PERFORMANCE DRIVEN PLACEMENT AND ROUTING ALGORITHMS

BY

TONG GAO

B.S. University of Illinois at Urbana-Champaign, 1988
M.S. University of Illinois at Urbana-Champaign, 1992

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 1994

Urbana, Illinois

ABSTRACT

As technology advances, the effect of intra-module delays become less significant, while the effect of inter-module interconnection delays become more prominent. Also, as power dissipation becomes an important issue in VLSI design, it is desirable for the signals to arrive at the inputs of the modules at the same time in order to reduce the number of unwanted transient switches. To minimize the signal arrival times at the primary output pins and the signal skews at the inputs of the modules, we developed a net-based performance driven placement algorithm and a path-based performance driven placement algorithm. As chip architectures become more specific (e.g., FPGA), it is important to consider the physical design information during logic design steps. Therefore, we developed a placement driven technology mapping algorithm for FPGA circuits. Finally, as technology advances, interconnection wires are placed in closer proximity and circuits operate at higher frequencies. Consequently, reduction in crosstalks between interconnection wires becomes an important consideration in VLSI design. To satisfy the crosstalk constraints and to minimize the total crosstalk among all the nets in a design, we developed a track permutation algorithm for gridded channel routing problems. We also developed a wire segment assignment algorithm for both channel routing problems and switchbox routing problems. The experimental results indicate that our algorithms are very promising.

To my parents
Shunquan Gao and Xiangjin Yao

ACKNOWLEDGEMENTS

I am deeply indebted to Professor C. L. Liu for his guidance, kindness, wisdom, and friendship. I also wish to thank the members of my committee, Professors Farid Najm, Bill Kubitz, Sung Mo Kang, and Pravin Vaidya for their guidance. I wish to thank especially Professor Pravin Vaidya for his help on my research.

My special thanks to President Stanley Ikenberry of University of Illinois for inviting me here.

I wish to thank the present and past members of Professor Liu's research group, Cheng Chang, Ki-Seok Chung, Jason Cong, Sai-Keung Dong, Don Gillies, Ran Libeskind-Hadas, Nany Hasan, Taewhan Kim, Anmol Mathur, Unni Krishnan Narayanan, Chaeryung Park, Peichen Pan, Srilata Raman, Prashant Saxena, Yachyang Sun, and Too-Seng Tia.

I wish to thank Gayanne Carpenter, Barb Cicone, Becky Green, Kathy Johnson, Don Ray, and Teresa Schurg in the Department of Computer Science for making my staying here easier and more enjoyable.

My profound thanks go to my family for their constant support, encouragement, and love: My mother, Xianjin Yao O'malley; my step father, James O'malley; my sisters, Yao Gao and Hui Gao; my brother in law, Xiangli Chen; and my girl friend, Jane Wang.

TABLE OF CONTENTS

CHAPTER

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

Introduction

The design process of a VLSI circuit is usually divided into several steps, namely, high level synthesis, logic synthesis, physical design, and transistor and mask level design. In this thesis, we shall mostly focus on the physical design process. The physical design process determines where the circuit modules should be placed and where the interconnection wires should be laid out. It has become very important for the physical design algorithms to satisfy certain performance requirements such as the timing delay requirements and the crosstalk requirements. Since placement and routing are the two most important steps in the physical design process, it is important to develop good performance driven placement and routing algorithms.

During the placement process, the exact locations of the electric components (modules) in the circuit are determined so that certain objectives such as the total wire length and placement area are minimized. As speed becomes one of the most important considerations in circuit design, timing delay becomes an important issue during placement. Delays of the components in a circuit determines the signal arrival times at the primary output pins, which in turn, determine the maximum speed at which the circuit can operate. Circuit delay consists of the internal delays of the modules and the delays of the nets that interconnect the modules. As technology advances, the effect of inter-module interconnection delays becomes more prominent in circuit delay. Since the interconnection delays are primary determined by the placement of the modules, it is important to study *performance driven placement algorithms* which are capable of producing placements that satisfy the *timing delay requirements* at the primary output pins.

While circuits become more complex and operate at higher frequencies, power dissipation increases. Large power dissipation is undesirable from the point of view of both reliability and cost. In CMOS circuits, power dissipation occurs during signal switchings. The time difference between the latest arriving signal and the earliest arriving signal at a module is called the *signal skew* at the module. Signal skews causes transient switchings, which in turn, cause power dissipation (Figure 1.1). To reduce the number of unwanted transient signal switchings (thus reducing the power dissipation), it is important for the input signals to arrive at the modules more or less at the same time. We will refer to the requirements at the inputs of the modules as *signal skew requirements*. The signal skews depend on the signal arrival times at the modules, which in turn, depends on the placement of the modules. Therefore, it is important for the placement algorithm to satisfy the signal skew requirements.

For ASIC designs, it is important to have short turn around times. At the same time, it is important to keep the design cost low. Field Programmable Gate Arrays (FPGAs) can be easily programmed and reprogrammed, and therefore, are natural choices for rapid and low cost prototyping. In a typical FPGA design, logic design process is usually carried out independent

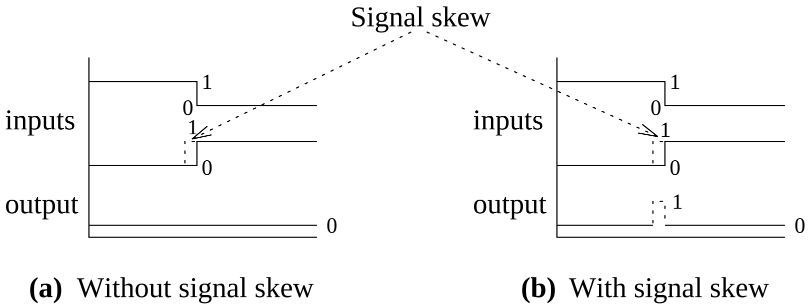


Figure 1.1. Transient switching in an AND gate caused by signal skew

of physical design process. However, due to the specific FPGA architectures, a circuit generated without considering the physical design process might be hard to place and route, and lead to unsatisfactory design. Therefore, it is important to consider the physical design information during logic design process. Since the routing of the interconnection wires depends very much on the placement of the modules, it is most important to consider the placement information during the logic design process.

During the routing process, interconnections between modules are made so that certain objectives such as routing area and total wire length are minimized. As fabrication technology advances, minimum feature sizes for VLSI chips continue to decrease, and devices and interconnection wires (nets) are placed in closer proximity in order to reduce interconnection delay and routing area. As interconnection delay and circuit switching delay become smaller, circuits will be able to operate at higher frequencies. Consequently, reducing crosstalks between nets becomes an important consideration in VLSI design. Crosstalks are mostly caused by coupling capacitance between nets, which in turn, are determined by the relative positions of the nets. Since the relative positions of the nets are determined by the routing of the nets, it is very important to have routing algorithms which are capable of producing routings that satisfy the crosstalk constraints.

To address these new issues in physical design process, we developed several new performance driven placement and routing algorithms. This thesis mainly consists of two parts, namely, the placement part and the routing part. In Chapter 2, we shall study the performance driven placement problems with timing delay requirements and signal skew requirements. In Chapter 3, we shall present a net-based performance driven placement algorithm which is capable of satisfying both the timing delay requirements and signal skew requirements [24,25,23]. In Chapter 4, we shall present a path-based performance driven placement algorithm which is capable of satisfying the timing delay requirements [19]. In Chapter 5, we shall study the technology mapping problems for circuits with Xilinx's FPGA architecture. We shall then present a new placement driven technology mapping algorithm [18]. In Chapter 6, we shall study the routing problems with crosstalk constraints. In Chapter 7, we shall present a channel routing algorithm which is capable of generating routing solutions that satisfy crosstalk constraints [21,20]. In Chapter 8, we shall present a switchbox routing algorithm which is capable of generating routing solutions that satisfy crosstalk constraints [22]. The switchbox routing algorithm presented in Chapter 8 can also be used to solve channel routing problems with crosstalk constraints.

CHAPTER 2

Performance Driven Placement Problems

2.1 Introduction

Performance driven placement problems are difficult due to the incompatibility between performance requirements and physical placement requirements. Since the delay between two points in a circuit depends on all the paths between them, performance requirements are path oriented. Since net wire lengths are determined by the relative positions of the modules, placement requirements are usually net oriented. Due to the incompatibility between performance requirements and placement requirements, conventional area driven (or wire length driven) physical placement algorithms cannot be easily adapted to solve the performance driven placement problems.

In this chapter, we shall first study the existing performance driven placement algorithms. We shall then formally define the performance driven placement problem to be solved. We shall also present one way to translate the timing delay requirements and signal skew requirements into a set of linear constraints on net delays. To get a better understanding of the performance driven placement problem, we shall also interpret the placement problem in the n-dimensional space where n is the number of nets in the circuit.

2.2 Existing Algorithms

The performance driven placement problem has been studied extensively in the past [2,11,13,14, 24,25,19,27,29,30,32,34,37,38,39,43,44,45,51,50,40,42,41,51]. According to the basic approaches taken, most previous work can be divided into two major categories, namely, net-based approach and path-based approach.

In a net-based algorithm [2,11,14,27,32,34,37,45,51,50], timing delay requirements are first translated into physical design constraints, such as net constraints. The placement algorithm then tries to generate a placement under the guidance of the net constraints. In some net-based algorithms [14,2,32,45], weights are assigned to nets to reflect the timing criticalities of the paths that the nets are on. The timing criticalities are determined either dynamically or statically by a timing verifier. A placement solution is then generated according to the weights of the nets. In some other net-based placement algorithms [11,27,33,34,37,51,50], a timing analysis of the circuit is first carried out to generate a set of upper-bounds on net wire lengths.

The upper-bounds are chosen so that the signal arrival times at the primary output pins are guaranteed to be no larger than the required signal arrival times if the wire length of each net in a physical placement does not exceed its corresponding upper-bound. These upper-bounds are then used to guide the placement process. Utilizing upper-bounds resolves the incompatibility between timing delay requirements and placement requirements. Since most existing placement algorithms are capable of handling net constraints well, utilizing upper-bounds enables us to use these placement algorithms to solve performance driven placement problems. Most upper-bound approaches, however, do not have a way to deal with other performance issues, such as signal skew in the circuits. Additionally, upper-bound approaches are often criticized for over-constraining the placement problem. Since timing delay requirements are path oriented, representing timing delay requirements by net constraints usually over-constrains the problem. Such over-constraints, in turn, may exclude many of the possible placement solutions from consideration. Furthermore, since the choice of the upper-bounds is not unique, upper-bounds that are not chosen properly would also over-constrain the problem adversely. In [27], Hauge presented an algorithm which computes an arbitrary set of upper-bounds. In [33], linear programming algorithm is used reduce the randomness of the upper-bounds generated. In [37,50], linear and quadratic programming formulations are presented to compute the upper-bounds. However, as is pointed out in [50], the sizes of the linear and quadratic programming problems may be unreasonably large, even for relatively small circuits. Moreover, due to the formulations, many nets may have values of 0 for the upper-bounds on their wire lengths. Since no two modules may overlap, no net may have length of zero. Therefore, the upper-bounds generated by these linear and quadratic programming algorithms could be severely over-constraining.

Obviously, to avoid the difficulty of over-constraint, the upper-bounds should be chosen in such a way that the placement algorithm will be provided with maximum flexibility.

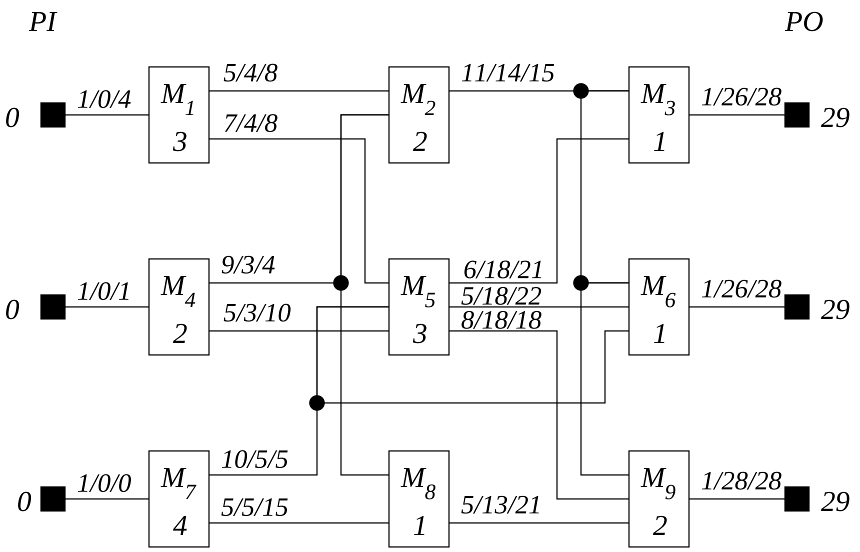
In a path-based algorithm [13,19,29,30,38,39], timing delay requirements are considered explicitly, and the placement algorithm tries to satisfy both timing delay requirements and placement requirements simultaneously. One typical path-based approach is to formulate the placement problem as an optimization problem with both timing delay requirements and placement requirements as constraints [29,30,38,39]. Since timing delay requirements are used explicitly in the computation, little path information is lost in the process of generating a placement. However, it is difficult to satisfy all the placement requirements during the optimization process, and the solution to the optimization problem usually contain overlapping modules. Therefore, the placement solution need to be repeatedly perturbed until all the placement constraints are satisfied. After each perturbation, the optimization problem need to be solved again. To avoid everything from clustering into one point in the solution generated by the optimization algorithm, it is usually required that the I/O pins have pre-determined positions. This early commitment of I/O pins to their positions without any information about the possible positions of the modules could lead to bad placement results. Also, the perturbation process destroys the optimality of the solution generated by the optimization process. Additionally, the size of the optimization problem is usually very large, which could result in exceedingly long computation times even for problems of moderate size.

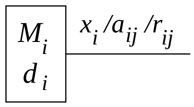
2.3 Problem Formulation

Input to the placement algorithm consists of circuit specification and timing specification. A circuit consists of a set of m modules $M_1, M_2, \ldots, M_m$, a set of n nets $N = \{N_1, N_2, \ldots, N_n\}$, and a set of p primary input pins (*PI*s) and primary output pins (*PO*s) $R_1, R_2, \ldots, R_p$. Depending on the technology used, different physical placement constraints exist. For gate-array technology, all modules have the same shape and size and are to be fitted into the slots pre-determined on the placement area. For macro-cell technology, modules have different shapes and sizes, and the dimensions $w_i \times h_i$ of M_i for all the modules are given in the circuit specification. The placement area has dimensions $W \times H$ and is given in the circuit specification. A net is an interconnection of a set of pins on the boundaries of the modules, and net wire length is estimated by the half perimeter of the bounding rectangle of the pins in the net. As will be shown in the experimental results in Chapter 3, half perimeter is a simple yet effective estimation for wire length and interconnection delay. For simplicity in computation, we assume that all pins are located at the centers of the modules and a net is viewed as an interconnection of the modules where the pins reside. Depending on the placement algorithm, the locations of the primary input pins and the primary output pins can be either pre-determined or determined together with the modules. Denote the jth pin of module M_i as P_{ij}. Timing specification of a circuit includes signal arrival times at the *PI*s (the signal arrival time at *PI* R_i is denoted a_i), the required signal arrival times at the *PO*s (the required signal arrival time at *PO* R_k is denoted r_k), internal delay d_i of module M_i, and the maximum allowable signal skew C_i at module M_i for all the modules. The net wire length of net N_i is denoted x_i, and the signal skew at module M_i is denoted c_i. Delay in a net is proportional to the lumped capacitance of the net; the lumped capacitance, in turn, is proportional to the net wire length. To simplify our notation, we consider net wire length and net delay as equivalent.

A *legitimate placement* of a circuit is a physical placement of the modules in the placement area that satisfies certain placement requirements. For a gate-array circuit, a legitimate placement of the circuit is an assignment of the modules to the slots in the placement area so that no two modules are assigned to the same slot. For a macro-cell circuit, a legitimate placement of the circuit is a placement of the modules inside of the placement area so that no two modules overlap. The objective of our placement algorithm is to find a legitimate placement of the circuit that satisfies all the timing delay requirements and signal skew requirements. Such a placement is called a *feasible placement.* Another objective of the placement algorithm is to minimize the total wire length.

