only one layer is used, the relative positions of wires are fixed. Therefore, the only way to reduce the crosstalk between two neighboring nets is to position them a little far away. This can be regarded as inserting space between them. Because of the fixed pin positions and monotonic route requirement, the space which can be used is limited and must be allocated properly to fulfill the crosstalk constraints. In this sense, the CCRR problem might be transformed to and solved by a space allocation problem. But the transformation is not trivial because there is no one-to-one correspondence between space insertion and crosstalk reduction. There are two problems. One is that crosstalk can be reduced without space insertion and the other is that space insertion sometimes can not reduce crosstalk. We explain them by two simple examples. In Figure 2.2(a), two nets are routed as close to each other as possible, hence have the same space between them. Crosstalk is decreased when they change their shapes. In Figure 2.2(b), one more unit of space is inserted between two nets, but crosstalk between them does not decrease.

Analyzing the situation happened in Figure 2.2(a), we can find that crosstalk is decreased because more bends are introduced. The basic reason for this comes from the fact that at the concave side of a bend the nearest wire is at least two units apart. Since bends are beneficial on crosstalk avoidance, we are interested in

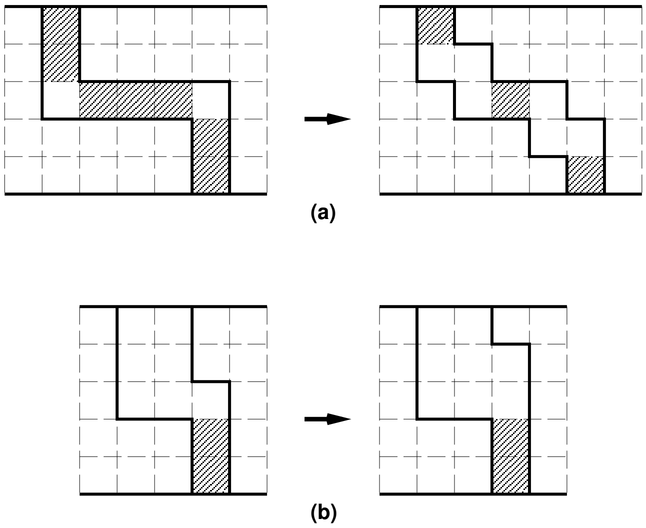

Figure 2.2: (a)Crosstalk reduction without space insertion (b)Space insertion without crosstalk reduction

those routes with as many bends as possible. For each net, such routes are defined as *MC* (stands for Minimal-Crosstalk) *routes*, since they will have minimal crosstalk if neighboring nets are routed closely along them. It will be shown that, if there is a feasible routing solution, then there will also be one composed of MC routes. That is, considering only MC routes is sufficient to find an optimal solution for CCRR problem. Besides that, MC routes have many good properties. One of them is that space must be inserted in order to reduce crosstalk between two MC routes. This avoids the problem in Figure 2.2(a) and helps in modeling the CCRR problem by space allocation. Details about MC routes are presented in Section 2.3.1.

In order to measure the crosstalk reduction requirement for each pair of nets, we need to compute the maximum crosstalk between their MC routes. We define this crosstalk as *potential crosstalk* of the two nets, which is the crosstalk when no space is inserted. Obviously, the difference between potential crosstalk and crosstalk constraint defines the requirement. Based on the properties of MC routes, the same amount of space is needed to get a feasible solution. The space which can be used to reduce potential crosstalk is defined as *MC space*. It may be decomposed into sub-regions called *MC blocks*, each of which is shared by a set of net pairs. Furthermore, if we restrict the maximum amount of MC space allocated to each pair of net, we can also avoid the problem shown in Figure 2.2(b). Under this restriction, we establish a one-to-one correspondence between MC space insertion and crosstalk reduction. Therefore, the CCRR problem is transformed into a space allocation problem. It can then be modeled by a flow network and solved by a maximum flow computation. Details about the space allocation and network flow are presented in Section 2.3.2 and Section 2.3.3, respectively.

2.3.1 MC (minimal-crosstalk) routes

Without considering crosstalk, river routing is a well-solved problem. Given a river routing problem, there exists an $O(n^2)$ time algorithm which decides whether there is a routing solution and gives one if possible [16, 30, 35]. In CCRR problem, among all those possible solutions, one with crosstalk under constraints must be selected. In order to specify the flexibility of the selection, we are interested in two extreme routing solutions. In one solution, all nets are routed as far to the left as possible, as shown in Figure 2.3(a). In the other solution, all nets are routed as far to the right as possible, shown in Figure 2.3(b). For each net, we call these routes left and right *legal boundaries*, respectively (Figure 2.3(c)). It is easy to see that a route is legal if and only if it is within the legal boundaries.

As observed from Figure 2.2(a), bends are useful in reducing crosstalk. Therefore, we prefer to perform routing using as many bends as possible.

Definition 1 (stair) *A stair is a route segment composed of alternating horizontal and vertical grid edges.*

To illustrate the definition, a route is shown in Figure 2.4(a), where the encircled segments are stairs. It is easy to see that, except the first and last edges

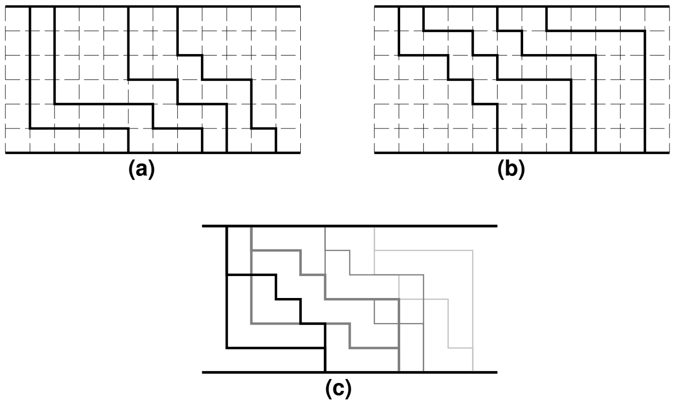

Figure 2.3: (a)Leftmost routes; (b)Rightmost routes; (c)Legal boundaries

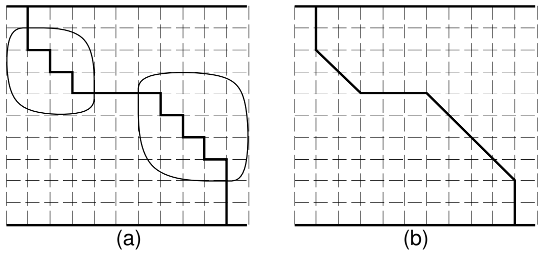

Figure 2.4: Stairs and their representations

which may induce crosstalk on one side, the middle part of a stair can not have crosstalk with any wire. In order to simplify our presentation, in the rest of the chapter, we will use 45° line to represent the middle part of a stair, which can never have crosstalk. In addition, we will call other segments *straight lines*, or specifically *horizontal/vertical lines* when the orientations are emphasized. For example, Figure 2.4(b) will be used to represent Figure 2.4(a).

Based on stairs and straight lines, for each net, we can define a binary relation on the set of its legal routes.

Definition 2 (derivation) *Let w and w' be two legal routes of a given net, we say w' is derived from w if its straight lines are all on w.*

It is easy to check that the derivation relation is reflexive, transitive, and anti-symmetric, hence a partial order. Based on the definition, in Figure 2.5, the route in (b) is derived from that in (a). Dashed lines in the figure are legal boundaries, drawn to show that both routes are legal.

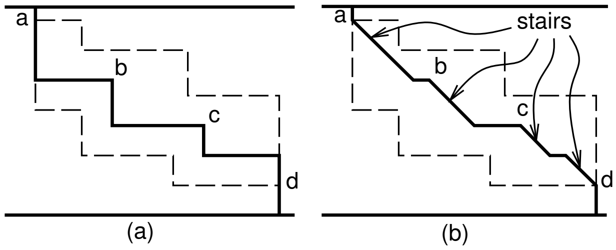

Figure 2.5: (a)Legal route; (b)Derived MC route

Since only straight lines can induce crosstalk, if w' is derived from w, it usually will not induce more crosstalk than w. Formally, this is stated as the following lemma.

Lemma 1 *Suppose w' is a legal route derived from route w. If w is feasible, then w' is also feasible.*

Proof: Consider a feasible routing solution which includes route w. Substitute w with w'. According to the definition of derivation, only stairs of w' may get cross with the neighboring route. If this happened, we also change the crossed parts of the neighboring route and route them as close as possible to w'. It is not difficult to see that the changed parts are all stairs with the first and last edges on the original route. Continue the process until there is no crossing between routes. This is always possible, because w' is legal and we always route the changed parts as close to each other as possible. Because all new parts are middle parts of stairs, crosstalk can not increase (usually, they decrease). $\square$

The above lemma shows that, if w' is derived from w, considering only w' is sufficient to find a feasible solution. Therefore, we are interested in a set of special routes defined as follows.

Definition 3 (MC route) *A legal route is called an MC (minimal-crosstalk) route if its only derived route is itself.*

We can check that the route in Figure 2.5(b) is actually an MC route. The following theorem shows that considering only MC routes is sufficient in the CCRR problem.

Theorem 1 *If there is a routing solution in which crosstalk between net i and net $i+1$ is R_i, for $1 \leq i \leq n-1$, then there will be a routing solution composed of MC routes in which crosstalk between net i and net $i+1$ is R'_i such that $R'_i \leq R_i$, for $1 \leq i \leq n-1$.*

Proof: Suppose r is the given routing solution, if all of its routes are MC routes, we already have the result. Otherwise, there must be a route w which is not an MC route. By the process in the proof of Lemma 1, we can change w to an MC route without increasing crosstalk. That process will not change other MC routes to non-MC ones because every new part is a stair. Repeat the process for each non-MC route, we will finally get a routing solution composed of all MC routes. $\square$

Similar to legal routes, MC routes of a given net are also restricted within some boundaries.

Definition 4 (MC boundary and MC region) *For each net, the MC route derived from the left legal boundary is called the left MC boundary; the MC route derived from the right legal boundary is called the right MC boundary. The regions surrounded by the left and right MC boundaries are called MC regions.*

The MC boundaries and MC regions of one net are shown in Figure 2.6. The dashed lines are the left and right legal boundaries of the net.

We summarize the properties of MC routes in the following lemma.

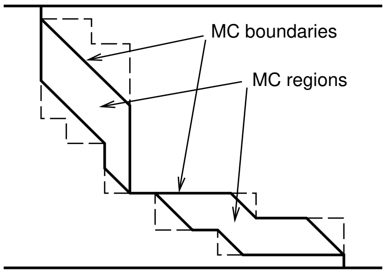

Figure 2.6: MC boundaries and MC regions

Lemma 2 *For each net, we have*

1. All MC routes are within the MC boundaries;

2. Every MC route has the same length of horizontal lines and the same length of vertical lines;

3. The horizontal and vertical lines of a MC route, if both exist, must lie within separate MC regions.

Proof: Suppose property 1 is not true, that is, there is an MC route w which goes outside of, say, left MC boundary. According to the definition of MC route, w must be within the left legal boundary. Let s denote a segment lying between the legal boundary and MC boundary. It is easy to see that s can be changed to a segment on the MC boundary, which is always a stair. This is a contradiction to the fact that w is an MC route.

For properties 2 and 3, first we will show that, on the left MC boundary which is also an MC route, horizontal and vertical lines can not be boundaries of the same MC region.

Suppose there are one horizontal and one vertical line on the boundary of a region. Without loss of generality, we may assume that the top pin is at the left of the bottom pin. Because of the monotonicity requirement, the two straight lines have only two relative positions, which are shown in Figure 2.7. In Figure 2.7(a), vertical line is to the upper left of horizontal line. The left MC boundary from a to

b must meet the right boundary. Otherwise, we can "push" the stair up and right and get a longer stair. In Figure 2.7(b), horizontal line is to the upper left of vertical line. The stairs start from a and b will both meet the right boundary. Otherwise, we can substitute the straight line by the corresponding stair. This, then, means the right MC boundary meets some point between a and b. In both positions, horizontal and vertical lines can not lie on boundary of the same region.

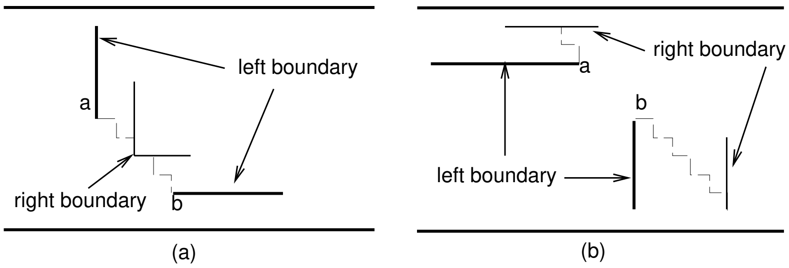

Figure 2.7: Relative positions of straight lines

For any other MC route, we only need to prove that it has the same length of horizontal and vertical lines as the left MC boundary in any separate MC region. This can be easily shown by comparing the coordinates of the the two points where the left MC boundary meets the right MC boundary. Every route must go through these two points, and if the horizontal distance of them is larger (smaller) than the vertical distance then every MC route must have horizontal (vertical) lines whose length will be the difference of the two distances. □

2.3.2 Space allocation

To make the presentation clear, in previous section, we discuss each net independently as though other nets do not exist. In this section, we will consider the interaction among different nets. Theorem 1 allows us to focus only on MC routes.

We will first consider the worst-case crosstalk between two neighboring nets.

Definition 5 (potential crosstalk) *The maximum crosstalk between any MC routes* *of net i and net $i+1$ is called their potential crosstalk and is denoted by p_i.*

Since only straight lines can induce crosstalk, the potential crosstalk happens when two MC routes have their straight lines as close to each other as possible. In fact, a general way to compute potential crosstalk p_i is to route net i along its right MC boundary and route net $i+1$ as close to it as possible. For example, given two neighboring nets with their MC boundaries as shown in Figure 2.8(a), Figure 2.8(b) shows the general way to compute the potential crosstalk, Figure 2.8(c) gives one of other situations when potential crosstalk happens.

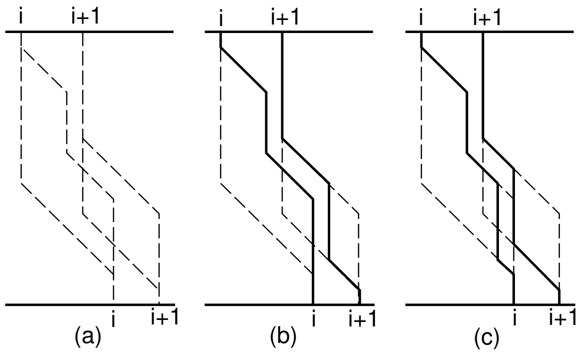

Figure 2.8: Potential crosstalk

If $p_i \leq C_i$, net i and net $i+1$ can always be routed close to each other without crosstalk violation. But if $p_i > C_i$, we can not route net i and net $i+1$ closely. The reason is that we are now using MC routes and, according to Lemma 2, changing shape can not reduce the length of straight lines, hence closely routed MC routes of net i and net $i+1$ will always have crosstalk p_i. Comparing Figure 2.8 (b) and (c) illustrates this conclusion. Therefore, to reduce the crosstalk between two MC routes, space must be inserted. For example, to reduce crosstalk in Figure 2.8(c), as shown in Figure 2.9(a), we can change straight line ab to $a'b'$ or straight line cd to $c'd'$, which is equivalent to inserting space between ab and $a'b'$, or between cd and $c'd'$. It must also be noticed that, for each space insertion, there is a limit on the maximum achievable crosstalk reduction. Beyond that limit, inserting space can not reduce crosstalk. Such an example is shown in Figure 2.9(b).

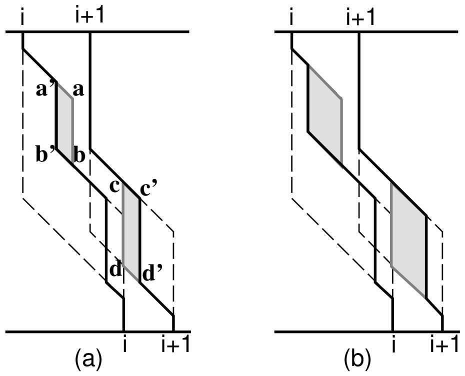

Figure 2.9: Crosstalk reduction vs. space insertion

Since each net must be routed in an MC route, the space which can be used to reduce crosstalk is restricted within its left and right MC boundaries (which are the left- and right-most MC routes according to Lemma 2), that is, the space in the MC regions. But the space in each MC region can not be counted independently. Different nets may share the space in their MC regions. For example, consider once again the two nets shown in Figure 2.8(a). When net i is routed along its right MC boundary, as shown in Figure 2.10(a), the shaded region in net $i+1$ can not be used. Similarly, as shown in Figure 2.10(b), if net $i+1$ is routed along the left MC boundary, the shaded region of net i can not be used, too. In fact, the two shaded regions must be treated as the same block shared by the two nets.

Space sharing is not only happened between neighboring nets, it may span among a set of consecutive nets. Generally, *shared space* can be identified as follows. Here we call a net a *falling net* if its top pin is at the left of its bottom pin, or a *rising net* if its top pin is at the right of its bottom pin.

Definition 6 (shared space) *Given net i and net j with $i < j$, we can move net j left and down both by $j - i$ units if it is a falling net; or move it left and up both*

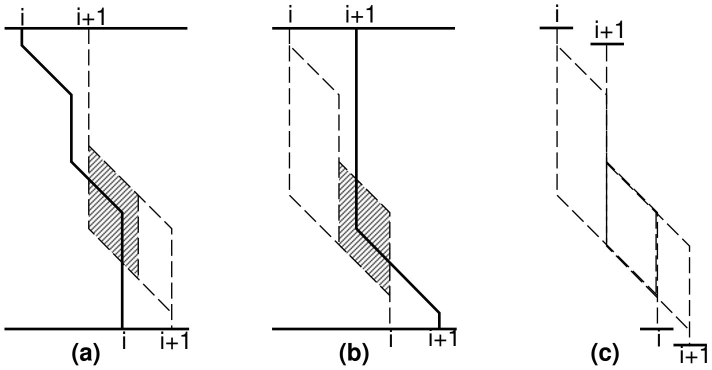

Figure 2.10: Space sharing among nets

by $j - i$ units if it is a rising net. The overlapped space in their MC regions then is their shared space.

For the two nets shown in Figure 2.8(a), we will move net $i+1$ left and down both by $(i+1) - i = 1$ unit, as shown in Figure 2.10(c). The overlapped blocks in their MC regions are actually the shaded blocks in Figure 2.10 (a) and (b).

After identifying the shared space among different nets, we can now count the actual space which can be used to reduce crosstalk.

Definition 7 (MC space and MC blocks) *For each net i, where $1 \leq i \leq n$, move it left and down both by $i - 1$ units if it is a falling net; or move it left and up both by $i - 1$ units if it is a rising net. The space within the MC regions then (shared space is counted only once) is called MC space. The whole MC space may be divided into blocks by MC boundaries. Each block is called an MC block.*

To make it clear, we consider the following example composed of four falling nets. The nets together with their MC boundaries are given in Figure 2.11(a). As shown in Figure 2.11(b), nets 2, 3, and 4 are moved left and down by 1, 2, and 3 units, respectively. The MC space is the space within the MC regions. It is composed of 3 MC blocks: S_1, S_2, S_3, which are separated by some MC boundaries.

Showing each net separately in Figure 2.11(c), we can find that S_1 is shared by net 1 and net 2, S_2 is shared by all four nets, and S_3 is shared by net 3 and net 4.

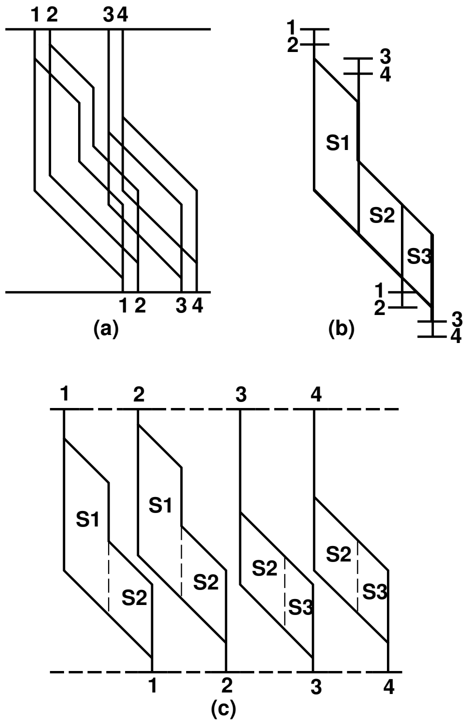

Figure 2.11: Identifying MC blocks and MC space

The following lemma summarizes the relationship between MC space insertion and crosstalk reduction.

Lemma 3 *Let p_i be the potential crosstalk between net i and net $i+1$. If, in a routing solution composed of MC routes, the crosstalk between the two nets is R_i, then at least $p_i - R_i$ units of MC space is inserted between them.*

Proof: According to the definition of potential crosstalk, the two MC routes will have crosstalk p_i when there is no MC space inserted. Because the crosstalk comes only from straight lines, and all MC routes of a net will have the same length of straight lines, there must be $p_i - R_i$ unit length of wires on the straight lines which have distance of at least two from the other net. This means at least $p_i - R_i$ units of MC space are between these two nets. □

In summary, if we put two neighboring MC routes close to each other, the potential crosstalk p_i will be induced. Based on the properties of MC routes, in order to reduce one unit crosstalk, one unit of MC space must be inserted. The MC space is composed of MC blocks, each of which is shared by a set of nets. Therefore, in order to get a feasible routing solution, MC space must be allocated properly to each pair of nets. The space providers are the MC blocks, each with a finite amount of space. The space consumers are all pairs of neighboring nets, each with a requirement of $\max\{p_i - C_i, 0\}$. Of course, each MC block has a limit on maximum achievable crosstalk reduction (recall the case in Figure 2.9(b)). Furthermore, a set of blocks may also have such limit in their total crosstalk reduction (will be explained in next section). These restrictions, together with the space allocation problem, can be modeled by a flow network. This will be presented in the next section.

2.3.3 Network flow

In this section, we solve the space allocation problem derived in the previous section. We have MC blocks, each has a finite amount of MC space which will be allocated to a set of pairs of neighboring nets. Each pair of neighboring net, based on their potential crosstalk p_i and crosstalk bound C_i, requires $\max\{p_i - C_i, 0\}$ units of MC space to be feasible. Since the objective of MC space allocation is to reduce crosstalk, the amount allocated from a set of blocks to each pair of nets must be restricted to make sure the same amount of crosstalk can actually be reduced. We will model the space allocation as a maximum flow problem. The flow network can be constructed as follows.

Suppose we identified m MC blocks in the CCRR problem. For each MC block S_i, where $1 \leq i \leq m$, a node S_i is introduced. Also, there will be $n-1$ nodes $D_1, D_2, \ldots, D_{n-1}$ which represent the $n-1$ pairs of neighboring nets. We plan to use the flow from S_i to D_j to represent the amount of MC space allocated from block S_i to the pair of nets j and $j+1$. Therefore, we add a source node s and a sink node t. For $1 \leq i \leq m$, there is an edge from s to each S_i, whose capacity is the amount of space in block S_i, used to model the available space. For $1 \leq j \leq n-1$, there is also an edge from each D_i to t with capacity of $\max\{p_i - C_i, 0\}$, representing the amount of space requirement.

The network structure between nodes S_i, for $1 \leq i \leq m$, and nodes D_j, for $1 \leq j \leq n-1$, is used to control the space allocation, such that, if an amount of space is allocated from a block, the same amount of crosstalk can be reduced. It can be constructed independently for each pair of nets. Consider net j and net $j+1$. Node S_i can have flows into D_j if and only if the two nets may have crosstalk within or along the boundary of block S_i. This amount of crosstalk also gives the limit on the maximum achievable crosstalk reduction by S_i, and will be used as the capacity on the edge going from S_i. For adjacent blocks with common boundaries, the maximum crosstalk they can reduce together is smaller than the summation of their limits. In this case, a *restriction node* R_k is introduced. The edges from these blocks are made to enter R_k and the capacity on the edge from R_k is set to control the flow.

To make it clear, we consider a pair of nets with their MC boundaries shown in Figure 2.12(a). Suppose their MC blocks are identified as in Figure 2.12(b). The dimension of each block is given at the upper-right corner, where the first number is the length of the straight boundary, and the second number is the width of the block. Here, net i and net $i+1$ could be routed along the right boundary of block S_1 and induce 7 units of crosstalk on the straight lines. Therefore, S_1 can reduce at most 7 units of crosstalk for them. They could also be routed closely in block

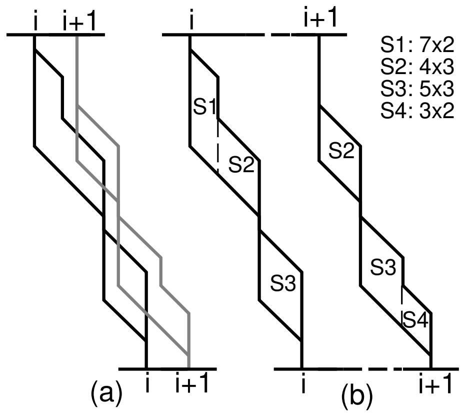

Figure 2.12: MC blocks of two neighboring nets

S_2 and induce 4 units of crosstalk, which is the length of straight lines they have in block S_2. Therefore, S_2 can reduce at most 4 units of crosstalk. But S_1 and S_2 have a common boundary and together can reduce at most 7 units of crosstalk. Hence, a restriction node R_1 is introduced. As shown in Figure 2.13, three edges are used to control the flows: (S_1, R_1) with capacity of 7, (S_2, R_1) with capacity of 4, and (R_1, D_i) with capacity of 7. Similarly, S_3, S_4 can reduce at most 5 and 3 units of crosstalk separately, but only 5 units together. These are controlled by the edges $(S_3, R_2), (S_4, R_2)$ and (R_2, D_i). Doing the construction for every pair of neighboring nets, we will get a network shown in Figure 2.13.

The correctness of our approach is given by the following theorem.

Theorem 2 *Suppose $G = (V, E, s, t, C)$ is the flow network constructed from a given CCRR problem, where s and t are the source and sink, respectively. The given problem has a feasible routing solution if and only if there is a flow which saturates all edges entering t.*

Proof:

$\Leftarrow$) Suppose the given CCRR problem has a feasible routing. According to Theorem 1, we can assume it is composed of MC routes. By Lemma 3, at least

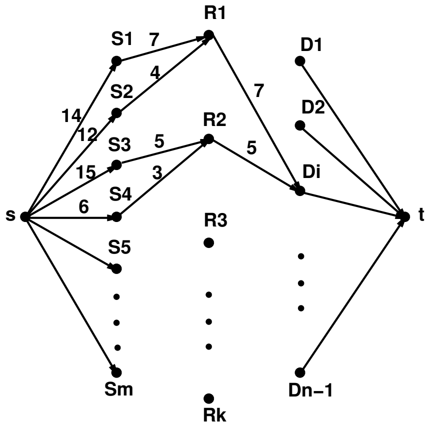

Figure 2.13: Flow network for space allocation

$p_i - C_i$ units of MC space are between net i and net $i+1$, for $1 \leq i \leq n-1$. This gives a flow in G saturating all edges entering t.

$\Rightarrow$) If G has a flow which saturates all edges entering t, a feasible routing solution can be constructed by using the information from the flow. The nets will be routed one by one from left to right. When routing each net, say net i, two conditions must be satisfied. One is that the crosstalk between net $i-1$ and net i, R_{i-1}, is no more than C_{i-1}. The other is that if the total flows from S_j to $D_i, D_{i+1}, \ldots, D_{n-1}$ is M then at least M units of spaces in S_j are at the right of net i. For the first net, net 1, routing trivially along its left MC boundary fulfills both conditions. Now suppose net i has been routed. We initially route net $i+1$ as close to net i as possible. Consider each flow to D_i with value z. If it comes from a block presented only in MC regions of net i, at least z units of spaces have been left between the route of net i and its right MC boundary. Hence at least z units of crosstalk has been reduced from potential crosstalk p_i. Otherwise, it comes from a block in MC regions of net $i+1$. In this case, we can insert z units of spaces from the block along the straight lines of net $i+1$. This will also reduce z units of crosstalk. Since the total amount of flows to D_i is $p_i - C_i$ if $p_i > C_i$, the final crosstalk R_i is no more than $p_i - (p_i - C_i) = C_i$. Since we use only spaces flowing to D_i, after routing of net $i+1$, the second condition is also kept.

To make it clear, consider again the example in Figure 2.12. Suppose net i has been routed as in Figure 2.14(a), and the flows from S_1, S_2, S_3 and S_4 to D_i are $2, 4, 3$ and 0, respectively. Then net $i+1$ will be routed as in Figure 2.14(b).

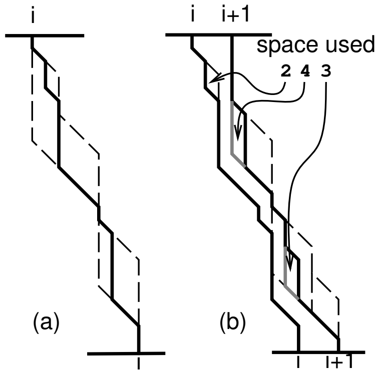

Figure 2.14: Construct routing from maximum flow

$\square$

Based on the above theorem, a routing solution which maximizes the min-slack can be easily constructed.

Corollary 2.1 *Suppose a CCRR problem has a feasible solution and the flow network derived from it is $G = (V, E, s, t, C)$. If we simultaneously increase the capacities on all edges entering t while keeping them saturated, the last maximum flow will give a routing solution with maximum min-slack.*

2.3.4 Algorithm description

Summarizing all contents in the previous sections, the pseudo-code of the optimal algorithm is presented in Figure 2.15.

It is easy to see that, in the optimal algorithm, the maximum flow computation will be the dominant part of time complexity. Suppose there are n nets, and

```
Algorithm  Optimal Algorithm for CCRR problem
   Construct legal boundaries;
   Derive MC boundaries;
   Compute potential crosstalk;
   Identify MC space and MC blocks;
   Build flow network G = (V, E, s, t, C);
   Compute max-flow of G;
   if the flow does not saturate edges entering t
      then ‘‘No feasible solution’’;
   else {
      Increase capacities on edges entering t simultaneously by 1;
      Augment current flow;
   } while flow saturates edges entering t;
   Route nets according to the flow.
```

Figure 2.15: The optimal algorithm for CCRR problem

the width and length of the channel are W and L, respectively. It can be proved that the number of MC blocks is bounded by $O(n)$. In the worst case, each net can have $O(n)$ blocks in its MC regions. So the numbers of nodes and edges in the flow network are $O(n^2)$. Since the total amount of spaces is at most WL, simply using Ford-Fulkerson algorithm gives us $O(n^2WL)$ worst-case running time [18].

2.4 Post-processing

As we can see, routing solutions given by the above algorithm are composed of MC routes, which may have many stairs. Although stairs are useful to reduce crosstalk, not all of them are necessary in the final solution. To reduce the number of bends, we design a heuristic as a post-processing procedure for the above algorithm.

Intuitively, the procedure is an iterative “rip-up and reroute” process. Each time, only one net is chosen to be rerouted, and other nets are all fixed. We want to find a new route with fewer bends than the old one, but more importantly, it can not increase the crosstalk (on itself or others). To fulfill this requirement, the *contour* of a chosen net can be defined as follows. Given a routing solution, the *left contour* of net i is its leftmost route such that the segments inducing crosstalk are on its given route; the *right contour* of net i is its rightmost route such that the segments inducing crosstalk are on its given route. For example, in the given routing solution shown in Figure 2.16, the dashed lines shows the contour of net i.