Timing analysis of a circuit has been studied in detail in [28]. Consider a combinational circuit (Figure 2.1) without feedback loops. The arrival time a_{ij} of the signal at pin P_{ij} and the arrival time a_k at primary output pin R_k are computed as a function of net wire length by a forward traversal of the circuit starting from the *PI*s. If a module receives input signals from different nets, the values of the outputs of the module will not be correct until the latest arriving signal arrives. Accordingly, the signal arrival time at the output of the module is decided by the latest arriving signal. Therefore, for a module that receives input signals from different nets, we assume that the early arriving signals wait for the latest arriving signal. For example, module M_2 in Figure 2.1 has 2 inputs to it, one with signal arrival time 8 and the other with signal arrival time 11. The signal arrival time at M_2 is taken as 11, and the signal arrival time at the output of M_2 is computed as $11 + 2 = 13$. The required signal arrival time r_{ij} of the signal





M_i = module number

d_i =module delay

x_i =net delay

a_{ij} =arrival time at module output j

r_{ij} = required time at module output j

Figure 2.1. Signal arrival times and required signal arrival times

at pin P_{ij} and the required signal arrival time r_k at the primary input pin R_k are computed as a function of net wire lengths by a backward traversal of the circuit starting from the POs. If a net has multiple outputs, the earliest required signal arrival time among them is used to compute the required signal arrival times of the input signals to the modules. For example, module M_4 in Figure 2.1 has 2 outputs, one with required signal arrival time 3 and the other with required signal arrival time 9. Therefore, the required signal arrival time at M_4 is taken as 3 and the required signal arrival time at the input of M_4 is computed as $3 - 2 = 1$. The slack s_{ij} at the jth pin of module M_i is the difference between between r_{ij} and a_{ij}. The slack s_k at the primary output pin R_k is the difference between r_k and a_k (Figure 2.1). The signal skew c_i at module M_i is the difference between the arrival times of the latest arriving signal and the earliest arriving signal at module M_i. Obviously, a given placement satisfies the timing delay requirements and signal skew requirements if and only if $a_k \leq r_k$ for all POs R_k and $c_i \leq C_i$ for all modules M_i. The condition $a_k \leq r_k$ is referred to as a *timing delay requirement*, and the condition $c_i \leq C_i$ is referred to as a *signal skew requirement.*

2.4 Timing Delay Requirements and Signal Skew Requirements

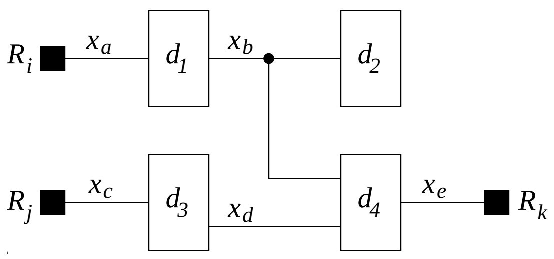


Figure 2.2. Generating timing constraints for timing delay requirements

For a path between a primary input pin R_i and a primary output pin R_k on which each module has only one input (Figure 2.2a), timing delay requirement can be expressed as a linear constraint:

$$\sum_{all\ M_u} d_u + \sum_{all\ N_v} x_v \leq r_k - a_i \tag{2.1}$$

where the M_u's and N_v's are the modules and nets on the path from R_i to R_k, respectively. For a path on which modules receive multiple inputs, timing delay requirements can be expressed by a set of linear constraints in a similar way. For example, for the path in Figure 2.2b, because early input signals must wait for the latest input signal, the timing delay requirement $a_k \leq r_k$ for the primary output pin R_k can be expressed as

$$\max(a_i + x_a + d_1 + x_b, a_j + x_c + d_3 + x_d) + d_4 + x_e \leq r_k \tag{2.2}$$

which in turn is equivalent to

$$x_a + d_1 + x_b + d_4 + x_e \leq r_k - a_i \tag{2.3}$$

and

$$x_c + d_3 + x_d + d_4 + x_e \leq r_k - a_j \tag{2.4}$$

Notice that the left hand side of Equation 2.3 is the sum of the delays of the modules and nets on the path from primary input pin R_i to primary output pin R_k, and the left hand side of Equation 2.4 is the sum of the delays of the modules and nets on the path from primary input pin R_j to primary output pin R_k. In general, timing delay requirements can be expressed by a set of linear (timing) constraints of the form in Equation 2.1 for all possible paths from the primary input pins to the primary output pins. Since there may be a large number of paths between all primary input pins and primary output pins, the number of linear constraints that specify the timing delay requirements may also be large.

With respect to the signal skew, if module M_i has only one input signal, the signal skew c_i is always equal to 0. If M_i has $q > 1$ input pins $P_{i1}, P_{i2}, \ldots, P_{iq}$, the signal skew requirement $c_i \leq C_i$ can be expressed as a set of linear constraints

$$a_{ij} - a_{il} \leq C_i \tag{2.5}$$

for all input pins P_{ij} and P_{il}, $j \neq l$, of module M_i. Equation 2.5 is equivalent to

$$(a_s + \sum_{all\ M_u} d_u + \sum_{all N_v} x_v) - (a_t + \sum_{all\ M_w} d_w + \sum_{all\ N_y} x_y) \leq C_i \tag{2.6}$$

for all the possible paths from any primary input pin R_s to pin P_{ij} and all possible paths from any primary input pin R_t to pin P_{il}. M_u's and N_v's are the modules and nets on the path from R_s to P_{ij}, respectively. In Equation 2.6, M_w's and N_y's are the modules and nets on the path from R_t to P_{il}, respectively. Again, the number of linear constraints in Equation 2.6 that specify the signal skew requirements may be large.

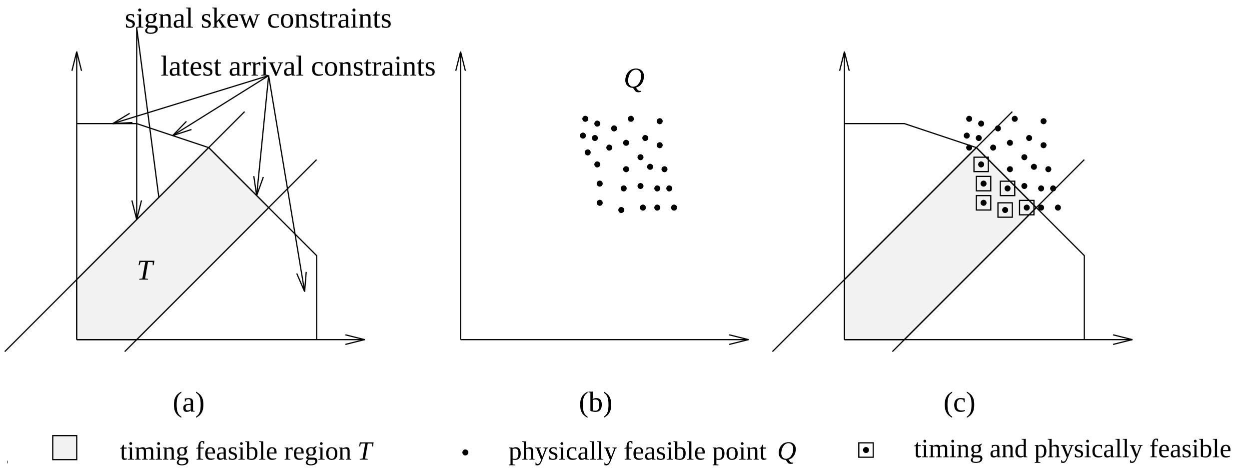


Figure 2.3. Timing feasibility, physically feasibility, and solution to the problem

2.5 Solutions to the Performance Driven Placement Problem

Since both timing delay constraints and signal skew constraints can be expressed by linear constraints, it is possible to express the timing requirements as a set of linear constraints on the net wire lengths. Since no net can have negative wire length, we call the set of *positive* wire lengths $X = \{x_1, x_2, \ldots, x_n\}$ for the nets in $N = \{N_1, N_2, \ldots, N_n\}$ *timing feasible* if it satisfies the linear constraints in Equations 2.1 and 2.6, and call it *physically feasible* if there is a physical placement with net wire lengths equal to that in X. A set of wire lengths X can be represented by a point in n dimensional space. Let T denote the set of all timing feasible points which will be referred to as the *timing feasible region* (Figure 2.3a), and Q denote the set of all physically feasible points which will be referred to as the *physically feasible region* (Figure 2.3b). Therefore, a solution to the placement problem should have its net wire lengths corresponding to some point in $T \cap Q$ (Figure 2.3c). Notice that T is a convex polytope bounded by linear constraints in Equations 2.1 and 2.6, whereas Q can be either a continuous region or a set of discrete points, depending on what kind of technology is used for the placement algorithm. Since the distribution of physically feasible points is determined by the placement problem, the distribution of the physically feasible points is in general very hard to predict.

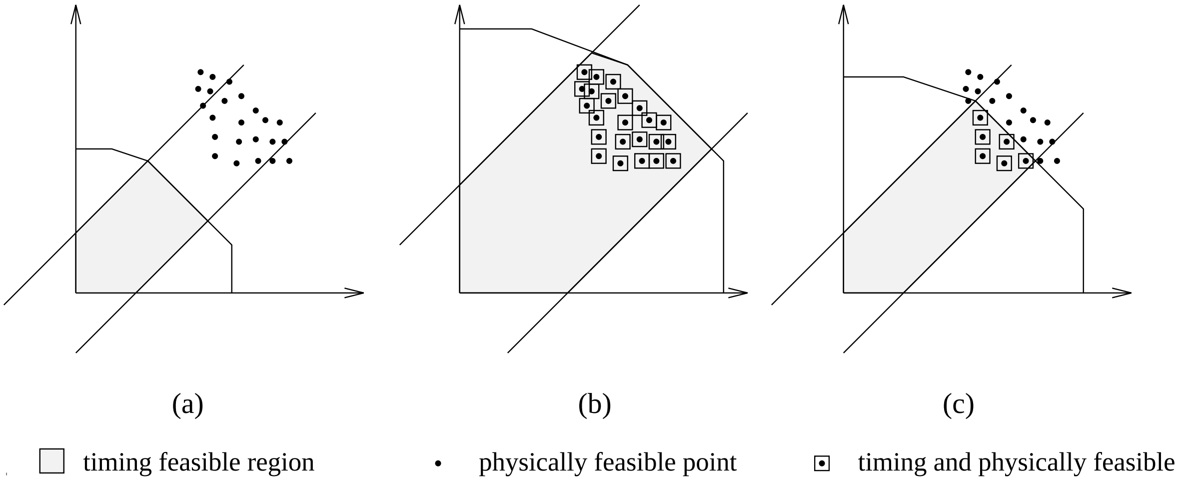


Figure 2.4. Relationship between T, Q, and $T \cap Q$

For a given placement problem, one of the following situations will arise. If the required signal arrival time at the PO pins or the maximum allowable signal skew at the modules is very small, the timing feasible region T might not intersect with the physically feasible region Q at all (Figure 2.4a). In this case, there is no solution to the placement problem. If the required signal arrival time at the PO pins and the maximum allowable signal skews at the modules are very large, T may enclose all the points in Q. In this case, any physical placement is a solution to the placement problem (Figure 2.4b). Between these two extremes, in reality, it is usually the case that some physically feasible points lie within the timing feasible region (Figure 2.4c). In this case, a good placement algorithm should lead to a placement corresponding to such a physically feasible point.

CHAPTER 3

A Net-Based Performance Driven Placement Algorithm

3.1 Introduction

In a net-based performance driven placement algorithm, timing delay requirements are first translated into physical design constraints, such as net constraints. The placement algorithm then tries to generate a placement under the guidance of the net constraints. In this chapter, we shall present a net-based performance driven placement algorithm for gate-array circuits, Xilinx's FPGA circuits, and macro-cell circuits [24,25,23]. Our placement algorithm is capable of satisfying both the timing delay requirements and the signal skew requirements.

As was mentioned in Chapter 2, upper-bound approaches are often criticized for over-constraining the placement problem. To avoid the difficulty of over-constraint, the upper-bounds should be chosen in such a way that the placement algorithm will be provided with maximum flexibility. Since the upper-bounds may be over-constraining, if one set of upper-bounds does not lead to a physical placement that satisfies the timing delay requirements, it is still possible that there are other sets of upper-bounds that will lead to such a placement. An effective way to handle the situation is to have a procedure for modifying a set of upper-bounds so that a feasible placement solution can be discovered. Our algorithm consists of the following steps:

(1) Using convex programming techniques, deduce from the timing delay requirements a set of upper-bounds on net wire lengths. Also, deduce from the signal skew requirements a set of lower-bounds on net wire lengths.

(2) Use a modified min-cut placement algorithm to obtain a physical placement of the modules. The min-cut algorithm is guided by the upper- and lower-bounds obtained in step 1.

(3) Check whether all timing requirements are satisfied in the placement obtained in step 2.

(4) If all timing requirements are satisfied, the placement obtained in step 2 is accepted. If not, modify the set of upper- and lower-bounds obtained in step 1 and repeat step 2 and 3.

By utilizing a powerful convex programming algorithm, we are able to generate upper-bounds that provide the placement algorithm with maximum flexibility in a reasonable amount of time. Incorporating lower-bounds enables us to generate placements that satisfy signal skew requirements as well as timing delay requirements. Finally, by introducing additional lower-bounds on the minimum separations between modules, we are able to solve the placement problem for both macro-cell and gate-array circuits.

In Section 3.2, we shall describe how to compute the initial set of upper- and lower-bounds. In Section3.3, we shall describe the modified min-cut placement algorithm. In Section 3.4, we shall describe the procedure for modifying the upper- and lower-bounds. In Section 3.5, we shall present some experimental results.

3.2 Computing the Bounds

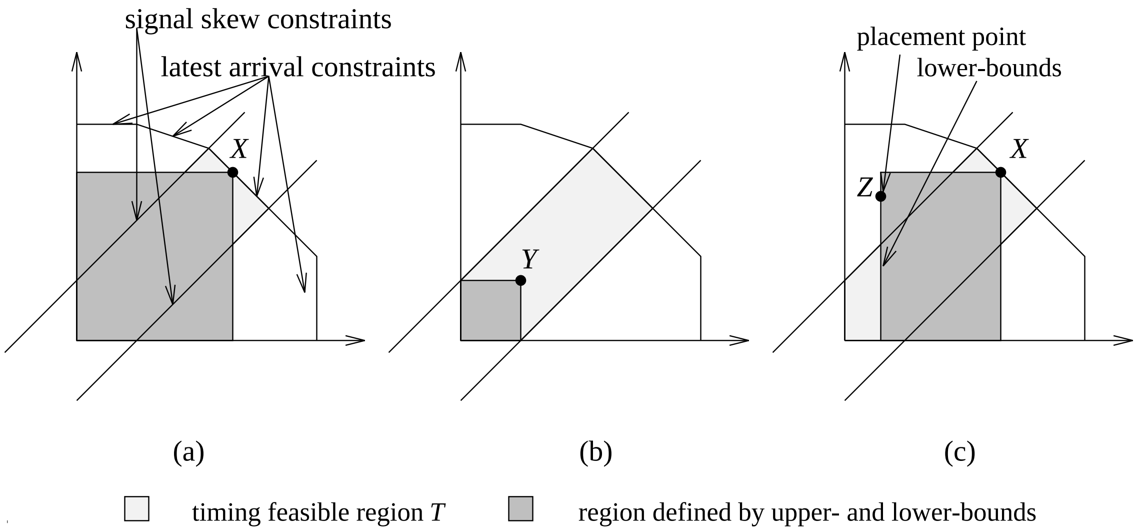


Figure 3.1. Upper- and lower-bounds

Let $X = \{x_1', x_2', \ldots, x_n'\}$ be a timing feasible point. Consider the box bounded by the following hyper planes, denoted by B_X:

$$x_1 = x_1',\ x_2 = x_2',\ \ldots,\ x_n = x_n' \tag{3.1}$$

$$x_1 = 0,\ x_2 = 0,\ \ldots,\ x_n = 0 \tag{3.2}$$

Since the linear constraints in Equation 2.1 have only positive coefficients, any point in B_X satisfies the constraints in Equation 2.1 (Figure 3.1a). In contract, since some of the coefficients in Equation 2.6 are negative, in general, not all points in B_X will satisfy the constraints in Equation 2.6 (Figure 3.1a). Therefore, for a timing feasible point X, if the coordinates of X are used as the upper-bounds on net wire lengths, a physical placement with net wire lengths satisfying the upper-bounds is guaranteed to satisfy the timing delay requirements, but is not guaranteed to satisfy the signal skew requirements. To assure that all signal skew constraints are also satisfied, we can choose a set of upper-bounds such as the point Y in Figure 3.1b. However, the box B_Y usually is very small and excludes a large portion of the timing feasible region. Thus, upper-bounds generated in this way tend to severely over-constrain the placement problem. An alternative way to satisfy the signal skew requirements is to introduce lower-bounds on some of the net wire lengths to force the placement into the timing feasible region (Figure 3.1c). This alternative approach provides the placement algorithm with more flexibility and has been demonstrated to be quite effective.

Looking at all the timing feasible points, it is intuitively clear that we should choose the set of upper-bounds such that the box B_X will enclose a large number of physically feasible points. However, as noted above, the distribution of the physically feasible points is hard to predict. Moreover, at the beginning of the placement process, there is no information available about where the physically feasible points will be. Intuitively, in the case that very little is known about the distribution of the physically feasible points, the larger the volume of B_X is, the better the chance that more physically feasible points will be enclosed. Consequently, to give the algorithm maximum flexibility during placement, we should choose the set of upper-bounds as the timing feasible point X that gives the largest box B_X. To get such a point, we need to solve an optimization problem that maximize the product $x_1 x_2 \cdots x_n$ subject to the linear constraints in Equations 2.1 and 2.6. We use the more general objective function

$$\alpha_1 \log x_1 + \alpha_2 \log x_2 + \cdots + \alpha_n \log x_n \tag{3.3}$$

where $\alpha_1, \alpha_2, \ldots, \alpha_n$ are constants determined by our algorithm. The constants $\alpha_1, \alpha_2, \ldots, \alpha_n$ give us the flexibility to choose different sets of upper-bounds by adjusting the values of these constants. Initially, all the α_i's are set to 1, and maximizing the objective function

$$\alpha_1 \log x_1 + \alpha_2 \log x_2 + \cdots + \alpha_n \log x_n \tag{3.4}$$

is equivalent to maximizing the function $x_1 x_2 \cdots x_n$.

Even though a set of upper-bounds can be obtained by solving the convex programming problem defined above, the large number of linear constraints in the system prevents us from using an ordinary convex programming algorithm. In order to deal with the large number of constraints, we integrate the upper-bound problem into the convex programming algorithm presented in [46]. One novel feature of this convex programming algorithm is that the computational complexity of the algorithm does not depend on the number of constraints, but rather depends on the number of variables in the system, provided that if some constraints are violated, one such constraint can be found in linear time. This algorithm is faster than previously known algorithms for such problems as, for example, the ellipsoid algorithm [31], by a factor proportional to the number of variables. The details of the convex programming algorithm can be found in [46]. Here, we shall merely summarize the algorithm and explain how it is used to solve the upper-bounds problem.

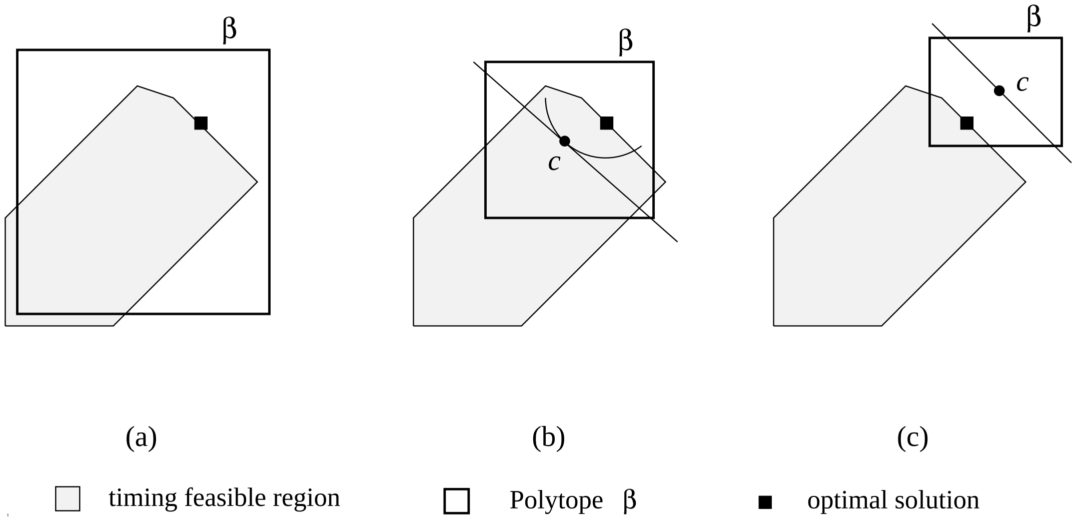


Figure 3.2. Solving the convex programming problem

The convex programming algorithm starts with a initial bounding box β that encloses the optimal solution (Figure 3.2a), and iteratively generates cutting planes that cut off the part of β that does not contain the optimal solution (Figures 3.2b, 3.2c). The cutting plane is generated as follows. First, the center (see [46] for definition) of the polytope β is computed. Then the feasibility of the center is checked. If the center is feasible (i.e., it satisfies all timing constraints), then a cutting plane tangent to the surface of the objective function at the center is generated (Figure 3.2b). If the center is outside of the feasible region, then at least one of the (timing) constraints is violated. For this case, one such constraint is found, and a cutting plane passing through the center and parallel to the violated constraint is generated (Figure 3.2c).

The cutting plane divides the polytope β into two parts. Since both the objective function and the feasible region are convex, the part that contains the optimal solution can be determined easily. This part will be used as the new polytope β for the next iteration. The cutting process continues until the polytope β is small enough that a good estimation of the optimal solution can be made. Obviously, the smaller the initial β is, the faster the algorithm will generate a solution. Since each cutting plane goes through the center of the polytope β, its volume is guaranteed to shrink rapidly [46]. In the remainder of this section, we shall describe more details of the convex programming algorithm.

First, we shall describe how to compute the initial bounding box. We can simply use 0 and the half perimeter of the placement area as the estimate of minimum and maximum possible wire length, respectively, for all the nets. However, in both gate-array and macro-cell technology, a careful analysis of the circuit will reduce the size of the initial bounding box and speed up the computation of the upper-bounds substantially.

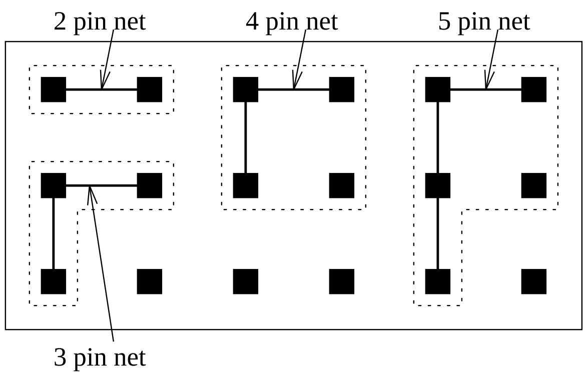


Figure 3.3. Minimum possible net wire length

In a gate array circuit, because modules are to be placed in different slots, net wire length is larger than the distance between the center of two adjacent slots. The minimum possible wire length of a net can be computed as a function of the net size. For example, assuming the distance between the center of two adjacent slots to be 1, for the nets that connect 2, 3, 4, and 5 modules, respectively, the minimum possible net wire length are 1, 2, 2, 3, respectively (Figure 3.3). In general, the minimum possible wire length for a net connecting q modules can be computed by

$$\lceil\sqrt{q}\rceil + \lceil\frac{q}{\lceil\sqrt{q}\rceil}\rceil - 2 \tag{3.5}$$

This formula computes the smallest possible half parameter among all the rectangles which enclose at least q modules. The correctness of the formula can be easily proven by mathematical induction.

For a macro-cell circuit, let $L_j = \max(w_j, h_j)$ for module M_j. Since modules cannot overlap in a placement, the net wire length for net N_i must be larger than or equal to $\frac{L_q+L_r}{2}$ where M_q and M_r are the two modules with the largest L values among the modules connected by net N_i.

The size of the initial bounding box can be further reduced by a timing analysis of the circuit using the minimum possible wire lengths computed above as the wire lengths for the nets. Since the wire lengths for all the nets in the circuit are known, the signal arrival times at all the module output pins and the required signal arrival times at all the module input pins can be computed as described in Section 2. The difference between the latest required signal arrival time at net N_i's outputs (inputs to modules) and the arrival time at net N_i's input (outputs to modules) gives the maximum possible wire length V_i for net N_i. Since no nets can have wire length longer than the half perimeter of the placement area, the maximum possible wire length for net N_i is be $\min(W + H, V_i)$.

To check the feasibility of the center computed by the convex programming algorithm, we let the coordinates of the center be the wire lengths of the nets, and run a timing analysis to calculate the arrival times at the input pins of the modules and the arrival times at the primary output pins. For any module M_i in the circuit, if the time difference between the earliest arrival and the latest arrival at its input pins is larger than C_i, a signal skew constraint is violated and the center is not feasible. Also, if any primary output signal arrives at its primary output pin later than the corresponding required signal arrival time, a timing delay constraint is violated and the center is not feasible.

In cases where the center is not feasible, a violated constraint is found as follows: During the process of computing arrival times (while checking the feasibility of the center), the nets with the latest and the earliest arrival times are recorded for each module. If the center is not feasible, there must be at least one latest arrival time constraint in Equation 2.1 or one signal skew constraint in Equation 2.6 that is violated. In the former case, a critical path corresponding to the violated constraint can be found by (following the net that connects to a pin with the latest arrival time and) backtracking from the PO for which signal arrival time is later than signal required signal arrival time. In the latter case, a critical path corresponding to the violated signal skew constraint at module M_i can be found by (following the net that connects to a pin with the latest arrival time and) backtracking from the two signal input pins of M_i with the the latest and the earliest signal arrival time, respectively.

3.3 The Placement Algorithm

After obtaining a set of upper-bounds, we use a modified min-cut placement algorithm to generate a placement that satisfies as many upper- and lower-bounds as possible. As will be discussed later, the placement algorithm also tries t the same time to minimize the total wire length. The basic idea of our algorithm is to modify the original min-cut algorithm presented in [15,16] so that, at each level of the partitioning hierarchy, net wire lengths are estimated, and the estimated values are then used to guide the partitioning process. In our algorithm, before refining a partition, the estimated wire lengths of the nets in the partition are first compared with the corresponding upper- and lower-bounds. Different weights are assigned to the nets according to the outcome of the comparison. The partitioning algorithm then tries to partition the modules so that the sum of the weights in the cut set is minimized. Depending on whether the gate-array or macro-cell technology is used, different steps are taken during the placement process. Section 4.1 describes the placement algorithm for gate-array circuits, and Section 4.2 describes the placement algorithm for macro-cell circuits.

3.3.1 The Placement Algorithm for Gate-Array Circuits

At any intermediate level of the partitioning hierarchy, although the exact wire lengths are not known, the maximum and minimum possible wire length for each net can be computed. Because a module could eventually be placed in any slot within a given partition, the maximum possible wire length max_i for net N_i is the half perimeter of the smallest bounding rectangle that contains all the slots in the partitions where the modules of net N_i are located. The minimum possible wire length min_i for net N_i is the half perimeter of the smallest bounding rectangle that contains at least as many slots in any partition as the number of modules connected to N_i in that partition. Figure 3.4 shows the maximum and the minimum possible wire length for the net connecting the modules M_a, M_b, M_c and M_d, which are shown in their corresponding partitions.

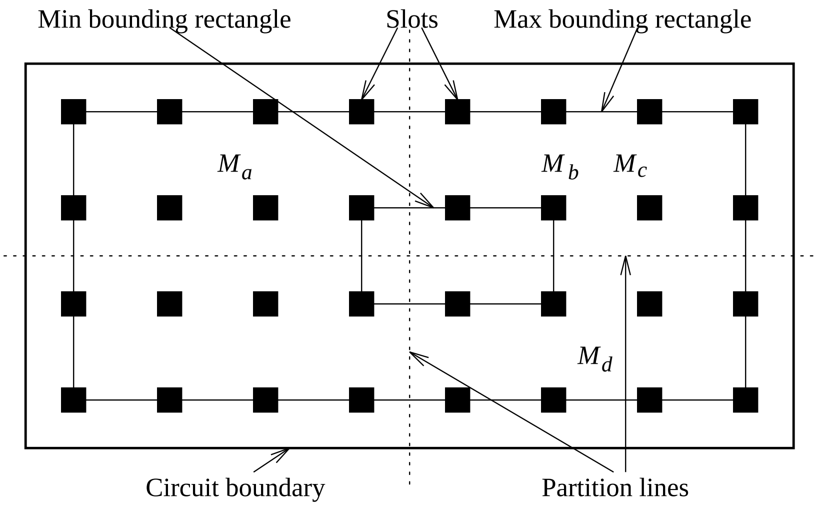


Figure 3.4. Estimation of the length of the net connecting M_a, M_b, M_c, and M_d

Note that at higher levels of the partitioning hierarchy, the maximum and minimum possible wire length for a net may differ quite substantially. In other words, the estimation of the net wire length is not precise at higher levels of the partitioning hierarchy. As we go down the levels in the partitioning hierarchy, the estimation becomes more and more precise. The inaccuracy at higher levels of the partitioning hierarchy gives the placement algorithm more freedom in placing the modules at these levels.

The weight f_i for signal net N_i is computed as the sum of f_{i1} and f_{i2} where f_{i1} and f_{i2} are computed as follows: Recall that x_i denotes the upper-bound on the net wire length of N_i. If $max_i \leq x_i$, then the upper-bound x_i will be satisfied no matter how the partitions are further refined. Therefore, f_{i1} is set to 0. If $min_i > x_i$, then the wire length will exceed x_i no matter how the partitions are further refined. Therefore, f_{i1} is set to a predetermined maximum weight $maxcrit$. if $max_i \leq x_i \leq min_i$, f_{i1} is computed according to the formula

$$round(\frac{max_i - x_i}{max_i - min_i} \times maxcrit) \qquad (3.6)$$

where $round(x)$ represents the smallest integer greater than x if the fractional part of x is greater than 0.5, and represents the largest integer less than or equal to x otherwise. For a net with an upper-bound close to the maximum possible wire length, there is a high likelihood that the upper-bound will be satisfied by the placement algorithm. Therefore, a smaller weight is computed for this net. For a net with upper-bound close to the minimum possible wire length, it is harder to satisfy the upper-bound. Therefore, a larger weight is computed for the net. Similarly, for the lower-bound l_i on the net wire length of N_i, if $min_i \geq l_i$, then f_{i2} is set to 0.

If $max_i \leq l_i$, then f_{i2} is set to $-maxcrit$. If $min_i \leq l_i \leq max_i$, f_{i2} is computed according to the formula

$$-round(\frac{l_i - min_i}{max_i - min_i} \times maxcrit) \qquad (3.7)$$

The gain in moving net N_i out of the cut set is computed as $w_1 + w_2 \times f_i$, where w_1 and w_2 are user chosen parameters. If we choose $w_1 = 1$ and $w_2 = 0$, the algorithm becomes the classical min-cut algorithm, which tries to minimize the size of the cut set. If we choose $w_1 = 0$ and $w_2 = 1$, the algorithm tries to satisfy as many upper- and lower-bounds as possible. Besides these two extremes, if we choose a relatively large value for w_1, the algorithm emphasizes more on minimizing the size of the cut set; and if we choose a relatively large value for w_2, the algorithm emphasizes more on satisfying the upper- and lower-bounds. A relatively large w_1 usually leads to a placement with smaller total wire length, and a relatively large w_2 usually leads to a placement with more upper- and lower-bounds satisfied.

3.3.2 The Placement Algorithm for Macro-Cell Circuits

A similar modified min-cut algorithm is used to generate placements for macro-cell circuits. To control overlaps between modules, we use a minimum separation between each pair of modules. Since the placement algorithm deals only with net wire lengths rather than with distances between modules, an artificial net is introduced between each pair of modules. Denote the kth artificial net by N_{n+k}. (Nets $N_1, N_2, \ldots, N_n$ are signal nets in the circuit.) The minimum separation requirement between a pair of modules is expressed as lower-bounds on the length of the corresponding artificial net. Note that, because these artificial nets are introduced to enforce minimum separation between modules, there are no upper-bounds on wire length of these nets.