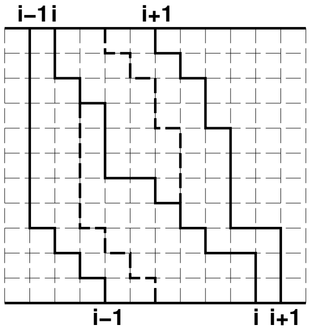

Figure 2.16: Contour

It is easy to see that, in a feasible routing solution, changing one route within its contours will end up with another feasible routing solution. The post-processing procedure consists of two passes. In the first pass, net 1 to net n are changed, one by one in that sequence. For each net, the contour is first derived, then a line-search algorithm (such as [37]) is used to find a leftmost route with minimum bends within the contour. The second pass processes in the reverse order, from net n to net 1. Instead, for each net, a rightmost minimum bend route is found within the contour. An example result of the post-processing procedure is shown in Figure 2.17.

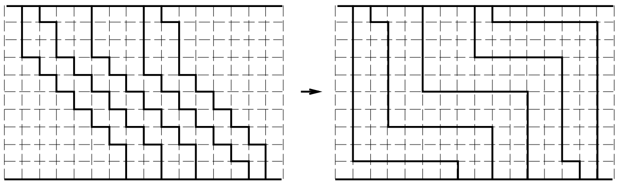

Figure 2.17: Post-processing

Table 2.1: Experimental results

data	#nets	chnl width	#MC blocks	#nodes	#arcs	time(sec.)
P_1	20	7	4	26	30	0.01
P_2	30	10	29	123	219	0.01
P_3	35	35	33	396	948	0.03
P_4	50	30	126	705	1488	0.04
P_5	100	50	239	1497	3221	0.10
P_6	200	100	1142	8882	19844	0.55
P_7	200	100	2112	8710	18704	0.33
P_8	400	100	881	4156	8739	0.24
P_9	500	100	1323	8720	19351	0.57
P_{10}	500	200	2169	17914	45619	1.26

2.5 Experimental results

To test the efficiency of our algorithm, we implemented it on an IBM RS6000 machine. After computing potential crosstalk and identifying MC blocks, the program constructs a flow network and writes it out in the DIMACS format. A maximum flow program (here we use **gold_hlf** [3]) is then run on the network to compute the maximum flow. Since there is no benchmarks for river routing, let along those with crosstalk constraints, routing instances are generated by a program. For each instances, we give two parameters: channel width and number of nets. The pin positions and crosstalk constraints are randomly generated. Here, we report the results on 10 such instances $P_1, P_2, \ldots, P_{10}$. Besides the running time, for each instance, the number of identified MC blocks and the number of nodes and arcs in the networks are also reported in Table 2.1.

2.6 Conclusions

In this chapter, we presented an optimal algorithm to generate routing solutions for the CCRR problem. In the formulation of the problem, crosstalk constraints are specified on each pair of neighboring nets and routes are required to be monotonic.

But in practice, crosstalk constraints may be given on each net. In this case, our algorithm can follow a pre-processing step which translates "net constraints" into "net-pair constraints". Furthermore, although monotonic routes are sufficient for the traditional river routing problem, it is possible to find examples where non-monotonic routes are needed to satisfy crosstalk constraints. Whether the CCRR problem with non-monotonic routes can be solved in polynomial time is still an open problem.

Chapter 3

Maze Routing with Crosstalk Constraints

3.1 Introduction

In this chapter, we deal with how to control crosstalk in maze routing. Maze routing belongs to the category of sequential routing algorithms, that is, it routes only one single net at a time. After a net has been routed, the area occupied by its wiring will be marked as obstacles to the unrouted nets. When routing a net in the presence of obstacles and routed nets, besides crosstalk induced on the current net, extra crosstalk may be created on other routed nets by the current one. Therefore, we need to consider how to simultaneously control crosstalk on each net when routing a new net. Under a simplified crosstalk model and the assumption of two-terminal nets, we formulate it as a *crosstalk-constrained maze routing problem*. It is then modeled as a *multi-constrained shortest path problem* and shown to be strongly NP-complete. Based on the Lagrangian relaxation technique, an effective heuristic algorithm is designed to solve it. This approach is so robust that it can be easily modified to accommodate more general crosstalk models and multi-terminal nets.

This chapter is organized as follows: Section 3.2 gives the formulation of the crosstalk-constrained maze routing problem. Section 3.3 discusses the complexity of the problem. In Section 3.4 we present a heuristic algorithm for the problem which is based on the Lagrangian relaxation technique. Section 3.5 considers extensions to more general crosstalk models and multi-terminal nets. Experimental results are reported in Section 3.6, and Section 3.7 concludes the chapter.

3.2 Problem formulation

In maze routing, the entire routing area is represented as a k-layer grid graph. All pins, edges of bounding boxes that enclose the cells are aligned on the grids. Each edge in the graph represents a track that can accommodate exactly one wire. If obstacles obstruct the routing region, the obstructed edges are deleted from the graph and a subgraph of the k-layer graph results. To simplify our presentation, we use only one layer here. Extension to k layers is straight-forward. Given a set of *nets*, each of which is defined by a set of pins need to be connected, a *routing solution* is defined as a collection of node-disjoint trees each connects all pins in one net.

Generally speaking, crosstalk between two parallel straight wires is proportional to the coupling capacitance between them, which in turn is proportional to their coupling length and inversely proportional to their separating distance. Since the coupling capacitance between two wires decreases rapidly as the distance between them increases, we assume that crosstalk only exists between wires in adjacent rows or columns. Furthermore, we will set the proportion constant to 1 and treat the adjacent length between two wires as their crosstalk. The total crosstalk on one net is then defined as the summation of crosstalk induced on it. This is a simplified crosstalk model which was also used in [19, 20, 68]. It is used here to simplify the presentation. More general models can be easily adopted and will be presented in

Section 3.5.

Suppose there are n nets. Then, for each net i, where $1 \leq i \leq n$, a constant C_i will be given as the maximum crosstalk net i can tolerate. For any routing solution, let R_i denote the crosstalk on net i for $1 \leq i \leq n$. If $R_i \leq C_i$, for all $1 \leq i \leq n$, the routing solution is called a *feasible routing solution.* For example, under the above crosstalk model, the crosstalk of the nets shown in Figure 3.1 is: $R_1 = 2+7+1 = 10, R_2 = 2+2 = 4, R_3 = 5+3+7+2+1 = 18, R_4 = 5+3 = 8$.

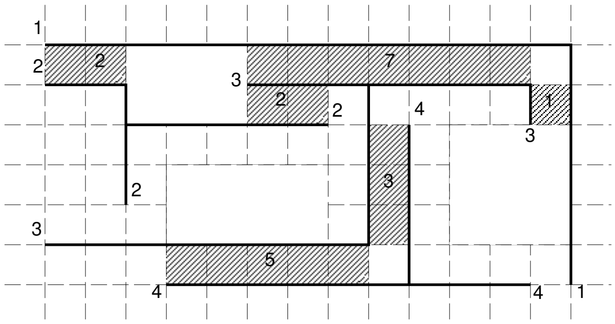

Figure 3.1: A routing solution and its crosstalk

Since simultaneously routing all nets is prohibitively complicated, a sequential routing approach is adopted here. First, an order is decided on all nets. Then, nets are routed one at a time according to that order. After routing one net, the area used by that net is no longer available to other nets and will be treated as obstacles. Obviously, net ordering is of great importance in this approach. But there is no generally agreed method to compute it. In practice, rip-up and re-route is heavily employed to alleviate the dependence on net ordering. Therefore, the main problem in maze routing is how to route one net in the presence of obstacles and routed nets. We first consider only two-terminal nets here. Extension to multi-terminal nets will be given in Section 3.5.

When routing a net among previously routed nets, besides crosstalk on the current net, there is also extra crosstalk induced on other routed nets. Therefore, it is necessary to control the crosstalk on all nets simultaneously. This can be formulated as the following problem.

Problem 1 (Crosstalk-Constrained Maze Routing (CCMR)) *Suppose we have a grid routing graph and nets $1, 2, \ldots, m-1$ are already routed on it. Given net m with two pins p_1, p_2 and crosstalk constraints C_i, for $1 \leq i \leq m$, find a shortest route for net m such that $R_i \leq C_i$ for all $1 \leq i \leq m$.*

Traditional maze routing algorithms route a net by a shortest path or a minimum bend path and can also be modified to accommodate various constraints and path selection criteria [40]. But it is hard to modify them to solve the CCMR problem, since what required here is not minimizing the total crosstalk but balancing crosstalk among different nets. This can be illustrated by the example in Figure 3.2.

3.3 Complexity

Given an instance of the CCMR problem, we can construct a vector-weighted graph as follows. The grid nodes and edges which are occupied by the obstacles and previously routed nets are removed from the grid graph. Each remaining edge e will have a vector $(v_1^e, v_2^e, \ldots, v_m^e)$ as its weight, where v_i^e equals to the crosstalk induced on net i if net m is routed on edge e. For example, given a CCMR problem shown in Figure 3.3(a), the correspondent graph is constructed in Figure 3.3(b).

It can be shown that solving the CCMR problem is equivalent to solving the following problem.

Problem 2 (Multi-Constrained Shortest Path (MCSP)) *Given a vector-weighted graph, two nodes s and t in the graph, and a set of constants $B_1, B_2, \ldots, B_n$, find a shortest path P from s to t such that $\sum_{e \in P} v_i^e \leq B_i, \forall 1 \leq i \leq n$.*

By treating the edge length as another dimension in the weight vectors, the decision version of the above problem can be stated as follows.

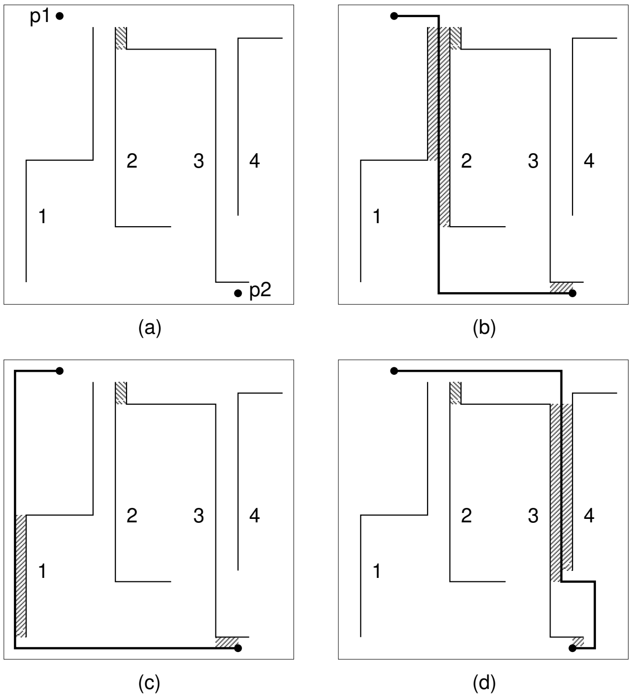

Figure 3.2: (a)A CCMR problem: nets 1, 2, 3, 4 are routed and net 5 is (p_1, p_2); $C_1 = 2, C_2 = 3, C_3 = 30, C_4 = 36, C_5 = 35$. (b)Shortest path route (not feasible): $R_1 = 12, R_2 = 20, R_3 = 4, R_4 = 0, R_5 = 32$. (c)Minimum total crosstalk route (not feasible): $R_1 = 11, R_2 = 2, R_3 = 4, R_4 = 0, R_5 = 13$. (d)Optimal solution: $R_1 = 0, R_2 = 2, R_3 = 19, R_4 = 15, R_5 = 32$

Problem 3 (MCSP-Decision) *Given a vector-weighted graph, two nodes s and t in the graph, and a set of constants $B_1, B_2, \ldots, B_{n+1}$, find a path P from s to t such that $\sum_{e \in P} v_i^e \leq B_i, \forall 1 \leq i \leq n+1$.*

This problem is similar to the Shortest Weight-Constrained Path (SWCP) problem in [21]. But SWCP problem is weakly NP-complete, and can even be solved in polynomial time if all weights are equal or all lengths are equal. However, for MCSP problem, we have the following result.

Theorem 3 *MCSP-Decision problem is strongly NP-complete.*

Proof: Given an instance of MCSP-Decision problem, a path from s to t can be

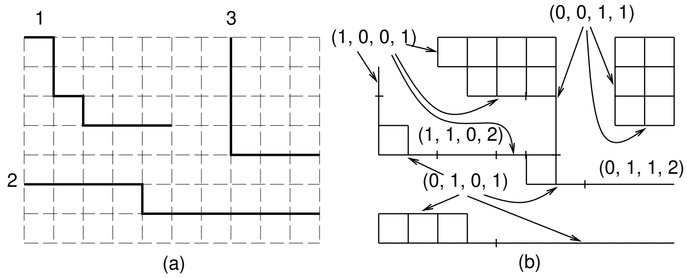

Figure 3.3: (a)A CCMR instance (b)Corresponding vector-weighted graph (edges not labeled have weight (0,0,0,0))

checked in polynomial time to see whether the constraints are obeyed. This means that MCSP-decision problem is in NP.

We will show that MCSP-Decision is strongly NP-hard by transforming 3-PARTITION [21] to it. 3-PARTITION is defined as follows.

> INSTANCE: Set A of $3m$ elements, a bound $B \in Z^+$, and a size $s(a) \in Z^+$ for each $a \in A$ such that $B/4 < s(a) < B/2$ and such that $\sum_{a \in A} s(a) = mB$.
>
> QUESTION: Can A be partitioned into m disjoint sets $A_1, A_2, \ldots, A_m$ such that, for $1 \leq i \leq m, \sum_{a \in A_i} s(a) = B$?

Given one of the instances described above, we can construct an instance of MCSP-Decision as follows. Denote the elements in A by $a_1, a_2, \ldots, a_{3m}$. Corresponding to each $a_i, 1 \leq i \leq 3m$, we will have $m+1$ nodes $s_i, t_{i1}, t_{i2}, \ldots, t_{im}$ and m edges $(s_i, t_{i1}), (s_i, t_{i2}), \ldots, (s_i, t_{im})$. The weight vector of edge (s_i, t_{ij}) is defined as follows: all elements are 0 except the jth element, which will be $s(a_i)$. The nodes and edges for different elements will be connected by adding edges (t_{ij}, s_{i+1}) for $1 \leq i \leq 3m-1, 1 \leq j \leq m$. We will add one more node t and m more edges $(t_{3m,j}, t), 1 \leq j \leq m$. The weights of all the newly-added edges will be 0. This will give us a weighted graph as shown in Figure 3.4. We will set the constraints $B_1 = B_2 = \cdots = B_m = B$.

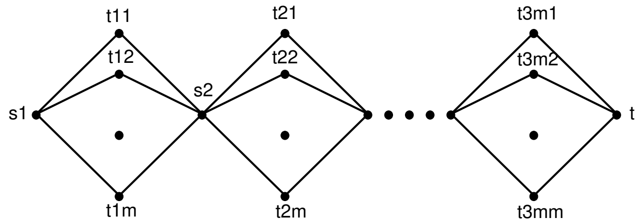

Figure 3.4: An instance constructed from 3-PARTITION

Now, suppose there is a path P from s_1 to t such that $\sum_{e \in P} v_j^e \leq B, \forall 1 \leq j \leq m$. According to the construction, there are only $3m$ non-zero weight edges on the path, that is,

$$\begin{aligned} \sum_{e \in P} v_j^e &= \sum_{1 \leq i \leq 3m, t_{ij} \in P} v_j^{(s_i, t_{ij})} \\ &= \sum_{1 \leq i \leq 3m, t_{ij} \in P} s(a_i) \\ &\leq B \end{aligned}$$

Hence, the following partition is a YES witness for the 3-PARTITION:

$$A_j = \{a_i | t_{ij} \in P\}, 1 \leq j \leq m$$

On the other hand, suppose there is a partition $A_1, A_2, \ldots, A_m$ such that, for $1 \leq i \leq m, \sum_{a \in A_i} s(a) = B$. Then, we can choose t_{ij} to be on the path if and only if $a_i \in A_j$. It is easy to check that such a path fulfills the constraints of MCSP-Decision problem. $\square$

3.4 A heuristic based on Lagrangian relaxation

As we have found in the NP-completeness proof, the difficulty of MCSP problem comes from the requirement to balance costs among different dimensions of the weight vector. Since minimizing the weighted sum of all dimensions in the weight vector can be easily solved, an alternative way to balance the costs is first giving relative weight for each dimension and then minimizing the weighted sum of them. Lagrangian relaxation is a way to search better relative weights by iteratively adjusting them, and hence, will be used to solve the MCSP problem.

Given an MCSP problem, by relaxing the constraints into the objective function, that is, giving relative weights to the constraints, we have the following problem.

Problem 4 (Lagrangian Sub-Problem (LSP)) *Given an instance of MCSP and a set of non-negative constants $\mu_1, \mu_2, \ldots, \mu_n$, find a path P in the graph such that $\sum_{e \in P} 1 + \sum_{i=1}^{n} \mu_i (\sum_{e \in P} v_i^e - B_i)$ is minimized.*

Usually, the constants $\mu_1, \mu_2, \ldots, \mu_n$ are called Lagrangian multipliers. Note that

$$\begin{aligned} & \sum_{e \in P} 1 + \sum_{i=1}^{n} \mu_i (\sum_{e \in P} v_i^e - B_i) \\ = \quad & \sum_{e \in P} (1 + \sum_{i=1}^{n} \mu_i v_i^e) - \sum_{i=1}^{n} \mu_i B_i, \end{aligned}$$

where $\sum_{i=1}^{n} \mu_i B_i$ is constant for fixed multipliers. Therefore, by treating $1+\sum_{i=1}^{n} \mu_i v_i^e$ as the weight on edge e, we can solve the LSP problem efficiently by Dijkstra's shortest path algorithm [14]. Let $L(\mu_1, \ldots, \mu_n)$ denote the optimal objective of the above problem, that is,

$$L(\mu_1, \ldots, \mu_n) = \min_P \{ \sum_{e \in P} 1 + \sum_{i=1}^{n} \mu_i (\sum_{e \in P} v_i^e - B_i) \}.$$

The Lagrangian multiplier problem for MCSP is as follows.

Problem 5 (Lagrangian Multiplier Problem (LMP))

$$\begin{aligned} \textit{Maximize} \quad & L(\mu_1, \ldots, \mu_n) \\ \textit{subject to} \quad & \mu_i \geq 0, \forall 1 \leq i \leq n \end{aligned}$$

It is obvious that for any $\mu_i \geq 0, 1 \leq i \leq n$,

$$
\begin{aligned}
& \min_P \{ \sum_{e \in P} 1 : \sum_{e \in P} v_i^e \leq B_i, 1 \leq i \leq n \} \\
\geq \quad & \min_P \{ \sum_{e \in P} 1 + \sum_{i=1}^{n} \mu_i (\sum_{e \in P} v_i^e - B_i) : \\
& \sum_{e \in P} v_i^e \leq B_i, 1 \leq i \leq n \} \\
\geq \quad & \min_P \{ \sum_{e \in P} 1 + \sum_{i=1}^{n} \mu_i (\sum_{e \in P} v_i^e - B_i) \} \\
= \quad & L(\mu_1, \ldots, \mu_n).
\end{aligned}
$$

Because the above inequality is true for any $\mu_i \geq 0, 1 \leq i \leq n$, it is also true for specific μ_is which make $L(\mu_1, \ldots, \mu_n)$ maximum, that is,

$$
\begin{aligned}
& \min_P \{ \sum_{e \in P} 1 : \sum_{e \in P} v_i^e \leq B_i, 1 \leq i \leq n \} \\
\geq \quad & \max_{\mu_1, \ldots, \mu_n} L(\mu_1, \ldots, \mu_n).
\end{aligned}
$$

This means that the optimal solution of the LMP problem is always a lower bound of the MCSP problem. This property also gives the following optimality criteria for the MCSP problem.

Theorem 4 *If we have a path P such that $\sum_{e \in P} v_i^e \leq B_i$ for all $1 \leq i \leq n$, and $\sum_{e \in P} 1 = L(\mu_1, \ldots, \mu_n)$ for some $\mu_i \geq 0, 1 \leq i \leq n$, then P is an optimal solution to the MCSP problem, and μ_i, for $1 \leq i \leq n$, is an optimal solution to the corresponding LMP problem.*

As stated by the following theorem, the LMP problem has a very good property for optimization.

Theorem 5 *The LMP is a convex programming problem.*

Based on the theorem, non-linear programming techniques can be used to solve the LMP problem. Since $L(\mu_1, \ldots, \mu_n)$ as a function is not everywhere differentiable, sub-gradient method is the usual approach. The pseudo-code of this approach is described in Figure 3.5.

```
Algorithm  Sub-Gradient Method for LMP
  k = 0; μ_i = 0, ∀1 ≤ i ≤ n;
  while stop criteria not satisfied {
    Solve the LSP by Dijkstra's algorithm;
    μ_i = max{0, μ_i + θ_k(Σ_{e∈P} v_i^e − B_i)}, ∀1 ≤ i ≤ n;
    k = k + 1;
  }
```

Figure 3.5: The sub-gradient method

The general theory for sub-gradient method states that if $\theta_k \to 0$ and $\sum_{j=1}^{k} \theta_j \to \infty$ as $k \to \infty$, the above algorithm will converge to the optimal solution [1].

The only problem with this approach is that there may exist a "gap" between MCSP and its corresponding Lagrangian multiplier problem. That is, it is possible that

$$\min\{\sum_{e \in P} 1 : \sum_{e \in P} v_i^e \leq B_i\} > \max L(\mu_1, \ldots, \mu_n).$$

Actually, solving MCSP problem is just one step in the whole scenario of a sequential maze routing. So, it is not necessary to spend too much time to get an optimal solution for MCSP. We will set up an upper bound on the number of iterations. During the iterations, we also record feasible solutions, that is, those paths P such that $\sum_{e \in P} v_i^e \leq B_i$. If the sub-gradient method did not give the optimal solution within those iterations, we will use the shortest one of the recorded feasible paths as our solution.

3.5 Extensions

In previous sections, to simplify the presentation, we based our work on a simplified crosstalk model and two-terminal nets. Actually, our Lagrangian relaxation based approach is such a robust method that it can be easily extended to multi-terminal nets and more general crosstalk models.

3.5.1 Multi-terminal nets

For multi-terminal nets, our objective is to find a minimum length Steiner tree route such that the crosstalk on each net is under constraint. But even without considering crosstalk, finding a minimum length Steiner tree is NP-hard [21]. Fortunately, there are many heuristic algorithms exist in the literature [24]. Many of them have proven worst-case upper bounds and produce good results in practice.

Using our Lagrangian relaxation based approach, when routing a multi-terminal net, the crosstalk constraints are also relaxed into the objective function. Instead of a path P, the resulting LSP needs to find a Steiner tree T, such that $\sum_{e \in T} 1 + \sum_{i=1}^{n} \mu_i (\sum_{e \in T} v_i^e - B_i)$ is minimized. Similarly, this can be solved by finding a minimum Steiner tree on a weighted graph. Although this can not be solved exactly, any minimum Steiner tree heuristic on weighted graphs can be used to find an approximate solution.

3.5.2 More general crosstalk model

In practice, besides coupling capacitance, crosstalk between two nets also depends on their functional and temporal sensitivities [27] and the relative drive strengths of their drivers. Therefore, two nets may not have crosstalk even they are routed closely, and for two adjacent wires, crosstalk induced on one wire may not equal to crosstalk induced on the other. In this case, we only need to change the weight vectors to represent the actual crosstalk induce on each net.

In previous sections, crosstalk is only considered between adjacent wires. That is a very reasonable model since, on one hand, the coupling capacitance decreases rapidly as the distance increases; on the other hand, if there is space between wires, shielding wires can be inserted when necessary. However, if crosstalk between non-adjacent wires must be considered, our approach can also be extended. In this case, the coupling capacitance can be computed as follows.

$$\textit{coupling-capacitance} = \alpha \frac{\textit{length}}{\textit{distance}^{\beta}},$$

where β is a small constant which was estimated to be about 2 in [48].

The only difference between non-adjacent model and the adjacent one is that, in non-adjacency model, we must consider the effect of shielding. That is, when a new net is routed between two previously routed ones, it may shield one of them from the other, and therefore decrease crosstalk on them. This can be illustrated in Figure 3.6. This effect can be taken into consideration by changing the weights of edges as follows. Suppose edge e lies between routed nets i and j, and c_{ij} is the crosstalk between net i and net j which might be shielded by e. Let c_i and c_j denote the crosstalk between e and nets i and j, respectively. Then we can set $v_i^e = c_i - c_{ij}$ and $v_j^e = c_j - c_{ij}$ instead of $v_i^e = c_i$ and $v_j^e = c_j$. If $c_i < c_{ij}$ or $c_j < c_{ij}$, negative weight elements are introduced. Therefore, Dijkstra's algorithm can not be used. Instead, we can use Bellman-Ford algorithm to compute the shortest path [14].

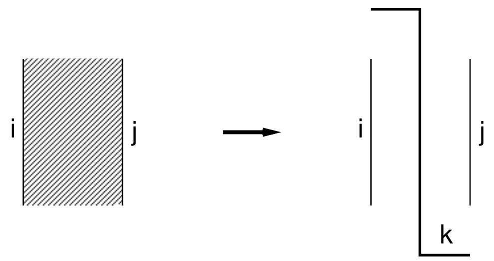

Figure 3.6: Crosstalk between nets i and j is shielded by a newly routed net k

Note that it can not be guaranteed to find a shortest path in terms of the weighted sum of crosstalk. This is because when a net is routed in an empty U-form as shown in Figure 3.7, it will shield some segments of itself from other nets. Therefore, the total weight of a path may not be equal to the sum of the weights of its component paths, and a component of an optimal path may not be an optimal one. But since we do not favor such empty U-forms in practice, the little overestimation may help us to avoid it.

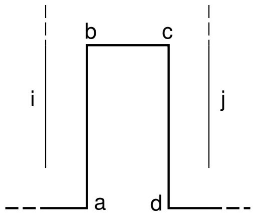

Figure 3.7: An empty U-form routing shields segment ab from net j, and segment cd from net i

3.6 Experimental results

We implement the Lagrangian relaxation based mazing routing algorithm in C++ on a Sun Sparc 5 workstation. Although our algorithm can be applied to multi-layer routing, the current implementation is only designed for two-layer routing in the HV model (i.e. one layer for horizontal wiring and one layer for vertical wiring). We do the experiments on the MCM benchmarks originated from 1993 Physical Design Workshop. These benchmarks are composed of multi-terminal nets and are intended for multi-layer routing. In the experiments, we first use the general maze routing (shortest path without considering crosstalk) to route as many nets as possible. This gives us a maximum number of nets which can be routed on a pair of layers. We then measure crosstalk on each net in this routing solution and call it the *base crosstalk* of that net. We systematically set the crosstalk constraint for each net to be a percentage of its base crosstalk and run our algorithm on it using the same net ordering. For comparison, we run once again the general maze routing on those nets. This time, after routing a net, we also compute its crosstalk. If the crosstalk violate the constraints, the current net is removed. We compare the numbers of feasibly routed nets by these two approaches and find that our algorithm can feasibly route more than 30% more nets than the general maze routing in average.

Figure 3.8 plots one of such experimental results on **mcm1** by setting the constraint of each net respectively to be 70%, 60%, 50%, 40%, and 0% of its base crosstalk.

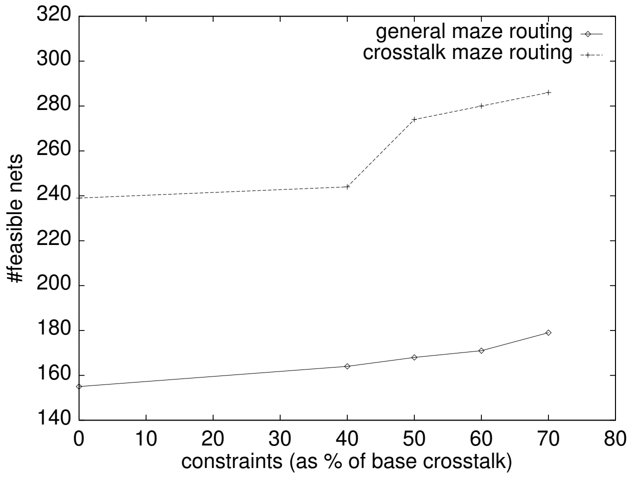

Figure 3.8: Comparison of the numbers of feasibly routed nets by the two algorithms

3.7 Conclusions

In this chapter, we considered how to control crosstalk in maze routing. When routing one net among other routed nets, besides the current net, there is crosstalk induced on other nets. Thus, the problem is a multi-constraint problem. We shown that the problem is strongly NP-hard. Based on Lagrangian relaxation, we designed a heuristic algorithm to solved the problem. Experimental results show that our approach is effective.

Chapter 4

Global Routing with Crosstalk Constraints

4.1 Introduction

Most previous works on crosstalk avoidance are focused on detailed routing [9, 19, 20, 26, 55, 68], where the estimation of crosstalk is accurate but the flexibility to avoid it is restricted. Therefore, it is often not possible to find a crosstalk-feasible solution in detailed routing if the global routing is crosstalk-blinded.

To make a global routing solution crosstalk risk-free, Xue et al. [64] proposed a post global routing adjustment procedure to modify a given routing solution. It used a gridded model and assumed that the crosstalk only exists between adjacent wires. That is, if there is a spare track between two wires, then there is no crosstalk between them. The crosstalk risk estimation is divided into two coupled procedures: bound partitioning and net ordering. Bound partitioning partitions the crosstalk bound of each net into the regions it routes through. Net ordering orders the nets in each region to require as few spare tracks as possible. Their approach is totally region-based, that is, crosstalk risks are defined by the regions requiring extra tracks and the number of extra tracks needed. If a region still requires extra tracks after bound partitioning and net ordering, some nets in the region are ripped up and re-routed using other regions.

In this chapter, we consider crosstalk avoidance during global routing. We use a crosstalk model which is more general than that of [64]. In order to have an accurate crosstalk estimation, we extend global routing to include layer and track assignments. Based on this extension, a novel *Crosstalk-Constrained Global Routing* problem is defined. Many sub-problems of this problem are shown to be NP-hard. Therefore, a two-stage heuristic approach is used to control crosstalk. First, an initial solution is constructed based on a new Steiner tree formulation to minimize the total crosstalk. After that, crosstalk on each net is estimated. The nets having crosstalk violations are then ripped up. They are re-routed by a Lagrangian relaxation technique to satisfy crosstalk constraints.

The rest of the chapter is organized as follows. Section 4.2 introduces the general crosstalk model we use. Section 4.3 extends the global routing to include the layer/track assignment and defines the crosstalk-constrained global routing problem. In Section 4.4 we show many sub-problems of the crosstalk-constrained global routing problem are NP-hard. Section 4.5 presents our global router for crosstalk control. Section 4.6 presents the experimental results. And Section 4.7 concludes the chapter.

4.2 Crosstalk model

Consider two coupling nets as shown in Figure 4.1(a). We can model them by a circuit shown in Figure 4.1(b). In this figure, v_1, v_2 represents the voltages at the outputs of the drivers, R_1, R_2 represents the driver resistances, C represents the coupling capacitance between the nets, and C_1, C_2 represents all other capacitances besides C. The crosstalk effect on net 2 can be thought of as the difference of voltages on node O with or without the signal switching in net 1. Here, we can find that besides the coupling capacitance C, the crosstalk effect also depends on driver strengths (R_1 and R_2), other load capacitances (C_1 and C_2), and the voltages (v_1 and v_2).