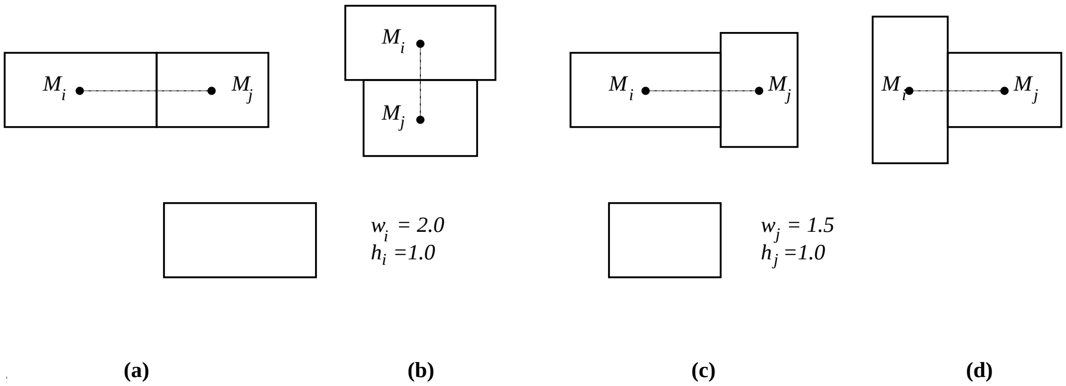


Figure 3.5. Non-overlapping placements that satisfy different lower-bounds

To guarantee that M_i and M_j will not overlap, it is sufficient to have $x_{n+k} \geq \frac{(w_i+w_j+h_i+h_j)}{2}$ where x_{n+k} is the wire length of the artificial net between M_i and M_j. However, in most cases, this requirement is more stringent than necessary. For example, for the two modules in Figure 3.5b, a placement with no overlap is possible if $x_{n+k} = 1.0$, while $\frac{(w_i+w_j+h_i+h_j)}{2} = 2.75$. Clearly, excessively stringent requirements may cause the placement algorithm to fail even when a feasible placement does exist.

To avoid over-constraining the placement problem, five different lower-bounds are computed for each artificial net according to the sizes and shapes of the modules that the net connects. The five lower-bounds for net N_{n+k} are:

$$l_{k1} = \frac{w_i + w_j}{2} \tag{3.8}$$

$$l_{k2} = \frac{h_i + h_j}{2} \tag{3.9}$$

$$l_{k3} = \frac{w_i + h_j}{2} \tag{3.10}$$

$$l_{k4} = \frac{h_i + w_j}{2} \tag{3.11}$$

$$l_{k5} = \frac{w_i + w_j + h_i + h_j}{2} \tag{3.12}$$

We include the lower-bounds l_{k3} and l_{k4} because the placement algorithm allows a module to be rotated by 90 degrees. Note that each of the first four lower-bounds corresponds to a non-overlapping arrangement of M_i and M_j, as shown in Figure 3.5. In the placement phase, the placement algorithm tries to find a placement that satisfies as many of the lower-bounds as possible for each artificial net. Clearly, the more lower-bounds that are satisfied, the better the chance that the modules will not overlap. Note that it is not necessary to satisfy all the lower-bounds for the artificial nets in order to have a non-overlapping placement.

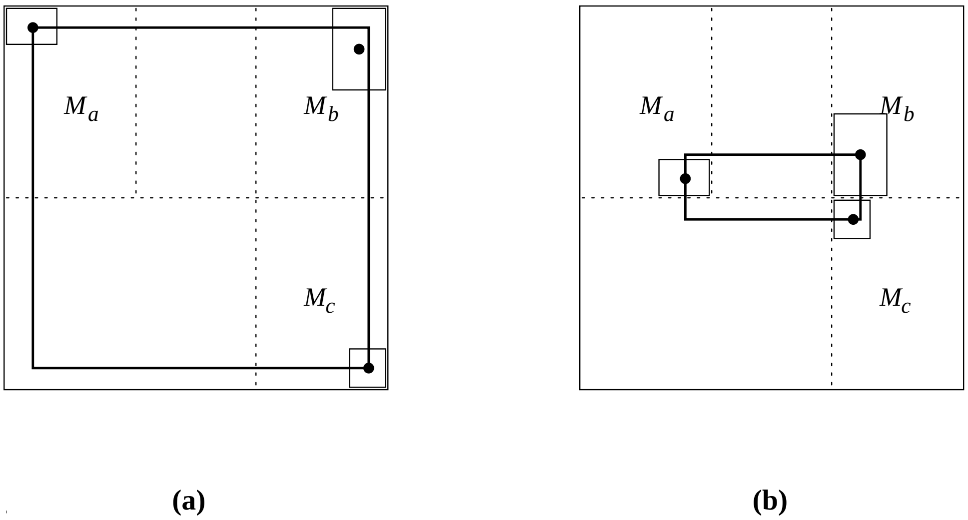


Figure 3.6. Max_i and min_i for net N_i connecting M_a, M_b, and M_c

As in gate-array circuits, at any intermediate level of the partitioning hierarchy, the maximum and minimum possible wire lengths for each net are computed. Because a module can eventually be placed anywhere within its partition in the chip, the maximum wire length max_i for net N_i is the half perimeter of the smallest bounding rectangle that contains all the points at which the centers of the modules in the net may be located (Figure 3.6a), and the minimum possible wire length min_i is estimated as the half perimeter of the smallest bounding rectangle that contains at least one possible location for the center of each module in the net (Figure 3.6b). In Figure 3.6, the dotted lines represent the cut lines.

The weight f_i for signal net N_i is computed in the same way as for the gate-array circuits. Here, we shall discuss how the weight f_{n+k} is computed for artificial net N_{n+k}. Since the artificial nets are introduced to ensure minimum separation between modules, it is desirable to include them in the cut-set. Thus, the weight for an artificial net should always be non-positive. The value of f_k lies within a predetermined range, from $-maxcrit$ to 0. It is computed as follows: If the two modules M_i and M_j connected by N_{n+k} belong to different partitions, f_k is set to 0 because these two modules will not overlap. If the two modules are in the same partition of dimensions $width \times length$, f_k is computed as

$$round(-maxcrit \times \frac{9-z}{9}) \tag{3.13}$$

where z is, among the 9 inequalities listed below, the number of inequalities that are satisfied.

$$width - \frac{w_i + w_j}{2} \geq l_{k1} \tag{3.14}$$

$$length - \frac{w_i + w_j}{2} \geq l_{k1} \tag{3.15}$$

$$width - \frac{h_i + h_j}{2} \geq l_{k2} \tag{3.16}$$

$$length - \frac{h_i + h_j}{2} \geq l_{k2} \tag{3.17}$$

$$width - \frac{w_i + h_j}{2} \geq l_{k3} \tag{3.18}$$

$$length - \frac{w_i + h_j}{2} \geq l_{k3} \tag{3.19}$$

$$width - \frac{h_i + w_j}{2} \geq l_{k4} \tag{3.20}$$

$$length - \frac{h_i + w_j}{2} \geq l_{k4} \tag{3.21}$$

$$length + width - \frac{w_i + h_i + w_j + h_j}{2} \geq l_{k5} \tag{3.22}$$

Because a module may be rotated by 90 degrees, each of the inequalities corresponds to a possible non-overlapping placement of the two modules. Thus, if more of these inequalities are satisfied, the algorithm will have a better chance of finding a non-overlapping placement of the two modules.

The gain in moving a net out of the cut set is computed in the same way as for gate-array circuits. After the gains are computed, Fiduccia's min-cut algorithm is called to divide each partition A into two sub-partitions, B and C. Since we are partitioning a collection of macro-cells, some area balancing rule is needed. Experimental results suggested that

$$\frac{area_A}{2} - maxmod \leq area_B \leq \frac{area_A}{2} + maxmod \tag{3.23}$$

is a reasonable balancing rule, where $maxmod$ is defined as the area of the largest module in partition A [16]. After A is partitioned into B and C, the cut line is shifted to further balance the areas of the partitions according to the areas of the modules in B and C. Suppose that the coordinates of the lower left corner of partition A are (llx, lly), and the coordinates of the upper right corner of partition A are (urx, ury). If the cut line is vertical, the new cut line will be positioned at

$$llx + \frac{area_B(urx - llx)}{area_B + area_C} \tag{3.24}$$

If the cut line is horizontal, the new cut line will be positioned at

$$lly + \frac{area_B(ury - lly)}{area_B + area_C} \tag{3.25}$$

Note that, if the shapes of the modules are flexible, by shifting the cut line, we can guarantee that each partition will have enough area for all the modules in the partition, provided that the total module area is no larger than the placement area.

The partitioning process continues on the sub-partitions until each sub-partition contains at most one module and the area of the sub-partition is small enough so that a good estimation of the module position can be made.

Because the macro-cell placement algorithm is already capable of dealing with lower-bounds, introducing extra lower-bounds on the artificial nets does not complicate the placement algorithm. However, the placement algorithm does not guarantee that all of the lower-bounds will be satisfied. Therefore, if the shapes of the macro-cells are rigid, there might still be some overlaps in the placement generated. These overlaps can be removed by local movement and rotation of the modules. Because the modules are moved locally, the timing performance of the placement will not be changed substantially.

3.4 Iterative Modification of the Bounds

Even though the placement algorithm tries to satisfy as many of the upper- and lower-bounds as possible, it does not guarantee that all the timing requirements will be satisfied. Therefore, after generating a placement, we carry out a timing analysis for that placement. Note that it is not necessary to satisfy all the upper- and lower-bounds in order to generate a placement that satisfies all the timing requirements because the bounds are usually over-constraining. If the placement satisfies all the timing requirements, then the placement is accepted. Otherwise, the type of violated timing requirement is found. Depending on whether a timing delay requirement or a signal skew requirement is violated, either the set of upper-bounds or the set of lower-bounds will be modified for the next iteration of the placement algorithm. By adjusting the bounds in accordance with information obtained from previous placements, the quality of placement is improved iteratively.

Even though the distribution of physically feasible points is in general hard to predict, they are usually clustered in groups. This can be easily seen by starting from a placement, exchanging the positions of a pair of adjacent modules, and obtaining a new placement corresponding to a new physically feasible point that is close to the original physically feasible point. For a given set of upper-bounds X, suppose that the placement generated by the placement algorithm does not satisfy all the timing delay requirements. Let Y denote the physically feasible point corresponding to this placement. Clearly, it is desirable for us to choose a new set of upper-bounds by moving X closer to Y because there are many other physically feasible points in the neighborhood of Y. This is achieved by adjusting the weights α_i's in the objective function. The new α_i's are calculated as

$$\alpha_i = \alpha_i' \times \frac{1 - (x_i - y_i)}{u} \tag{3.26}$$

where α_i' is the old weight for x_i in the objective function, y_i is the wire length of net N_i in the placement, x_i is the old upper-bound for net N_i, and $u = max_{i=1} |x_i - y_i|$. According to Equation 3.26, α_i will be larger than α_i' if the wire length of N_i is greater than its upper-bound, and α_i will be smaller than α_i' if the wire length of N_i is smaller than its upper-bound. Since in the new objective function, the coefficients are larger for nets whose wire lengths exceed the corresponding upper-bounds, while the coefficients are smaller for nets whose wire lengths are smaller than the corresponding upper-bounds, the convex programming system will try to increase the values of the upper-bounds for the nets in the former case and decrease the values of the upper-bounds in the latter case. Consequently, the new optimal point (i.e., the new upper-bounds) will be closer to the point that corresponding to the physical placement that was generated.

On the other hand, if a signal skew requirement is violated (Figure 3.7a,b), in order to force the point corresponding to the physical placement (e.g., Y in Figure 3.7a,b) into the timing feasible region, we can either decrease the values of some of the upper-bounds, as shown

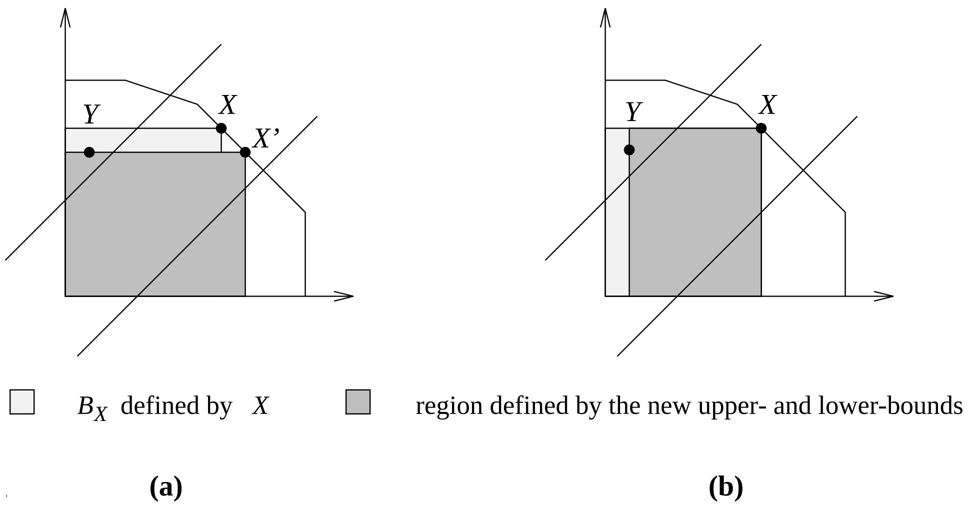


Figure 3.7. Modifying the upper- and lower-bounds

in Figure 11a (changing the upper-bounds from X to X'), or increase the values of some of the lower-bounds, as shown in Figure 3.7b. If we want to decrease the values of some of the upper-bounds, we need to recompute a set of upper-bounds for all the nets (Figure 3.7a). Recomputing the set of upper-bounds is very time consuming. Therefore, in the case that a signal skew requirement is violated, we choose to increase the values of the lower-bounds. If the signal skew requirement at M_i is violated, we can, by backtracking from the net which carries the earliest arriving signal at M_i, find a path p from some primary input pin R_j to M_i, the length of which should be increased in order to satisfy the constraint $c_i \leq C_i$. Consequently, the lower-bounds for the nets on path p should be increased so that

$$a_j + \sum_{all\ N_a \in p} l_a + \sum_{all\ M_k \in p} d_k = a_{il} - C_i \tag{3.27}$$

where a_j is the signal arrival time at R_j, and a_{il} is the arrival time of the latest arriving signal at M_i. In order to give the placement algorithm maximum flexibility, we want to recompute the lower-bounds for the nets on the path p such that, for all N_a in p,

(1) $\prod_{all\ N_a \in p} (x_a - l_a)$ is maximized

(2) $\sum_{all\ N_a \in p} l_a = a_{il} - C_i - a_j$

(3) $l_a \geq 0$

We shall denote $a_{il} - C_i - a_j$ by q. The following pseudo code computes such a set of lower-bounds with no lower-bounds negative.

T = all the nets on p;

$t = |T|$;

for $(i = 1; i < t; i++)$

$\quad l_i = x_i - \frac{\sum_{N_j \in T} x_j - q}{t}$;

while ($l_i < 0$ for some net $N_i \in T$)

{

$\quad T'$ = the set of nets in T with non-negative lower-bounds;

$\quad t' = |T'|$

$\quad$ for (net i in $T - T'$)

$\quad\quad l_i = 0$;

$\quad$ for (net i in T')

$\quad\quad l_i = x_i - \frac{\sum_{N_j \in N'} x_j - q}{t'}$;

$\quad T = T'$;

$\quad t = |T|$;

}

3.5 Experimental Results

Our algorithm was implemented for both gate-array and macro-cell circuits and executed on a Sparc 2 workstation. Since we were not able to find any gate-array or macro-cell placement benchmark circuits with timing information, it was hard to compare results generated by our algorithm with results generated by other performance driven placement algorithms. We did not use MCNC gate-array and macro-cell placement benchmark circuits because timing information and signal flow directions are not given for most of these circuits. To check the quality of the results produced by our algorithm, we tested our algorithm on eight randomly generated gate-array circuits and eight macro-cell circuits. The results generated by our algorithm were compared with the results generated by Fiduccia's min-cut placement algorithm. To compare our algorithm with some well accepted placement algorithms, we customized our algorithm for Xilinx's FPGA circuits, which have gate-array architecture. A performance driven simulated annealing placement algorithm is used by Xilinx's tools. Since the simulated annealing algorithm used in Xilinx's tools does not consider signal skews in the design, our algorithm was modified for the FPGA circuits so that no signal skews are considered. Since Xilinx's tools place I/O pins (I/Os) together with modules (*CLB*s), we modified our placement algorithm to place I/O pins together with modules. To test the algorithms on some benchmark circuits, we used both our placement algorithm and Xilinx's placement algorithm to place eight FPGA circuits mapped from MCNC multi-level benchmark circuits. To validate the delay estimations for the interconnecting wires used in our placement algorithm, we use Xilinx's routing algorithm to route the placement results generated by the two placement algorithms, and compare the delay in the final designs with the estimation of the delay before routing.

circuit no.	no. of modules	no. of nets	no. of *PI*	no. of *PO*	board width	board height	arr.	req.	skew
1	78	236	8	8	8	8	0	80	24
2	272	482	16	16	16	16	0	160	28
3	272	532	16	16	16	16	0	165	33
4	277	582	16	16	16	16	0	175	33
5	282	632	16	16	16	16	0	195	33
6	284	682	16	16	16	16	0	190	36
7	286	732	16	16	16	16	0	200	33
8	286	832	16	16	16	16	0	230	40

Table 3.1. Circuit specifications for the gate-array circuits

circuit no.	no. of modules	no. of nets	no. of *PI*	no. of *PO*	board width	board height	arr.	req.	skew
1	72	145	16	16	16	16	0	100	22
2	83	172	16	16	16	16	0	125	22
3	97	197	16	16	16	16	0	110	22
4	97	222	16	16	16	16	0	110	22
5	102	213	16	16	16	16	0	110	22
6	102	236	16	16	32	16	0	110	22
7	104	219	16	16	32	16	0	115	22
8	102	229	16	16	32	16	0	115	22

Table 3.2. Circuit specifications for the macro-cell circuits

The specifications of the gate-array circuits are summarized in Table 3.1, and the specifications of the macro-cell circuits are summarized in Table 3.2. The column "no. of modules" in the tables gives the number of modules, and the column "no. of nets" gives the number of nets in the circuits. The columns "board width" and "board height" specify the dimensions of the placement area. The columns "no. of *PI*" and "no. of *PO*" specify the number of primary input pins and the number of primary output pins, respectively. The column "arr." specifies the signal arrival time at the primary input pins, and the column "req." specifies the required signal arrival time at the primary output pins of the circuits. Although our program is capable of handling different arrival times at the primary input pins and different required signal arrival times at the primary output pins, for simplicity in the presentation of our results, we set the signal arrival times at all primary input pins to 0, and we set required signal arrival time at all primary output pins to be the same in all of our test runs. Note that, if the arrival times at the primary input pins are different, the arrival times can be made the same by adding extra delays at the primary input pins. Similarly, if the required signal arrival times at the primary output pins are different, the required signal arrival times can be made the same by adding extra delays at the primary output pins during the process of computing the upper-bounds. The column "skew" in the tables specifies the maximum allowable signal skew at the modules of the circuits. Again, to simplify the presentation, although our program is capable of handling different maximum signal skews at different modules, we set them to be the same.

circuit name	no. of modules	no. of nets	no. of *PI*	no. of *PO*	arr.	package no.
cordic	17	40	23	2	0	3020PC84-100
count	34	69	35	16	0	3020PC84-100
comp	68	100	32	3	0	3020PC84-100
9sym	96	105	9	1	0	3042PQ100-100
term1	117	44	34	10	0	3042PQ100-100
C499	70	111	41	32	0	3030PC84-100
C880	123	183	60	26	0	3042PG132-100
alu2	216	226	10	6	0	3064PG132-100

Table 3.3. Circuit specifications for the FPGA circuits

The specifications of the FPGA circuits are summarized in Table 3.3. The column "package no." specifies the actual FPGA chips used for the circuits. Since the required signal arrival times at the primary output pins are not given in the benchmark circuit, we used a binary search method to search for the placement with the smallest latest signal arrival time at the primary output pins.

The results generated by Fiduccia's min-cut placement algorithm [16] for the gate-array circuits and the macro-cell circuits are summarized in Table 3.4 and Table 3.5, respectively. The column "wire length" gives the total wire length, the column "arrival time" gives the latest signal arrival time at the primary output pins, and the column "signal skew" gives the maximum signal skew at the modules.

circuit no.	wire length	arrival time	signal skew
1	822	94	36
2	2135	210	61
3	2537	218	66
4	2695	222	72
5	3056	257	68
6	3370	226	59
7	3612	256	70
8	4174	281	78

Table 3.4. Results generated by min-cut algorithm for gate-array circuits

circuit no.	wire length	arrival time	signal skew
1	910.5	88.1	26.6
2	1057.3	171.5	48.8
3	917.3	105.2	34.4
4	1045.2	136.1	42.3
5	1043.8	96.2	31.1
6	1334.9	132.7	47.9
7	1057.2	94.3	25.9
8	1165.4	116.2	39.3

Table 3.5. Results generated by min-cut algorithm for macro-cell circuits

The results generated by Xilinx's tools for the FPGA circuits are summarized in Table 3.6 which shows both the estimated results before routing and the final results after routing. The column "placement time" specifies the running time of the placement algorithm, and the column "unrouted pins" specifies the number of unrouted pins in the final design. As is shown in the table, estimated latest signal arrival times at the primary output pins using bounding boxes as interconnection delays are reasonably close to the actual values computed after routing in all test cases. Therefore, the estimations used for the delays in our placement algorithm are good measurements for the actual delays. Notice however that as the circuit size gets larger, the actual maximum circuit delay tends to be larger than the estimated value. This is because that in Xilinx's FPGA chips, there are very limited routing resources, and it is harder to route circuits with relatively more interconnection wires.

	before routing			after routing	
circuit name	placement time (min.)	wire length	arrival time(ns)	unrouted pins	arrival time(ns)
cordic	1.3	378	46.5	0	41.3
count	4.7	632	71.5	0	70.0
bw	3.0	354	32.1	0	22.3
f51m	1.8	574	92.7	0	82.9
frg1	3.4	1024	79.7	0	70.1
comp	7.7	1143	90.9	0	88.0
my_add	2.5	470	97.1	0	96.1
9sym	21.5	1202	81.2	0	81.3
term1	24.3	2030	92.9	0	79.1
C499	9.5	1871	79.6	0	97.3
C880	28.5	2510	128.3	0	136.7
alu2	91.1	3226	192.3	0	206.8

Table 3.6. Results generated by Xilinx's tools

To run our placement algorithm, we need first to choose the values for the parameters w_1, w_2, and $maxcrit$. Recall that the gain in moving net N_i out of the cut set in our placement algorithm is computed as $w_i + w_2 \times f_i$, where f_i lies in the range of $-maxcrit$ and $maxcrit$. For our experiments, we set $maxcrit$ to be 15.

To illustrate the trade off between total wire length and timing performance, we tried two different sets of w_1 and w_2 for the gate-array circuits. First, we set $w_1 = w_2 = 1$. Because $maxcrit = 15$, by setting $w_1 = w_2 = 1$, we are putting much more emphasis on optimizing the timing performance than on minimizing the total wire length. The experimental results are summarized in Table 3.7. The columns "red. in %" show the reduction in wire length, arrival time and signal skew in the new placements as compared to the placements obtained using the min-cut algorithm. On average, we were able to obtained a 29.1% reduction in timing delay and a 49.3% reduction in maximum signal skew. Note that, because the placement algorithm puts more emphasis on optimizing the timing performance, the total wire length increases by 16.1% on average. We then set $w_1 = 5$ and $w_2 = 1$. The experimental results are summarized in Table 3.8. In comparison to the min-cut algorithm, we were still able to generate placements with 21.8% reduction in timing delay and 38.6% reduction in maximum signal skew, while increasing the total wire length by only a small percentage on average. The average running times for computing the bounds and generating placements for the gate-array circuits are listed in Table 3.9. Note that the computation of the upper-bounds consumes a large portion of the total computation time because a large convex programming problems must be solved in each iteration.

circuit no.	wire length	red. in %	arrival time	red. in %	signal skew	red. in %
1	937	-14.0	74	21.3	24	33.3
2	2535	-18.7	147	30.0	28	54.1
3	2914	-14.9	153	29.8	32	51.5
4	3214	-19.3	158	28.8	31	57.0
5	3451	-12.9	166	35.4	32	53.0
6	3880	-15.1	169	25.2	36	39.0
7	4214	-16.7	178	30.5	34	51.4
8	4896	-17.3	191	32.0	35	55.1
average		-16.1		29.1		49.3

Table 3.7. Results generated by our algorithm for gate-array circuits with $w_1 = w_2 = 1$

circuit no.	wire length	red. in %	arrival time	red. in %	signal skew	red. in %
1	830	-1.0	82	12.8	30	16.7
2	2291	-7.3	143	31.9	30	50.8
3	2540	-0.1	166	23.9	45	31.8
4	2987	-10.8	168	24.3	33	54.2
5	3178	-4.0	199	22.6	38	44.1
6	3491	-3.6	189	16.4	44	25.4
7	3779	-4.6	200	21.9	38	45.7
8	4484	-7.4	223	20.6	47	39.8
average		-4.9		21.8		38.6

Table 3.8. Results generated by our algorithm for gate-array circuits with $w_1 = 5$, $w_2 = 1$

circuit no.	1	2	3	4	5	6	7	8
upper-bounds	3.6	11.0	28.2	12.2	13.9	15.2	26.5	26.8
placement	0.1	0.4	0.4	0.4	0.4	0.4	0.4	0.5

Table 3.9. Running time of the gate-array circuits in minutes

For the macro-cell circuits, we set $w_1 = w_2 = 1$. The experimental results are summarized in Table 3.10. On average, in comparison to the placements generated by the min-cut algorithm, we were able to generate placements with a 25.6% reduction in timing delay and a 49.6% reduction in maximum signal skew, while increasing the total wire length by only a small percentage.

circuit no.	place time (min.)	wire length	red. in %	arrival time	red. in %	signal skew	red. in %
1	3.8	882.6	3.1	68.6	22.1	19.5	26.7
2	5.3	1079.3	-1.5	107.9	37.1	19.6	59.8
3	6.7	1043.2	-13.7	85.3	18.9	21.0	39.0
4	7.9	1117.6	-6.9	84.9	37.6	16.7	60.5
5	7.7	1242.5	-19.0	85.0	11.7	21.9	29.6
6	9.1	1333.6	0.1	86.6	34.7	20.5	57.2
7	8.0	1235.5	-16.9	83.2	11.8	20.6	20.5
8	8.5	1213.6	-4.1	78.0	32.9	19.8	49.6
average			-7.4		25.6		42.9

Table 3.10. Results generated by our algorithm for macro-cell circuits with $w_1 = w_2 = 1$

circuit 8	arrive	skew
spec.	115.0	22.0
iter. 1	120.6	44.2
iter. 2	82.5	31.2
iter. 3	78.0	19.8

Table 3.11. Intermediate results of placing macro-cell circuit 8

To illustrate the iterative procedure presented in section 5, intermediate results of the iterations when macro-cell circuit number 8 was placed are listed in Table 3.11. The row "spec." is the specified required signal arrival time at the primary output pins and maximum allowable signal skew of the circuit. After the first iteration, both the timing delay and the signal skew requirements were violated. Therefore, the set of upper-bounds was modified. After the second iteration, the timing delay requirements were satisfied but the signal skew requirements were still violated. Therefore, the set of lower-bounds was modified. After the third iteration, the placement generated satisfies all the timing requirements.

For the FPGA circuits, we set $w_1 = 3$ and $w_2 = 1$. The experimental results are summarized in Table 3.12. Again, both the estimated results before routing and the final results after routing are listed. As in the results shown in Table 3.6, estimated latest signal arrival times at the primary output pins used in our placement algorithm are reasonably close to the actual values after routing in all test cases. On average, in comparison to the placement results generated by Xilinx's placement algorithm, we were able to obtain a 59.0% reduction on running time, 0.9% reduction in total wire length, and 10.4% reduction on the latest signal arrival time before routing. After routing, we were able to obtain a 9.6% reduction on the latest signal arrival time on average.

circuit name	before routing						after routing		
	place time (min.)	red. in %	wire length	red. in %	arrival time(ns)	red. in %	unrouted pins	arrival time(ns)	red. in %
cordic	0.8	38.5	337	10.8	37.9	18.5	0	35.9	13.1
count	1.7	63.8	672	-6.3	63.5	11.2	0	56.5	19.3
comp	3.1	59.7	1049	8.2	84.4	7.2	0	78.9	10.3
9sym	6.5	69.8	1198	0.3	78.6	3.2	0	76.6	5.8
term1	9.0	63.0	2162	-1.1	79.5	14.4	0	74.3	6.1
C499	7.2	24.2	1918	-2.5	75.0	5.8	0	82.8	14.9
C880	16.8	41.1	2643	-5.3	111.5	13.1	0	117.3	14.2
alu2	30.9	66.1	3130	3.0	173.5	9.8	0	199.1	3.7
average		59.0		0.9		10.4			9.6

Table 3.12. Results generated by our algorithm for FPGA circuits with $w_1 = 3, w_2 = 1$

3.6 Summary

In this chapter, we presented a unique and effective upper- and lower-bound approach to the performance driven gate-array and macro-cell placement problem. In our approach, the timing delay requirements, the signal skew requirements, and the non-overlapping requirements between macro-cells are first translated into a set of upper- and lower-bounds on net wire lengths. A modified min-cut algorithm is then used to generate a placement that satisfies as many of the upper- and lower-bounds as possible. In case the placement generated does not satisfy all the timing delay requirements and signal skew requirements, the upper- and lower- bounds are modified iteratively to improve the timing performance of the placement. Experimental results indicated that our algorithm are quite effective.

CHAPTER 4

A Path-Based Performance Driven Placement Algorithm

4.1 Introduction

In a path-based performance driven placement algorithm, path delays are considered explicitly during the placement process. One typical such approach is to formulate the placement problem as an optimization problem with both timing requirements and physical placement requirements as constraints [29,30,38,39]. However, it is difficult to satisfy all the placement constraints during the placement process, and the solution to the placement problem usually contain overlapping modules. To satisfy the placement constraints, the placement solution need to be repeatedly perturbed until all the placement constraints are satisfied. After each perturbation, the optimization problem also need to be solved again. Obviously, The perturbation process destroys the optimality of the solution generated by the optimization process. In order to avoid all the modules from clustering into one point, the exact locations of the I/O pins usually need to be pre-specified. Such an early commitment of I/O pins to their locations without knowing the possible placement of the circuit modules could leads to poor placement results. Therefore, it is desirable for the placement algorithm to place I/O pins together with modules in the circuit.

For most performance driven placement algorithms, the required signal arrival times at the primary output pins need to be pre-specified. However, for many circuits, it is not known what the reasonable required signal arrival times at the primary output pins should be, and it is desirable for the placement algorithm to place the circuits so that the placements generated have minimum signal arrival times at the primary output pins. Also, in many circuits, the required signal arrival times at the primary output pins might be much larger than their minimum possible values, and placements that satisfy the timing delay requirements do not necessary give the best timing performance. Therefore, instead of trying to satisfy the timing delay requirements, it is more desirable for the placement algorithm to generate placements with minimum signal arrival times at the primary output pins.

In this chapter, we shall present a hierarchical partitioning placement algorithm which tries to minimize the total wire length and the latest signal arrival times at the primary output pins (maximum circuit delay) for gate-array circuits [19]. Our algorithm does not rely on pre-specified information on the required signal arrival times at the primary output pins. Neither doe our algorithm rely on the locations of the primary input and output pins. Consequently, our placement algorithm overcomes the shortcomings of other path-based performance driven placement algorithms. In our placement algorithm, the total wire length and maximum circuit delay are computed and updated explicitly at each step of the hierarchical partitioning step, and no path information is lost during the placement process.

As was shown in Chapter 3, there is usually a trade off between minimizing the total wire length and minimizing the maximum circuit delay. Therefore, it is important for the designer to have the flexibility to decide the relative importance between the total wire length and maximum circuit delay. In our algorithm, we define the objective function as a weighted sum of the total wire length and maximum circuit delay. By specifying different relative weights for the total wire length and maximum circuit delay, the placement algorithm is capable of placing different amount of emphases on the two objectives.

4.2 Hierarchical Partitioning Placement Algorithm

In our hierarchical partitioning placement algorithm, a region is defined to be a rectangular area within the placement area which encloses a number of module and I/O slots. The general idea of a hierarchical partitioning placement algorithm is as follows. The placement algorithm keeps a queue of the regions which are to be partitioned into sub-regions. At each iteration of the partitioning process, the region at the top of the queue is partitioned into two sub-regions and the resulting sub-regions are placed at the end of the queue. If a sub-region is small enough so that the locations of the components in the sub-region can be determined, the sub-region will not be further partitioned and thus will not be placed at the end of the queue. The partitioning process continues until the locations of all the components are determined. During the partitioning process, balancing rules are used to guarantee that the placement generated is valid (e.g., no two modules are placed in the same slot). Balancing rules also have a big effect on the quality of the placement generated. The objective of the partitioning process is to minimize the objective function.

One well known hierarchical partitioning placement algorithm is the min-cut placement algorithm [15,16]. The min-cut placement algorithm tries to minimize a weighted sum of the cut set of the nets between the two sub-regions that a region is partitioned into. Intuitively, a small cut set leads to small total wire length and small channel densities. Because cut sets are net-based, they are not capable of carrying circuit delay information.