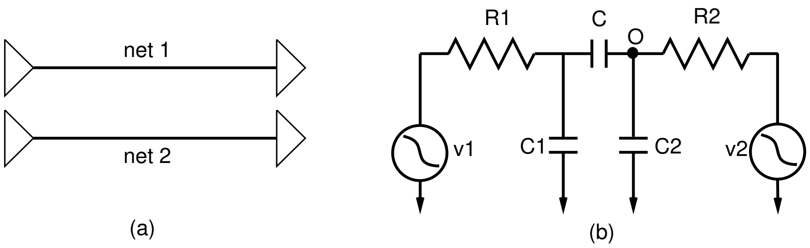

Figure 4.1: Modeling two coupling nets

Among all parameters affecting crosstalk effect, we assume that only the coupling capacitance is controllable in layout design. Other parameters are either not changeable or not preferred to be changed. However, they are useful information for crosstalk control in layout design. For example, if we know net 1 and net 2 will never switch at the same time, then we can safely route them together since the crosstalk effect will be zero even they are heavily coupled. We can model all parameters except coupling capacitance by a *crosstalk coefficient.* For each net i, the *crosstalk coefficient* from net j is a real number $e_{ij} \in [0, 1]$ which is the crosstalk on net i contributed by one unit of coupling capacitance from net j. For example, if $e_{ij} = 2e_{ik}$, then the crosstalk induced on net i by net j is twice of that by net k if they have the same coupling capacitance.

For a net, the crosstalk effects from other nets may not happen at the same time. But characterizing all cases requires exhaustive timing analysis, which in turn depends on crosstalk. Therefore, in the worse case, we can use the summation of all the effects from other nets as the total crosstalk on one net. That is,

$$X_i = \sum_{j \neq i} e_{ij} C_{ij}, \qquad (4.1)$$

where C_{ij} is the coupling capacitance between net i and net j.

Generally speaking, each element in a chip is coupled with every other element. But coupling capacitance decreases rapidly when an element is out of the neighborhood of the other element. For interconnection, coupling capacitance between perpendicular wires is also very small. Therefore, we assume coupling capacitance only exists between neighboring parallel wires and is given by the following formula:

$$C = \alpha \frac{length}{distance^{\beta}}, \tag{4.2}$$

where β is a constant which was estimated to be about 2 in [48].

4.3 Problem formulation

In traditional design flow, routing is divided into global and detailed routing. In global routing, the set of regions a net goes through is decided. In detailed routing, the actual route in each region is then computed. From the previous section, we know coupling is critically dependent on wire adjacency. However, in traditional global routing, for each net, only routing regions it goes through are decided. The adjacency information in each region is not available. Therefore, it is not possible to get accurate crosstalk estimation in that situation.

In order to estimate crosstalk during global routing, some "detailed" information must be known. Of course, doing the whole task of global and detailed routing together would be very complicated. Therefore, the idea is to do a "simplified" detailed routing "on the fly" in the process of global routing. This detailed routing must be simple enough to be merged into the global routing process, and at the same time, must be detailed enough for crosstalk estimation.

In fact, each region in the global routing stage can be viewed as a multi-layer rectangular area. We also assume that each layer is composed of tracks with equal spacing and each track can be used only by one net. During global routing, besides deciding which regions a net goes through, layer and track assignments in each region are also decided. Formally, we are given an undirected graph $G = (V, E)$, where each edge e represents a region $r(e)$ which has a length $l(e)$ and may have k layers each with a capacity $c_i(e)$, for $1 \leq i \leq k$. Also specified are n nets each is a subset of V. An *extended global routing solution* is an edge labeling S of the form $S(e) = (s_1, s_2, \cdots, s_k)$, where each s_i for $1 \leq i \leq k$ is a sequence of nets. For $1 \leq i \leq n$, the edges including i must form a Steiner tree for net i. If $s_i = (\cdots, 2, 1, 3, \cdots)$, this means that nets 1, 2, and 3 are routed through region $r(e)$ and placed in layer i in the order $2, 1, 3$. For example, an extended global routing solution is shown in Figure 4.2, where each region has only one layer.

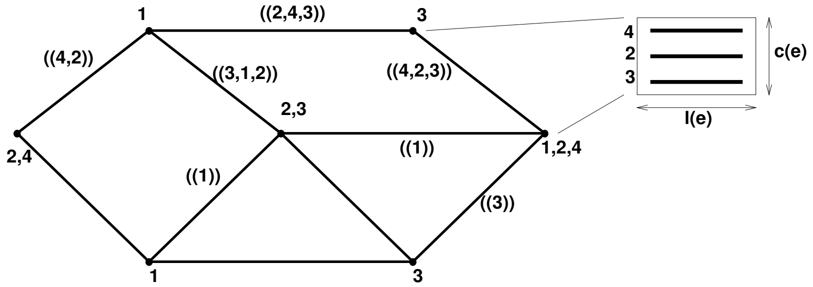

Figure 4.2: An extended global routing solution

Given an extended global routing solution, the adjacency information is known in each edge. Suppose nets i and j are adjacent on layer l of edge e, the coupling capacitance between them can be computed as $C = \alpha \frac{l(e)}{(c_l(e)/|N_l(e)|)^\beta}$, where $N_l(e)$ represents the set of nets on layer l of edge e. Therefore, crosstalk on each net can be computed. Using $X_i(S)$ to represent crosstalk on net i under an extended global routing solution S, the problem we need to solve can be defined as follows.

Problem 6 (Crosstalk-Constrained Global Routing (CCGR)) *Given a routing graph $G = (V, E)$, n nets, and the crosstalk constraints $C_1, \cdots, C_n$, find an extended global routing solution S such that $X_i(S) \leq C_i$ for all $1 \leq i \leq n$.*

4.4 Complexity results

In the CCGR problem, both routing topology and adjacency relations in each region can be changed. Therefore, the following methods can be use to reduce crosstalk: *shielding* (inserting a spare wire between coupling nets), *track assignment* (changing neighboring relations of nets), *layer assignment* (distributing coupling wires to different layers). Restricting the CCGR problem to each of these methods, we can formulate the following sub-problems.

Problem 7 (Shielding for Crosstalk Control) *Suppose the routing topology, layer/track assignments in all regions are fixed, and only one shielding wire can be inserted in each region. Find a shielding position in each region such that each net satisfies its crosstalk constraint.*

Problem 8 (Track Assignment for Crosstalk Minimization) *Suppose the routing topology is fixed, and so are track assignments in all regions except one. Find a track assignment in that region such that the total crosstalk on the nets is minimized.*

Problem 9 (Track Assignment for Crosstalk Control) *Suppose the routing topology is fixed, and so are track assignments in all regions except one. Find a track assignment in that region such that each net satisfies its constraint.*

Problem 10 (Layer Assignment for Crosstalk Control) *Suppose there are only two layers in each region, and the routing topology and net orderings in all regions are fixed. Find a layer assignment in each region such that each net satisfies its crosstalk constraint.*

As stated by the following theorem, all the above problems are intractable.

Theorem 6 *Problems 7, 8, 9, 10 are all NP-hard.*

Proof: We show NP-hardness of these four problems by reductions from two problems: PARTITION and HAMILTONIAN PATH. These problems are defined as follows [21].

PARTITION

INSTANCE: Finite set A and a size $s(a) \in Z^+$ for each $a \in A$.

QUESTION: Is there a subset $A' \in A$ such that $\sum_{a \in A'} s(a) = \sum_{a \in A - A'} s(a)$?

HAMILTONIAN PATH

INSTANCE: Graph $G = (V, E)$.

QUESTION: Does G contain a Hamiltonian path?

Specifically, we reduce PARTITION to the problem of shielding for crosstalk control and the problem of layer assignment for crosstalk control, and reduce HAMILTONIAN PATH to the problem of track assignment for crosstalk minimization and the problem of track assignment for crosstalk control.

Let the problem of shielding for crosstalk control has 3 nets $1, 2$, and 3, which are routed by the same topology and layer assignment and fixed in the sequence $1, 2, 3$ in each region. Also each region is one-to-one correspondent to an $a \in A$, that is, crosstalk between two adjacent wires in the region is $s(a)$. The crosstalk constraints are $C_1 = C_3 = \sum_{a \in A} s(s)/2$, and $C_2 = \infty$. It is easy to check that there is a solution to this problem of shielding for crosstalk control if and only if PARTITION has a yes answer.

Very similarly, let the problem of layer assignment for crosstalk control has 3 nets $1, 2, 3$, which are routed by the same topology and net ordering. The crosstalk between adjacent wires in a region is also $s(a)$ for each $a \in A$. The crosstalk constraints are the same as in the above reduction. It is also easy to check that there is a solution to this problem of layer assignment for crosstalk control if and only if PARTITION has a yes answer.

Given $G = (V, E)$ as an instance of HAMILTONIAN PATH, for each $v \in V$, we will have a net v in the problem of track assignment for crosstalk minimization. For any pair of nets u and v, its crosstalk coefficient $e_{uv} = e_{vu} = 0$ if $(u, v) \in E$; or $e_{uv} = e_{vu} = 1$ otherwise. Each of these nets is routed through different regions except in one region where all nets are routed together. It is easy to check that there is a track assignment with 0 total crosstalk if and only if G contains a Hamiltonian path.

We construct the problem of track assignment for crosstalk control exactly the same as in the above reduction. Furthermore, we set the crosstalk constraint for each net to be 0. In this case, it is also easy to check that there is a solution to the problem if and only if G contains a Hamiltonian path. $\square$

4.5 A global router with crosstalk control

As shown in the previous section, many sub-problems of the CCGR problem are NP-hard. We use a heuristic to deal with this problem. It is a sequential approach, that is, routing is done net by net. The approach consists of two stages. The first stage constructs a routing solution to minimizes the summation of crosstalk on all nets. Each time, a Steiner tree is computed for a net which minimizes the total crosstalk and its topology is then fixed for the remaining nets. In the second stage, crosstalk on each net in the initial solution is estimated. The nets having constraint violations are then ripped up. During re-routing, a method based on Lagrangian relaxation is used to balance crosstalk according to constraints. Because there is no forbidden region in the process, a new route can have the same topology as the old one but with different layer/track assignment. This means layer/track re-assignment is also included in this procedure.

4.5.1 Minimum crosstalk Steiner tree

Since we use a sequential approach in our routing, the kernel procedure is how to construct a Steiner tree and decide the layer/track assignment in each region such that the total crosstalk is minimize. Formally, this can be defined as the following problem.

Problem 11 (Minimum Crosstalk Steiner Tree) *Given a routing graph* $G = (V, E)$ *and an extended global routing solution* S *for nets* $1, \cdots, m-1$ *which minimized the total crosstalk, find an extended global routing solution* S' *for nets* $1, \cdots, m$ *such that routing topologies for nets* $1, \cdots, m-1$ *are kept and* $X(S') = \sum_{i=1}^{m} X_i(S')$ *is minimized.*

Let $X^e(S)$ denote the summation of crosstalk on all nets in edge e under solution S. The objective function in the above problem can be rewritten as follows.

$$\begin{aligned} X(S') &= \sum_{e \in E} X^e(S') \\ &= \sum_{e \in T_m} X^e(S') + \sum_{e \notin T_m} X^e(S') \\ &= \sum_{e \in T_m} (X^e(S') - X^e(S)) + \sum_{e \in E} X^e(S) \end{aligned} \tag{4.3}$$

where T_m represents the Steiner tree for net m. For each edge e, $X^e(S') - X^e(S)$ is the difference of the minimum total crosstalk in edge e before or after inserting net m in the edge. Given a set of nets going through an edge, the minimum crosstalk in the edge can be computed by an optimal layer/track assignment. Therefore, we can compute $X^e(S') - X^e(S)$ and use it as the cost of edge e. Since $\sum_{e \in E} X^e(S)$ in (4.3) is a constant, the minimum crosstalk Steiner tree problem becomes a minimum Steiner tree problem.

Unfortunately, according to Theorem 6, layer/track assignment for crosstalk minimization in one region is NP-hard. Therefore, a heuristic is designed to do the job. It works as follows. First, we decide an ordering on all nets. Then, each net is added sequentially to the region according to its order. When adding one net, the existing nets keep their relative ordering, and the new net is inserted to a position which has the minimum increase of crosstalk.

If we use the same net ordering for sequential routing in the above heuristic, the layer/track assignments can be merged into the routing process. In this case, when computing edge costs, there is no need to do layer/track re-assignments for the existing nets. Suppose S is the given solution for nets $1, \cdots, m-1$, the cost of routing net m on edge e is the minimum increase of crosstalk by inserting net m in one sequence of $S(e)$. If e is chosen by a minimum Steiner tree algorithm, such insertion will give $S'(e)$. An example of the procedure is given in Figure 4.3.

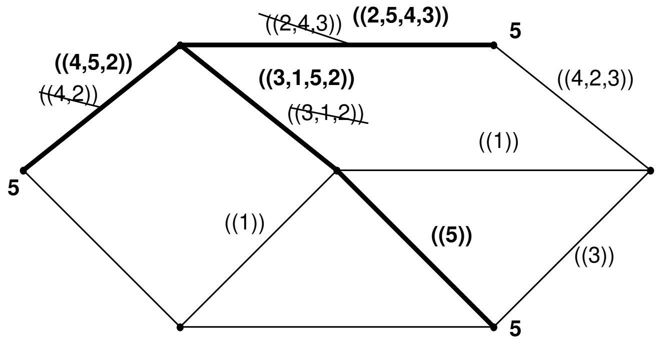

Figure 4.3: Routing net 5

In practice, crosstalk is not the only objective we want to minimize. We also need to consider wire length and congestion. In such case, we can use the following edge cost function.

$$cost(e) = \alpha \cdot length(e) + \beta \cdot overflow(e)^2 + \gamma \cdot xtalk(e)$$

where $overflow(e)$ is the extra space needed when compared with the capacity of the edge, and $xtalk(e)$ is the increase of crosstalk on the edge. Based on this cost function, any minimum Steiner tree heuristic on general weighted graph can be used to construct a Steiner tree which simultaneously optimizes the wire length, congestion, and crosstalk.

The minimum Steiner tree heuristic we use in our global router is based on the shortest path and the minimum spanning tree. According to [24], these heuristics are guaranteed to have solutions within $2(1-1/l)$ of the optimum, where l is the number of leaves in the optimal solution. We use Dijkstra's algorithm [14] for the shortest path and Prim's algorithm [14] for the minimum spanning tree. The reason we use Prim's algorithm instead of Kruskal's is that Dijkstra's algorithm and Prim's algorithm are both working in a tree expansion fashion, hence can be easily combined. The pseudo-code of the procedure is given in Figure 4.4.

```
Algorithm Minimum Steiner tree heuristic
  l(s) = 0 for one pin-node s, and l(v) = ∞ for all other nodes;
  insert s into a priority queue Q and mark all other pin-nodes;
  while (Q is not empty) {
    u = extract-min(Q);
    if (u is not marked)
      for (each neighbor v of u) {
        if (l(v) < l(u) + cost(u, v)) {
          l(v) = l(u) + cost(u, v);
          parent(v) = u;
          insert v in Q if it is not;
        }
      }
    else {
      unmark(u); l(u) = 0; insert u in Q;
      x = parent(u);
      while (x is not a pin-node) {
        l(x) = 0; insert x in Q;
        x = parent(x);
      }
    }
  }
```

Figure 4.4: Minimum Steiner tree heuristic

4.5.2 Rip-up and re-route based on Lagrangian relaxation

Given an initial solution constructed by the minimum crosstalk Steiner tree algorithm, nets with crosstalk violations are ripped up. We re-route them one net at a time in the order of decreasing violations. When routing a net, besides crosstalk on the current net, extra crosstalk can be induced on previously routed nets. Therefore, we need to solve the following problem.

Problem 12 (Re-Routing for Crosstalk Control) *Given a routing graph $G = (V, E)$ and an extended global routing solution S for nets $1, \cdots, m-1$ such that $X_i(S) < C_i$ for $1 \leq i \leq m-1$, find an extended global routing solution S' for nets $1, \cdots, m$ such that routing topologies of nets $1, \cdots, m-1$ are kept, the wire length is minimized, and $X_i(S') < C_i$ for all $1 \leq i \leq m$.*

Compared with the minimum crosstalk Steiner tree problem which minimizes the total crosstalk, the above problem requires to balance the crosstalk on each net according to its constraint, hence is more complicated. However, we can use Lagrangian relexation technique to satisfy all constraints simultaneously. By relaxing constraints into objective function, we have the following problem.

Problem 13 (Lagrangian Sub-Problem (LSP)) *Given a routing graph $G = (V, E)$, an extended global routing solution S for nets $1, \cdots, m-1$, and constants $\mu_1, \cdots, \mu_m$, find an extended global routing solution S' for nets $1, \cdots, m$ such that the routing topology for nets $1, \cdots, m-1$ is kept, and $length(S') + \sum_{i=1}^{m} \mu_i (X_i(S') - C_i)$ is minimized. Where $length(S')$ is the wire length in solution S'.*

Let $length_i(S')$ be the wire length of net i under solution S'. The objective function of above problem can be rewritten as follows.

$$
\begin{aligned}
& length(S') + \sum_{i=1}^{m} \mu_i (X_i(S') - C_i) \\
= \quad & \sum_{i=1}^{m} (length_i(S') + \mu_i X_i(S')) - \sum_{i=1}^{m} \mu_i C_i
\end{aligned}
$$

Since $\sum_{i=1}^{m} \mu_i C_i$ is a constant, by treating $\mathit{length}_i(S')+\mu_i X_i(S')$ as "crosstalk" on net i, the Lagrangian sub-problem becomes the minimum crosstalk Steiner tree problem, hence can be solved by the same algorithm.

Let $L(\mu_1, \cdots, \mu_m)$ denote the optimal solution of the Lagrangian sub-problem. For any $\mu_i \geq 0, 1 \leq i \leq m$, we have

$$\begin{aligned} L(\mu_1, \cdots, \mu_m) &= \min_{S'}\{\mathit{length}(S') + \sum_{i=1}^{m} \mu_i (X_i(S') - C_i)\} \\ &\leq \min_{S'}\{\mathit{length}(S') + \sum_{i=1}^{m} \mu_i (X_i(S') - C_i) : X_i(S') \leq C_i, 1 \leq i \leq m\} \\ &\leq \min_{S'}\{\mathit{length}(S') : X_i(S') \leq C_i, 1 \leq i \leq m\}. \end{aligned}$$

This means, for any fixed $\mu_i \geq 0, 1 \leq i \leq m$, the solution of the Lagrangian sub-problem $L(\mu_1, \cdots, \mu_m)$ is a lower bound of the re-routing for crosstalk control problem. Therefore, if we find $\mu_1^*, \cdots, \mu_m^*$ such that

$$L(\mu_1^*, \cdots, \mu_m^*) = \min_{S'}\{\mathit{length}(S') : X_i(S') \leq C_i, 1 \leq i \leq m\},$$

we know that S' is an optimal solution. Therefore, we need to solve the following problem.

Problem 14 (Lagrangian Multiplier Problem (LMP))

$$\begin{array}{ll} \mathit{Maximize} & L(\mu_1, \cdots, \mu_m) \\ \mathit{subject\ to} & \mu_i \geq 0, \forall 1 \leq i \leq m \end{array}$$

As stated by the following theorem, the LMP problem has a very good property for optimization.

Theorem 7 *The LMP is a convex programming problem.*

Proof: Given a multiplier $(\mu_1^0, \cdots, \mu_m^0)$, for every positive ϵ and any multiplier $(\mu_1^1, \cdots, \mu_m^1)$ such that

$$|(\mu_1^1, \cdots, \mu_m^1) - (\mu_1^0, \cdots, \mu_m^0)| < \frac{\epsilon}{\max_{S'} \sum_{i=1}^{m} |X_i(S') - C_i|},$$

we have

$$
\begin{aligned}
&|L(\mu_1^1, \cdots, \mu_m^1) - L(\mu_1^0, \cdots, \mu_m^0)| \\
&= \ |\min_{S'}\{\mathit{length}(S') + \sum_{i=1}^{m} \mu_i^1 (X_i(S') - C_i)\} - \min_{S'}\{\mathit{length}(S') + \sum_{i=1}^{m} \mu_i^0 (X_i(S') - C_i)\}| \\
&\leq \ \max\{|(\mathit{length}(S_1') + \sum_{i=1}^{m} \mu_i^1 (X_i(S_1') - C_i)) - (\mathit{length}(S_1') + \sum_{i=1}^{m} \mu_i^0 (X_i(S_1') - C_i))|, \\
&\qquad |(\mathit{length}(S_0') + \sum_{i=1}^{m} \mu_i^1 (X_i(S_0') - C_i)) - (\mathit{length}(S_0') + \sum_{i=1}^{m} \mu_i^0 (X_i(S_0') - C_i))|\} \\
&\leq \ \max\{\sum_{i=1}^{m} |\mu_i^1 - \mu_i^0||X_i(S_1') - C_i|, \sum_{i=1}^{m} |\mu_i^1 - \mu_i^0||X_i(S_0') - C_i|\} \\
&\leq \ |(\mu_1^1, \cdots, \mu_m^1) - (\mu_1^0, \cdots, \mu_m^0)| \max_{S'} \sum_{i=1}^{m} |X_i(S') - C_i| \\
&< \ \epsilon
\end{aligned}
$$

where S_0' and S_1' are solutions giving $L(\mu_1^0, \cdots, \mu_m^0)$ and $L(\mu_1^1, \cdots, \mu_m^1)$, respectively. This shows that $L(\mu_1, \cdots, \mu_m)$ is continuous.

The following inequality shows that $L(\mu_1, \cdots, \mu_m)$ is a concave function. For any $0 \leq \lambda \leq 1$, and $(\mu_1^0, \cdots, \mu_m^0), (\mu_1^1, \cdots, \mu_m^1)$,

$$
\begin{aligned}
&\lambda L(\mu_1^0, \cdots, \mu_n^0) + (1-\lambda) L(\mu_1^1, \cdots, \mu_n^1) \\
&= \ \lambda \min_{S'}\{\mathit{length}(S') + \sum_{i=1}^{m} \mu_i^0 (X_i(S') - C_i)\} \\
&\qquad + (1-\lambda) \min_{S'}\{\mathit{length}(S') + \sum_{i=1}^{m} \mu_i^1 (X_i(S') - C_i)\} \\
&\leq \ \lambda(\mathit{length}(S^*) + \sum_{i=1}^{m} \mu_i^0 (X_i(S^*) - C_i)) + (1-\lambda)(\mathit{length}(S^*) + \sum_{i=1}^{m} \mu_i^1 (X_i(S^*) - C_i)) \\
&= \ \mathit{length}(S^*) + \sum_{i=1}^{m} (\lambda \mu_i^0 + (1-\lambda)\mu_i^1)(X_i(S^*) - C_i) \\
&= \ L(\lambda \mu_1^0 + (1-\lambda)\mu_1^1, \cdots, \lambda \mu_m^0 + (1-\lambda)\mu_m^1)
\end{aligned}
$$

where S^* is a solution giving $L(\lambda \mu_1^0 + (1-\lambda)\mu_1^1, \cdots, \lambda \mu_m^0 + (1-\lambda)\mu_m^1)$. □

Based on the theorem, non-linear programming techniques can be used to solve the LMP problem. Since $L(\mu_1, \cdots, \mu_m)$ as a function is not everywhere differentiable, we will use the sub-gradient method to solve it. The pseudo-code of the sub-gradient method is given in Figure 4.5.

```
Algorithm  Sub-Gradient Method for LMP
    k = 0;
    for i = 1 to m
        μ_i = 0;
    while ( stop criteria not satisfied) {
        Solve the LSP by the Steiner tree heuristic;
        for i = 1 to m
            μ_i = max{0, μ_i + θ_k(X_i(S') − C_i)};
        k = k + 1;
    }
```

Figure 4.5: Pseudo-code of sub-gradient method

The general theory for sub-gradient method states that if $\theta_k \rightarrow 0$ and $\sum_{j=1}^{k} \theta_j \rightarrow \infty$ as $k \rightarrow \infty$, it will converge to an optimal solution [1]. Intuitively, the procedure can be explained as follows. First we give a relative weight to each net as the importance of its crosstalk control. Then we minimize a weighted sum of crosstalk on all nets. After that, we estimate the crosstalk on each net. For those nets with violations, we increase their weights according to the violations; for those nets with slacks, we decrease their weights according to the slacks.

The only problem with this approach is that there may exist a "duality gap" between the problem of re-routing for control crosstalk and its corresponding Lagrangian multiplier problem. That is, it is possible that

$$\min\{\mathit{length}(S') : X(i, S') \leq C_i\} > \max L(\mu_1, \cdots, \mu_m).$$

Furthermore, solving the problem of re-routing for crosstalk control is just one step in the whole process of re-routing. Hence, it is not necessary to spend too much time to get an optimal solution. So, we set up an upper bound on the number of iterations. During the iterations, we also record the best routing, that is, the ever seen solution S' with shortest wire length such that $X(i, S') \leq C_i$ or a solution S' such that $\max_i(X(i, S') - C_i)$ is minimum. If the sub-gradient method did not give the optimal solution within those iterations, we will use the best routing as our solution.

4.6 Experimental results

We implement the global router described in Section 4.5 on a Solaris/PC platform. Five circuits from the CBL/NCSU building block benchmarks: ami33, ami49, apte, hp, and xerox are used. Since these benchmarks do not come with placement information, we use a floorplanner based on simulated annealing [62] to give placements. Using different parameters, we obtain two different placements for both ami33 and ami49: ami33.1, ami33.2, ami49.1, ami49.2, and three different placements for xerox: xerox.1, xerox.2, xerox.3. Using different minimum track spacing and crosstalk constraints, different test problems are produced for the same placement, which are indicated by the suffixes: a, b, etc.

In order to measure the effectiveness of our global router, we compare it with other routing approaches. "Traditional GR" only considers wire length and overflow, and nets are ordered sequentially in each region. "Post-GR layer/track assignment" uses the same topologies as in traditional GR but use the greedy heuristic given in Section 4.5.1 to do layer/track assignment in each region. For each approach, the numbers of nets having crosstalk violations and the maximum violations are reported in Table 4.1. For our global router, both results after the first stage (i.e., minimizing total crosstalk) and the second stage (i.e., rip-up & re-route by Lagrangian relaxation) are reported. As we can see, after rip-up and re-route, almost all violations are solved. For those few nets which still have violations, the violations are so small that it could be easily solved in detailed routing.

Table 4.1: Experimental results

test problems	Traditional GR		post-GR layer/track assignment		our approach			
					first stage		second stage	
	#vio.	max	#vio.	max	#vio.	max	#vio.	max
ami49.1.a	329	285.29	301	191.91	121	81.46	1	0.84
ami49.1.b	252	86.27	143	35.46	46	15.23	1	0.14
ami49.1.c	303	114.50	265	46.78	146	25.93	3	0.90
ami49.2	325	100.46	283	46.16	156	16.92	2	0.19
ami33.1	70	82.15	65	31.90	35	8.97	5	0.39
ami33.2	64	83.07	55	31.87	24	8.89	1	0.19
apte.a	19	21.93	11	7.56	3	4.54	0	0.00
apte.b	17	5.15	11	2.10	5	1.74	0	0.00
xerox.1	49	5.16	9	4.34	3	3.83	2	0.08
xerox.2	57	17.17	18	5.87	2	0.32	0	0.00
xerox.3	70	49.70	34	31.23	22	15.98	0	0.00

4.7 Conclusions

To enlarge the flexibility to control crosstalk, we considered global routing with crosstalk constraints in this chapter. Since crosstalk is heavily dependent on neighboring relations among wires, traditional global routing is extended to include simple layer and track assignments. Based on this extension, a two-stage global router is designed to give feasible routing solutions with respect to crosstalk constraints. The first stage is used to minimize the total crosstalk and is formulated as a new Steiner tree problem. The second stage is used to control crosstalk on each net according to its constraint and is solved by Lagrangian relaxation. Experimental results show that our router is effective in crosstalk control.

Chapter 5

Low Power AND/OR Gate Decomposition

5.1 Introduction

With the remarkable growth of portable application and the increasing frequency and integration density, power is being given comparable weight to speed and area in IC designs. Power dissipation in digital CMOS circuits is dominated by the *dynamic dissipation*, which is mainly the charging and discharging of the load capacitances [46]. In a circuit consisting of n gates, it can be modeled as

$$P = 0.5V_{dd}^2 f_{clk} \sum_{i=1}^{n} C_L^i E_{sw}^i$$

where V_{dd} is the supply voltage, f_{clk} is the clock frequency, C_L^i is the load capacitance at the output of gate i, and E_{sw}^i (referred to as the *switching activity*) is the expected number of gate i transitions per clock cycle. As we can see, V_{dd} and f_{clk} are fixed by the technology, but C_L^i and E_{sw}^i can be controlled in design process.

In technology mapping, the subject netlist is usually first decomposed into a netlist composed of only inverters and two-input NAND (or NOR) gates. How the decomposition is done can have a significant impact on the power dissipation of the final implementation [38, 56, 59, 60]. We deal with the *low-power* AND/OR *gate decomposition* (or simply low-power gate decomposition) problem in this chapter.

The problem appears in a few recent papers. Tiwari *et al.* [56] mentioned the importance of a good decomposition on the final result of technology mapping, but did not give any solution. At the same time, Tsui *et al.* [59, 60] analyzed the problem and found that Huffman's algorithm [23] can only be used in domino dynamic logic. For static logic which is more important in low-power applications, only a greedy heuristic called the *modified Huffman algorithm* is given. Murgai *et al.* [38] also considered the decomposition problem, but their minimization objective was the power consumptions due to glitches.

Since the problem for dynamic logics can be easily solved, we only consider static logics. In our approach, we first study the structure of an optimal decomposition tree. This is given by a set of properties an optimal tree must have. Then, based on these properties, we designed an exact algorithm for the construction of an optimal decomposition tree. The time complexity of the algorithm is $O(n2^n)$, which, though still exponential, should be regarded as efficient considering the total more than $(2n-1)^{n-1}$ trees in the solution space.

As a by-product, a heuristic algorithm can be easily derived from the exact algorithm. Its running time is $O(n \log n)$, which is much faster than the $O(n^2 \log n)$ running time of the modified Huffman algorithm [59, 60]. Since the heuristic is strongly based on the optimality properties, it also performs much better than the modified Huffman algorithm. In fact our experimental results show that our heuristic gives optimal results in most cases.

The rest of the chapter is organized as follows. In Section 5.2, we define the low-power gate decomposition problem. In Section 5.3, we describe Huffman's algorithm for tree construction and identify two special cases of our problem which can be solved. Section 5.4 studies the properties of an optimal decomposition tree. Based on these properties, Section 5.5 presents two algorithms: one exact algorithm and one heuristic. Section 5.6 gives the experimental results. Finally, Conclusions and extensions are provided in Section 5.7.

5.2 Problem formulation

In AND/OR gate decomposition, we need to decompose a multi-input AND (or OR) gate into a tree of two-input gates. Since AND and OR are dual operations, a decomposition algorithm for one gives an algorithm for the other. Thus, we need only to consider how to decompose an AND gate into a tree of two-input AND gates. This is simply called *gate decomposition.*

We treat the signals in a circuit as random variables and define the *signal probability* of a signal x as the probability of x being 1, denoted by $p(x)$. As in [56, 59, 60], We assume the *zero delay model* where gate delays are assumed to be zero and thus signal transitions due to glitching are ignored. We also assume the inputs of the AND gate are uncorrelated (*spatial independent*) and the present input signal value is independent of those in the past (*temporal independent*). Under these assumptions, given input signal probabilities and a decomposition tree, the probabilities of internal signals can be computed as follows. Start from the primary inputs, for each $z = x$ AND y, let $p(z) = p(x)p(y)$. Thus, the signal probability of any node v is equal to the product of all leaf probabilities in the subtree rooted at v. For example, Figure 5.1 shows one gate decomposition and all signal probabilities of the nodes.