In our algorithm, the total wire length and the maximum circuit delay are estimated and updated directly (rather than using some intermediate quantities such as the size of the cut set in the min-cut algorithm) in a dynamic way throughout the partitioning process. When a region is partitioned into two sub-regions, the gain of a component is computed as the decrement in the value of the objective function when the component is moved across the cut line. The *module balance factor* of a region is defined to be the ratio between the number of modules and the number of the module slots in the region, and the *I/O balance factor* of a region is defined to be the ratio between the number of I/O pins and the number of I/O slots in the region. The basic steps for partitioning a region r into sub-regions r_1 and r_2 are as follows:

(1) Generate a cut line which divides r into r_1 and r_2. Distribute the components in r into r_1 and r_2 in an arbitrary order so that the module balance factors and the I/O balance factors

in the two sub-regions are as close as possible. Compute the gains of the components in r_1 and r_2.

(2) Generate balancing rules for r_1 and r_2.

(3) Repeatedly select and move the components across the cut line in a way similar to the min-cut algorithm [16] under the guidance of the balancing rules and the gains of the components to minimize the value of the objective function. The gains of the components in r_1 and r_2 are updated dynamically as the components are moved.

Generating the balancing rules and computing the gains of the components are the key steps in our placement algorithm, which will be discussed in detail in Section 4.3. We shall describe here the initial partitioning process in Step 1 and the component selecting process in Step 3.

When region r is partitioned, a cut line is first generated to divide the slots in r into r_1 and r_2 as evenly as possible. To partition the placement area evenly, cut lines are drawn alternately in horizontal and vertical directions in successive hierarchies. After the cut line is generated, components in r are distributed into r_1 and r_2 one by one in an arbitrary order. To distribute the modules into r_1 and r_2 evenly, before a module is distributed, the module balance factors of the two sub-regions corresponding to the modules that have already been distributed are computed and the new module is placed in the sub-region with a smaller module balance factor. I/O pins are distributed in a similar way.

After the initial partitioning, components are selected and moved across the cut line in order to reduce the value of the objective function. Each time before a component is moved, the gains of the components in the two sub-regions are computed and the module and the I/O pin with the largest gains in each sub-region are recorded. These four components (two I/O pins and two modules in the two sub-regions) are candidates for moving. If a sub-region does not contain any I/O pin or module which can be moved, the number of possible candidates will be less than four. If moving any candidate component across the cut line violates any balancing rule, the component is excluded from the candidate list. Among the candidates, the component with the largest gain is moved. Tie is broken by choosing the component that will lead to the most balanced distribution of components after it is moved. The moving process continues until there is no more candidate in the candidate list. To avoid getting stuck at local minimums, we use a moving procedure similar to the one used in [16].

4.3 Generating Balancing Rules and Computing Gains

Balancing rules are the rules that the placement algorithm uses to control the balance between the number of I/O pins (modules) and the number of I/O slots (module slots) in each region. The quality of the placement depends heavily on the balancing rules. Overly restrictive balancing rules will restrict the exploration of possible placement solutions and thus lead to poor placement results, and overly loose balancing rules may lead to unevenly distributed and possibly congested sub-regions that are hard to place and route later on. It is important to have balancing rules that will lead to placements with evenly distributed components while not overly restricting the partitioning process. However, there is one exception to above criteria for balancing rules. If the number of components in the circuit is much smaller than the number of corresponding slots, distributing the components evenly among the sub-regions will lead to an overly sparse placement which will have large net wire lengths and large circuit delay. In this case, unevenly distributed regions should be allowed.

In our algorithm, similar balancing rules are used for the modules and the I/O pins. Here, we shall only describe the balancing rules for the modules in terms of the *maximum allowable module balance factor*. Recall that the module balance factor B_r is defined to be $\frac{C_r}{S_r}$, where C_r is the number of modules in r and S_r is the number of module slots in r. After r is initially partitioned, the maximum allowable module balance factor β_{r_1} for r_1 is computed as follows:

1. $\beta_{r_1} = \frac{B_r+1}{2}$;
2. if $(C_{r_1} + 1 > \beta_{r_1} \times S_{r_1})$
3.   {
4. $\beta_{r_1} = \frac{C_{r_1}+1}{S_{r_1}}$;
5.   if $(\beta_{r_1} > 1)$
6. $\beta_{r_1} = 1$;
7.   }
8. if $(\beta_{r_1} < P)$
9. $\beta_{r_1} = P$;

The maximum allowable module balance factor β_{r_2} for r_2 can be computed in a similar way. To distribute modules evenly between r_1 and r_2, it is desirable to have B_{r_1} and B_{r_2} as close as possible. However, in order not to overly restrict the partitioning process, we should give the partitioning algorithm a certain amount of leeway. Therefore, the initial value of β_{r_1} is computed as in line 1. Line 2 checks whether the value of β_{r_1} computed in line 1 will forbid the movement of any module into r_1. If that is the case, line 4 increases the value of β_{r_1} to allow one module to be moved into r_1. Lines 5 and 6 make sure that the value of β_{r_1} computed in line 4 does not exceed 1 so that the number of modules in r_1 does not exceed the corresponding number of modules slots in r_1. In the case that r was sparse, lines 8 and 9 increase the value of β_{r_1} to avoid the generation of an overly sparse placement. P is a pre-determined value for the minimum possible value of β_{r_1}, and experimental results suggest that 0.8 is a good choice for the value of P.

Special attentions are needed if the number of modules is equal to the number of module slots in r. Since there is no empty module slot in r, there is no empty module slot in r_1 and r_2, and moving any module across the cut line violates the balancing rules. To be able to move the modules across the cut line under such circumstance, we define a special type of move which consists of moving a module from r_1 (or r_2) to r_2 (or r_1) followed immediately by moving another module from r_2 (or r_1) back to r_1 (or r_2). Notice that this type of move does not change the number of modules in r_1 and r_2, and therefore will not violate the balancing rules. Moves of this type are only allowed when r has no empty module slot. Similar moves are defined for I/O pins in case the number of I/O pins is equal to the number of I/O slots in r.

To compute the gain of a component, the reduction in the value of the objective function corresponding to moving the component across the cut line need to be computed. Recall that the objective function is of the form $w_T \times T + w_D \times D$ where T is the total wire length, D is the maximum circuit delay, w_T is the weight for the total wire length, and w_D is the weight for the maximum circuit delay. At any intermediate step of the partitioning process, it is impossible to compute the exact values of the net wire lengths because the exact locations of the components have not been determined. However, since each component can be placed only in a slot within its region, lower- and upper-bounds on net wire lengths can be computed. For net N_i, let $U_{i,urx}$ denote the x-coordinate of the rightmost slot that the components connected to N_i may occupy, $U_{i,llx}$ denote the x-coordinate of the leftmost slot that these components may occupy, $U_{i,ury}$ denote the y-coordinate of the upper most slot that these components may occupy, and $U_{i,lly}$ denote the y-coordinate of the lowest slot that these components may occupy. $U_{i,llx}, U_{i,lly}, U_{i,urx}$, and $U_{i,ury}$ can be computed easily by going through the list of components that N_i connects. The upper-bound on the length of N_i, denoted by MAX_i, can be computed by the formula

$$MAX_i = U_{i,urx} - U_{i,llx} + U_{i,ury} - U_{i,lly}. \tag{4.1}$$

Similarly, let $L_{i,llx}$ denote the smallest x-coordinate of the rightmost slots that the components connected by N_i may occupy, $L_{i,urx}$ denote the largest x-coordinate of the leftmost slots that these components may occupy, $L_{i,lly}$ denote the smallest y-coordinate of the upper most slots that these components may occupy, and $L_{i,ury}$ denote the largest y-coordinate of the lowest slots that these components may occupy. Since net wire length may not be negative in the x and the y direction, the lower-bound MIN_i on the length of N_i can be computed as follows:

if $(L_{i,urx} > L_{i,llx})$
  $MIN_i = L_{i,urx} - L_{i,llx}$;
else $MIN_i = 0$;
if $(L_{i,ury} > L_{i,lly})$
  $MIN_i = MIN_i + L_{i,ury} - L_{i,lly}$;
else $MIN_i = L_{i,ury} - L_{i,lly}$;

A careful study of the gate array placement problems suggests that the lower-bounds on net wire lengths can be further improved using Formula 3.5.

Figure 4.1 shows the lower- and upper-bounds on the net wire length of a net connecting three modules M_a, M_b, and M_c, where the modules are shown in their corresponding regions. The upper-bound on its wire length is 5 and the lower-bound on its wire length is 3.

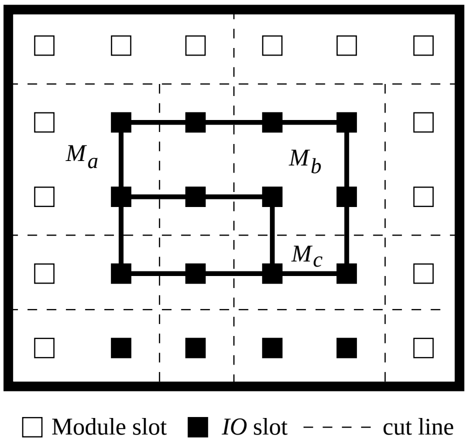


Figure 4.1. Estimating bounds on net wire length

For a set of n nets, their wire lengths can be represented conveniently by a point in the n-dimensional space. Specifically, we shall let the net wire lengths be the coordinates of the point. For a given partition of the placement area (intermediate or final), let $(MIN_1, MIN_2, \ldots, MIN_n)$ be the point corresponding to the lower-bounds on the net wire lengths, and $(MAX_1, MAX_2, \ldots, MAX_n)$ be the point corresponding to the upper-bounds on the net wire lengths. The *solution space* that corresponds to a partition is defined to be the set of points $(x_1, x_2, \ldots, x_n)$ in the n-dimensional space such that $MIN_i \leq x_i \leq MAX_i$ for $1 \leq i \leq n$. A placement generated by further partitioning the regions in the current partition corresponds to a point in the solution space. However, at an intermediate step, it is not known where this point will be in the solution space. Therefore, we shall estimate the net wire lengths by the average values of their corresponding upper- and lower-bounds. As the partitioning process continues, the solution space becomes smaller and smaller, and the estimations become more and more accurate.

Notice that the estimated net wire lengths will change if and only if the solution space changes, which happens when a region is initially partitioned and when a component is moved across the cut line. In order to compute the value of the objective function efficiently, it is necessary to determine the solution space (the bounds on wire lengths) and the maximum circuit delay efficiently.

When region r is initially partitioned into r_1 and r_2, only the lower- and upper-bounds on the lengths of the nets which are connected to components in both r_1 and r_2 might change. One straight forward way to update the bounds on the wire lengths of these nets is to go through the list of components in r and recompute the lower- and upper-bounds on the wire lengths of the nets connected to them. A more efficient way to update the bounds is to compute the changes in the lower- and upper-bounds from their previous values. Without loss of generality, we assume that the cut line is a vertical line and r_1 is to the left of r_2. Since the slots in r_1 and r_2 are subsets of the slots in r, by placing M_i in r_1, $U_{j,llx}$ and $L_{j,urx}$ remain unchanged for net N_j which is connected to M_i. However, $U_{j,urx}$ and $L_{j,llx}$ might decrease. If the original value of $U_{j,urx}$ is equal to the x-coordinate of the rightmost slot M_i may occupy in r before the partitioning, the value of $U_{j,urx}$ might decrease and need to be recomputed. If the original value of $L_{j,llx}$ is greater than the x-coordinate of the rightmost slot that M_i may occupy in

r_1, the value of $L_{j,llx}$ should be decreased to the x-coordinate of this slot. The values of $U_{j,lly}$, $U_{j,ury}$, $L_{j,lly}$, and $L_{j,ury}$ can be updated in a similar way. After the lower- and upper-bounds are updated, the estimated net wire lengths can be updated easily. Because many net wire lengths may change after the partitioning, it is more efficient to recompute the maximum circuit delay after r is partitioned and before any component is moved across the cut line.

Recall that the gain of a component is the reduction in the value of the objective function when the component is moved across the cut line. Since the objective function is a weighted sum of the total wire length and the maximum circuit delay, an efficient algorithm is needed to compute the changes in the total wire length and the maximum circuit delay after a component is moved. Notice that when a component is moved, only the wire length of the nets connected to the component might change. Without loss of generality, we assume that we want to compute the gain of moving a module from r_1 to r_2 where r_1 is to the left of r_2. Let $r_{1,llx}$ denote the x-coordinate of the leftmost module slot in r_1, $r_{1,urx}$ denote the x-coordinate of the rightmost module slot in r_1, $r_{2,llx}$ denote the x-coordinate of the leftmost module slot in r_2, and $r_{2,urx}$ denote the x-coordinate of the rightmost module slot in r_2. Because the module is moved to the right, for a net N_j connected to the module, the value of $U_{j,urx}$ should become $r_{2,urx}$ if $U_{j,urx} < r_{2,urx}$, and should remain unchanged otherwise. Similarly, the value of $L_{j,urx}$ should become $r_{2,llx}$ if $L_{j,urx} < r_{2,llx}$, and should remain unchanged otherwise. If $U_{j,llx} = r_{1,llx}$, $U_{j,llx}$ might become larger and need to be recomputed. Otherwise, $U_{j,llx}$ remains unchanged. Similarly, if $L_{j,llx} = r_{1,urx}$, $L_{j,llx}$ might become larger and need to be recomputed. Otherwise, $L_{j,llx}$ remains unchanged. $U_{j,lly}$, $U_{j,ury}$, $L_{j,lly}$, and $L_{j,ury}$ can be updated in a similar way.

To compute the maximum circuit delay, it suffices to compute the signal arrival times at the outputs of all the modules and I/O pins. Changes in the delays of a few nets usually lead to changes in signal arrival times at the outputs of many components. When a component is moved, a straight forward way to update the signal arrival times at component outputs is to recompute them. However, since components are moved many times during the partitioning process, recomputing the signal arrival times at all the component outputs every time when a component is moved is not practical. Notice that the signal arrival time at a component output might change if and only if the wire length of a net that is on a path from a primary input pin to the component is changed. To speed up the timing analysis process, for each component, a list of components the signal arrival times of which might change as a result of the movement is kept. Such a list Q_i for component M_i can be determined as follows. First, $M_i \in Q_i$ because the wire lengths of the nets which carry the input signals to M_i might change as M_i moves. Also, any component M_j which shares an input signal with M_i is in Q_i because the wire length of the net carrying the shared input signal might change as M_i moves. Finally, any component that receives signals from any of the components in Q_i is also in Q_i. A topological sort can be carried out to search for components that are reachable from the components already in Q_i. Since the components in Q_i are determined in topological order, the signal arrival times at the outputs of components in Q_i can be updated in the same order after M_i is moved. Notice that Q_i needs to be computed only once at the beginning of the placement process.

4.4 Experimental Results

Our placement algorithm was implemented in C and tested on a Sparc IPX workstation. Since there is no available placement benchmark circuits with timing information, our algorithm was tested on 8 randomly generated combinational circuits the specifications of which are summarized in Table 4.1. Even though our algorithm is capable of handling circuits with different module and I/O pin delays, for simplicity of presentation, the internal delay of each module and I/O pin was set to 1. The spacing between a module and an I/O slot is also set to 1.

cir. no	specification		min-cut		
	no. of modules	no. of nets	total length	max. delay	run time
1	300	450	1693	361	0.1
2	450	600	2637	447	0.1
3	550	700	3027	481	0.2
4	600	800	3598	491	0.2
5	650	850	3875	759	0.3
6	700	950	3360	502	0.3
7	750	1150	3372	381	0.4
8	800	1100	5066	557	0.5

Table 4.1. Circuit specifications and results generated by min-cut algorithm

For comparison, Fiduccia's min-cut algorithm [16] is first run on the circuits and the results are summarized in Table 4.1. So that the comparisons will be fair and meaningful, Fiduccia's min-cut algorithm was modified so that I/O pins are placed together with the modules. The results obtained by our performance driven placement algorithm are summarized in Table 4.2. In Table 4.2, two different sets of values of w_D and w_T are used in the objective function. In the first set, $w_D = 1$ and $w_T = 0.3$. In the second set, $w_D = 1$ and $w_T = 0.7$. For the placement generated using the first set of weights, we were able to obtain 1.5 percents of improvement on total wire length and 16.8 percents of improvement on maximum circuit delay on the average over the min-cut algorithm. For the placement generated using the second set of weights, we were able to obtain 3.5 percents of improvement on total wire length and 15.8 percents of improvement on maximum circuit delay on the average over the min-cut algorithm. Since a larger weight is assigned to the total wire length in the second set, the results obtained using the second set of weights have larger maximum circuit delay and smaller total wire length. Since vast amount of computation is involved in our algorithm, the running time of our algorithm is invariably longer than the running time of the min-cut algorithm. However, the updating techniques enable us to keep the running time of our algorithm within a reasonable range (the run times in the tables are in minutes).

cir.	$w_D = 1, w_T = 0.3$			$w_D = 1, w_T = 0.7$		
no.	total length	max. delay	run time	total length	max. delay	run time
1	1695	301	6	1660	310	5
2	2483	352	28	2385	352	25
3	2927	369	45	2915	389	39
4	3624	409	66	3512	412	73
5	3965	653	99	3860	660	142
6	3330	404	141	3259	422	147
7	3352	336	133	3293	330	180
8	4878	494	230	4844	483	253

Table 4.2. Results generated by our placement algorithm

4.5 Summary

In this chapter, we presented a hierarchical partitioning placement algorithm which tries to minimize the total wire length and the maximum circuit delay explicitly during the partitioning process. The placement algorithm gives the designer the flexibility to decide how much weight the placement algorithm puts on the two different objectives. Effective techniques to estimate and update the total wire length and the maximum circuit delay enable use to generate placements for reasonably large circuits in reasonable amount of time.