The switching activity E_{sw} depends on the implementation logic style. In p-domino logic designs, the gate outputs are pre-discharged to 0, thus the switching activity of a node is equal to the probability of being 1. Let $T = (V, E)$ represent the decomposition tree, and $p(v)$, for any $v \in V$, denote the output signal probability of node v. The objective function we want to minimize in domino logic is $\sum_{v \in V} p(v)$.

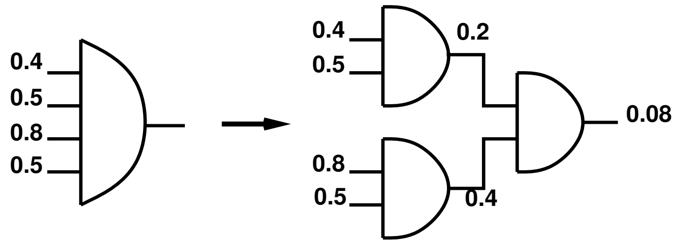

Figure 5.1: Gate decomposition

Because of this simple objective function, it can be shown that Huffman's algorithm can be used to give an optimal decomposition tree in domino logic designs [59, 60].

Because of the pre-discharges or pre-charges, domino logic designs dissipate more power than static logic designs, which never do extra charges or discharges. In static logic, under the temporal independence assumption, the switching activity E_{sw} of signal x can be written as

$$\begin{aligned} E_{sw}(x) &= Pr[x : 0 \to 1] + Pr[x : 1 \to 0] \\ &= Pr[x = 0]Pr[x = 1] + Pr[x = 1]Pr[x = 0] \\ &= 2Pr[x = 1]Pr[x = 0] \\ &= 2p(x)(1 - p(x)) \end{aligned}$$

The problem we will solve in this chapter can be defined as follows.

Low-power gate decomposition problem: Given an n-input AND gate with inputs $s_1, s_2, \ldots, s_n$ and their signal probabilities $p(s_1), p(s_2), \ldots, p(s_n)$, construct a tree $T = (V, E)$ of 2-input AND gates with $s_1, s_2, \ldots, s_n$ as its leaves such that

$$E_{sw}(T) = \sum_{v \in V} p(v)(1 - p(v))$$

is minimized.

According to Knuth [28], the number of different *labeled oriented* binary trees with n leaves is $\binom{2n-1}{n-1}(2n-2)!/2^{n-1}$. In a decomposition tree, only leaves are labeled, the internal nodes are indistinguishable. Therefore, the number of different decomposition trees is

$$\frac{\binom{2n-1}{n-1}(2n-2)!}{2^{n-1}(n-1)!} > (2n-1)^{n-1}.$$

Thus, an exhaustive enumeration method is prohibitively expensive. It was found by Tsui *et al.* [59, 60] that Huffman's algorithm can not solve this problem. Instead, they gave a heuristic which was called modified Huffman algorithm. It starts with a forest composed of all the inputs, and incrementally combines two trees into one until there is only one tree. It is a greedy algorithm, and each time tries all pairs and chooses the combination which gives the minimum increase on the objective function. The time complexity of the algorithm is $O(n^2 \log n)$ [59, 60].

This algorithm is by far not optimal. This can be shown by a simple example. Here we have six input signals with the following probabilities: 0.4, 0.4, 0.4, 0.94, 0.94, 0.95. The decomposition tree constructed by the modified Huffman algorithm is shown in Figure 5.2(a), where the summation of switching activities is 1.3337. Nevertheless, a decomposition tree shown in Figure 5.2(b) has 1.22748 as its total switching activities.

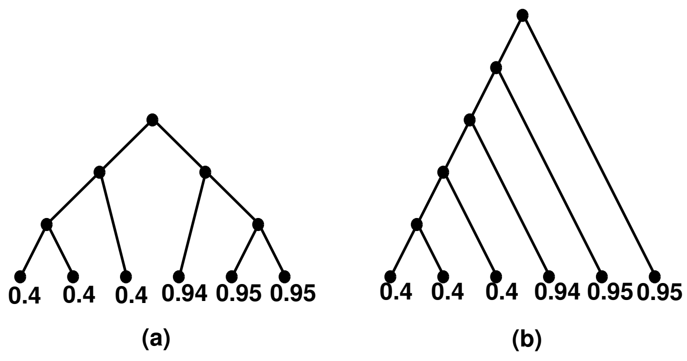

Figure 5.2: (a) Decomposition tree by modified Huffman has switching activities 1.3337; (b) A decomposition tree with switching activities 1.22748

5.3 Huffman's algorithm

Given n leaves $v_1, v_2, \ldots, v_n$ with weights $w(v_1), w(v_2), \ldots, w(v_n)$, Huffman [23] gave an algorithm to construct a binary tree with minimum weighted path length $\sum_{i=1}^{n} w(v_i)l_i$, where l_i is the path length from the root to v_i. The algorithm can be described as follows. Starting from a forest composed of all the leaves, it combines two trees with the minimum weights, use the summation of the weights as the weight of the combined tree and substitute the two trees by the combined one; this process is continued until there is only one tree.

If, for each internal node r with two children u and v, we define the weight $w(r) = w(u) + w(v)$, then

$$\sum_{i=1}^{n} w(v_i)l_i = \sum_{u \in V} w(u),$$

where V represents the set of internal nodes. This formulation leads us to consider Huffman's algorithm for the low-power gate decomposition problem. Unfortunately, it can not solve the problem in general case [59, 60]. However, we find that under some conditions Huffman's algorithm can give optimal solutions. Before we give these conditions, we will describe two variations of Huffman's algorithm, which are a little different with the original one.

> *Min-Huffman algorithm*: Start with all the input signals; combine the two signals of minimum probabilities and substitute the two signals with the new signal; continue the process until there remains only one signal.

> *Max-Huffman algorithm*: Start with all the input signals; combine the two signals of maximum probabilities and substitute the two signals with the new signal; continue the process until there remains only one signal.

We first state a lemma which is useful in the proofs.

Lemma 4 *Given two sets of signals $u_1, u_2, \ldots, u_n$ and $v_1, v_2, \ldots, v_n$ such that $0.5 \leq p(u_i) \leq p(v_i)$ for all $1 \leq i \leq n$, and let $0 < \alpha < 1$. If we change the probability of u_i from $p(u_i)$ to $\alpha p(u_i)$, and the probability of v_i from $p(v_i)$ to $p(v_i)/\alpha$, for all $1 \leq i \leq n$, then the summation of their switching activities will decrease.*

Proof: Consider the difference between the summation of changed switching activities and that of the original ones

$$\sum_{i=1}^{n}(\alpha p(u_i)(1-\alpha p(u_i)) + \frac{1}{\alpha}p(v_i)(1-\frac{1}{\alpha}p(v_i))) - \sum_{i=1}^{n}(p(u_i)(1-p(u_i)) + p(v_i)(1-p(v_i)))$$

$$\begin{aligned}
&= \sum_{i=1}^{n}((\alpha p(u_i) - \alpha^2 p(u_i)^2) + (\frac{1}{\alpha}p(v_i) - \frac{1}{\alpha^2}p(v_i)^2) - (p(u_i) - p(u_i)^2) - (p(v_i) - p(v_i)^2)) \\
&= \sum_{i=1}^{n}(p(u_i)(\alpha-1) - p(u_i)^2(\alpha^2-1) + p(v_i)(\frac{1}{\alpha}-1) - p(v_i)^2(\frac{1}{\alpha^2}-1)) \\
&= \sum_{i=1}^{n}((\alpha-1)(p(u_i) - \frac{1}{\alpha}p(v_i)) - (\alpha^2-1)(p(u_i)^2 - \frac{1}{\alpha^2}p(v_i)^2)) \\
&= \sum_{i=1}^{n}(\alpha-1)(p(u_i) - \frac{1}{\alpha}p(v_i))(1-(\alpha+1)(p(u_i) + \frac{1}{\alpha}p(v_i)))
\end{aligned}$$

Since $\alpha < 1$, $p(u_i) - p(v_i)/\alpha < p(u_i) - p(v_i) \leq 0$, and $p(u_i) + p(v_i)/\alpha > p(u_i) + p(v_i) \geq 0.5 + 0.5 = 1$, we know $(\alpha-1)(p(u_i) - p(v_i)/\alpha)(1-(\alpha+1)(p(u_i) + p(v_i)/\alpha))) < 0$. Therefore the above difference is negative, which proves our lemma. $\square$

As stated by the following theorem, two special cases can be solved efficiently.

Theorem 8 *If all input signal probabilities are not greater than 0.5, the low-power gate decomposition problem can be solved by the Min-Huffman algorithm; If the product of all input signal probabilities is not less than 0.5, it can be solved by the Max-Huffman algorithm.*

Proof: We will prove the theorem by showing that each combination in the algorithms actually appears in the optimal tree. Before doing that, we must notice that switching activity is monotonically increasing with signal probability when the signal probability is not greater than 0.5; and is monotonically decreasing with signal probability when it is greater than 0.5. In fact, their relation can be shown in Figure 5.3.

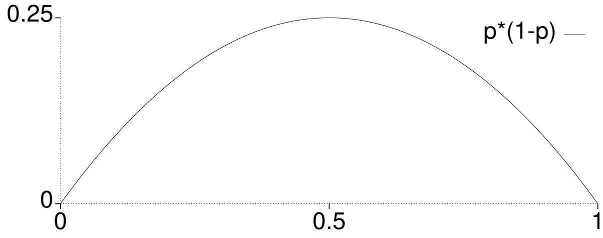

Figure 5.3: Switching activity as a function of signal probability

We claim that, given a set of signals whose probabilities are all not greater than 0.5, the two with minimum probabilities must be combined in an optimal tree. Suppose this is not true, that is, we have three signals u, v and w such that $p(u) < p(w), p(v) < p(w)$ but u, w are combined first. Without loss of generality, their relation can be illustrated in Figure 5.4, where, t denotes their least common parent, l and r are t's two children, and p is u and w's parent. Because r is an ancestor of v (or it can be v itself), we have $p(r) \leq p(v) < p(w)$. Now consider exchanging w with the subtree T_1 rooted at r. Only the probabilities of the nodes on the path from p to l are changed, and actually they are decreased. Since all of them are not greater than 0.5, the total switching activities are decreased, which is a contradiction to the optimality of the tree.

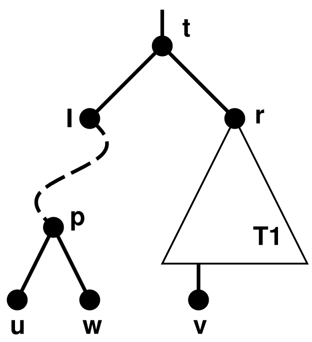

Figure 5.4: Two nodes with minimum probabilities must be combined

We also claim that, given a set of signals whose product of probabilities is greater than 0.5, the two with maximum probabilities must be combined in an optimal tree. Suppose this is not true, then we must have three signals u, v and w such that $p(u) > p(w), p(v) > p(w)$ but u and w are combined first. Similarly, let t denote their least common parent. Here we have two cases.

Case 1 is shown in Figure 5.5(a), where v is combined with an ancestor of u and w. In this case, we can exchange v with w. Because $p(v) > p(w)$, only the probabilities on the path from p to l increased, others did not change. Since none of the signal probabilities is less than 0.5, this will reduce the total switching activities, which is a contradiction.

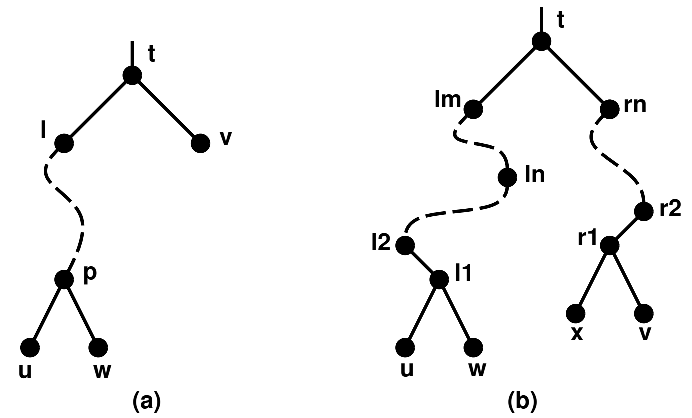

Figure 5.5: Two nodes with maximum probabilities must be combined: (a) Case 1 (b) Case 2

Case 2 is shown in Figure 5.5(b), in which v is combined with another signal x such that $p(x) < p(u)$ and $p(x) < p(v)$. Label the nodes on the path from u to t by $l_1, l_2, \ldots, l_m$, and the nodes on the path from v to t by $r_1, r_2, \ldots, r_n$. Since u, w and v, x are symmetric, without loss of generality, we can assume that $m \geq n$. We can also assume $p(r_n) \leq p(l_n)$, this is because that, if $p(r_n) > p(l_n)$, we can simply exchange the two subtrees rooted at l_n and r_n. This can only increase the probabilities of $l_{n+1}, \ldots, l_m$ (nothing if $m = n$), and does not increase the whole switching activities. Furthermore, we can assume $p(r_i) \leq p(l_i)$, for $1 \leq i \leq n$. The reason is that, if $p(r_i) \leq p(l_i)$ for all $j+1 \leq i \leq n$ but $p(r_j) > p(l_j)$, then r_j and l_j will have siblings r'_j and l'_j, respectively, such that $p(r'_j) < p(l'_j)$. Hence we can relabel r_j, r'_j by x, v and l_j, l'_j by u, w.

Now consider exchanging v with w. Let $\alpha = p(w)/p(v) < 1$, it only increases the signal probability of l_i from $p(l_i)$ to $p(l_i)/\alpha$ for $1 \leq i \leq m$, and decreases the signal probability of r_j from $p(r_j)$ to $\alpha p(r_j)$ for $1 \leq j \leq n$. By Lemma 4, the switching activities of l_i, r_i for $1 \leq i \leq n$ is decreased. Furthermore, since their probabilities are greater than 0.5 and are increased, the switching activity of l_j, for $n+1 \leq j \leq m$, are also decreased. This is a contradiction to the optimality of the tree. $\square$

5.4 Optimality properties

In previous section, we identified two special cases of the low-power gate decomposition problem which can be solved efficiently. In order to solve the general case, in this section, we will study the properties of an optimal decomposition tree.

First, we have the following simple observations.

Lemma 5 *On any path from a leaf to the root in a decomposition tree, the signal probabilities are decreasing. Each subtree in an optimal decomposition tree is also optimal.*

Proof: The first part is trivial because every signal probability is smaller than 1 and the probability of an internal node is the product of those of its children.

If there is a subtree S in an optimal tree which is not optimal, then S can be locally reconstructed to get another S' which has smaller switching activities. Since the root probabilities of the two subtrees are equal, it will not change switching activities in other parts. This is a contradiction. $\square$

Further analysis gives us the following result.

Lemma 6 *In an optimal decomposition tree, all inputs whose probabilities are not greater than 0.5 must form a separate subtree.*

Proof: Suppose the conclusion is not true. Then there must be a node v having the following property: its left subtree has an input x, its right subtree has a y, such that $p(x), p(y) \leq 0.5$, and at least one of them, say the left one, has an input z with $p(z) > 0.5$. This is shown in Figure 5.6. Let l and r denote the two children of v, from Lemma 5, we have $p(l) < p(x) \leq 0.5$ and $p(r) \leq p(y) \leq 0.5$. Now consider the path from z to l in the left subtree. Since $p(z) > 0.5$ and $p(l) < 0.5$, there exist two succeed nodes u, w on the path such that $p(u) > 0.5 \geq p(w)$. Now exchange the subtree T_1 rooted at r with the subtree T_2 rooted at u. Only those nodes on the path from w to l have their probabilities decreased. According to Lemma 5, these probabilities are originally not greater than $p(w) \leq 0.5$. Therefore, the switching activities are decreased, which is a contradiction. □

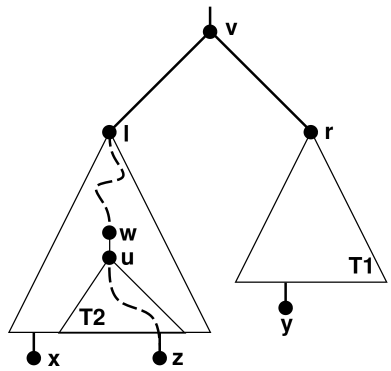

Figure 5.6: Inputs whose probabilities ≤ 0.5 must form a subtree

Lemma 6 tells us, in order to construct an optimal decomposition tree, we can always combine the signals whose probabilities are not greater than 0.5 into a subtree. By Lemma 5, this subtree needs to be an optimal one. According to Theorem 8, it can be constructed by the Min-Huffman algorithm. In fact, since the product of two smallest probabilities is still the smallest, in the Min-Huffman algorithm, signals are combined sequentially from low probability to high probability.

Similar analysis gives the following lemma.

Lemma 7 *In an optimal decomposition tree, the internal nodes whose probabilities are not greater than 0.5 form a path.*

Proof: Suppose those nodes form a tree other than a path. Then we can find a node v with two children l, r such that l, r are both internal nodes and $p(l), p(r) \leq 0.5$. Since the signals whose probabilities are not greater than 0.5 must be combined sequentially in an optimal tree , one of the two subtrees rooted at l and r, say the left one, must have an input z such that $p(z) > 0.5$. This give the same situation as shown in Figure 5.6. Using the same exchanging technique in the proof of Lemma 6, we can get a contradiction. $\square$

In order to present the next optimality property, we need to define two labels for each node in an optimal decomposition tree. For each v, let $level(v)$ be the distance of v from the root. That is, the root has level 0, its children have level 1, etc. For each v, if v is an internal node and $p(v) \leq 0.5$, then let $rank(v) = 0$. Otherwise, let $rank(v)$ be the minimum distance of v from any node in rank 0. The property can be stated as follows.

Theorem 9 *Let u and v be any two nodes in an optimal decomposition tree. If $rank(u) = rank(v) \neq 0$ and $level(u) < level(v)$, then $p(u) \geq p(v)$.*

Proof: We prove by induction on the rank.

First, we prove the lemma is true for the nodes of rank 1. If it is not the case, then we must have two nodes u and v such that $rank(u) = rank(v) = 1$ and $level(u) = level(v) + 1$ but $p(u) > p(v)$. This is illustrated by Figure 5.7, where $rank(x) = rank(y) = 0$ (which means $p(x), p(y) \leq 0.5$). Now consider exchanging the subtrees rooted at u and v. Since the leaves of the subtree rooted at y are kept the same, $p(y)$ does not change. Only $p(x)$ is changed and it is decreased. Since $p(x) \leq 0.5$, the switching activity of x is also decreased. This means the new tree has smaller switching activities than the old one, which is a contradiction.

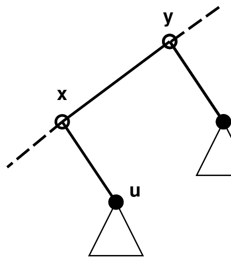

Figure 5.7: Nodes of rank 1 have non-increasing probabilities with respect to their level

Suppose the lemma is true for all ranks up to k. Now we will show that the probabilities of rank $k+1$ are also non-increasing with respect to levels. If it is not the case, then there must exist two nodes u_1 and v_1 such that $rank(u_1) = rank(v_1) = k+1$ and $level(u_1) < level(v_1)$ but $p(u_1) > p(v_1)$. This is illustrated by Figure 5.8, where $x_1, x_2, \ldots, x_m$ are nodes of rank 0. Now consider exchanging the two subtrees rooted at u_1 and v_1. With the same argument as in the base case, the probability of $p(x_m)$ does not change. The nodes whose probabilities are changed come from three paths: $x_1, x_2, \ldots, x_{m-1}$, $u_2, u_3, \ldots, u_k$, and $v_2, v_3, \ldots, v_k$. Because $p(u_1) > p(v_1)$, the probabilities of x_i and u_j, for $1 \leq i \leq m-1$ and $2 \leq j \leq k$, are decreased, but the probabilities of v_j, for $2 \leq j \leq k$, are increased. Since $p(x_i) \leq 0.5$ for $1 \leq i \leq m-1$, the switching activities of them are decreased. Based on the induction hypothesis, we have $p(u_i) \leq p(v_i)$ for $2 \leq i \leq k$. Let $\alpha = p(v_1)/p(u_1) < 1$, we know that u_i changes probability from $p(u_i)$ to $\alpha p(u_i)$ and v_i changes probability from $p(v_i)$ to $p(v_i)/\alpha$ for $2 \leq i \leq k$. According to Lemma 4, the summation of their switching activities is also decreased. Because we assume the original tree is optimal, this is a contradiction.

$\square$

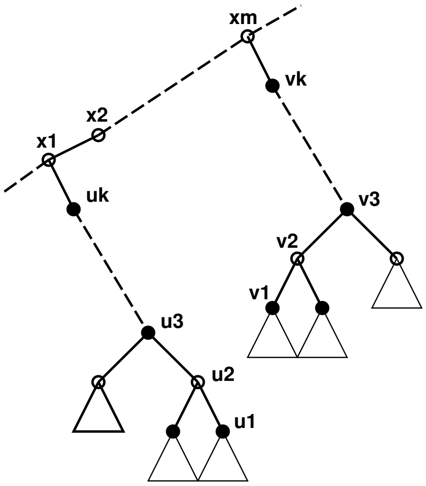

Figure 5.8: Nodes of rank k have non-increasing probabilities with respect to their levels

This theorem states that, in an optimal decomposition tree, for the nodes in the same rank other than 0, the probabilities are non-increasing with respect to their levels. According to the definition, the probability of each internal node in rank 1 is greater than 0.5. By Theorem 8, each subtree rooted at rank 1 node can be constructed by the Max-Huffman algorithm. Therefore, it is possible to arrange each subtree in such a way that, in each rank, the probabilities is non-decreasing from left to right. Under these arrangements, an optimal decomposition tree can be visualized in Figure 5.9, where the nodes in rank 0 form a path, and the probabilities in other ranks are non-decreasing along the arrows.

In order to prove our next optimality property, we will first prove the following lemma.

Lemma 8 *Given a set of signals $u_1, u_2, \ldots, u_k$ and two constants α_1 and α_2 such that $0 < \alpha_1 < 1 < \alpha_2$, either*

$$\sum_{i=1}^{k} \alpha_1 p(u_i)(1 - \alpha_1 p(u_i)) < \sum_{i=1}^{k} p(u_i)(1 - p(u_i))$$

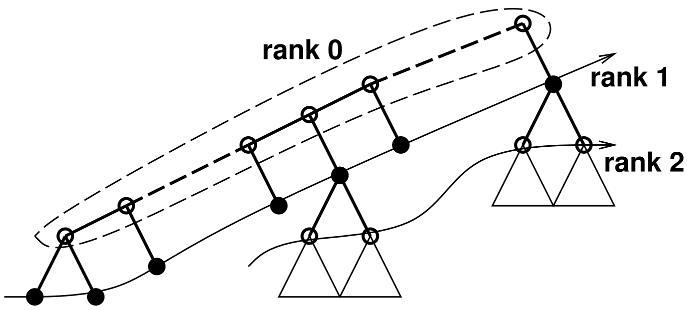

Figure 5.9: Probabilities are non-decreasing along the arrows in an optimal tree

or

$$\sum_{i=1}^{k} \alpha_2 p(u_i)(1-\alpha_2 p(u_i)) < \sum_{i=1}^{k} p(u_i)(1-p(u_i)).$$

Proof: We have

$$\begin{aligned}
&\sum_{i=1}^{k} \alpha p(u_i)(1-\alpha p(u_i)) - \sum_{i=1}^{k} p(u_i)(1-p(u_i)) \\
&= \sum_{i=1}^{k}(\alpha p(u_i) - \alpha^2 p(u_i)^2) - \sum_{i=1}^{k}(p(u_i) - p(u_i)^2) \\
&= (1-\alpha^2)\sum_{i=1}^{k} p(u_i)^2 - (1-\alpha)\sum_{i=1}^{k} p(u_i) \\
&= (1-\alpha)((1+\alpha)\sum_{i=1}^{k} p(u_i)^2 - \sum_{i=1}^{k} p(u_i))
\end{aligned}$$

If $(1-\alpha_1)((1+\alpha_1)\sum_{i=1}^{k} p(u_i)^2 - \sum_{i=1}^{k} p(u_i)) < 0$, we are done. Otherwise, since $1-\alpha_1 > 0$, we must have $(1+\alpha_1)\sum_{i=1}^{k} p(u_i)^2 - \sum_{i=1}^{k} p(u_i) \geq 0$. Therefore,

$$\begin{aligned}
&(1+\alpha_2)\sum_{i=1}^{k} p(u_i)^2 - \sum_{i=1}^{k} p(u_i) \\
&> (1+\alpha_1)\sum_{i=1}^{k} p(u_i)^2 - \sum_{i=1}^{k} p(u_i) \\
&\geq 0
\end{aligned}$$

Since $1 - \alpha_2 < 0$, this makes

$$\begin{aligned}
&\sum_{i=1}^{k} \alpha_2 p(u_i)(1 - \alpha_2 p(u_i)) - \sum_{i=1}^{k} p(u_i)(1 - p(u_i)) \\
&= \quad (1 - \alpha_2)((1 + \alpha_2) \sum_{i=1}^{k} p(u_i)^2 - \sum_{i=1}^{k} p(u_i)) \\
&< \quad 0
\end{aligned}$$

$\square$

The following theorem gives another important property of an optimal decomposition tree.

Theorem 10 *Let u and v be any siblings in an optimal decomposition tree such that $0.5 < p(u) < p(v)$, there can not exist node y in the tree such that $p(u) < p(y) < p(v)$.*

Proof: Suppose there is a node y such that $p(u) < p(y) < p(v)$. Given a node z such that $p(z) > 0.5$, we will use $root(z)$ to represent the ancestor of z whose rank is 1. According to Theorem 8, u, v, y can not exist in the same subtree rooted at a node of rank 1, that is, $root(u) = root(v) \neq root(y)$. Based on whether y has a sibling whose probability is greater than 0.5, we have two cases.

Case 1: y has no such sibling. This means $y = root(y)$ and $rank(y) = 1$. By Theorem 9, since $p(root(y)) = p(y) > p(u) > p(root(u))$, we have $level(root(y)) < level(root(u))$. Denote the nodes on the path from the parent of u and v to the sibling of y (whose probability is not greater than 0.5) by $u_1, u_2, \ldots, u_k$, as shown in Figure 5.10. Let $\alpha_1 = p(y)/p(v)$ and $\alpha_2 = p(y)/p(u)$, then $0 < \alpha_1 < 1 < \alpha_2$. Exchanging v with y will change the probability of u_i from $p(u_i)$ to $\alpha_1 p(u_i)$ for $1 \leq i \leq k$; exchanging u with y will change the probability from $p(u_i)$ to $\alpha_2 p(u_i)$ for $1 \leq i \leq k$. According to Lemma 8, one of them will decrease the switching activities. But this is a contradiction with the optimality of the tree.

Case 2: y has a sibling z whose probability is greater than 0.5. Without loss of generality, we can assume $level(root(y)) < level(root(u))$ (the other case can be dealt

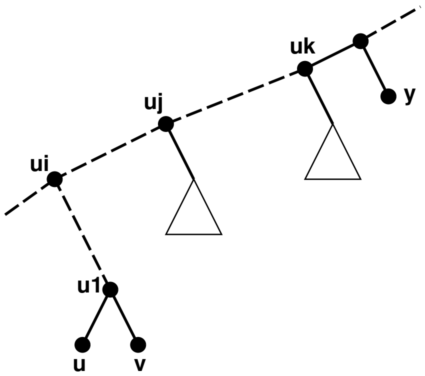

Figure 5.10: Case 1 in proof of Theorem 10

with by simply exchanging u, v with y, z). According to Theorem 9, because $p(v) > p(y)$, we must have $rank(u) > rank(y)$. As shown in Figure 5.11, we can label the nodes on the path from the parent of y to $root(y)$ by $y_1, y_2, \ldots, y_m$. Similarly, label the same number of nodes on the path starting from the parent of u by $v_1, v_2, \ldots, v_m$. Furthermore, also label the nodes on the path from the parent of v_m to the sibling of y_m by $u_1, u_2, \ldots, u_k$. Based on the probabilities of v_m and y_m, we have 3 cases here.

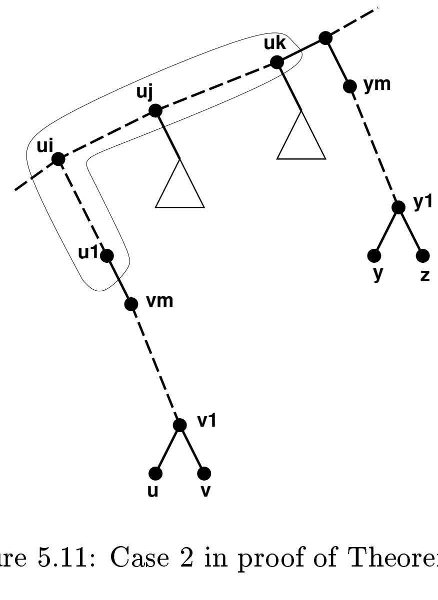

Figure 5.11: Case 2 in proof of Theorem 10

Case 2.1: $p(v_m) > p(y_m)$. Without loss of generality, we can assume that $p(v_i) \geq p(y_i), \forall 1 \leq i < m$. (This is because if $p(v_i) \geq p(y_i)$ for $j+1 \leq i \leq m$ but $p(v_j) < p(y_j)$, then v_j, y_j will have siblings v'_j, y'_j, respectively, such that $p(v'_j) > p(y'_j)$. Therefore, we can relabel v_j, v'_j, y_j, y'_j by u, v, y, z.) Now consider exchanging y with u. This will change the probabilities of $v_1, \ldots, v_m; y_1, \ldots, y_m; u_1, \ldots, u_k$ in Figure 5.11. Let $\alpha = p(u)/p(y) < 1$, for all $1 \leq i \leq m$, the probabilities of y_i and v_i will change from $p(y_i)$ and $p(v_i)$ to $\alpha p(y_i)$ and $p(v_i)/\alpha$, respectively. According to Lemma 4, the summation of their switching activities is decreased. The exchanging also increases $p(u_i)$ to $p(u_i)/\alpha$ for all $1 \leq i \leq k$. It must increase their total switching activities, otherwise, we will have a contradiction. On the other hand, according to Lemma 8, if this increasing of probabilities does increase the total switching activities of $u_1, \ldots, u_k$, then decreasing them can decrease the switching activities. This means that exchanging the two subtrees rooted at v_m and y_m will decrease switching activities of the whole tree, which is also a contradiction.

Case 2.2: $p(v_m) < p(y_m)$. This case is very similar to case 2.1. We can also assume that $p(v_i) \leq p(y_i)$ for all $1 \leq i < m$ and show that by exchanging v and y the activities of $v_1, \ldots, v_m; y_1, \ldots, y_m$ decrease. The exchanging also decreases the probabilities of $u_1, \ldots, u_k$ by a factor of $p(y)/p(v)$. If it does not increase the activities of them, we get a contradiction. Otherwise, by Lemma 8, increasing the probabilities of them will decrease the activities. This can be done by exchanging subtrees rooted at v_m and y_m, which also introduces a contradiction.