CHAPTER 5

A Placement Driven Technology Mapping Algorithm for Xilinx's FPGAs

5.1 Introduction

For ASIC designs, it is important to have short turn around times. At the same time, it is important to keep the design cost low. Field programmable Gate Arrays (FPGAs) can be easily programmed and reprogrammed, and therefore, are natural choices for rapid and low cost prototyping. Special FPGA architectures make FPGA designs different from conventional designs, and special logical and physical design tools are needed.

There are two most common FPGA architectures, one is based on the programmable cells of small granularity such as the multiplexers, and the other one is based on lookup-tables which can realize complex functions. Lookup-table based FPGAs, where each lookup-table can realize any function with the number of inputs not exceeding some given number, are

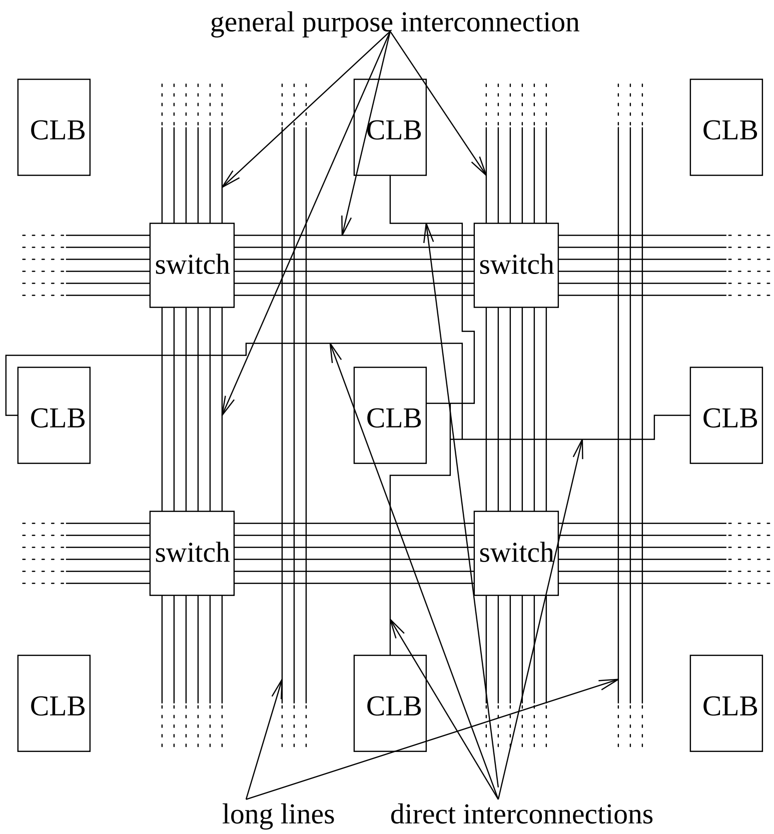


Figure 5.1. Xilinx's LCA

particularly interesting because of their flexibilities and the challenges posed to the design process. In this chapter, we shall consider specifically Xilinx's FPGA architecture (Figure 5.1) which is lookup-table based [48]. A Xilinx's FPGA is called a logic cell array (LCA). An LCA consists of an array of configurable logic blocks (CLBs), a set of configurable input/output blocks (IOBs), and a set of switching matrices which can be programmed for interconnections (Figure 5.1). A CLB can be programmed to implement one or two lookup-tables with a limited number of inputs. The IOBs can be programmed either as inputs or outputs of the LCA, and they are located at the peripheral of the LCA. Three kinds of routing resources are supplied in an LCA, namely, the general purpose interconnections, the direct interconnections, and the long lines (Figure 5.1).

The programmable switches for interconnection introduce extra circuitries and extra delays. To reduce the interconnection delays and the size of the LCAs, the programmable switches are designed to be partially programmable (e.g., only a subset of the possible interconnections can be programmed). As a result, relatively more restrictive physical design constraints exist in LCAs in comparison with other architectures.

Since Xilinx's FPGAs have relatively more restrictive physical design constraints, a circuit generated by a technology mapping algorithm without considering the possible placement and routing information of the resulting circuit could be hard to place and route, and consequently, the final design may have large maximum circuit delay. Therefore, it is important for the technology mapping algorithm to consider physical design constraints. Unfortunately, most existing technology mapping algorithms for FPGAs are not capable of doing so. Even though some performance-driven technology mapping algorithms exist [6,17], most of them try to generate circuits with minimum number of levels of CLBs. The circuits produced by these technology mapping algorithms will have small maximum circuit delay under the assumption that all the nets have the same delay after placement and routing. However, different nets always have different delays in reality. Since net delays are mostly decided by the placement of the CLBs, and since the routing of the nets is mostly decided by the placement, it is most important to consider the placement information during the technology mapping process. In [35], a layout driven technology mapping algorithm was presented. In this chapter, we shall present a new technology mapping algorithm which uses the placement information to guide the technology mapping process [18].

5.2 Placement Driven Technology Mapping

As was suggested in [6], a simple gate circuit in which each gate has only two inputs is a good starting point for the technology mapping process. There are two reasons to use the technology mapping algorithm on simple gate circuits. First, we want to limit the number of inputs to each gate so that it can be fitted into a CLB. Second, if we think of technology mapping as a process of packing gates into CLBs, intuitively, smaller gates will be more easily packed. Therefore, before technology mapping, the input circuit is transformed into a simple gate circuit. We shall call the simple gate circuit before technology mapping the *initial circuit*, and the circuit after technology mapping the *final circuit*. As was discussed in the introduction, the technology mapping algorithm should consider as much placement information of the final circuit as possible. Since the final circuit is not known until the end of the technology mapping process, it is impossible to obtain the precise placement information of the final circuit before or during the technology mapping process. However, if the technology mapping algorithm which maps the simple gates locally (e.g., the algorithm tries to map adjacent gates in the placement into the same CLB) in the initial circuit, the placement topology of the initial circuit will reflect the placement topology of the final circuit. Therefore, the placement information of the initial circuit can be used to approximate the placement information of the final circuit. In our algorithm, the performance driven placement algorithm described in Chapter 3 is first used to place the initial simple gate circuit. The placement information is then extracted from the simple gate circuit placement, and a modified FlowMap [7] algorithm is used to map the simple gate circuit under the guidance of the placement information.

Before the simple gate circuit can be placed, an artificial placement area with a set of IOBs and CLBs is generated to hold the simple gates and the primary I/O pins. The number of CLBs (simple gates) in the initial circuit is much larger than the number of CLBs in the final design, while the number of IOBs (primary I/O pins) in the initial circuit is the same as the number of IOBs in the final design. To make the placement topology of the initial circuit close to the placement topology of the final circuit, the same number of IOB slots are first assigned to the same locations in the artificial LCA as in the LCA used for the final design. The number of CLB slots needed to hold the simple gates are then computed and the CLB slots are distributed in a similar fashion as in the LCA used for the final design. To make the delay information of the simple gate circuit reflect the delay information of the final circuit as much as possible, the delay of the IOBs and the delay per unit length of interconnecting wire are set to be the same as the corresponding values in the LCA used for the final design. Since several simple gates may be mapped into one CLB by technology mapping, the CLB delay in the artificial LCA is set to be a fraction of the CLB delay in the LCA used for the final design.

After the placement, interconnection delay information of the placement can be easily extracted, and the delay information is used to guide the technology mapping process. To map the simple gate circuit under the guidance of the delay information, a modified FlowMap algorithm was developed which guarantees a minimum-delay mapping solutions under any given net delay estimation [8]. During the technology mapping process, to simplify the computation of the interconnection delays, it is assumed that the delay of a net which is completely mapped into a CLB (e.g., all the simple gates that the net interconnects are mapped into the same CLB) becomes 0, and the delay of a net which is not completely mapped into a CLB remains unchanged. The objective of the modified FlowMap algorithm is to minimize the maximum delay from the primary input pins to the primary output pins in the final circuit.

Since a performance driven placement algorithm is used to place the simple gate circuit, the simple gates which are on the same critical path will be placed close together. Similarly, since the modified FlowMap algorithm tries to minimize the delays on the critical paths, it try to map the CLBs on a critical path into the same CLB. In other words, the modified FlowMap algorithm tries to map adjacent simple gates in the simple gate placement into the same CLBs. Therefore, the placement topology of the simple gate circuit reflects the placement topology of the final circuit. Under the assumption that the placement topology of the simple gate circuit reflects the placement topology of the final circuit, the modified FlowMap algorithm generates a circuit with the minimum maximum circuit delay.

One side effect of using the modified FlowMap algorithm is that the number of CLBs in the final circuits might be significantly larger than the number of CLBs in the circuits generated by the original FlowMap algorithm [6]. The increase in the number of CLBs are mainly caused by the heuristics used in the original and modified FlowMap algorithms during the post-processing process for minimizing the number of CLBs. In the FlowMap algorithms, a cut set that gives the minimum delay and the maximum volume (number of simple gates) is chosen to be mapped into an CLB to minimize the maximum circuit delay and the number of CLBs. In the unit delay model which is used in the original FlowMap algorithm [6], since all the nets have the same delay, there are many minimum delay cuts and it is easy to find a cut that packs large number of simple gates. However, in the non-unit delay model which is used in the modified FlowMap algorithm [7], there are few minimum delay cuts and they often pack fewer simple gates. Since the CLBs generated in the non-unit delay model contain fewer simple gates, more CLBs are needed to implement the initial circuit. More CLBs usually leads to more CLB delays, more and longer interconnections, and larger maximum circuit delay. The increases in the number of CLBs and interconnections also cause the resulting circuit to be hard to place and route. In order to reduce the number of CLBs in the circuits generated by the modified technology mapping algorithm, we round off the net delays to reduce the possible number of different net delay values. By rounding off the net delays, we are trading the accuracy of delay information with the number of CLBs. In [7], a relaxation technique was also introduced to reduce the number of CLBs by relaxing the minimum delay requirement.

5.3 Experimental Results

circuit name	specification no. of gates	no. of nets	package no.	Xilinx's tools no. of CLBs	no. of nets	max. delay	our mapping algorithm no. of CLBs	no. of nets	max. delay
cordic	68	91	3020PC84-100	17	40	41.3	23	46	39.6
count	129	164	3020PC84-100	34	69	70.0	34	69	62.5
comp	204	236	3020PC84-100	68	100	88.0	76	98	91.8
9sym	274	283	3042PQ100-100	96	105	81.3	97	106	81.1
my-add	208	241	3020PC84-100	24	57	96.1	30	66	87.7
C499	398	439	3030PC84-100	70	111	97.3	77	118	83.1
C880	418	478	3042PG132-100	123	183	136.7	140	200	-
alu2	492	502	3064PG132-100	216	226	206.8	247	257	-

Table 5.1. Placement driven technology mapping

In order to be able to compare the results generated by our technology mapping algorithm with the results generated by Xilinx's technology mapping algorithm, both our placement and placement driven technology mapping algorithms were implemented for Xilinx's 3000 series FPGAs. Our algorithms were implemented in C and integrated into Berkeley logic synthesis tool *sis*. The algorithms were tested on 8 MCNC multilevel benchmark circuits which are summarized in Table 5.1. The final results generated by Xilinx's physical design tools using Xilinx's technology mapping algorithm and our mapping algorithm are summarized in Table 5.1. To reduce the number of CLBs, the circuits generated by our technology mapping algorithm were obtained by allowing only 4 different net delay values for all the nets. However, as one can see, the modified FlowMap algorithm still generate more CLBs than the circuits generated by Xilinx's mapping algorithm. Notice that even though the circuits generated by our technology mapping algorithm still have more CLBs than the circuits generated by Xilinx's mapping algorithm, in most case, we were able to obtain final designs with smaller maximum circuit delays comparing with the results obtained by Xilinx's tools. We were not able to generate final designs for the last two circuits because they requires more CLBs than the number of CLBs in the Xilinx's FPGA chip.

5.4 Summary

In this chapter, we presented a placement driven technology mapping algorithm for Xilinx's FPGAs. In our approach, the simple gate circuit before technology mapping is first placed, and the placement information is used to guide the technology mapping process. The experimental results are encouraging.

CHAPTER 6

Routing Problems with Crosstalk Constraints

6.1 Introduction

As fabrication technology advances, minimum feature size for VLSI chips continues to decrease, and interconnection wires are placed in closer proximity in order to reduce interconnection delay and routing area. As circuit delay decreases, circuits will operate at higher frequencies. Consequently, reduction of crosstalks between interconnection wires is an important consideration in VLSI design. Crosstalks are mostly caused by the coupling capacitance between interconnection wires (nets). The crosstalk between two wires is proportional to their coupling capacitance, which in turn, is determined by their relative positions. The coupling capacitance between a pair of parallel wires is proportional to their coupling length (the length of their overlapping segments), and is inversely proportional to their separating distance. The coupling capacitance between a pair of orthogonal wires is negligible in comparison with the coupling capacitance between a pair of parallel wires. Consequently, it is reasonable to assume that there is crosstalk only between parallel wires. The crosstalk between two interconnection wires also depends on the frequencies of the signals traveling on the wires.

Crosstalks together with signal reflection and cell switching produce noises which could lead to unexpected circuit behavior. The total amount of crosstalk that a net can tolerate depends on the power of the circuit that drives the net and the sensitivity of the circuits that the net drives. Usually, in the design specification, the maximum tolerable crosstalk for each net that will guarantee the proper functionality of the circuit is given. Crosstalk in each net should not exceed the corresponding maximum tolerable crosstalk in a valid design. We call such constraints *crosstalk constraints*. To reduce noises in a design, it is also desirable that the total crosstalk among all the nets is minimized. Notice that crosstalks depend not only on the coupling capacitance between the nets, but also on the frequencies of the signals traveling on the nets. To simplify our presentation, we shall assume that the circuit operates at a fixed frequency and the value of the crosstalk in a net is directly proportional to the coupling capacitance between the net and its neighboring nets. Moreover, we shall use the terms "coupling capacitance" and "crosstalk" interchangeably.

Crosstalks between interconnection wires are determined by the routing of the interconnection wires. Therefore, it is important to design routing algorithms that are capable of generating routing solutions that satisfy the crosstalk constraints and minimize the total crosstalk amount all the nets. As traditional routing problems with objectives such as minimizing the total routing area are already known to be difficult problems, routing problems with the second objective of controlling the crosstalks between interconnection wires becomes even more difficult. The reason for this is that the crosstalks are determined not only by how each individual wire is routed, but also by the relative positions of the interconnection wires.

In order to reduce crosstalks in the nets, it is desirable to avoid long coupling wire segments, which is also a desirable feature from a yield enhancement point of view. Moreover, reduction in the coupling capacitance and crosstalks in the nets also lead to smaller signal propagation delays and smaller power dissipation.

6.2 Existing Algorithms

There are prior works on the crosstalk problem for the gridless routing model. In [4], Chen presented a gridless channel routing algorithm which generates routings that satisfy the crosstalk constraints. In the algorithm, a weighted constraint graph is used to represent the wire separation requirements [3,4]. Because it is not possible to obtain information about the actual spacing between the nets before they are routed, no crosstalk information is presented in the initial weighted constraint graph. As each net is routed, the crosstalk in the net due to all the routed nets is evaluated, and the spacing requirements between the routed nets and unrouted nets are updated. Since the routing model is gridless, it is always possible to satisfy all the crosstalk constraints by varying the spacings between the nets. Since the routing algorithm does not consider the crosstalk effect of the net that is being routed on the unrouted nets, it is possible that the nets that were routed early will force the remaining nets to be routed far apart. In [12], Dai presented a gridless general routing algorithm which is also capable of satisfying crosstalk constraints. Rubber bands are used to approximate the routings of the nets. If potential crosstalk problems are detected on some of the nets, their corresponding rubber bands are stretched to allow more spacings between these nets. Because the rubber bands are elastic and the routing region is not gridded, the crosstalk constraints can always be satisfied.