Case 2.3: $p(v_m) = p(y_m)$. Without loss of generality, we can assume that $p(v_i) = p(y_i), \forall 1 \leq i \leq m$. This is because if $p(v_j) \neq p(y_j)$ we can relabel them and their siblings as u, v, y, z. Similarly, by Lemma 4, we can show that either exchanging u, y or exchanging v, y can decrease the activities of $v_1, \ldots, v_m; y_1, \ldots, y_m$. One of them will increase the probabilities of $u_1, \ldots, u_k$ by a factor of $p(y)/p(u)$, and the other will decrease them by a factor of $p(y)/p(v)$. According to Lemma 8, at lease one of them will decrease the activities of $u_1, \ldots, u_k$. Contradiction is unavoidable.

Base on the above analysis, the theorem is proved. $\square$

5.5 Decomposition algorithms

In the previous section, we have derived some properties an optimal decomposition tree must have. Since these properties are necessary conditions of an optimal tree, other trees which do not observe them need not to be considered during the optimization process. This can reduce the search space and help us to design an efficient algorithm for the low-power gate decomposition problem.

The following theorem combines all optimality properties given in previous section and is the basis of our exact algorithm.

Theorem 11 *Given n input signals $s_1, s_2, \ldots, s_n$ such that $p(s_1) \leq p(s_2) \ldots \leq p(s_n)$, there is an optimal decomposition tree where s_n either is combined with s_{n-1} or is a direct child of the root.*

Proof: We have two cases based on $p(s_{n-1})$.

Case 1. $p(s_{n-1}) \leq 0.5$. We claim s_n must be a direct child of the root in an optimal tree. Here we have $p(s_i) \leq 0.5$ for all $1 \leq i \leq n-1$. If $p(s_n) \leq 0.5$, according to Theorem 8, the optimal tree can be constructed by the Min-Huffman algorithm and s_n will be a direct child of the root. On the other hand, if $p(s_n) > 0.5$, according to Lemma 6, signals $s_1, s_2, \ldots, s_{n-1}$ must form a separate subtree, which will finally be combined with s_n. This also means s_n is a direct child of the root.

Case 2. $p(s_{n-1}) > 0.5$. We show there is an optimal tree where s_n either is combined with s_{n-1} or is a direct child of the root. Denote the sibling of s_n in an optimal tree by s. According to Lemma 5, we have $p(s) \leq p(s_{n-1})$. If $p(s) > 0.5$ then it must be that $p(s) = p(s_{n-1})$. Otherwise, we will have $0.5 < p(s) < p(s_{n-1}) < p(s_n)$, which contradicts with Theorem 10. But if $p(s) = p(s_{n-1})$, we can always exchange the subtree rooted at s with s_{n-1} and get an optimal tree where s_n is combined with s_{n-1}. On the other hand, if $p(s) \leq 0.5$, then, let v be the parent of s and s_{n-1}, we will have $p(v) \leq 0.5$. This means $rank(v) = 0$ and hence $rank(s_n) = 1$. Since $p(s_n)$ is the maximum, according to Theorem 9, $level(s_n)$ must be the minimum. Therefore, $level(s_n) = 1$, which means s_n is a direct child of the root. □

In other words, the theorem says that there is always an optimal tree between the two configurations shown in Figure 5.12. More specifically, if $p(s_{n-1}) \leq 0.5$ it must be configuration I; otherwise, it can be either configuration I or configuration II.

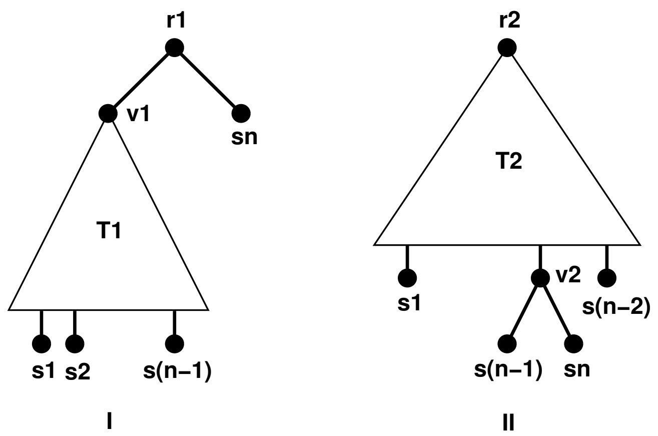

Figure 5.12: Two configurations of an optimal decomposition tree

Based on Theorem 11, we can design an exact algorithm for the low-power gate decomposition problem as follows. Given n input signals, we first sort them according to their probabilities such that $p(s_1) \leq p(s_2) \ldots \leq p(s_n)$. If $p(s_{n-1}) \leq 0.5$, we construct configuration I; otherwise, we construct both configurations I and II and output the one with the minimum switching activities. According to Lemma 5, the subgraphs T_1 and T_2 in Figure 5.12 must also be optimal. Since their input sizes are both only $n - 1$, we can construct them recursively. This algorithm is called

ExDecomp and its pseudo-code is given in Figure 5.13.

The correctness of the algorithm comes directly from Theorem 11 and can be stated as the following corollary.

Corollary 11.1 *The* **ExDecomp** *algorithm exactly solves the low-power gate decomposition problem.*

```
Algorithm ExDecomp
Input: signals S = {s_1, ..., s_n} with p(s_1) ≤ ... ≤ p(s_n)
Output: a decomposition tree T
ExDecomp(S)
{
  if |S| = 1
    return s_n;
  T_1 = ExDecomp(S − {s_n});
  if (p(s_{n−1}) ≤ 0.5)
    return combine(T_1, s_n);
  s = combine(s_{n−1}, s_n);
  T_2 = ExDecomp(S + {s} − {s_{n−1}, s_n});
  if (E_sw(combine(T_1, s_n)) < E_sw(T_2))
    return combine(T_1, s_n);
  else return T_2;
}
```

Figure 5.13: The **ExDecomp** algorithm

At each recursion in **ExDecomp**, we need to store the current configuration, which is upper bounded by n. Since the recursion depth is at most n, the space usage in the worst case is n^2. Let $T(n)$ represent the running time of **ExDecomp** on an instance of size n. It is easy to see that

$$T(n) = 2T(n-1) + n.$$

This gives us $T(n) = O(n2^n)$. Although in the worst case it is still need exponential time, compared with the total of more than $(2n-1)^{n-1}$ decomposition trees, it is efficient.

Besides the exact algorithm, the optimality properties can also be used to derive a set of efficient heuristic algorithms. As we can see, the complexity of **ExDecomp** comes from the fact that it is not known beforehand which configuration in Figure 5.12 will give the minimum switching activities. Trade accuracy for speed, we can use heuristics to choose only one configuration at each recursion. This gives us the algorithm scheme shown in Figure 5.14 which can be tuned into different heuristic algorithms based on different decision criteria.

```
Algorithm  HeuDecomp
Input:  signals S = {s_1, ..., s_n} with p(s_1) ≤ ... ≤ p(s_n)
Output:  a decomposition tree T
HeuDecomp(S)
{
  if (p(s_{n-1}) ≤ 0.5 or choose configuration I) {
    T_1 = HeuDecomp(S - {s_n});
    return combine(T_1, s_n);
  }
  else {
    s = combine(s_{n-1}, s_n);
    T_2 = HeuDecomp(S + {s} - {s_{n-1}, s_n});
    return T_2;
  }
}
```

Figure 5.14: The **HeuDecomp** algorithm

The heuristic we used in our implementation can be described as follows. Since the structures of T_1 and T_2 in Figure 5.12 are not known until we recursively construct them, we can not compare their switching activities beforehand. But we can find that, for the two trees, except one leaf, all other $n-2$ leaves are the same. Therefore, we can assume the difference between the internal switching activities of T_1 and T_2 is not too much. We also know $p(r_1) = p(r_2)$. So the only concern comes from the difference between v_1 and v_2. Our decision criteria then is: if $E_{sw}(v_1) < E_{sw}(v_2)$, choose configuration I, otherwise choose configuration II.

Since only one configuration is chosen at each recursion in **HeuDecomp**, we need only keep one copy of the tree structure, hence the space usage is only n. Implemented by the priority queue data structure [14], the running time can also be upper bounded by $O(n \log n)$, which is much faster than the modified Huffman algorithm. Furthermore, since **HeuDecomp** is strongly based on the optimality properties, its result should be better than that of the modified Huffman algorithm. This is supported by our experimental results.

5.6 Experimental results

We implement both the exact algorithm **ExDecomp** and the heuristic algorithm **HeuDecomp** in C++ on a Sun Sparc 5 workstation. Our experiments focus on two aspects: the running time of the exact algorithm and the performance of the heuristic. In order to compare the performance of the heuristic, we also implement the modified Huffman algorithm [59, 60].

According to Lemma 6 and Theorem 8, the input signals whose probabilities are not greater than 0.5 can be easily combined into a subtree by the Min-Huffman algorithm. Therefore, the complexity only depends on the number of signals whose probabilities are greater than 0.5. In our experiments, the input signal probabilities are randomly generate, and based on the above reason, all signal probabilities are generated to be greater than 0.5.

On each different input size ranging from 5 to 20, we randomly generated 100 sets of input probabilities. We run **ExDecomp**, **HeuDecomp** and the modified

Table 5.1: Experimental results

	ExDecomp	Modified Huffman			**HeuDecomp**		
#in	time(sec.)	#bad	MaxR	AvgR	#bad	MaxR	AvgR
5	0.0009	43	5.51%	0.76%	6	0.430%	0.010%
6	0.0014	66	5.59%	1.29%	6	0.187%	0.006%
7	0.0028	83	7.33%	2.30%	9	0.288%	0.008%
8	0.0057	89	11.31%	4.05%	9	0.268%	0.007%
9	0.0105	93	9.95%	5.02%	8	0.254%	0.008%
10	0.0187	93	14.87%	5.97%	5	0.265%	0.005%
11	0.0424	96	14.61%	7.55%	1	0.108%	0.001%
12	0.0752	97	16.17%	8.45%	2	0.111%	0.002%
13	0.1377	100	19.41%	9.78%	0	0.000%	0.000%
14	0.3050	97	19.97%	9.83%	0	0.000%	0.000%
15	0.5465	100	20.44%	11.32%	0	0.000%	0.000%
16	1.1539	98	20.82%	11.09%	0	0.000%	0.000%
18	3.9232	99	25.23%	12.25%	0	0.000%	0.000%
20	14.2497	100	29.55%	12.32%	0	0.000%	0.000%

Huffman algorithm on each of them. We compute the average running time of **ExDecomp** on each input size. To measure the results of **HeuDecomp** and the modified Huffman algorithm, we compare their solutions with the optimal solution given by **ExDecomp**. The number of non-optimal solutions is counted. For each problem I, let $Opt(I)$ represent the optimal solution, we use the ratio

$$R = \frac{S(I) - Opt(I)}{Opt(I)}$$

to measure the non-optimality of solution $S(I)$. For each algorithm, the maximum and average ratios are computed.

Based on the results reported in Table 5.1, we have the following conclusions. First, **ExDecomp** is efficient in practice. For 20 input probabilities which are greater than 0.5, the average running time is less than 15 seconds. In reality, usually only half of the input probabilities are greater than 0.5. This means a problem with 40 inputs can be solved in less than 15 seconds. Second, the performance of **HeuDecomp** is very good. Among all the 1400 solutions reported in Table 5.1, only 46 of them are not optimal. Among these non-optimal solutions, the largest deviation from the optimal solution is only 0.43%. Finally, an interesting phenomenon is that, with the increasing of the input size, **HeuDecomp** performs better and better. Starting from 13 inputs, all solutions given by **HeuDecomp** are optimal. Based on this phenomenon and the fact that **ExDecomp** runs very fast when the input size is not too large, we can use the following strategy for the low-power gate decomposition problem: if the input size is not too large, use **ExDecomp**; otherwise, use **HeuDecomp**.

5.7 Conclusions

In this chapter, we studied the low-power AND/OR decomposition. We proved many properties for optimal decompositions. Based on these properties, we designed an exact algorithm, which is currently the fastest in the literature, to solve the problem. Derived from the exact algorithm is an $O(n \log n)$ time heuristic algorithm which outperforms known heuristics.

In our decomposition problem, we assumed a zero-delay model as in [56, 59, 60]. This ignored the power dissipation due to glitches and tends to underestimate the total power dissipation. However, since there is a good correlation between the estimated power under a zero delay model and a real delay model [52], this assumption is reasonable.

Other assumptions we made were that the inputs of the gate are temporally uncorrelated and pairwise independent. In reality, the circuit inputs may be temporally correlated. Therefore, the switching activity of signal x, that is $E_{sw}(x)$, can no longer be written as $2p(x)(1-p(x))$. However, in their recent work [63], Wu *et al.* showed that, the expected switching activity among all signals of probability p is also $2p(1-p)$. This means that when different signal sequences are equally applied our solution will minimize the average switching activity.

Due to next state computation and re-convergent paths in the circuits, the inputs of a gate are likely correlated. However, considering these correlations has several drawbacks. First, we must use the conditional probabilities to calculate the signal probabilities of the internal nodes. This means that conditional probabilities among all subsets of inputs are needed, which is exponential in size of the inputs. Second, under the correlation model, many of our properties can not be proved. Therefore, the **ExDecomp** algorithm can **not** guarantee to find an optimal solution.

The main contribution of our work is to show that, by carefully analysis and reasoning, an efficient and exact algorithm can be derived for the "hard" low-power gate decomposition problem. Although the optimality is currently based on the above assumptions, it also provides some insight into more general cases and we hope it can stimulate researches along this line.

In CMOS logic, NAND and NOR gates are more naturally implemented than AND and OR gates. Therefore, it is usually required to decompose an multiple input AND gate into a tree of two input NAND gates and inverters. However, it can be proved that substituting each AND gate in our decomposition by a connection of an inverter and a NAND gate gives an optimal NAND decomposition.

Chapter 6

Low Power XOR Gate Decomposition

6.1 Introduction

Most previous efforts in power reduction have been focused on reducing power consumption in AND/OR or NAND/NOR based circuits [56, 59, 60, 70]. However, there are major classes of circuits which have sizable AND/OR representations, but have very compact AND/XOR representations [50, 58]. Standard examples are arithmetic functions such as adders, multipliers, and error-correcting codes [47, 58]. Recently, there has been some success in achieving area reduction by employing optimization techniques specifically targeted towards initial AND/XOR representations in the well known Fixed Polarity Reed Muller (FPRM) form [58].

For low power AND/OR gate decomposition, if the implementation technology is dynamic logic, it can be solved by Huffman's algorithm in $O(n \log n)$ time [59]. But it is more complicated for static logic. Our results in Chapter 5 give the best exact algorithm and a very good heuristic algorithm in the current literature. For low power XOR gate decomposition, to the best of our acknowledgment, the only result in the literature is an algorithm by Narayanan and Liu [39]. Their algorithm is for static logic and requires that each input has both polarities available. But as we will show later, it is not optimal.

This chapter has two contributions to the low power XOR gate decomposition. The first is an optimal algorithm for dynamic logic. For this problem, we first study the structure of an optimal decomposition tree. This is given by a set of necessary properties an optimal tree must have. Then, based on these properties, we design an algorithm to construct an optimal decomposition tree. The time complexity of the algorithm is $O(n \log n)$.

The second contribution is another optimal algorithm for low power XOR gate decomposition in static logic. We find that Narayanan and Liu's algorithm [39], which requires both polarities for inputs, is **not** optimal. Based on the properties of decompositions in static logic, we design an $O(n \log n)$ time optimal algorithm which does not even require both polarities available for the inputs.

The rest of the chapter is organized as follows. In Section 6.2, we define the low-power XOR gate decomposition problems, both in dynamic logic and static logic. Section 6.3 presents the solution for dynamic logic. Section 6.4 points out the problems in Narayanan and Liu's algorithm [39] and presents the optimal algorithm for static logic. Finally, experimental results and conclusions are presented in Section 6.5 and Section 6.6.

6.2 Problem formulation

Technology decomposition is the step before technology mapping which decomposes a network with multiple-input gates into a network with only two-input gates. In XOR gate decomposition, we need to decompose a multi-input XOR gate into a tree of two-input XOR gates.

Same as in Chapter 5, we treat each signal in a circuit as a random variable and define the *signal probability* of a signal x as the probability of x being 1. For simplification, in the sequel, we use the same symbol to represent both a signal and its probability. Its meaning can be distinguished from the context. As in [39, 56, 59, 60, 70], We assume the *zero delay model* where gate delays are assumed to be zero and thus signal transitions due to glitching are ignored. We also assume the inputs of the XOR gate are uncorrelated (*spatial independent*) and the present input signal value is independent of those in the past (*temporal independent*). Under these assumptions, given input signal probabilities and a decomposition tree, the probabilities of internal signals can be computed as follows. Start from the primary inputs, for each gate $z = x$ XOR y, the probability $z = x + y - 2xy$. For example, Figure 6.1 shows one XOR gate decomposition and the signal probabilities of all nodes.

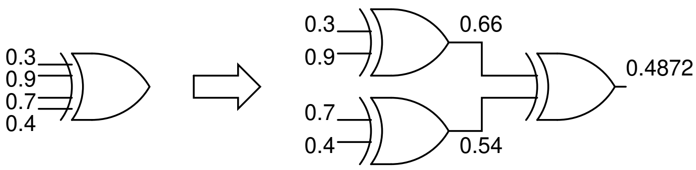

Figure 6.1: XOR gate decomposition

Given the probability, the switching activity of a signal in a circuit really depends on the implementation logic style. If implemented in dynamic logic styles, in each clock cycle, a gate output is first pre-charged to 0 (pre-charged to 1), and then evaluated to the result value. Therefore, if the output probability is x, the switching activity will be $2x$ ($2(1-x)$), where one x (one $1-x$) comes from the pre-charging and one x (one $1-x$) comes from the evaluation.

In static logic, under the temporal independence assumption, the switching activity E_{sw} of signal x can be written as

$$E_{sw}(x) \quad = \quad Pr[x : 0 \rightarrow 1] + Pr[x : 1 \rightarrow 0]$$

$$
\begin{aligned}
&= Pr[x=0]Pr[x=1] + Pr[x=1]Pr[x=0] \\
&= 2Pr[x=1]Pr[x=0] \\
&= 2x(1-x)
\end{aligned}
$$

Plotting the switching activity as a function of signal probability both in dynamic logic and static logic, we have Figure 6.2. As we can see, if considered only in term of switching activity, static logic is always better than dynamic logics. But we can not come to the same conclusion in terms of power dissipation, since dynamic logics usually have smaller load capacitances.

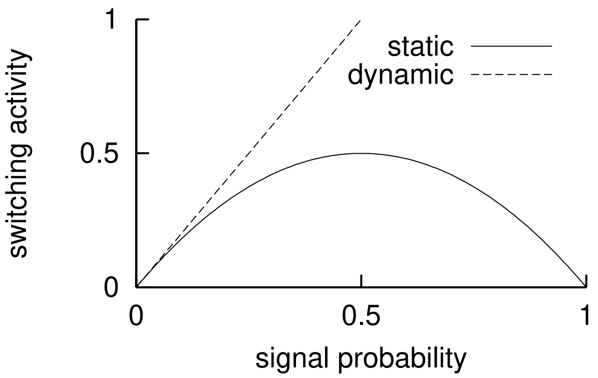

Figure 6.2: Relations between switching activity and signal probability

Based on the above discussion, the problem of low power XOR gate decomposition in dynamic logic and static logic, respectively, can be defined as follows.

Low power XOR *gate decomposition problem – Dynamic logic*: Given an n-input XOR gate with inputs $I = \{s_1, s_2, \ldots, s_n\}$, construct a tree $T = (V, E)$ of 2-input XOR gates with $s_1, s_2, \ldots, s_n$ as its leaves such that

$$E_{sw}^{dynamic}(T) = \sum_{v \in V - I} 2v$$

is minimized.

Low power XOR *gate decomposition problem – Static logic*: Given an n-input XOR gate with inputs $I = \{s_1, s_2, \ldots, s_n\}$, construct a tree $T = (V, E)$ of 2-input XOR gates with $s_1, s_2, \ldots, s_n$ as its leaves such that

$$E_{sw}^{static}(T) = \sum_{v \in V - I} 2v(1-v)$$

is minimized.

6.3 Decomposition for dynamic logic

In this section, we solve the low power XOR gate decomposition in dynamic logic. First, the structures of optimal decompositions are analyzed and the optimality properties are given in Section 6.3.1. Then, based on these properties, an optimal algorithm is designed and analyzed in Section 6.3.2.

6.3.1 Optimality properties

Given two input signals with probabilities x and y, the probability of the output of an XOR gate is given as $z = f(x, y) = x + y - 2xy$. The partial derivatives of f are given as follows.

$$\frac{\partial f}{\partial x} = 1 - 2y$$
$$\frac{\partial f}{\partial y} = 1 - 2x$$

This means that, when $x < 0.5$ ($y < 0.5$), the function f is strictly increasing with y (x); when $x > 0.5$ ($y > 0.5$), it is strictly decreasing with y (x). Combined with the fact that $f(0.5, y) = f(x, 0.5) = 0.5$, the values of f can be characterized by Figure 6.3.

The truth table of the operation XOR is shown in Table 6.1. Comparing Figure 6.3 with Table 6.1, we have the following lemma.

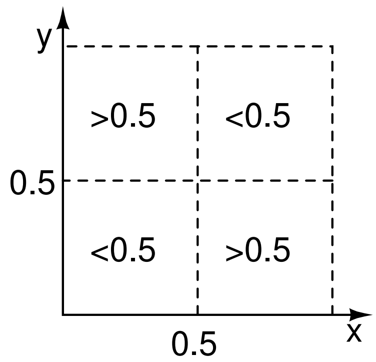

Figure 6.3: Output probability as function of input probabilities

Table 6.1: Truth table of XOR

XOR	0	1
0	0	1
1	1	0

Lemma 9 *The probability of a node is equal to* 0.5 *if and only if the subtree rooted at the node has at least one node whose probability is* 0.5*; otherwise, the probability is greater than* 0.5 *if and only if any subtree rooted at the node has odd numbers of leaves whose probabilities are greater than* 0.5*.*

Based on the above lemma, we have the following theorem.

Theorem 12 *In an optimal* XOR *gate decomposition, except the root, no internal node could have probability greater than* 0.5*.*

Proof: We prove it by contradiction. Suppose there exist an internal node $v > 0.5$ in the optimal decomposition tree as shown in Figure 6.4. From Lemma 9, one of its children, say x, has probability greater than 0.5; another child, say y, has probability smaller than 0.5. Denote the sibling of v by u, and their parent by p. Depending on u, we have two cases. In case 1 we have $u > 0.5$. After exchanging the subtrees rooted at y and u, the probability of v is decreased to smaller than 0.5. Since no

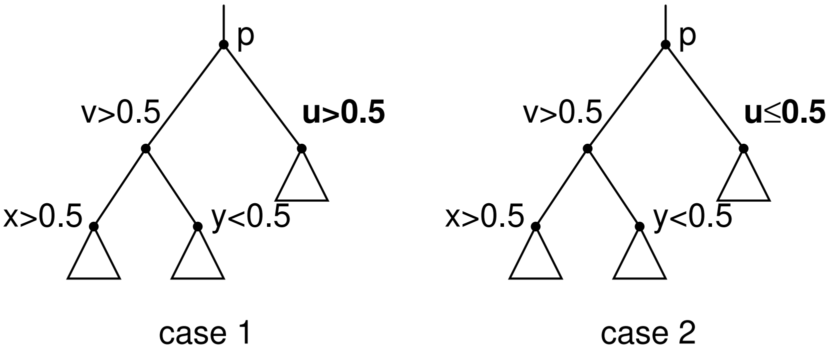

Figure 6.4: Two cases where a tree have non-root internal node of probability greater than 0.5

other node changes probability, this is a contradiction with the fact that the given decomposition is optimal. In case 2 we have $u \leq 0.5$. Similarly, exchanging the subtree rooted at x and u will decrease the total probabilities of the tree, thus a contradiction, too. $\square$

This means that, except at the root, inputs of probabilities greater than 0.5 can not be combined with probabilities smaller than 0.5. We can also prove that, no of them, except one, can be combined with probability 0.5. This is stated formally as the following theorem.

Theorem 13 *In an optimal* XOR *gate decomposition, there is at most one input of probability greater than* 0.5 *which can be combined with probability* 0.5.

Proof: Suppose there are two inputs u and v of probabilities greater than 0.5 which are combined with two 0.5 probabilities x and y. Depending on whether one of the 0.5 probabilities is an ancestor of the other, we have two cases as shown in Figure 6.5. In case 1, probability x is an ancestor of y. Without loss of generality, we can assume that these two combination is "consecutive", that is, no siblings of the node on the path between x and y have probabilities greater than 0.5. Consider exchanging u with the subtree rooted at y. The probability of the parent of v is decreased to below 0.5. The probability of any other node on the path between v and

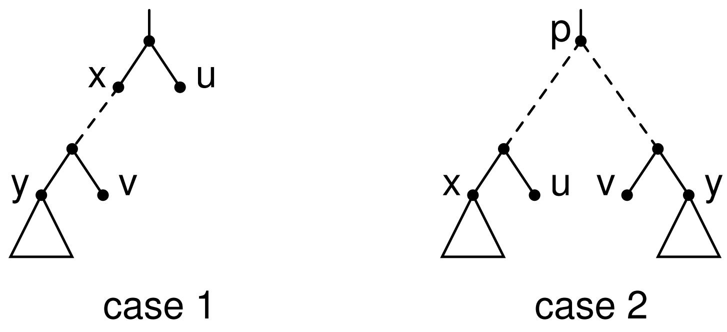

Figure 6.5: Two cases where a tree have non-root internal node of probability greater than 0.5

x can not be increased since one of its children is not greater than 0.5 and the other child can only decrease. Since the parent of x is kept to be 0.5, other probabilities can not be changed. All of these form a contradiction to the optimality of the tree.

In case 2, let p be the lowest common ancestor of x and y. Without loss of generality, we can assume the nodes on the path between x and p have no sibling of probability greater than 0.5. Otherwise, it becomes case 1. Similarly, exchanging v with the subtree rooted at x decreases the probability of the parent of u, but increases no other probabilities. It is also a contradiction. □

Combining the results from Theorem 12 and Theorem 13, we can have the following result.

Corollary 13.1 *In an optimal* XOR *gate decomposition, all inputs of probabilities greater than* 0.5 *must be combined pairwisely, except one when there are odd numbers of them, which is either a child of the root or combined with a probability of* 0.5.

By combining inputs of probabilities greater than 0.5 pairwisely, we get signals whose probabilities are smaller than 0.5. When there are odd numbers of them, the one left is also combined with probability 0.5. Therefore, we are facing a problem of low power XOR gate decomposition where no input has probability greater than 0.5. We will show that this problem can be solved by Huffman's algorithm. That is, each time, the two signals with the smallest probabilities are combined until there is only one signal. We call the tree constructed by Huffman's algorithm Huffman tree.

If we use $w(T) = [v_1, v_2, \cdots, v_n]$ to represent the non-decreasing sequence of probabilities on internal nodes of a tree T, then that means, for Huffman tree T, we have

$$\sum_{i=1}^{n} w(T)_i \leq \sum_{i=1}^{n} w(S)_i$$

for any tree S. In fact, we can prove a much stronger result which will also be used in Section 6.4. First, we will define a binary relation "$\preceq$" on probability sequences.

Definition 8 *We say $w(T) \preceq w(S)$ if for all h such that $1 \leq h \leq n$, we have*

$$\sum_{i=1}^{h} w(T)_i \leq \sum_{i=1}^{h} w(S)_i.$$

As we can check, the relation $\preceq$ is reflexive, anti-symmetric, and transitive, thus is a partial order. The following theorem shows that the Huffman tree has the minimum probability sequence with respect to $\preceq$.

Theorem 14 *If no input probability is greater than 0.5 and T is a Huffman tree, we have $w(T) \preceq w(S)$ for any tree S.*

Proof: Without loss of generality, we can assume $w(S)$ is a local minimal with respect to $\preceq$, that is, there is no S' such that $w(S') \preceq w(S)$.

If $w(S)$ is the same as $w(T)$, we are done. Otherwise, at some point in the construction of S, we must have three nodes u, v, w such that $u < w$ and $v < w$, but u and w are combined. As shown in Figure 6.6, let p be their lowest common ancestor. Depending on whether v is a child of p, we have two cases.

Case 1 has v as a child of p. Since $u < w \leq 0.5$, exchanging the subtrees rooted at v and w decreases the probability of the parent of u, but can not increase any other probabilities on the path from u to p since no probability is greater than

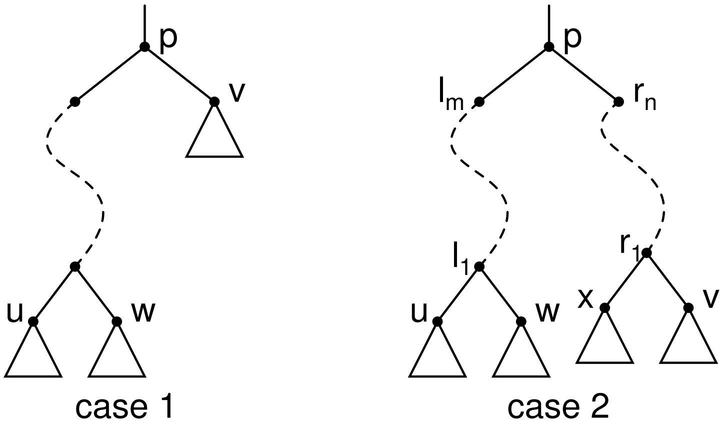

Figure 6.6: Two cases when tree S differs from Huffman tree.

0.5. Since no other probability is changed, this gives a smaller sequence with respect to $\preceq$. It is a contradiction to the fact that $w(S)$ is minimal.

In case 2, node v is combined with x such that $x > u$ and $x > v$. Label the nodes on the path from u to p by $l_1, l_2, \ldots, l_m$, and the nodes on the path from v to p by $r_1, r_2, \ldots, r_n$. We claim that $l_i \neq 0.5$ for $1 \leq i \leq m$ and $r_i \neq 0.5$ for $1 \leq i \leq n$. If this is not true, since $u, v < w \leq 0.5$, without loss of generality, we can assume $l_1 = r_1 = 0.5$. Exchanging the subtrees rooted at v and w will decrease the probability of l_1 and may only decrease the probabilities on other nodes from l_1 to l_m. Since no other probabilities will be changed, it is a contradiction.

Since u, w and v, x are symmetric, without loss of generality, we can assume that $m \geq n$. We can also assume $r_n \geq l_n$, this is because that, if $r_n < l_n$, we can simply exchange the two subtrees rooted at l_n and r_n. This can only decrease $l_{n+1}, \ldots, l_m$ (nothing if $m = n$). Furthermore, we can assume $r_i \geq l_i$, for $1 \leq i \leq n$. The reason is that, if $r_i \geq l_i$ for all $j + 1 \leq i \leq n$ but $r_j < l_j$, then r_j and l_j will have siblings r'_j and l'_j, respectively, such that $r'_j > l'_j$. Hence we can relabel r_j, r'_j by x, v and l_j, l'_j by u, w.

Now consider exchanging v with w. The probabilities $l_{n+1}, \ldots, l_m$ will decrease. For each $1 \leq i \leq n$, the probability l_i will decrease but r_i will increase. Since XOR is an associative operation, we can write $l_i = \alpha_i + w - 2\alpha_i w$ and $r_i = \beta_i + v - 2\beta_i v$. Because $l_i \leq r_i$ and $v < w$, we have $\alpha_i < \beta_i$. Let l'_i and r'_i be the probabilities after the exchanging, then

$$\begin{aligned}
&(l'_i + r'_i) - (l_i + r_i) \\
&\quad = (\alpha_i + v - 2\alpha_i v) + (\beta_i + w - 2\beta_i w) - (\alpha_i + w - 2\alpha_i w) - (\beta_i + v - 2\beta_i v) \\
&\quad = 2\alpha(w - v) + 2\beta(v - w) \\
&\quad = 2(\alpha - \beta)(w - v) \\
&\quad < 0
\end{aligned}$$

This fact $l_i \leq r_i$ and $l'_i < r'_i$ means that, in any summation in the definition of $\preceq$, if there is a r_i, there is also l_i. Together with the above inequality, this means the sequence $w(S)$ will becomes smaller by exchanging v and w. Contradiction can not be avoided. $\square$

Now the only remaining thing is how we can pair-up the input probabilities which are greater than 0.5. As we already know, when $x > 0.5$, the function $f(x, y) = x + y - 2xy$ is monotonically deceasing with y. Thus, combining the largest two probabilities will give the smallest probability. But this will leave the remaining probabilities larger. However, as we can prove in the next theorem, it actually gives the optimal solution.

Theorem 15 *In an optimal* XOR *gate decomposition, the probabilities greater than* 0.5 *are paired up from largest to smallest.*

Proof: The proof is much similar as that for Theorem 14.

First we show that, if there are odd numbers of inputs whose probabilities are greater than 0.5, the smallest one is left out. Since it is otherwise combined with a probability of 0.5, we can alway assume the one left-out is a child of the root.

Thus, if the above claim is not true, we will have $u, v, w > 0.5$ and $v > w$, but v is a child of the root. This is illustrated in Figure 6.7(a). Exchanging v and w decreases the probability of the parent of u and increases no other probabilities. This is a contradiction.

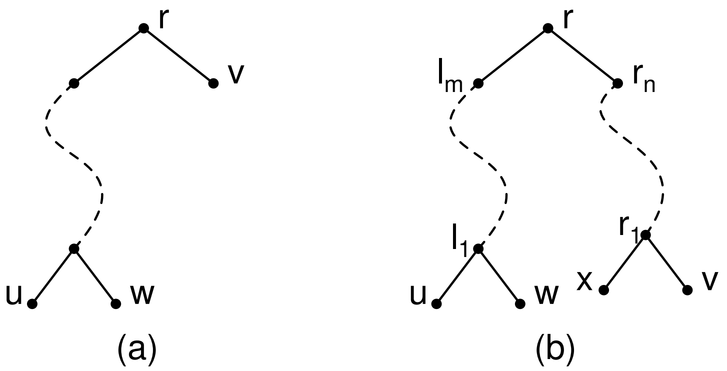

Figure 6.7: Two cases when probabilities are not paired up from largest to smallest.

Suppose the probabilities greater than 0.5 are not pair up from largest to smallest, we will have a situation shown in Figure 6.7(b), where $u, v > w, x > 0.5$. Since we have $l_1, r_1 < 0.5$, similarly as in the proof for Theorem 14, we can show $l_i < 0.5$ for $1 \leq i \leq m$ and $r_i < 0.5$ for $1 \leq i \leq n$. Thus, we can assume $m \geq n$ and $l_i \leq r_i$ for $1 \leq i \leq n$. Similarly, we can write $l_i = \alpha_i + w - 2\alpha_i w$ and $r_i = \beta_i + v - 2\beta_i v$. Because $l_i \leq r_i$ and $v > w > 0.5$, we have $\alpha_i > \beta_i > 0.5$. Let l'_i and r'_i be the probabilities after the exchanging v and w, then

$$(l'_i + r'_i) - (l_i + r_i) = 2(\alpha - \beta)(w - v) < 0.$$

This is a contradiction. ∎

6.3.2 Algorithm

Based on the properties shown in the previous section, we have the following algorithm to solve the low power XOR gate decomposition problem in dynamic logic.

First, sort the input probabilities which are greater than 0.5. Then, combine them pairwisely from the largest to the smallest. Finally, iteratively combine the two signals with the smallest probabilities until there is one signal.

For example, consider the following problem with 6 inputs: $a = 0.1, b = 0.2, c = 0.5, d = 0.7, e = 0.8, f = 0.9$. By the algorithm, we first combine e and f and get a signal with probability $0.8+0.9-2\cdot0.8\cdot0.9 = 0.26$. We then iteratively combine the two signal with the smallest probabilities to get a tree shown in Figure 6.8.

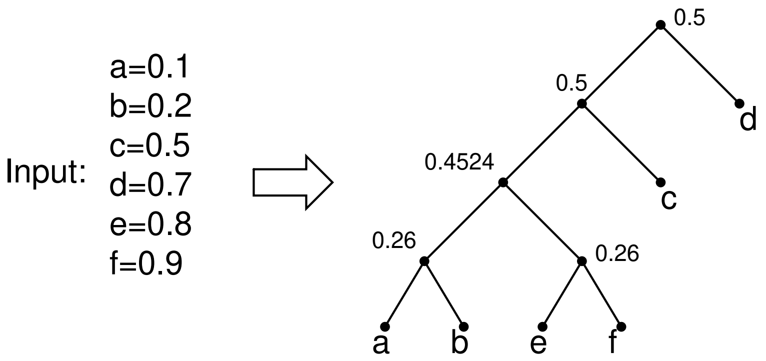

Figure 6.8: An optimal solution in dynamic logic

As we already know, a signal with probability 0.5 is special since it will always produce probability 0.5 no matter which signal it combines with. Thus, if we consider a subtree whose internal nodes all have probability 0.5, we can change it to any tree topology without increasing the switching activities. This gives us the flexibility to optimize other parameters without compromising on power dissipation.

One of such possibilities is the delay optimization. It works as follows. When the combination iterations in the above algorithm come to a point when there are at most one probability which are smaller than 0.5, the process is stopped. The arrival time of each current signal is then computed. And the remaining combinations are based on arrival times to give a root with minimum arrival time. That is, the two signals with the smallest arrival times are combined iteratively. But we must take care of one special case when none of the two earliest signals has probability of

0.5 (this could happen when one is smaller than 0.5 and the other is greater than 0.5). In this case, the earliest signal will instead be combined with the third earliest signal which has probability 0.5. For example, if all inputs arrive at the same time, a solution with delay consideration to the problem in Figure 6.8 is given in Figure 6.9.

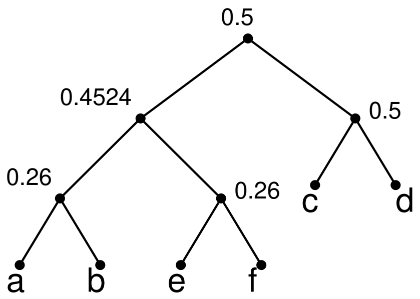

Figure 6.9: A solution with delay consideration

6.4 Decomposition for static logic

In the previous section, we solved the lower power XOR gate decomposition in dynamic logic. In this section, we will solve the problem in static logic.

Narayanan and Liu [39] formulated the problem of lower power XOR gate decomposition in static logic and provided the following algorithm to solve it: *Let k be the number of primary inputs to the* XOR *tree with probabilities greater than* 0.5*. If k is even replace all k inputs by their respective complements. If k is odd then replace any $k-1$ inputs by their respective complements. Then simply use the Huffman algorithm to build the optimal tree.*

In fact, the above algorithm is **not** optimal. This can be shown by a simple example in Figure 6.10. The problem has three inputs: $0.1, 0.5$, and 0.9. Since there is only one signal whose probability is greater than 0.5, by Narayanan and Liu's algorithm, no signal will be changed to its complement. Thus, as shown in

Figure 6.10(a), probabilities 0.1 and 0.5 are first combined, and then combined with 0.9. The switching activities on the two internal nodes is $2 \cdot 0.5(1-0.5) + 2 \cdot 0.5(1-0.5) = 1$. But if we first combine 0.1 and 0.9, then 0.5, as shown in Figure 6.10(b), the corresponding switching activities are $2 \cdot 0.82(1-0.82) + 2 \cdot 0.5(1-0.5) = 0.6476$, which is smaller.

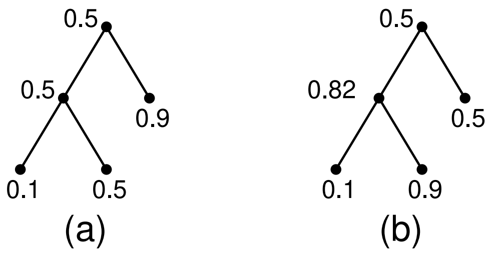

Figure 6.10: (a) Solution given by Narayanan and Liu (b) A solution with smaller switching activities

Furthermore, since the above example has only one input whose probability is greater than 0.5, it is "LSB (low signal biased)" according to the definition in [39]. Thus, it also shows that Theorem 2.3 (LSB/HSB Optimality Theorem) in [39] is not correct.

To design an optimal algorithm for the low power XOR gate decomposition in static logic, we need to analyze the properties of the decompositions. We start with the following observation by Narayanan and Liu [39].

Theorem 16 (Narayanan and Liu) *Suppose we have an* XOR *tree T consisting of n inputs. If we replace any subset of the n inputs by their complement, the switching activities in the tree remain unchanged.*

This result comes from two facts. One is the property of XOR gate: complementing one fanin will complement the output. The other is the property of static logic: a signal has the same switching activity as its complement.

Therefore, we will first solve the decomposition problem for a special case when no input probability is greater than 0.5. As in [45], we give the following definition.

Definition 9 *A function $G : U^n \to R$ is* Schur concave *if*

$$(x_i - x_j)(\frac{\partial G}{\partial x_i} - \frac{\partial G}{\partial x_j}) \leq 0 \text{ for all } x_i, x_j \in U, 1 \leq i, j \leq n.$$

Schur [51] and Ostrowski [43] gave the following result which characterizes the relation between a function being Schur concave and the result of applying it on a sequence of weights.

Theorem 17 (Schur and Ostrowski) *Suppose G is increasing, then $G(\mathbf{a}) \leq G(\mathbf{b})$ for all sequences $\mathbf{a} \preceq \mathbf{b}$ if and only if G is Schur concave.*

In fact, we can easily check that the cost function $E_{sw}^{static}(T) = \sum_{v \in T} 2v(1-v)$ for XOR gate decomposition in static logic is Schur concave and increasing when no probability is greater than 0.5. Therefore, by combining Theorem 14 and Theorem 17, we have the following result.

Theorem 18 *If no input probability is greater than 0.5, the Huffman algorithm gives an optimal solution to the low power* XOR *gate decomposition in static logic.*

For the general problem with arbitrary input probabilities, we also have an optimal algorithm which does not even require both polarities for the inputs. It is stated as the following theorem.

Theorem 19 *For each input probability p, use $\min\{p, 1-p\}$ as its weight. Then the Huffman algorithm with the merging function $M(x,y) = x + y - 2xy$ gives an optimal* XOR *gate decomposition in static logic.*

Proof: Let T be the decomposition given in the theorem and T' be the same decomposition except the inputs whose probabilities are greater than 0.5 are changed to their complements. From the construction, we know that T' is a Huffman tree.

If T is not optimal, then there exists a decomposition S such that $E_{sw}^{static}(S) < E_{sw}^{static}(T)$. We can derive S' from S by changing the inputs of probabilities greater than 0.5 to their complements. From Theorem 16, we know that $E_{sw}^{static}(T') = E_{sw}^{static}(T)$ and $E_{sw}^{static}(S') = E_{sw}^{static}(S)$. Thus, $E_{sw}^{static}(S') < E_{sw}^{static}(T')$. This is a contradiction to Theorem 18. $\square$

As an example, we consider the following problem of low power XOR gate decomposition in static logic: we have 5 inputs with probabilities $a = 0.2, b = 0.5, c = 0.7, d = 0.7, e = 0.8$. The weights for these inputs are $w(a) = 0.2, w(b) = 0.5, w(c) = 0.3, w(d) = 0.3, w(e) = 0.2$. Thus by Huffman algorithm, we first combine a and e to get a new weight $0.2 + 0.2 - 2 \cdot 0.2 \cdot 0.2 = 0.32$, then we combine the two signal with smallest weights, that is c and d. Continue this, we get solution shown in Figure 6.11, with "weight/probability" labeled beside each node.

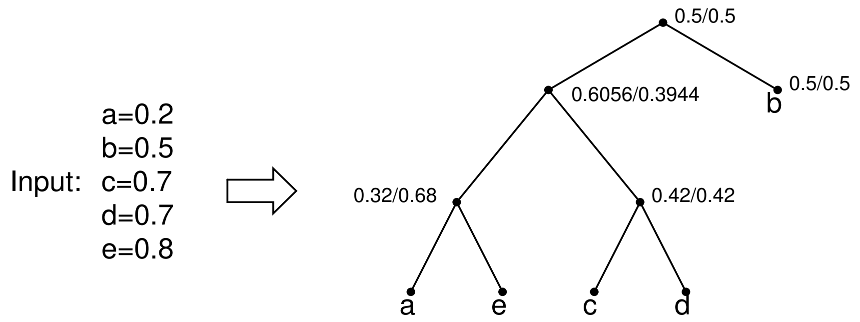

Figure 6.11: A solution given by our optimal algorithm (the label beside each node is "weight/probability")

Same as in dynamic logic, the property of probability 0.5 can be used to optimize other parameters such as delay without compromising on power dissipation. In this case, we will stop when there is at most one non-0.5 probability and use the Huffman algorithm on arrival times to minimize the arrival time of the root.

6.5 Experimental results

We implement the two optimal algorithms for low power XOR gate decomposition in dynamic logic and static logic. Both these algorithms have the time complexity of $O(n \log n)$. In fact, when running on a PC with 166MHz CPU and 64M memory, decomposing 1000 gates each with 100 inputs only takes about 4 seconds.

In order to measure the difference between the optimal algorithms and other simple heuristics, we also implement the Huffman algorithm. It comes in two versions. In one version which we called *MinHuffman*, two signals with the smallest probabilities are combined iteratively. In other version which we called *MaxHuffman*, on the contrary, two signals with the largest probabilities are combined iteratively. In static logic, we also compare with the algorithm given by Narayanan and Liu [39].

On each input size from 3 to 20, we randomly generated 1000 sets of input probabilities. We run *MinHuffman*, *MaxHuffman*, Narayanan and Liu's algorithm, and the two optimal algorithms on each of them. For each heuristic algorithm, the number of non-optimal solutions is counted. Given a problem I, let $Opt(I)$ represent the optimal solution, we use the ratio

$$R = \frac{S(I) - Opt(I)}{Opt(I)}$$

to measure the non-optimality of solution $S(I)$. For each heuristic, the maximum and average ratios are computed. The results for dynamic logic are given in Table 6.2 and the results for static logic are given in Table 6.3.

6.6 Conclusions

In this chapter, we have studied the low power gate decomposition problem for XOR gate. We optimally solved the problem for both static and dynamic logic. As in Chapter 5, we first proved the optimality properties of low power decompositions. Based on these properties, we designed polynomial time algorithms to give optimal

Table 6.2: Experimental results in dynamic logic

#in	MinHuffman			MaxHuffman		
	#bad	MaxR	AvgR	#bad	MaxR	AvgR
3	490	532.65%	13.94%	510	91.54%	10.94%
4	686	191.22%	15.52%	939	120.32%	16.26%
5	796	176.87%	16.26%	1000	138.76%	19.09%
6	887	145.49%	17.46%	1000	174.17%	20.98%
7	945	102.17%	16.51%	1000	167.04%	20.89%
8	965	141.20%	17.03%	1000	139.40%	21.55%
9	985	90.30%	16.79%	1000	101.26%	21.22%
10	985	83.11%	16.73%	1000	84.20%	21.29%
11	994	83.91%	16.25%	1000	106.09%	20.719%
12	997	80.83%	16.26%	1000	70.49%	20.94%
13	998	68.53%	16.45%	1000	79.53%	21.06%
14	1000	56.01%	16.04%	1000	73.82%	20.72%
15	1000	69.81%	16.28%	1000	70.01%	21.00%
16	1000	50.07%	16.80%	1000	67.64%	21.17%
17	1000	52.21%	16.16%	1000	65.75%	21.05%
18	1000	53.10%	16.42%	1000	60.25%	21.26%
19	999	56.09%	15.86%	1000	58.13%	21.05%
20	1000	47.58%	16.34%	1000	74.98%	21.14%

decompositions. The running times of the algorithms for both static and dynamic logic are both $O(n \log n)$. Experimental results show that previous known algorithm for static logic is far off optimal, and Huffman's algorithm can not be used to solve this problem.

Future work includes integrating our algorithms into the synthesis tool for arithmetic logic [58] and considering delay optimization together with power optimization.

Table 6.3: Experimental results in static logic

#in	MinHuffman			MaxHuffman			Narayanan-Liu		
	#bad	MaxR	AvgR	#bad	MaxR	AvgR	#bad	MaxR	AvgR
3	736	81.74%	7.98%	750	75.70%	7.89%	384	81.74%	4.36%
4	871	93.11%	10.29%	993	117.04%	10.96%	436	64.23%	3.85%
5	938	93.41%	9.99%	1000	100.23%	11.65%	473	43.89%	3.48%
6	967	104.65%	10.97%	1000	91.26%	12.15%	486	38.11%	3.77%
7	988	59.71%	10.33%	1000	107.18%	12.24%	500	26.47%	3.13%
8	995	105.01%	10.26%	1000	72.15%	11.99%	496	24.58%	3.03%
9	997	57.61%	10.10%	1000	76.03%	11.72%	464	18.24%	2.66%
10	998	53.65%	10.30%	1000	70.08%	11.75%	524	17.20%	2.90%
11	999	56.84%	9.49%	1000	52.00%	11.19%	481	14.80%	2.38%
12	1000	53.32%	9.43%	1000	50.99%	11.25%	484	13.57%	2.33%
13	1000	47.96%	9.51%	1000	49.27%	11.27%	491	12.32%	2.36%
14	1000	37.26%	9.14%	1000	51.35%	11.07%	499	10.60%	2.23%
15	1000	50.98%	9.19%	1000	47.45%	11.24%	535	11.87%	2.19%
16	1000	35.26%	9.34%	1000	39.96%	11.08%	496	8.89%	2.10%
17	1000	32.39%	8.99%	1000	40.42%	11.08%	488	8.16%	1.92%
18	1000	33.61%	9.09%	1000	41.31%	11.23%	480	7.87%	1.85%
19	1000	38.35%	8.76%	1000	37.43%	11.14%	492	7.73%	1.76%
20	1000	29.31%	8.98%	1000	47.09%	11.15%	517	7.05%	1.82%

Chapter 7

Floorplan Power and Area Optimization

7.1 Introduction

Floorplan design is an important step in the physical design of VLSI circuits. During this step, the topology (i.e., relative positions of the blocks) and a specific implementation for each block must be decided. There are two typical approaches for the problem. One approach first determines the topology primarily using the interconnection information among the blocks [29, 44], and then chooses implementations to minimize some cost measures. The other approach dynamically chooses the implementations during the topology construction [62]. No matter which approach is used, the floorplan optimization problem, that is, choosing an implementation for each block under a given topology, must be solved. Traditionally, when choosing the implementations, only geometrical dimensions are considered to minimize the total area. But since different implementations have different power consumptions, a floorplan design with minimum area could consume too much power. Besides the area, the total power consumption must also be minimized.

There are two kinds of floorplans: slicing and non-slicing. A slicing floorplan is a floorplan which can be obtained by recursively cutting a rectangle into two parts by either a vertical or a horizontal line. A non-slicing floorplan is a floorplan which is not a slicing floorplan. Since slicing floorplans have very simple solution representations (e.g. slicing tree [44], Polish expression [62], etc.), many hard problems for general floorplans have efficient algorithms when restricted to slicing floorplans. Furthermore, it was shown that the performance of a slicing floorplan will not be too worse than that of a general floorplan [65].

In [8], Chao and Wong formulated the slicing floorplan power and area optimization problem and proposed a dynamic programming approach for it. Similar as in the slicing floorplan area optimization [54], a list of non-redundant alternative implementations is first computed and an optimal solution is then chosen from them. During the computation, implementations are indexed by their power dissipation. Although they showed the algorithm uses exponential storages (therefore, has exponential running time), the complexity of the problem is not known. In this chapter, we first characterize the complexity of the problem. We show that it is NP-hard even when the number of values in height or in width is upper bounded. Besides their algorithm, we also give another pseudo-polynomial time algorithm whose time does not depend on the values of power dissipation.

The rest of this chapter is organized as follows. Section 7.2 gives preliminaries and formulates the slicing floorplan power and area optimization problem. Complexity results are given in Section 7.3. In Section 7.4, we give a pseudo-polynomial time algorithm whose time does not depend on the values of power dissipation. Section 7.5 summarizes the chapter and gives future research directions.

7.2 Problem formulation

A *circuit* can be represented by a hyper-graph $C = (B, N)$, where B is the set of blocks and N is the set of nets. A *floorplan* F for a circuit $C = (B, N)$ consists of an enveloping rectangle subdivided by horizontal and vertical line segments into $|B|$ non-overlapping rectangles, called *basic rectangles* or *rooms*, and a one-to-one correspondence between the basic rectangles and the blocks. A *slicing floorplan* is a floorplan where the rooms can be obtained by recursively cutting a rectangle into two parts by either a vertical or a horizontal line. For example, a slicing floorplan for a circuit with 7 blocks A, B, C, D, E, F, G is shown in Figure 7.1.

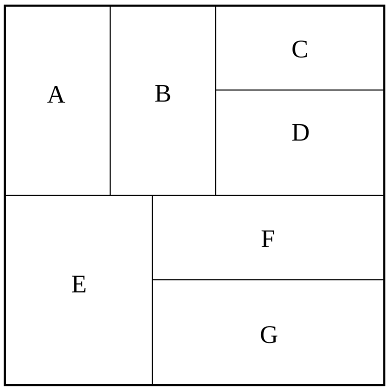

Figure 7.1: A slicing floorplan

Since a slicing floorplan has a simple structure, it can be represented by a tree. A *slicing tree* T of a slicing floorplan F is a binary tree where each leaf represents a room and each interior node represent a cut line. Each leaf is given a label which gives the name of the block mapping to that room and each interior node receives an orientation label from the set {**hor,ver**}, depending on whether the associated cut line runs horizontal or vertical. A slicing tree for the floorplan shown in Figure 7.1 is given in Figure 7.2.

Each block in the circuit $C = (B, N)$ could have more than one implementations. We will represent them by an *alternative implementation list*, which is a

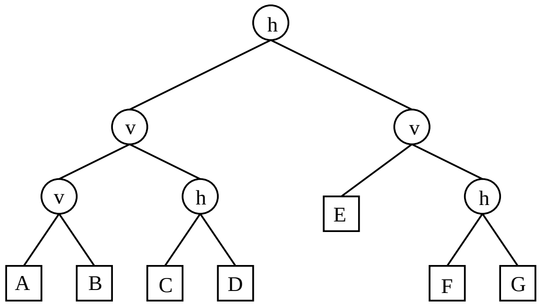

Figure 7.2: Slicing tree

finite set $L \subseteq \mathbf{R}^3$. Each element $(x, y, p) \in L$ represents an implementation whose width, height, and power dissipation are x, y, and p, respectively.

Given an implementation for each block, the corresponding room in the floorplan must be made large enough to accommodate that implementation. This, then, gives the minimum size of the enveloping rectangle. The total power consumption can be computed as the summation of power consumptions in the blocks. Formally, given a slicing tree T and an implementation for each leaf, the measures (that is, width, height, and power consumption) of the floorplan can be computed in a bottom-up fashion. Consider a node v which has two children with implementations (x_1, y_1, p_1) and (x_2, y_2, p_2). If v is labeled with **hor**, that is, the cut line is horizontal, then it will have the implementation $(\max(x_1, x_2), y_1 + y_2, p_1 + p_2)$. If v is labeled with **ver**, that is, the cut line is vertical, then it will have the implementation $(x_1 + x_2, \max(y_1, y_2), p_1 + p_2)$. The measures of the floorplan are given by the implementation at the root of the slicing tree.

We will consider the following problem.

Problem 15 (Slicing Floorplan Power and Area Optimization (SFPAO)) *Given a slicing tree T of a floorplan with m rooms, an alternative implementation list L_i for each room $i, 1 \leq i \leq m$, and a positive number $P > 0$. Determine an implementation (x_i, y_i, p_i) for each room $i, 1 \leq i \leq m$, such that $\sum_{i=1}^{m} p_i \leq P$ and* *the total area is minimized.*

Another version of the problem is to minimize the total power consumption under area constraint.

Without considering power consumption, the above problem becomes the classical *slicing floorplan area optimization* problem and has an efficient optimal algorithm which runs in polynomial time [54]. But corresponding problems for general non-slicing floorplans and for slicing floorplans without orientation labeling are both NP-hard [54, 31].

7.3 Complexity

Although Chao and Wong [8] gave an algorithm for the slicing floorplan power and area problem, it runs in exponential time and the complexity of the problem is not known. In this section, we characterize the complexity of the slicing floorplan power and area optimization (SFPAO) problem. In order to do that, the decision version of the SFPAO problem is given as follows.

Problem 16 (SFPAO-decision) *Given a slicing tree T of a floorplan with m rooms, an alternative implementation list L_i for each room $i, 1 \leq i \leq m$, and two positive numbers $A, P > 0$. Determine an implementation (x_i, y_i, p_i) for each room $i, 1 \leq i \leq m$, such that $\sum_{i=1}^{m} p_i \leq P$ and the total area is not greater than A.*

First, we define the following problem.

MCK(MULTIPLE CHOICE KNAPSACK)
INSTANCE: Finite set U, for each $u \in U$ a size $s(u) \in Z^+$ and a value $v(u) \in Z^+$, a partition of U into disjoint sets $U_1, U_2, \ldots, U_m$, and positive integer B and K.

QUESTION: Is there a choice of a unique element $u_i \in U_i$ for each $1 \le i \le m$ such that

$$\sum_{i=1}^{m} s(u_i) \le B \quad \text{and} \quad \sum_{i=1}^{m} v(u_i) \ge K?$$

For the above problem, we have the following result.

Lemma 10 *MULTIPLE CHOICE KNAPSACK is NP-complete.*

Proof: It is easy to see that MCK is in NP. We will show that MCK is NP-complete by a reduction from KNAPSACK [21]. The KNAPSACK problem is defined as follows.

INSTANCE: Finite set U, for each $u \in U$ a size $s(u) \in Z^+$ and a value $v(u) \in Z^+$, and positive integers B and K.
QUESTION: Is there a subset $U' \subseteq U$ such that

$$\sum_{u \in U'} s(u) \le B \quad \text{and} \quad \sum_{u \in U'} v(u) \ge K?$$

Given a set U with $s(u), v(u) \in Z^+$ for each $u \in U$ and two constants $B, K \in Z^+$ as an instance of KNAPSACK, we can construct an instance of MCK as follows. Here, we suppose U has m elements $u_1, u_2, \ldots, u_m$.

Let $W = U \cup \{NIL_i | 1 \le i \le m\}$ with partitions $W_i = \{u_i, NIL_i\}$. Also let $B' = B + m, K' = K + m$. For each $u \in W$, define the size $s'(u)$ and the value $v'(u)$ in W in the following way.

$$s'(u) = \begin{cases} s(u) + 1 & \text{if } u \in U \\ 1 & \text{otherwise} \end{cases}$$

$$v'(u) = \begin{cases} v(u) + 1 & \text{if } u \in U \\ 1 & \text{otherwise} \end{cases}$$

If the answer of MCK to this instance is "YES", that is, we have a choice of $w_i{}' \in W_i$, such that

$$\sum_{i=1}^{m} s'(w_i) \leq B' \quad \text{and} \quad \sum_{i=1}^{m} v'(w_i) \geq K'. \tag{7.1}$$

If we define $U' = \{w_i | w_i \in U\}$, (7.1) becomes

$$m + \sum_{u \in U'} s(u) \leq B' \quad \text{and} \quad m + \sum_{u \in U'} v(u) \geq K'.$$

That is,

$$\sum_{u \in U'} s(u) \leq B \quad \text{and} \quad \sum_{u \in U'} v(u) \geq K.$$

This means that KNAPSACK will answer "YES".

If the answer of KNAPSACK is "YES", that is, there is $U' \subseteq U$, such that

$$\sum_{u \in U'} s(u) \leq B \quad \text{and} \quad \sum_{u \in U'} v(u) \geq K. \tag{7.2}$$

Then, in MCK, we can construct a multiple choice $w_i \in W_i$ as follows.

$$w_i = \begin{cases} u_i & \text{if } u_i \in U' \\ NIL_i & \text{otherwise} \end{cases}$$

By adding m to (7.2), we have

$$\sum_{i=1}^{m} s'(w_i) \leq B' \quad \text{and} \quad \sum_{i=1}^{m} v'(w_i) \geq K'.$$

This means that MCK will answer "YES". □

Based on the above lemma, we have the following theorem.

Theorem 20 *SFPAO-decision is NP-complete.*

Proof: To show SFPAO-decision belongs to NP, a non-deterministic algorithm needs only choose an implementation for each cell and check in polynomial time whether these implements satisfy the power and area conditions. We will show SFPAO-decision is NP-complete by a reduction from MCK.

Suppose the set U with $s(u), v(u) \in Z^+$ for each $u \in U$, its partitions $U_1, U_2, \ldots, U_m$, and constants $B, K \in Z^+$ are given as an instance of MCK. We can construct an instance of SFPAO-decision as follows.

The floorplan has m rooms: $1, 2, \ldots, m$. All internal nodes of the slicing tree T have vertical orientation. (So we need not concern about the actual structure of it.) Each room i has an implementation list $L_i = \{(M - v(u), 1, s(u)) | u \in U_i\}$, where $M = \max_{u \in U}(v(u)) + 1$. And the constraints are $P = B$ and $A = m * M - K$.

If the answer of SFPAO to the instance is "YES", that is, there is an implementation $(x_i, y_i, p_i) \in L_i$ for each $1 \leq i \leq m$, such that

$$\sum_{i=1}^{m} p_i \leq P \quad \text{and} \quad Area = \max(y_i) * \sum_{i=1}^{m} x_i \leq A,$$

then, because $p_i = s(u_i), x_i = M - v(u_i), y_i = 1$, where, according to the construction of SFPAO, $u_i \in U_i$ for each $1 \leq i \leq m$, we have

$$\sum_{i=1}^{m} s(u_i) \leq P = B \quad \text{and} \quad \sum_{i=1}^{m} (M - v(u_i)) \leq A = m * M - K.$$

The last one implies $\sum_{i=1}^{m} v(u_i) \geq K$, which means that the answer of MCK will be "YES".

If the answer of MCK is "YES", that is, there is a multiple choice $u_i \in U_i$ for each $1 \leq i \leq m$, such that

$$\sum_{i=1}^{m} s(u_i) \leq B \quad \text{and} \quad \sum_{i=1}^{m} v(u_i) \geq K,$$

then, in SFPAO, we can choose the implementation $(M - v(u_i), 1, s(u_i))$ for each room $1 \leq i \leq m$ and have

$$\sum_{i=1}^{m} s(u_i) \leq P \quad \text{and} \quad \sum_{i=1}^{m} (M - v(u_i)) \leq A.$$

This means that the answer of SFPAO is "YES". ∎

7.4 Algorithms

In the slicing floorplan power and area optimization problem, the power dissipation and geometric dimensions of an implementation are real numbers. But they can be represented as integers as follows. Suppose the maximum common factors of the heights, width, and powers are δ_y, δ_x, and δ_p, respectively. An implementation (x, y, p) will then be represented by $(x/\delta_x, y/\delta_y, p/\delta_p)$, which are all integers. Similarly, A and P will also be scaled to $A/\delta_x\delta_y$ and P/δ_p, respectively. As we can see, this representation can simplify computations in computers, since integer comparison and summation are simpler than their floating point counterparts.

From the previous section, we know that the slicing floorplan power and area optimization problem is NP-hard. But as we will see later, it is *weak NP-hard*, that is, if we represent all values as integers, there exists an algorithm whose running time is a polynomial in terms of the maximum value. Such an algorithm is called a *pseudo-polynomial time algorithm.* In practice, pseudo-polynomial time algorithms can be made as efficient approximation algorithms as follows. We can "discretize" all values of the height, width, or power into a fix number of possible values. In the view of the above scheme, this means choosing δ_y, δ_x, or δ_p, and representing each height y, width x, power p by $[y/\delta_y]$, $[p/\delta_x]$, or $[p/\delta_p]$, such that $[y/\delta_y]$, $[x/\delta_x]$, or $[p/\delta_p]$ is upper bounded by fixed numbers. Since discretization loses accuracy, we look for polynomial algorithms which discretize as few parameters as possible.

The following result shows that discretizing only one geometric dimension is not enough to get a polynomial algorithm.

Theorem 21 *The slicing floorplan power and area optimization problem is still NP-hard even when the values of width (or height) are integers and upper bounded by a constant.*

Proof: This is a direct result from the proof for Theorem 20. In that proof, we reduce MCK to a special instance of SFPAO where the heights of all implementations are 1. We can similarly reduce MCK to an instance where the width of all implementations are 1. ∎

Although we could not prove the same result for power, based on our experience, we conjecture it is also true.

Conjecture 1 *The slicing floorplan power and area optimization problem is still NP-hard even when the values of power are integers and upper bounded by a constant.*

Next, we will give two algorithms for the slicing floorplan power and area optimization problem. One algorithm is due to Chao and Wong [8] and runs in polynomial time when power and one geometric dimension are discretized. The other runs in polynomial time when the two dimensions are discretized. The general idea here is simply the dynamic programming. Here, we define an implementation list L to be *non-redundant* if there is no implementations $(x_1, y_1, p_1), (x_2, y_2, p_2) \in L$ such that $x_1 \leq x_2, y_1 \leq y_2$ and $p_1 \leq p_2$. Given a slicing tree T and one implementation list for each of the leaves, a non-redundant implementation list is computed for all nodes by a post-order traversal of the tree. That is, the non-redundant implementation list of each node can be computed by combining the non-redundant lists of its children. After the non-redundant list associated with the root is obtained, we can select an optimal solution from it.

During the combination, we will partition a non-redundant list into a set of lists indexed by one parameter.

Definition 10 (Indexed Implementation List) An *X-indexed implementation list* is a non-redundant implementation list $\{(x_i, y_i, p_i) | 1 \leq i \leq n\}$ such that $x_i = x_j$ and $y_i \geq y_j$ for $1 \leq i < j \leq n$. Similarly, A *Y-indexed implementation list* is a non-redundant list such that $y_i = y_j$ and $x_i \geq x_j$ for $1 \leq i < j \leq n$, and a *P-indexed implementation list* is a non-redundant list such that $p_i = p_j$ and $x_i \geq x_j$

for $1 \leq i < j \leq n$. For an indexed implementation list L, we use $index(L)$ to denote the value of the index parameter.

Depending on which two parameters are discretized, we have different approaches to compute the non-redundant lists. Given by Chao and Wong [8] in an informal description, the first approach is suitable when power is among the two discretized parameters. Here, each non-redundant list is partitioned a set of P-indexed lists and the procedure `P-combine` described in Figure 7.3 is used to compute the non-redundant list of a node from the lists of its children. At the end of the procedure, a collection of P-indexed lists with the same index need to be merged into one. To do so, we can iteratively merge two lists into one. Each time, the pair of lists with the fewest elements are chosen and combined by the procedure `P-merge` shown in Figure 7.4. The complexity of the algorithm is given by the following theorem.

Theorem 22 ([8]) *Given an slicing floorplan with n blocks, if the values of power and height (or width) in the implementations are integers and upper bounded by P and L, then the power and area optimization problem can be solved in $O(nP^2L)$ time.*

On the other hand, if we want to get accurate result for power dissipation, both height and width need to be discretized. This is more complicated and works as follows. Suppose the non-redundant lists of the children of node v are known. If v is labeled with **ver**, the list of each child will be partitioned into a set of X-indexed lists and a procedure `X-combine` will be used to compute the list for v. If v is labeled with **hor**, the list of each child will be partitioned into a set of Y-indexed lists and a procedure `Y-combine` will be used. The procedure `X-combine` is described in Figure 7.5. Exchanging x and y in `X-combine` gives `Y-combine`.

The complexity of the above algorithm is given by the following theorem.

Theorem 23 *Given an slicing floorplan with n blocks, if the values of width and hight in the implementations are integers and upper bounded by X and Y, then the*

```
Algorithm P-combine
  for (i = 1; i ≤ m; i = i + 1) {
    for (j = 1; j ≤ l; j = j + 1) {
      Create an empty list L_v^{ij};
      if ( v has the orientation of hor) {
        a = 1; b = 1;
        while (a ≤ |L_u^i| and b ≤ |L_w^j|) {
          (x_a, y_a, p_a) = the ath element of L_u^i;
          (x_b, y_b, p_b) = the bth element of L_w^j;
          Insert the implementation (max(x_a, x_b), y_a + y_b, p_a + p_b)
          to the end of L_v^{ij};
          if (x_a ≤ x_b)
            b = b + 1;
          else a = a + 1;
        }
      }
      else {
        a = |L_u^i|; b = |L_w^j|;
        while (a ≥ 1 and b ≥ 1) {
          (x_a, y_a, p_a) = the ath element of L_u^i;
          (x_b, y_b, p_b) = the bth element of L_w^j;
          Insert the implementation (x_a + x_b, max(y_a, y_b), p_a + p_b)
          to the end of L_v^{ij};
          if (y_a ≤ y_b) {
            b = b - 1;
          else a = a - 1;
        }
      }
    }
  }
  Merge all P-indexed lists which have the same index into one;
```

Figure 7.3: Pseudo-code of P-Combine

```
Algorithm P-merge
  i = 1; j = 1;
  while (i ≤ m and j ≤ n) {
    if (x_i^1 ≤ x_j^2) {
      if (y_i^1 > y_j^2)
        Insert (x_j^2, y_j^2, p_j^2) to L;
      j = j + 1;
    } else {
      if (y_i^1 ≤ y_j^2)
        Insert (x_i^1, y_i^1, p_i^1) to L;
      i = i + 1;
    }
  }
  for (a = i; a ≤ m; a = a + 1) {
    Insert (x_a^1, y_a^1, p_a^1) to L;
  for (a = i; a ≤ n; a = 1 + 1) {
    Insert (x_a^2, y_a^2, p_a^2) to L;
```

Figure 7.4: Pseudo-code of P-merge

power and area optimization problem can be solved in $O(n \max(X,Y)^2 \min(X,Y))$ *time.*

Proof: We first show the complexity of procedure X-combine is $O(X^2Y)$. For every x, y such that $1 \leq x \leq X, 1 \leq y \leq Y$, there is at most one p such that (x,y,p) is not redundant. Therefore, for every node in the slicing tree, there are at most X of X-indexed lists and each indexed list has no more than Y elements. Hence, all instructions before the merge will take $O(X^2Y)$ time. Suppose $\{L_v^i | 1 \leq i \leq s\}$ are the X-indexed implementation lists of v. Then, we have $s \leq X$. For each $1 \leq k \leq s$, consider the set $X_k = \{(i,j) | index(L_u^i) + index(L_w^j) = index(L_v^k)\}$. They are the combinations which yield the lists of the same index. Note that no two ordered pairs in X_k has identical first component or identical second component. This means that

```
Algorithm  X-combine
  for (i = 1; i ≤ m; i = i + 1) {
    for (j = 1; j ≤ l; j = j + 1) {
      Create an empty X-indexed list L_v^{ij};
      a = 1; b = 1;
      while (a ≤ |L_u^i| and b ≤ |L_w^j|) {
        (x_a, y_a, p_a) = the ath element of L_u^i;
        (x_b, y_b, p_b) = the bth element of L_w^j;
        Insert the implementation (x_a + x_b, max(y_a, y_b), p_a + p_b)
        to the end of L_v^{ij};
        if (y_a ≤ y_b)
          b = b + 1;
        else a = a + 1;
      }
    }
  }
  Merge all X-indexed lists which have the same index into one;
```

Figure 7.5: Pseudo-code of `X-combine`

$|X_k| \leq \min(m, l) \leq X$. Therefore, to get L_v^k, the number of times when `X-merge` is called is $O(X)$. Since each `X-merge` run in $O(Y)$, the time to get each L_v^k is $O(XY)$. Therefore, the time for the merge is also $O(X^2Y)$.

Similarly, we can prove that the running time of `Y-combine` is $O(Y^2X)$.

When the list of a child is indexed by a different parameter from the requirement, the list needs to be re-partitioned. The additional cost for this is $O(XY \log XY)$, since we have at most XY implementations at each node and sorting can be used. Therefore, obtaining the list for one node needs $O(\max(X, Y)^2 \min(X, Y))$ time. For the total $n - 1$ internal nodes, the time complexity is $O(n \max(X, Y)^2 \min(X, Y))$. □

7.5 Conclusions

In this chapter, we revisited the slicing floorplan power and area optimization problem formulated by Chao and Wong [8]. We first showed that the problem is NP-hard. Therefore, it is necessary to discretize some of the parameters to get polynomial algorithms. We showed that it is not enough to discretize only height or width and conjectured this is also true for power. We also extended the power-indexed algorithm in [8] to discretize height and width.

Possible future works include proving or disproving the conjecture, extending the work to similar tree optimization problems such as tree-based technology mapping, etc.

Chapter 8

Simultaneous Routing and Buffer Insertion

8.1 Introduction

With the evolution of VLSI fabrication technology, interconnect delay, especially global interconnect delay, has become the dominant factor in deep sub-micron design. Today, in many systems, as much as 50% to 70% of clock cycle are consumed by interconnect delay [13]. As technology continues to scale down, the significance of interconnect delay is expected to further increase in the near future. Many techniques are employed to reduce interconnect delay, among them, buffer insertion has been shown to be an effective approach [6]. Because of the resistive and capacitive effects of a wire, its delay grows quadratically in terms of its length. Buffer insertion separates a long wire into short ones, thus can reduce interconnect delay significantly.

Traditionally, buffer insertion is considered in pre-layout phases and often referred to as *fanout optimization* [7, 57, 33, 53]. Buffers inserted prior to layout are then laid out in the same way as other components. The main problem is the difficulty of accurately taking into account the resistive and capacitive effects of interconnect. As a result of increasing importance of interconnect delay, recent researches took on during- or post-layout approaches where the physical information is available [61, 25, 32, 41, 42, 49]. The work in [61, 25, 32] considered buffer insertion when the routing is fixed, while [41, 42, 49] simultaneously did buffer insertion and tree construction.

During routing process, especially that for global nets, there are macro blocks placed within the area. These blocks form useful routing regions because wires are allowed to go over them. But since buffers are implemented by transistors, a buffer "over" a macro block must be actually put into that block. However, it is impossible to insert a buffer in a macro block if it is an IP (Intellectual Property) licensed from others, or it is a block such as memory which requires regular layout. Even in case when that is theoretically possible, since a re-design of the influenced block is needed, it is usually not allowed due to the design flow. Therefore, macro blocks present themselves as routing resources for wires but obstacles for buffers. We can not ignore macro blocks since buffering has become an inseparable part of interconnects. Neither can we treat them as routing blockage since they constitute a large amount of wiring area, and without them, routing can not be finished.

Currently, buffer location restrictions are taken care of only after routing stages. The work in post-layout buffer insertion [61, 25, 32] can be easily modified to avoid inserting buffers over macro blocks. But without considering macro blocks, a router may produce a route which runs so long over a block that later buffer insertion can not effectively reduce its delay. In maze routing, current approach first find a shortest path and then insert buffers on it. If there is no macro blocks, it can be proved that this two-stage "routing and buffer insertion" approach gives an optimal solution. However, with macro blocks, a shortest path no longer guarantees minimum delay. Therefore, buffer location restrictions make it necessary to combine buffer insertion and routing.

Although [41, 42, 49] did combine buffer insertion and routing tree construction, since they ignored macro blocks, they always use a shortest path to connect two points. In this chapter, we take the buffer location restrictions into consideration and solve the simultaneous maze routing and buffer insertion problem with these restrictions. Given two pins (a source and a sink) and a macro block placement which defines buffer location restrictions, we give a polynomial time exact algorithm to find a buffered path from source to sink such that the delay is minimized. When power and congestion are considered with delay minimization, the algorithm is extended to solve the problems in pseudo-polynomial times. We also discuss the issues on layer assignment for multi-layer designs.

The rest of the chapter is organized as follows. In Section 8.2, we describe the delay model and problem formulation. In Section 8.3, we give an algorithm which we called fast path algorithm to find a buffered path with minimum delay. In Section 8.4, we extend the fast path algorithm to consider power together with delay minimization. In Section 8.5, the fast path algorithm is extended to consider congestion in global routing. Section 8.6 combines layer assignment with the buffered path construction. Experimental results are reported in Section 8.7. Finally, Section 8.8 concludes the chapter.

8.2 Problem formulation

8.2.1 Delay models

As in most previous work on interconnect optimization, we adopt the Elmore delay model [17] for interconnect delay and use the common RC models for wires and buffers.

For a wire e, let l_e, c_e, r_e denote its length, capacitance, and resistance, respectively. Using c_0 and r_0 to represent the unit length capacitance and resistance, we can write

$$c_e = c_0 l_e,$$

$$r_e = r_0 l_e.$$

As shown in Figure 8.1, we use a π-model to represent a wire and a common RC-model to represent a buffer. Let C be the load capacitance of a wire and a buffer, the Elmore delay for each of them can be written down as follows.

$$D(e) = r_e \cdot (\frac{c_e}{2} + C)$$

$$D(b) = d_b + r_b \cdot C$$

The delay along a path is computed as the summation of delays on all elements in the path.

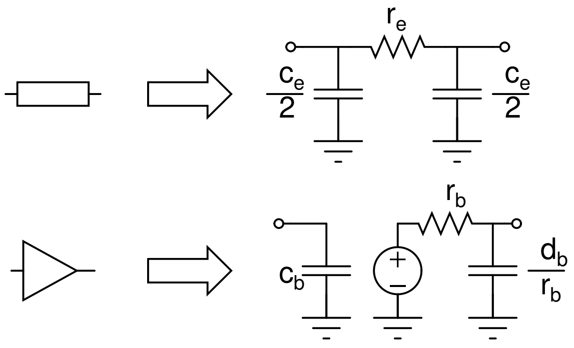

Figure 8.1: Models of a wire and a buffer

Given a wire, we can view it to be composed of different segments with different lengths. A good property of the π-model is that, by using it, the delay on a wire is invariant no matter how you segment it. This is because the delay on a wire of length l is equal to the delay on a path of two segments of length l_1 and l_2 such that $l_1 + l_2 = l$. That is,

$$r_0(l_1 + l_2)\left(c_0 \frac{l_1 + l_2}{2} + C\right) = r_0 l_1 \left(c_0 \frac{l_1}{2} + (c_0 l_2 + C)\right) + r_0 l_2 \left(c_0 \frac{l_2}{2} + C\right).$$

8.2.2 Buffered maze routing problem

In traditional maze routing, we are given a routing area where there are regions occupied by other nets and need to find a path of minimum length to connect two pins. As fabrication gets into the deep sub-micron era, buffering becomes an inseparable part of wiring. Therefore, macro blocks introduce a new type of regions which does not exist in traditional maze routing. These regions are transparent to wires but not feasible for buffer insertion. Without macro blocks, it can be shown that a shortest path followed by optimal buffer insertion gives a buffered path with minimum delay. Considering macro blocks forces us to combine together buffer insertion and routing.

As in Figure 8.2, we use dark boxes to represent the occupied regions where wiring is not allowed, and gray boxes to represent macro blocks where buffering is not allowed. Given such a routing area and the pin positions of a net, we want to find a buffered path with minimum delay, as illustrated in Figure 8.2.

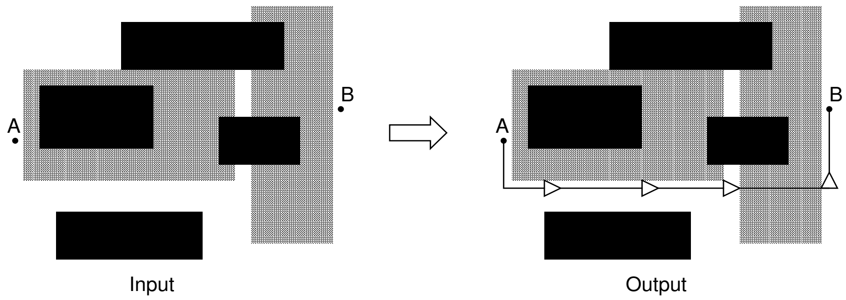

Figure 8.2: Buffered maze routing problem

To clearly illustrate the comparisons between different approaches, we will use the example in Figure 8.2 and set its dimensions as in Figure 8.3(a), where each unit has a length of $0.2mm$. We also set driver resistance at A and load capacitance at B to be 140Ω and $2fF$, and unit length capacitance and resistance to be $0.21fF/\mu m$ and $0.29\Omega/\mu m$, respectively. Furthermore, we assume there is only one type of buffer with $c_b = 2fF, r_b = 140\Omega$, and $d_b = 40ps$.

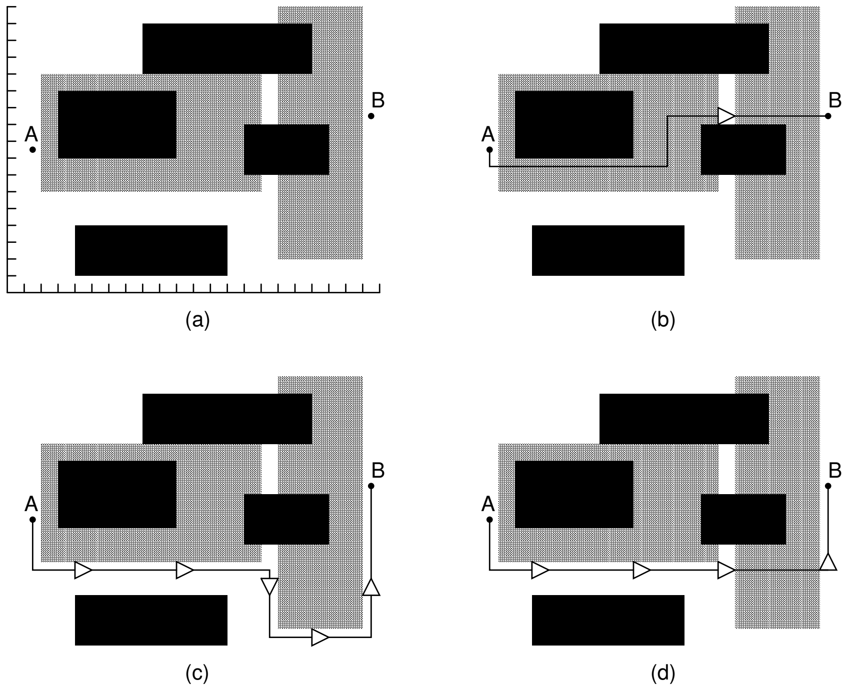

Figure 8.3: Comparison of different approaches: (a) Dimension of problem, each unit = $0.2mm$; (b) Shortest path + buffer insertion, delay = $621.81ps$; (c) Shortest path avoiding macro blocks + buffer insertion, delay = $680.62ps$; (d) Buffered maze routing considering macro blocks, delay = $521.73ps$

The first approach is the traditional one which ignores macro blocks during routing. Thus, it will find a shortest path first and then inserts buffers outside macro blocks. An optimal solution by this approach is shown in Figure 8.3(b) and its delay is $621.81ps$. As we can see, this shortest path has a very long part running over a macro block. Since the long over-block path can not be buffered and its delay is quadratic to the length, it produce a large delay. Taking this analysis into consideration, the second approach treats macro blocks as obstacles and avoid wiring over them. It is also composed of two stages: find a shortest path without running over macro blocks, then insert buffers to minimize delay. An optimal solution by this approach is given in Figure 8.3(c) and its delay is $680.62ps$. The drawback of the second approach is the waste of routing resources. Furthermore, although buffers can now be inserted anywhere, a route could become too long. On the contrary, wiring over macro blocks while wisely considering buffer insertion can give us a solution shown in Figure 8.3(d). Its delay is $521.73ps$.

In maze routing, the entire routing area is represented as a grid graph. Each edge in the graph represents a track that can accommodate exactly one wire. Edges occupied by other nets and wiring obstacles are deleted from the graph. Given such graph $G = (V, E)$, for each edge $(u, v) \in E$, we use $l(u, v)$, $c(u, v)$, and $r(u, v)$ to represent its length, capacitance, and resistance, respectively. On each node $v \in V$, there is a label $p(v) \in \{0, 1\}$ indicating whether buffer insertion is allowed on that node. The problem we need to solve can be defined as follows.

Problem 17 (Buffered Maze Routing) *Given a routing graph $G = (V, E)$, a buffer library B, and two nodes $s, t \in V$ with driver resistance R and load capacitance C, find a buffered path from s to t, that is, a sequence $(s = v_1, v_2 \ldots, v_k = t)$ and a labeling $b(v_i) \in B \cup \{0\}$ for $1 \leq i \leq k$, such that the Elmore delay is minimized and the buffer insertion is feasible, that is, $b(v_i) \in B \Rightarrow p(v_i) = 1$, for $1 \leq i \leq k$.*

8.3 Fast path algorithm

In this section, we will give an exact algorithm to solve the buffered maze routing problem. It extends Dijkstra's shortest path algorithm [15] to do a general labeling based on the Elmore delays. Since it finds a path with minimum delay instead of minimum length, we call it "fast path algorithm".

8.3.1 Basic algorithm

The key condition for Dijkstra's algorithm to work is the property that any sub-path of a shortest path is also a shortest one. If we substitute the length in the algorithm by the Elmore delay, the property no longer holds, since a path with minimum delay may have a sub-path near the sink with a longer delay but a much smaller capacitance. Therefore, during the labeling process, besides delays, our algorithm need also to record capacitances if it starts from the sink or to record resistances if it starts from the source.

To simplify the presentation, we consider only starting with the sink. Each node will be labeled with a set of capacitance-delay pairs, each represents a partial solution with that delay and capacitance. A capacitance-delay pair (c_1, d_1) is said to be *inferior* to another pair (c_2, d_2) if $c_1 \geq c_2$ and $d_1 \geq d_2$, since (c_1, d_1) in any solution can be substituted by (c_2, d_2) without increasing delay. A set of capacitance-delay pairs is said to be *non-redundant* if no one pair is inferior to any other.

The pseudo-code of the fast path algorithm is given in Figure 8.4 and can be described as follows. We use a quadruple (c, d, b, v) to represent each partial solution. Besides the capacitance c and delay d, we also record the node v and the buffer insertion on it. The algorithm uses a priority queue [14] Q to sort the partial solutions and it is initialized with the very first partial solution on sink t with capacitance C and delay 0. Each time, a partial solution with minimum delay is extracted from Q. If the capacitance of the partial solution is 0, then a path with minimum delay has already been found and we need to trace back to get the solution. If the node is not the source, it will be used to update the partial solutions on its neighbors, otherwise, we will construct a solution by multiplying the driver resistance and set its capacitance to be 0. In any case, if the current solution has no buffer inserted on the node, we will update the partial solutions of the current node by inserting each type of buffer from the library. When updating the partial solutions for a node, we check the redundancy of them and drop the inferior ones and delete them from Q.

The following theorem shows the correctness and running time of the fast

```
Algorithm Fast Path
  /* partial solution is represented as
  (capacitance, delay, buffer, node) */
  Initialize Q with only (C,0,0,t);
  while ( Q not empty) {
    Extract configuration (c,d,b,u) of minimum delay from Q;
    if c = 0
      return the optimal solution;
    if u = s {
      S = (0, d + Rc, b, u);
      Add S to solutions of u, prune, and update Q;
    }
    else
      for each (u,v) ∈ E {
        S = (c + c(u,v), d + r(u,v)(c + c(u,v)/2), 0, v);
        Add S to solutions of v, prune, and update Q;
      }
    if p(u) = 1 and b = 0
      for each b' ∈ B {
        S = (c_{b'}, d + r_{b'} c + d_{b'}, b', u);
        Add S to solutions of u, prune, and update Q;
      }
  }
```

Figure 8.4: The fast path algorithm

path algorithm.

Theorem 24 *The fast path algorithm computes a buffered path of minimum delay in $O(|B|^3|V|^2|E|\log|B||V|)$ time.*

Proof: We expand the graph $G=(V,E)$ to a new graph $G'=(V',E')$ such that

$$V' = \{(v,c)|v \in V \text{ and } c \text{ is any capacitance seen from } v\}$$

$$E' = \{((u,c_1),(v,c_2))|(u,v) \in E, c_2 = c_1 + c_0 l(u,v) \text{ or } \exists b \in B \text{ s.t. } c_2 = c_b\}.$$

For each $((u,c_1),(v,c_2)) \in E'$, we give a weight

$$w(((u,c_1),(v,c_2))) = \begin{cases} r(u,v)(\frac{1}{2}c(u,v)+c_1) + r_b(c(u,v)+c_1) + d_b & \text{if } c_2 = c_b \\ r(u,v)(\frac{1}{2}c(u,v)+c_1) & \text{otherwise} \end{cases}$$

From the construction, it is easy to see that each buffered path in G corresponds to a path in G' and the Elmore delay of the buffered path is equal to the sum of weights on its corresponding path. Therefore, a shortest path in G' gives a buffered path in G with minimum delay. Since all the weights in G' are non-negative, Dijkstra's algorithm finds a shortest path.

The fast path algorithm can be viewed as dynamically constructing G' in Dijkstra's algorithm. But it does not build all (v,c)'s for each node v. Node (v,c_1) with delay d_1 is pruned if there is another node (v,c_2) with delay d_2 such that $c_1 \geq c_2$ and $d_1 \geq d_2$. Since the sub-path from (v,c_1) to t in any path can be substituted by another sub-path from (v,c_2) to t without increasing the delay, the fast path algorithm correctly computes a buffered path with minimum delay.

To bound the running time of the algorithm, we show that, for each $v \in V$, the number of (v,c)'s not pruned is upper bounded by $|B||V|$. Consider all the paths from v to the sink t. If the first buffer is on v, the number of different capacitances seen from v is at most $|B|$. If the first buffer is on another node u, the number of different capacitances seen from v can become exponential since there can be exponential number of paths from u to v. But the number of capacitances not pruned is also bounded by $|B|$. This is because, without buffers on it, a path from u to v with minimum capacitance also has the minimum delay, thus makes all other paths pruned. Therefore, we can bound $|V'|$ by $|B||V|^2$. In the fast path algorithm, the number of `EXTRACT-MIN` operations on the priority queue is upper bounded by $|V'|$. But the dominant parts are the edge connection and the buffer insertion (the two **for** loops in the pseudo-code). The number of edge connecting operations is upper bounded by $|B||V||E|$. Since number of partial solutions kept in a node at any time is also bounded by $|B||V|$, and each of the `DELETE` and `DECREASE-KEY` operation in Q takes $O(\log |B||V|)$, the time for each edge connection is bounded by $O(|B||V| \log |B||V|)$. Thus, the total time for edge connections is $O(|B|^2|V|^2|E| \log |B||V|)$. On the other hand, the number of buffer insertion operations is upper bounded by $|B|^2|V||E|$, and the time for each operation is bounded by $|B||V| \log |B||V|$. Therefore, the total running time of the fast path algorithm is $O(|B|^3|V|^2 \log |B||V|)$. ∎

The above theorem only gives an upper bound on the running time of the fast path algorithm. In reality, the running time is much less, since many partial solutions will have the same capacitances and thus be pruned. This is confirmed by our experimental results.

Note that in the fast path algorithm, it is restricted that at most one buffer can be inserted on a given node. Therefore, cascaded buffers on one node is not allowed. But the algorithm can be generalized to include cascaded buffers as follows. First, we can show that cascaded buffers in a minimum delay path will be monotonically connected. That is,

Lemma 11 *If two buffers b_1 and b_2 are inserted on one node in a minimum delay path and b_1 drives b_2, then we must have $c_{b_1} < c_{b_2}$ and $r_{b_1} > r_{b_2}$.*

Proof: If $c_{b_1} \geq c_{b_2}$, we can simply delete b_1. Let R represent the driver resistant seen by b_1, the original delay $Rc_{b_1} + d_{b_1} + r_{b_1}c_{b_2}$ is greater than Rc_{b_2} which is the later delay. This is a contradiction. On the other hand, if $c_{b_1} < c_{b_2}$ but $r_{b_1} \leq r_{b_2}$, we can delete b_2. Let C represent the load capacitance see by b_2, the original delay $r_{b_1}c_{b_2} + d_{b_2} + r_{b_2}C$ is greater than $r_{b_1}C$ which is the later delay. Another contradiction. ∎

In order to consider cascaded buffers, we need to delete the condition for buffer insertion loop in the fast path algorithm. Based on the above property, there is also no need to consider all types of buffers if there are already buffers on the node.

In this case, we only need to consider buffer types satisfying Lemma 11. Since this modification increases the number of buffer insertion operations on each node from $|B|^2|V||E|$ to $|B||V||E|2^{|B|}$, but does not change the number of partial solutions on any node, the running time becomes $|B|^2|V|^2|E|2^{|B|}\log|B||V|$.

8.3.2 Minimum bends

In maze routing, the routing graph $G = (V, E)$ considered in previous section is a partial grid graph. Since there may exist more than one path with the minimum delay, it is preferred to find a path which has minimum bends among all paths of the minimum delay.

In order to find such a path, we need to modify the basic algorithm as follows. We extend a partial solution to include six parameters (c, d, b, v, n, o). That is, we add two more parameters: the number of bends n and the orientation o of the last edge. A partial solution with (c_1, d_1, n_1, o_1) is *inferior* to another one with (c_2, d_2, n_2, o_2) if (c_1, d_1) is inferior to (c_2, d_2), or when they are equal, has $n_1 > n_2$ or $n_1 = n_2$ and $o_1 = o_2$. The pseudo-code is similar to the basic algorithm, except that key in priority queue is the ordered pair (d, n), in stead of just delay. That is, we have $(d_1, n_1) \le (d_2, n_2)$ if and only if $d_1 < d_2$ or $d_1 = d_2$ and $n_1 \le n_2$. When a new partial solution is constructed, the number of bends is incremented if current orientation is different with previous one.

8.4 Power minimization

As we discussed in Chapter 1, power dissipation in VLSI circuits is proportional to the product of switching activity and load capacitance. In the case of buffered maze routing, since signals can not be changed, switching activities are kept the same. Minimizing power dissipation is equivalent to minimizing the total capacitance. If only power is minimized without considering delay, the solution is simply a shortest path without buffer insertion. But in practice, delay needs to be considered with power. There are two versions of the problem. One is to minimize power under the delay constraint; the other is to minimize delay under the power constraint. Since our algorithm for one can be easily modified for the other, for simplification, we only consider minimizing delay under the power constraint in the sequel.

The problem of *low power buffered maze routing* is defined the same as the problem of buffered maze routing except that a constant P is given and it is required that the total capacitance is upper bounded by P. Here, the total capacitance includes wire capacitance and buffer capacitance, and is different from the capacitance seen from a node because of the separation by buffers. We can extend the fast path algorithm to control the power dissipation. In order to do that, for each partial solution, besides the capacitance seen from the current node, we need also to remember the total capacitance in the solution. We use (c, d, p) to represent a partial solution where c represents the capacitance seen from the node, d the delay, and p the total capacitance in the solution. We say that a partial solution (c_1, d_1, p_1) is *inferior* to another partial solution (c_2, d_2, p_2) if and only if $c_1 \geq c_2$, $d_1 \geq d_2$, and $p_1 \geq p_2$. The pseudo-code is given in Figure 8.5. The algorithm runs in pseudo-polynomial time.

8.5 Application to global routing

As shown by Theorem 24, the running time of the fast path algorithm does not depend on the edge lengths or capacitance ranges. Therefore, besides maze routing, it can also be used on global routing, where edge lengths in the routing graph may be different.

However, since we only consider buffer insertions on vertices, edges of long lengths introduce limitations on the solutions. One way to deal with this is to first compute the minimum wire length between consecutive buffers and then use that length to direct the construction of the routing graph. Another way is to segment

```
Algorithm Low Power Fast Path
  /* partial solution is represented as
  (capacitance, delay, power, buffer, node) */
  Initialize Q with only (C,0,0,0,t);
  while ( Q not empty) {
    Extract configuration (c,d,p,b,u) of minimum delay from Q;
    if c = 0
      return the optimal path;
    if u = s {
      S = (0, d + R(s)c, p, b, u);
      Add S to solutions of u, prune, and update Q;
    }
    else
      for each (u,v) ∈ E
        if p + c(u,v) ≤ P {
          S = (c + c((u,v)), d + r((u,v))(c + c((u,v))/2), p + c(u,v), 0, v);
          Add S to solutions of v, prune, and update Q;
        }
    if b = 0
      for each b' ∈ B
        if p + c_b' ≤ P {
          S = (c_b', d + r_b' c + d_b', p + c_b', b', u);
          Add S to solutions of u, prune, and update Q;
        }
  }
```

Figure 8.5: The low power fast path algorithm

every long edge into a sequence of short segments [2].

In global routing, after a net is routed, the used edges are not deleted. Instead, the capacities of them are decreased and nets routed later are discouraged to use edges with smaller capacities. That is, for each $(u, v) \in E$ in $G = (V, E)$, another label $g(u, v) \in R$ is given to represent congestion cost. Given a delay constraint for a net, we want to find a buffered path with minimum congestion cost such that the delay constraint is satisfied. Formally, this can be defined as the following problem.

Problem 18 (Minimizing congestion under delay constraint) *Given a routing graph $G = (V, E)$, a buffer library B, two nodes $s, t \in V$ with the driver resistance R and load capacitance C, and a constant D, find a buffered path in G, that is, a sequence $(s = v_1, v_2 \ldots, v_k = t)$ and a labeling $b(v_i) \in B \cup \{0\}$, such that the congestion cost $\sum_{i=1}^{k-1} g(v_i, v_{i+1})$ is minimized, the Elmore delay is bounded by D, and the buffer insertion is feasible, that is, $b(v_i) \in B \Rightarrow p(v_i) = 1$, for $1 \leq i \leq k$.*

Similar to the problem of minimizing congestion under length constraint,[1] the above problem is intractable, as stated by the following theorem.

Theorem 25 *The problem of minimizing congestion under delay constraint is NP-hard.*

Proof: Given a graph $G = (V, E)$ with length $l(e)$ and congestion cost $g(e)$ for each $e \in E$ and a length bound L as a problem of minimizing congestion under length constraint, we can define a problem of minimizing congestion under delay constraint by restricting no buffer inserted anywhere, both driver resistance and load capacitance being 0 and the unit length capacitance and resistance being 1. Solving the problem with delay bound $L^2/2$ is then equal to solving the original problem. □

However, the fast path algorithm can still be extended to solve the problem of minimizing congestion under delay constraint. Here, we need to extend a partial solution to include a congestion parameter g. A solution (c_1, d_1, g_1) is *inferior* to (c_2, d_2, g_2) if and only if $c_1 \geq c_2, d_1 \geq d_2$, and $g_1 \geq g_2$. The priority queue uses g as its key. The pseudo-code of the algorithm is given in Figure 8.6.

In practice, we can choose a minimum length such that, with sufficient precision, all edge lengths can be represented as integeral multiples of the minimum length. Similarly, congestions can also be approximated by integers. We use L

[1] The problem of minimizing congestion under length constraint is the same as the SHORTEST WEIGHT-CONSTRAINED PATH problem given in Garey and Johnson [21], thus is NP-hard.

```
Algorithm Congestion Minimization
  /* partial solution is represented as
  (capacitance, delay, buffer, node, congestion) */
  Initialize Q with only (C,0,0,t,0);
  while ( Q not empty) {
    Extract solution (c,d,b,u,g) of minimum congestion from Q;
    if u = s {
      if d + Rc ≤ D
        return the optimal path;
    }
    else
      for each (u,v) ∈ E
        if d + r(u,v)(c + c(u,v)/2) < D {
          S = (c + c(u,v), d + r(u,v)(c + c(u,v)/2), 0, v, g + g((u,v)));
          Add S to the solutions of v, prune, and update Q;
        }
    if p(u) = 1 and b = 0
      for each b' ∈ B
        if d + r_b' c + d_b' < D {
          S = (c_b', d + r_b' c + d_b', b', v, g)
          Add S to the solutions of u, prune, and update Q;
        }
  }
```

Figure 8.6: The congestion minimization algorithm

to represent the summation of edge lengths in terms of the number of minimum lengths, and K to represent the summation of congestions. The running time of the congestion minimization algorithm can be bounded as follows.

Theorem 26 *The running time of the congestion minimization algorithm is bounded by* $O(K^2L^2|B|^3|E|\log KL|B||V|)$.

Proof: Similarly as the fast path algorithm, the algorithm expand a node $v \in V$ to a set of nodes (v,c,d,g). The number of nodes not pruned is upper bounded by the number of non-inferior triples (c,d,g). For any capacitance-congestion pair

(c, g), there can be only one delay value d. Since capacitance is upper bounded by $L|B|$ and congestion is bounded by K, the number of nodes derived from a node $v \in V$ is then bounded by $KL|B|$. By a similar analysis as that on the fast path algorithm, the running time is $O(K^2L^2|B|^3|E| \log KL|B||V|)$. $\square$

8.6 Multi-layer consideration

In practice, routings are usually done on multiple layers and it is possible that unit length resistances and capacitances are different on different layers. In this case, the fast path algorithm given in Section 8.3 still works. But its running time becomes pseudo-polynomial now.

If we can assume that all layers are available for each edge in the routing graph (which is usually true in global routing), layer assignment can be done with buffered routing in polynomial time. We first consider the following layer assignment problem.

Problem 19 (Layer Assignment) *Given a path $(s = v_1, v_2, \ldots, v_k = t)$ in $G = (V, E)$, and a set of m layers each with a unit length capacitance c_i and a unit length resistance r_i for $1 \le i \le m$, find a layer assignment $y(v_i, v_{i+1})$ for each edge (v_i, v_{i+1}) with $1 \le i \le k-1$ such that the Elmore delay from s to t is minimized.*

First, we observed that for two layers i and j, if $c_i \le c_j$ and $r_i \le r_j$, then there is no need to consider layer j, since we can always re-assign edges from layer j to i without increasing the delay. Here, without loss of generality, we can assume $c_i < c_j$ and $r_i > r_j$ for all $1 \le i < j \le m$. It can be shown that the optimal layer assignment has the following monotonic property.

Lemma 12 *For any two edges (v_i, v_{i+1}) and (v_j, v_{j+1}) such that $1 \le i < j \le k-1$, if y is an optimal layer assignment, then*

$$y(v_i, v_{i+1}) \ge y(v_j, v_{j+1}).$$

Proof: To simplify the presentation, we will denote (v_i, v_{i+1}) by e_i, for $1 \le i \le k-1$. Suppose for the seek of contradiction that $y(e_i) = a, y(e_j) = b$ and $a < b$, which means $c_a < c_b$ and $r_a > r_b$. Because y is an optimal layer assignment, re-assigning e_i from layer $y(e_i) = a$ to layer b could not decrease the delay. Hence, we have the following inequality.

$$\begin{aligned}\Delta D_1 &= \left(R + \sum_{h=1}^{i-1} l(e_h) r_{y(e_h)}\right) l(e_i)(c_a - c_b) + \\ &\quad \frac{1}{2} l(e_i)^2 (c_a r_a - c_b r_b) + l(e_i)(r_a - r_b)\left(C + \sum_{h=i+1}^{k-1} l(e_h) c_{y(e_h)}\right) \\ &\le 0\end{aligned}$$

where R and C are the driver resistance and load capacitance, respectively.

Similarly, re-assigning (v_j, v_{j+1}) from layer $y(e_j) = b$ to layer a could not decrease the delay, either. Therefore, we have

$$\begin{aligned}\Delta D_2 &= \left(R + \sum_{h=1}^{j-1} l(e_h) r_{y(e_h)}\right) l(e_j)(c_b - c_a) + \\ &\quad \frac{1}{2} l(e_j)^2 (c_b r_b - c_a r_a) + l(e_j)(r_b - r_a)\left(C + \sum_{h=j+1}^{k-1} l(e_h) c_{y(e_h)}\right) \\ &\le 0\end{aligned}$$

Therefore, we must have $\frac{\Delta D_1}{l(e_i)} + \frac{\Delta D_2}{l(e_j)} \le 0$. But,

$$\begin{aligned}&\frac{\Delta D_1}{l(e_i)} + \frac{\Delta D_2}{l(e_j)} \\ &= (c_b - c_a)\sum_{h=i}^{j-1} l(e_h) r_{y(e_h)} + \frac{1}{2}(l(e_i) - l(e_j))(c_a r_a - c_b r_b) + (r_a - r_b)\sum_{h=i+1}^{j} l(e_h) c_{y(e_h)} \\ &\ge (c_b - c_a) l(e_i) r_a + \frac{1}{2} l(e_i)(c_a r_a - c_b r_b) + \frac{1}{2} l(e_j)(c_b r_b - c_a r_a) + (r_a - r_b) l(e_j) c_b \\ &> (c_b - c_a) l(e_i) r_a + \frac{1}{2} l(e_i)(c_a r_a - c_b r_a) + \frac{1}{2} l(e_j)(c_b r_b - c_b r_a) + (r_a - r_b) l(e_j) c_b \\ &= \frac{1}{2}(c_b - c_a) l(e_i) r_a + \frac{1}{2}(r_a - r_b) l(e_j) c_b \\ &> 0\end{aligned}$$

This is a contradiction. ∎

Based on the above property, a candidate configuration can be generated by separating the $k-1$ edges into m groups, and this can be done by selecting $m-1$ positions for the separators. Therefore, the total number of candidate configurations are $\binom{m+k-2}{m-1}$. This also gives an algorithm to find the optimal layer assignment. In practice, we have only a small constant number of layers. Therefore, the number of configurations is a low degree polynomial in terms of the number of edges.

Combining layer assignment together with routing, we have the following problem.

Problem 20 (Routing with Layer Assignment) *Given a routing graph $G = (V, E)$, a set of m layers each has a different parasitic characteristics, and two nodes $s, t \in V$ with the driver resistance R and load capacitance C, find a path in G from s to t, that is, a sequence $(s = v_1, v_2 \ldots, v_k = t)$ and a layer assignment for the edges, such that the Elmore delay from s to t is minimized.*

Based on the property given in Lemma 12, no matter which route we use, the layer assignment must be monotonic. Therefore, there are $m-1$ nodes (some of them may be the same) on the route working as separators, and the edges which are not separated are assigned to the same layer. We must also observe that, when such $m-1$ separators are fixed, the route is also fixed, since the path between neighboring separators must form a shortest path. Therefore, the total number of candidate configurations is upper bounded by $O(|V|^{m-1})$.

Now we can combine layer assignment with buffered path construction in the fast path algorithm to solve the following problem.

Problem 21 (Buffered Path with Layer Assignment) *Given a routing graph $G = (V, E)$, a set of m layers with different parasitic characteristics, a buffer library B, and two nodes $s, t \in V$ with driver resistance R and load capacitance C, find a buffered path with layer assignment from s to t, that is, a sequence*

$(s = v_1, v_2 \ldots, v_k = t)$, a labeling $b(v_i) \in B \cup \{0\}$ for $1 \leq i \leq k$, and another labeling $1 \leq y(v_i, v_{i+1}) \leq m$ for $1 \leq i \leq k-1$, such that the Elmore delay is minimized and the buffer insertion is feasible, that is, $b(v_i) \in B \Rightarrow p(v_i) = 1$, for $1 \leq i \leq k$.

In order to include layer assignment in the fast path algorithm, we add another parameter y to represent the layer assigned to the edge driven by the current node. When adding a new edge, we only assign it to the layers which observe the property in Lemma 12. The pseudo-code is given in Figure 8.7.

```
Algorithm Fast Path with Layer Assignment
  /* partial solution is represented as
  (capacitance, delay, buffer, node, layer) */
  Initialize Q with only (C,0,0,t,1);
  while ( Q not empty) {
    Extract solution (c,d,b,u,y) of minimum delay from Q;
    if u = s and c = 0
      return the optimal path;
    if u = s {
      S = (0, d + Rc, 0, u, y);
      Add S to the solution of u, prune, and update Q;
    }
    else
      for each (u,v) ∈ E and each y ≤ i ≤ m {
        S = (c + c_i l(u,v), d + r_i l(u,v)(c + c_i l(u,v)/2), 0, v, i);
        Add S to the solutions of v, prune, and update Q;
      }
    if p(u) = 1 and b = 0
      for each b' ∈ B {
        S = (c_b', d + r_b' c + d_b', b', u, 1)
        Add S to the solutions of u, prune, and update Q;
      }
  }
```

Figure 8.7: The fast path with layer assignment algorithm

Based on the above discussion, the running time of the fast path with layer assignment algorithm is given by the following theorem.

Theorem 27 *The running time of the fast path with layer assignment algorithm is upper bounded by* $O(m|B|^3|V|^{2(m-1)} \log |B||V|)$.

8.7 Experimental results

We implement the fast path algorithm in C++ on a Soloris/PC platform. The LEDA (Library of Efficient Data types and Algorithms) [36] is used to implement the priority queue which is actually based on Fibonacci heap [14].

The technology parameters used in the experiments are based on the $0.18\mu m$ technology predicted by the National Technology Roadmap of Semiconductors [4] (also see [12]). That is, we have unit length capacitance $c_0 = 0.21fF/\mu m$ and unit length resistance $r_0 = 0.29\Omega/\mu m$. We also set the driver resistance and load capacitance to $1K\Omega$ and $1fF$, respectively. The buffer library has only one type of buffer with $r_b = 1K\Omega, c_b = 1fF$ and $d_b = 57ps$.

Each problem has a rectangular routing area with height and width ranging from $10mm$ to $17mm$. We randomly place some blocks in that area and also randomly generate two pins in it. The grid we use for maze routing has $0.1mm$ as its unit length. On a machine with a 180MHz Pentium Pro processor and 64M memory, the running time ranges from 14 seconds to 51 seconds. As a comparison, we also implement the traditional method of first finding a shortest path and then inserting buffers. In Table 8.1, we first report the dimension of each problem and the number of macro blocks in it. Then the delays from source to sink and the wire lengths by the two methods are reported. To make the comparison clear, we also give the percentage improvements by fast path algorithm over the traditional approach.

From these experiments, we can see that, with only a minor increase on wire length, the fast path algorithm can dramatically reduce the delay.

Table 8.1: Experimental Results

Tech: $C = 1fF, R = 1K\Omega, c_0 = 0.21fF/\mu m, r_0 = 0.29\Omega/\mu m, c_b = 1fF, r_b = 1K\Omega, d_b = 57ps$								
Data			Shortest+Buffer		Fast Path		Improvements	
name	size(mm)	#blk	delay(ps)	len(mm)	delay(ps)	len(mm)	delay	len
ran1	15.0x15.5	6	7414.47	26.0	3946.88	27.2	87.86%	-4.0%
ran2	16.2x11.5	8	4632.86	24.6	3564.04	24.6	29.99%	0%
ran3	14.0x11.9	9	5221.35	23.2	3946.88	27.2	32.29%	-17.0%
ran4	11.7x13.8	6	8885.30	37.0	6057.30	39.8	46.69%	-7.0%
ran5	12.5x12.3	5	5820.45	33.4	4880.73	33.4	19.25%	0%
ran6	12.8x13.3	6	11662.15	36.6	5473.80	37.4	113.05%	-2.0%
ran7	11.9x11.9	3	6388.17	33.0	4800.30	33.0	33.08%	0%
ran8	10.3x12.5	8	9660.66	36.6	5356.80	36.6	80.34%	0%
ran9	13.9x12.2	7	14038.36	37.0	5435.40	37.0	158.28%	0%
ran10	10.8x10.6	10	9023.52	31.8	4908.38	32.6	83.84%	-2.0%

8.8 Conclusions

In this chapter, we considered buffer location restrictions introduced by macro blocks. These restrictions make it necessary to combine buffer insertion with routing. For delay minimization, we give a polynomial time exact algorithm to solve the problem of simultaneous maze routing and buffer insertion with buffer location restrictions. If power and congestion are considered with delay minimization, the algorithm is extended to give pseudo-polynomial time algorithms. We also discuss the issues on layer assignment for multi-layer designs. The experimental results show that our approach is effective.

Since connecting two points is a basic operation in tree construction, our algorithm can also be used in any routing tree construction for multi-terminal nets. For example, we can easily embed the fast path algorithm into [49] to construct a buffered routing tree with buffer location restrictions.

Bibliography

[1] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network Flows: Theory, Algorithms, and Application.* Prentice Hall, 1993.

[2] C. Alpert and A. Devgan. Wire segmenting for improved buffer insertion. In *Proc. of the Design Automation Conf.*, pages 588–593, 1997.

[3] R. Anderson and J. Setubal. Maxflow. `ftp://dimacs.rutgers.edu/pub/netflow/submit/setubal`, 1992.

[4] Semiconductor Industry Association. National technology roadmap for semiconductors, 1994.

[5] Semiconductor Industry Association. National technology roadmap for semiconductors, 1997.

[6] H. B. Bakoglu. *Circuits, Interconnections, and Packaging for VLSI.* Addison-Wesley, 1990.

[7] C. L. Berman, J. L. Carter, and K. F. Day. The fanout problem: From theory to practice. In C. L. Seitz, editor, *Advanced Research in VLSI: Proceedings of the 1989 Decennial Caltech Conference*, pages 69–99, March 1989.

[8] K.-Y. Chao and D. F. Wong. Floorplan design with low power considera-

tions. *International Journal of High Speed Electronics and System*, 7(2):305–322, 1996.

[9] K. Chaudhary, A. Onozawa, and E. S. Kuh. A spacing algorithm for performance enhancement and cross-talk reduction. In *Proc. Intl. Conf. on Computer-Aided Design*, pages 697–702, Santa Clara, CA, November 1993.

[10] C.-P. Chen, H. Zhou, and D. F. Wong. Optimal non-uniform wire-sizing under the Elmore delay model. In *Proc. Intl. Conf. on Computer-Aided Design*, San Jose, CA, 1996.

[11] H. H. Chen and C. K. Wong. Wiring and crosstalk avoidance in multi-chip module design. In *Proc. Custom Integrated Circuits Conf.*, pages 28.6.1–28.6.4, Boston, MA, May 1992.

[12] J. Cong, L. He, K.-Y. Khoo, C.-K. Koh, and Z. Pan. Interconnect design for deep submicron ICs. In *Proc. Intl. Conf. on Computer-Aided Design*, pages 478–485, 1997.

[13] J. Cong, L. He, C. K. Koh, and P. Madden. Performance optimization of VLSI interconnect layout. *INTEGRATION, the VLSI journal*, 21:1–94, 1996.

[14] T. H. Cormen, C. E. Leiserson, and R. H. Rivest. *Introduction to Algorithms.* MIT Press, 1989.

[15] E. W. Dijkstra. A note on two problems in connection with graphs. *Numerische Math.*, 1:269–271, 1959.

[16] Danny Dolev, Kevin Karplus, Alan Siegel, Alex Strong, and Jeffrey D. Ullman. Optimal wiring between rectangles. In *Conference Proceedings of the Thirteenth Annual ACM Symposium on Theory of Computation*, pages 312–317, Milwaukee, Wisconsin, 11–13 May 1981.

[17] W. C. Elmore. The transient response of dampled linear networks with particular regard to wide-band amplifiers. *Journal of Applied Physics*, 19(1):55–63, January 1948.

[18] J. R. Ford and D. R. Fulkerson. *Flows in Networks.* Princeton University Press, 1962.

[19] T. Gao and C. L. Liu. Minimum crosstalk channel routing. In *Proc. Intl. Conf. on Computer-Aided Design*, pages 692–696, Santa Clara, CA, November 1993.

[20] T. Gao and C. L. Liu. Minimum crosstalk switchbox routing. In *Proc. Intl. Conf. on Computer-Aided Design*, pages 610–615, San Jose, CA, November 1994.

[21] M. R. Garey and D. S. Johnson. *Computers and Intractability.* W. H. Freeman and Co., 1979.

[22] A. Goel, K. Sajid, H. Zhou, A. Aziz, and V. Singhal. BDD based procedures for a theory of equality with uninterpreted functions. In *Proc. of the Computer Aided Verification Conf.*, July 1998.

[23] D. A. Huffman. A method for the construction of minimum redundancy codes. In *Proceedings of the IRE*, volume 40, pages 1098–1101, sept 1952.

[24] F. K. Hwang, D. S. Richards, and P. Winter. *The Steiner Tree Problem.* North-Holland, 1992.

[25] L. N. Kannan, P. R. Suaris, and H.-G. Fang. A methodology and algorithms for post-placement delay optimization. In *Proc. of the Design Automation Conf.*, pages 327–332, 1994.

[26] D. A. Kirkpatrick and A. L. Sangiovanni-Vincentelli. Techniques for crosstalk avoidance in the physical design of high-performance digital systems. In *Proc.*

Intl. Conf. on Computer-Aided Design, pages 616–619, San Jose, CA, November 1994.

[27] D. A. Kirkpatrick and A. L. Sangiovanni-Vincentelli. Digital sensitivity: Predicting signal interaction using functional analysis. In *Proc. Intl. Conf. on Computer-Aided Design*, San Jose, CA, 1996.

[28] D. E. Knuth. *Fundamental Algorithms*, volume 1 of *The Art of Computer Programming.* Addison-Wesley, second edition, 1973.

[29] L. Lauther. A min-cut placement algorithm for general cell assemblies based on a graph representation. *J. Digital Syst.*, 4(1):21–34, 1980.

[30] T. Lengauer. *Combinatorial Algorithms for Integrated Circuit Layout.* John Willey & Sons, 1990.

[31] T. Lengauer and R. Muller. The complexity of floorplanning based on binary circuit partitions. Technical Report 46, Department of Mathematics and Computer Science, University of Paderborn, 1988.

[32] J. Lillis, C. K. Cheng, and T. T. Lin. Optimal and efficient buffer insertion and wire sizing. In *Proc. Custom Integrated Circuits Conf.*, pages 259–262, 1995.

[33] S. Lin and M. Marek-Sadowska. A fast and efficient algorithm for determining fanout trees in large networks. In *Proc. European Conf. on Design Automation*, pages 539–544, 1990.

[34] I.-M. Liu, T.-H. Liu, H. Zhou, and A. Aziz. Simultaneous PTL buffer insertion and sizing for minimizing Elmore delay. In *Workshop Notes of Intl. Workshop on Logic Synthesis*, Lake Tahoe, CA, June 1998.

[35] Errol L. Lloyd and S. S. Ravi. One-layer routing without component constraints. *Journal of Computer and System Sciences*, 28(3):420–438, June 1984.

[36] Kurt Mehlhorn, Stefan Naher, and Christian Uhrig. LEDA: Library of Efficient Data types and Algorithms. `http://www.mpi-sb.mpg.de/LEDA/lead.html`, 1998.

[37] K. Mikami and Tabuchi K. A computer program for optimal routing of printed circuit connectors. In *IFIPS Proc.*, volume H47, pages 1475–1478, 1968.

[38] R. Murgai, R. K. Brayton, and A. Sangiovanni-Vincentelli. Decomposition of logic functions for minimum transition activity. In *Proceeding of the International Workshop on Low Power Design*, 1994.

[39] U. Narayanan and C. L. Liu. Low power logic synthesis for XOR based circuits. In *Proc. Intl. Conf. on Computer-Aided Design*, pages 570–574, 1997.

[40] T. Ohtsuki. Maze-running and line-search algorithms. In T. Ohtsuki, editor, *Layout Design and Verification.* Elsevier Publishers B.V. (North-Holland), 1986.

[41] T. Okamoto and J. Cong. Interconnect layout optimization by simultaneous Steiner tree construction and buffer insertion. In *Proc. ACM/SIGDA Physical Design Workshop*, pages 1–6, 1996.

[42] T. Okamoto and J. Cong. Buffered Steiner tree construction with wire sizing for interconnect layout optimization. In *Proc. Intl. Conf. on Computer-Aided Design*, pages 44–49, 1996.

[43] A. Ostrowski. Sur quelques applications des fonctions convexes et concaves au sens de I. Schur (offert en homage á P. Montel). *J. Math. Pures Appl.*, 31:253–292, 1952. (In French).

[44] R. H. J. M. Otten. Efficient floorplan optimization. In *Proc. Intl. Conf. on Computer Design*, pages 499–502, 1983.

[45] D. S. Parker. Conditions for optimality of the Huffman algorithm. *siam-computing*, 9(3):470–489, August 1980.

[46] M. Pedram. Power minimization in IC design: Principles and application. *ACM Transactions on Design Automation of Electronic Systems*, January 1996.

[47] I. S. Reed. A class of multiple-error-correcting codes and the decoding scheme. *IRE Transactions on Information Theory*, PGIT-4:38–49, 1954.

[48] T. Sakurai and K. Tamaru. Simple formulas for two and three dimensional capacitance. *IEEE Trans. Electron Devices*, 1993.

[49] A. H. Salek, J. Lou, and M. Pedram. A simultaneous routing tree construction and fanout optimization algorithm. In *Proc. Intl. Conf. on Computer-Aided Design*, 1998.

[50] T. Sasao. *Logic Synthesis and Optimization.* Kluwer Academic Publishers, 1983.

[51] I. Schur. Über eine klass von mittelbildungen mit anwendungen auf die determinantentheorie. *Sitzungsber. Berl. Math. Ges.*, 22:9–20, 1923. (In German).

[52] A. A. Shen, A. Ghosh, S. Devadas, and K. Keutzer. On average power dissipation and random pattern testability of CMOS combinational logic networks. In *Proc. Intl. Conf. on Computer-Aided Design*, November 1992.

[53] K. J. Singh and A. Sangiovanni-Vincentelli. A heuristic algorithm for the fanout problem. In *Proc. of the Design Automation Conf.*, pages 357–360, 1990.

[54] L. Stockmeyer. Optimal orientations of cells in slicing floorplan designs. *Information and Control*, 59:91–101, 1983.

[55] S. Thakur, K.-Y. Chao, and D. F. Wong. An optimal layer assignment algorithm for minimizing crosstalk for three layer VHV channel routing. In *Proc. Intl. Symposium on Circuits and Systems*, pages I–207–210, April 1995.

[56] Vivek Tiwari, Pranav Ashar, and Sharad Malik. Technology mapping for low power. In *Proc. of the Design Automation Conf.*, 1993.

[57] Hervé J. Touati. *Performance-Oriented Technology Mapping.* PhD thesis, University of California Berkeley, Electronics Research Laboratory, College of Engineering, University of California, Berkeley, CA 94720, November 1990. Memorandum No. UCB/ERL M90/109.

[58] C.-C. Tsai and M. Marek-Sadowska. Multilevel logic synthesis for arithmetic functions. In *Proc. of the Design Automation Conf.*, pages 68–73, 1996.

[59] C.-Y. Tsui, M. Pedram, and A. M. Despain. Technology decomposition and mapping targeting low power dissipation. In *Proc. of the Design Automation Conf.*, 1993.

[60] C.-Y. Tsui, M. Pedram, and A. M. Despain. Power efficient technology decomposition and mapping under an extended power consumption model. *IEEE Transactions on Computer Aided Design*, pages 1110–1122, 1994.

[61] L. P. P. P. van Ginneken. Buffer placement in distributed RC-tree networks for minimal Elmore delay. In *Proc. Intl. Symposium on Circuits and Systems*, pages 865–868, 1990.

[62] D. F. Wong and C. L. Liu. A new algorithm for floorplan design. In *Proc. of the Design Automation Conf.*, 1986.

[63] Q. Wu, M. Pedram, and X. Wu. A note on the relationship between signal probability and switching activity. In *Proc. Asian and South Pacific Design Automation Conference*, 1997.

[64] T. Xue, E. S. Kuh, and D. Wang. Post global routing crosstalk risk estimation and reduction. In *Proc. Intl. Conf. on Computer-Aided Design*, 1996.

[65] F. Y. Young and D. F. Wong. How good are slicing floorplans. *INTEGRATION, the VLSI journal*, nov 1997.

[66] H. Zhou and A. Aziz. Buffer minimization in pass transistor logic. In *Workshop Notes of Intl. Workshop on Logic Synthesis*, Lake Tahoe, CA, June 1998.

[67] H. Zhou, V. Singhal, and A. Aziz. How powerful is retiming? In *Workshop Notes of Intl. Workshop on Logic Synthesis*, Lake Tahoe, CA, June 1998.

[68] H. Zhou and D. F. Wong. An optimal algorithm for river routing with crosstalk constraints. In *Proc. Intl. Conf. on Computer-Aided Design*, San Jose, CA, 1996.

[69] H. Zhou and D. F. Wong. Crosstalk constrained maze routing based on Lagrangian relaxation. In *Proc. Intl. Conf. on Computer Design*, Austin, TX, 1997.

[70] H. Zhou and D. F. Wong. An exact gate decomposition algorithm for low-power technology mapping. In *Proc. Intl. Conf. on Computer-Aided Design*, San Jose, CA, 1997.

[71] H. Zhou and D. F. Wong. Global routing with crosstalk constraints. In *Proc. of the Design Automation Conf.*, June 1998.

[72] H. Zhou and D. F. Wong. Optimal river routing with crosstalk constraints. *ACM Transactions on Design Automation of Electronic Systems*, July 1998.

[73] Hai Zhou, D. F. Wong, I-Min Liu, and Adnan Aziz. Simultaneous routing and buffer insertion with restrictions on buffer locations. Proc. of the Design Automation Conf., 1999.

Vita

Hai Zhou was born in Bijie, Guizhou, China on December 2, 1969, the only son of Guanghan Zhou and Yiwen Wang. With the 2nd highest score in the college entrance examination in Guizhou province, he entered Tsinghua University in Beijing, China in 1987. He received the degrees of Bachelor of Engineering and Master of Engineering in 1992 and 1994, respectively, both from Tsinghua University. In September 1994 he entered the Graduate School of The University of Texas at Austin.

He has published more than 10 technical papers in leading journals and conferences in the area of VLSI design automation. When he was pursuing his doctoral degree, he was employed as a teaching and research assistant in the Department of Computer Sciences of The University of Texas as Austin. He also worked as a CAD engineer from June 1997 to August 1997 at Avant! Corporation, Fremont, California.

Permanent Address: c/o Ms. Qin Zhou
28 Eagledance Drive
Scarborough, ON M1V 4J9
Canada

This dissertation was typeset with $\LaTeX\,2_{\varepsilon}$[2] by the author.

[2]The macros used in formatting this dissertation were written by Dinesh Das, Department of Computer Sciences, The University of Texas at Austin.