6.3 Gridded Channel and Switchbox Routing Problems

In our algorithms, we will deal specifically with the gridded channel and switchbox routing problems. Unlike the gridless routing model, wire segments (a wire segment is a piece of interconnection wire between two vias) in the gridded routing model are to be placed on grid edges, and the grid spacings can not be adjusted to satisfy the crosstalk constraints. Consequently, the gridded routing problems with crosstalk constraints are more difficult to solve in comparison with the gridless routing problem with crosstalk constraints. In Chapter 7, we shall present a channel routing algorithm which is capable of satisfying crosstalk constraints [21]. In Chapter 8, we shall present a switchbox routing algorithm which is capable of satisfying crosstalk constraints and minimizing the total crosstalk among all the nets [22].

CHAPTER 7

A Gridded Channel Routing Algorithm with Crosstalk Constraints

7.1 Introduction

In this chapter, we shall present a channel routing algorithm which is capable of satisfying crosstalk constraints [21,20]. As was discussed in Chapter 6, it is difficult to consider the crosstalks between wires in a global way during the routing process. Therefore, in our algorithm, instead of developing heuristic routing algorithms which try to route the interconnections with crosstalk constraints in mind, we propose a new transformational approach which modifies a routing solution to obtain a new one that will satisfy the crosstalk constraints. In our approach, a conventional channel routing algorithm is first used to determine a routing solution with the objective of minimizing the channel height. The tracks in the routing solution are then permuted to produce a routing solution that will satisfy the crosstalk constraints. Permutation of the tracks will change the relative positions of the horizontal wire segments and the lengths of the vertical wire segments in the routing solution, which will in turn, change the values of the crosstalks in the nets. Note that the tracks in a routing solution can not be permuted freely. Rather, they must satisfy a set of vertical constraints on their relative vertical positions. Since the tracks are only permuted, the height of the channel remains unchanged after the permutation. One advantage of this approach is that it enables us to use existing channel routing algorithms that can produce very good routing solutions for the objective of minimizing the channel height. Thus, together with a good permutation algorithm, we will be able to obtain good routing solutions (in terms of channel height) which satisfy the crosstalk constraints with a substantial margin as will be shown latter. A similar permutation approach was used in [9] to transform a two-layer routing solution into a three or four-layer routing solution.

Since there are many good existing channel routing algorithms [1,49,36], we shall focus our attention on the problem of track permutation in a given initial routing solution with the objective of satisfying as many crosstalk constraints as possible. Since crosstalk between two wires decreases as the distance between them increases, and since it is likely that the crosstalk between two non-adjacent wires is shielded by wires between them, to reduce the computation complexity, it is reasonable to assume that crosstalks only exist between wires in adjacent rows or columns. The formal definition for the permutation problem under this assumption is given in Section 7.2. Section 7.3 shows that the permutation problem is NP-complete. In Section 7.4, we present a mixed integer linear programming formulation for the permutation problem. Section 7.5 extends the mixed ILP formulation to model crosstalks between wires which are not adjacent. To solve the mixed ILP problem efficiently, in Section 7.6, we present techniques to reduce the number of variables and constraints in the mixed ILP formulation. Experimental results are presented in Section 7.7. The novelty of the mixed ILP formulation together with effective procedures to reduce the number of variables and constraints enable us to obtain excellent results for large benchmark examples such as Deutsch difficult problem.

7.2 Problem Formulation

Since the separation between two adjacent rows or columns is fixed in a gridded channel, the crosstalk between two wire segments on adjacent rows or columns is proportional to the coupling length between them. Without loss of generality, we shall set the proportion constant to 1 and consider the coupling length to be the same as the crosstalk. The crosstalk C_k in net k is the sum of all the crosstalks in the wire segments of the net. As an example, the crosstalk in net 4 in Figure 7.1a is $3+3+4+5+2 = 17$. Note that the coupling lengths between the left vertical wire segment of net 4 and the vertical wire segments to its left and right are both 3, the coupling lengths between the horizontal wire segment of net 4 and the horizontal wire segments below and above it are 4 and 5 respectively, and the coupling length between the right vertical wire segment of net 4 and the vertical wire segment to its left is 2. The maximum tolerable crosstalks in nets are given constants derived by the designer according to the electrical property of the circuit components. The difference between the maximum tolerable crosstalk in net k and C_k is called the *crosstalk slack* of net k. The minimum value of the crosstalk slacks among all the nets is called the *minimum slack*, denoted by $minslack$. In order to satisfy all the crosstalk constraints, it is necessary that $minslack \geq 0$. Moreover, $minslack$ measures the safety margin of the crosstalks in the nets, and a good routing algorithm should generate routing solutions with *minslack* as large as possible.

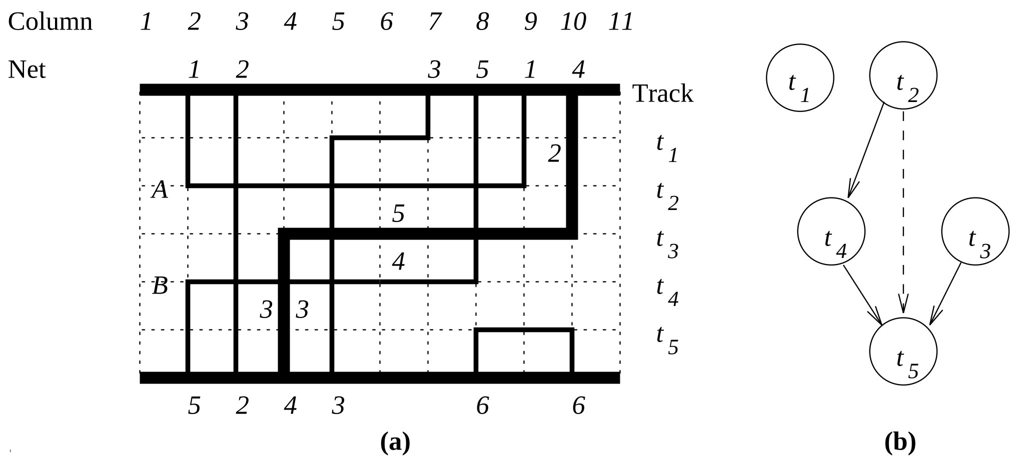


Figure 7.1. Crosstalk and vertical constraint graph

A channel is a rectangular routing region with pins placed on the top and the bottom boundaries of the region. For a gridded channel, it is assumed that there is an orthogonal gird superimposed on the channel. All pins along the top and bottom boundaries of the channel are at the grid points (Figure 7.1a). There are two layers available for routing, one for the horizontal wires and the other one for the vertical wires. In a valid routing solution, wires are routed only along grid edges and no two wires of different nets can overlap. The channel routing problem is to connect the pins in the same net so that the number of tracks is minimized and *minslack* is maximized.

For a given routing solution, track t_i is constrained to be placed above track t_j if there is a via in t_i which is connected to a pin on the top boundary that is directly above a via in t_j which is connected to a pin on the bottom boundary. Such a constraint will be referred to as a vertical constraint from t_i to t_j. For example, in Figure 7.1a, there is a vertical constraint from track t_2 to track t_4 because of via A and via B. Note that unlike the vertical constraints in [49] which are defined to be between nets and are independent of any routing solution, the vertical constraints here are defined as between tracks in a given initial routing solution. A vertical constraint from t_i to t_j and a vertical constraint from t_j to t_k *imply* a vertical constraint from t_i to t_k. For example, in Figure 7.1a, a vertical constraint from t_2 to t_4 and a vertical constraint from t_4 to t_5 imply a vertical constraint from t_2 to t_5. Vertical constraints which are not implied by other vertical constraints are called *essential vertical constraints*, and vertical constraints which are implied by other vertical constraints are called *extended vertical constraints*. Note that the vertical constraints for a valid routing solution are never circular. If all the essential vertical constraints are satisfied, then the extended vertical constraints are also satisfied. A *vertical constraint graph* is a directed graph with vertices representing the tracks and edges representing the vertical constraints (Figure 7.1b). In the vertical constraint graph, solid edges represent the essential vertical constraints and dotted edges represent the extended vertical constraints. Given an initial routing solution, by scanning the channel column by column, a set of vertical constraints which includes all the essential vertical constraints can be found. The vertical constraint graph can then be constructed easily from this set of vertical constraints. Since the essential vertical constraints for a given routing solution are never circular, the solid edges which represent the essential vertical constraints form a forest.

Given a routing solution which uses n tracks $t_1, t_2, ..., t_n$, a *valid permutation* of the tracks $t_{\pi_1}, t_{\pi_2}, ..., t_{\pi_n}$ (t_1 becomes t_{π_1}, t_2 becomes t_{π_2}, and so on) should satisfy all the vertical constraints. In the rest of this chapter, we shall refer to a "valid permutation" simply as a "permutation". Note that a permutation of the tracks corresponds to a topological ordering of the vertices in the vertical constraint graph. Obviously, a permutation of the tracks corresponds to a valid routing solution.

Since *minslack* measures the safety margin of the crosstalks in the nets, the objective of the permutation problem is to find a permutation of the tracks so that the value of *minslack* is maximized.

7.3 NP-Completeness of the Permutation Problem

Intuitively, the permutation problem is a difficult problem because crosstalks in the nets do not depend on the absolute locations of the tracks, but rather depend on the relative positions of the tracks. Here, we shall show that the permutation problem is a difficult problem by showing that it is NP-complete.

Definition: *Bottleneck Hamiltonian Path Problem*: Given a complete graph $G = (V, E)$ with m vertices $V = v_1, v_2, ..., v_m$ and positive weight w_{ij} for each edge $(v_i, v_j) \in E$, is there a Hamiltonian path such that the largest edge weight in the path is at most W, where W is a given positive number?

It is easy to show that the Bottleneck Hamiltonian Path Problem is an NP-complete problem [26].

Theorem: The permutation problem is NP-complete.

Proof: The permutation problem can be formulated as a decision problem which asks whether there is a permutation of the tracks in a given initial routing solution so that the resulting *minslack* is at least M where M is a given constant.

For a given permutation of the tracks, the crosstalks and the crosstalk slacks of the nets can be easily computed. Therefore, *minslack* can also be easily computed. After *minslck* is computed, it is easy to check if $minslack \geq M$. Therefore, the permutation problem is in the complexity class NP.

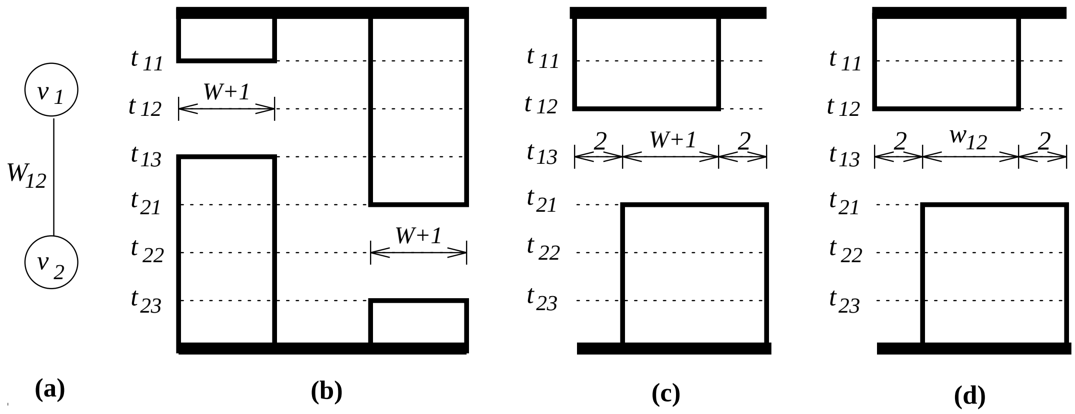


Figure 7.2. Translating Bottleneck Hamiltonian Path Problem to permutation problem

To prove that the permutation problem is NP-hard, we shall reduce the Bottleneck Hamiltonian Path Problem to the permutation problem. For a given instance of the Bottleneck Hamiltonian Path Problem, we shall construct a corresponding instance of the permutation problem so that the permutation problem has a solution if and only if the Bottleneck Hamiltonian Problem has a solution. The construction is carried out as follows. For each vertex

v_i in the Bottleneck Hamiltonian Path Problem, three tracks t_{i1}, t_{i2}, and t_{i3} are constructed. Moreover, the tracks in the initial routing solution is in the order t_{11}, t_{12}, t_{13}, t_{21}, t_{22}, t_{23}, ..., t_{m1}, t_{m2}, t_{m3} where m is the number of vertices in the complete graph in the Bottlenecked Hamiltonian Path Problem. For example, for the Bottleneck Hamiltonian Problem instance shown in Figure 7.2a, six tracks t_{11}, t_{12}, t_{13}, t_{21}, t_{22}, and t_{23} are constructed (Figure 7.2b). After the tracks are constructed, a set of nets is constructed and the nets are placed on the tracks. The nets are constructed in such a way that there is a Hamiltonian path v_i, v_j, ..., v_l in G with the largest edge weight no larger than W if and only if there is a corresponding permutation of the tracks t_{i1}, t_{i2}, t_{i3}, t_{j1}, t_{j2}, t_{j3}, ..., t_{l1}, t_{l2}, t_{l3} with the maximum crosstalk in the nets no larger than W. The maximum tolerable crosstalk in each net is W, and the permutation problem asks whether there is a permutation of the tracks so that the resulting routing solution has $minslack \geq 0$.

We shall now describe how to construct the corresponding nets for the complete graph in a Bottlenecked Hamiltonian Path Problem, and we shall illustrate the construction using the two vertex graph in Figure 7.2. As will be described, all the nets are constructed in such a way that no coupling vertical wire segments exist in any two adjacent columns, and crosstalks only exist between horizontal wire segments.

To force t_{i1}, t_{i2}, and t_{i3} to be adjacent and in the order of t_{i1}, t_{i2}, and t_{i3} after permutation, nets are added to the three tracks to force t_{i1} to be above but not adjacent to t_{i3} and to force t_{i2} not to be adjacent to any track except t_{i1} and t_{i3}. To force t_{i1} to be above but not adjacent to t_{i3}, a pair of nets with horizontal spans equal to $W + 1$ are added to t_{i1} and t_{i3} as shown in Figure 7.2b. The vertical constraint introduced this way forces t_{i1} to be above t_{i3}, and the coupling length of $W+1$ between t_{i1} and t_{i3} prevents t_{i1} from being adjacent to t_{i3} in the final permutation. To force t_{i2} not to be adjacent to any track except t_{i1} and t_{i3}, a pair of nets with horizontal spans equal to $W+3$ are added between t_{i2} and all other tracks. Figure 7.2b shows how to add such a pair of nets between t_{12} and t_{21}. Note the nets added this way do not impose any additional vertical constraints between the tracks. To avoid crosstalks between vertical wire segments, each pair of nets constructed this way is placed at a distance of at least 2 grid units away from other nets in the horizontal direction. Note that no crosstalk is caused by t_{i2} on t_{i1} and t_{i3} even though t_{i2} is adjacent to t_{i1} and t_{i3} in the final permutation.

Next, nets are added to the tracks so that there will be a pair of nets on t_{i1} and t_{j3} with crosstalk w_{ij} if the two tracks become adjacent in the final permutation. Moreover, other nets on tracks t_{i1} and t_{j3} has 0 crosstalk on them. To do this, a pair of nets with horizontal spans equal to $w_{12}+2$ are added to t_{i1} and t_{j3} for each pair of i and j. Figure 7.2c shows how to add such a pair of nets between track t_{11} and track t_{23}. Once again, to avoid crosstalks between vertical wire segments, each pair of nets constructed this way is placed at a distance of at least 2 away from other nets in the horizontal direction.

Obviously, a permutation of the tracks $t_{i1}, t_{i2}, t_{i3}, t_{j1}, t_{j2}, t_{j3}, ..., t_{l1}, t_{l2}, t_{l3}$ with $minslack \geq 0$ (no net has crosstalk larger than W) exists if and only if there exists a Hamiltonian path v_i, v_j, ..., v_l in the graph for the Bottlenecked Hamiltonian Path Problem with the maximum edge weight not exceeding W.

7.4 A Mixed Integer Linear Programming Formulation

Since the track permutation problem is an NP-complete problem, instead of designing heuristics for the solution of the problem, we shall formulate the problem as a mixed ILP problem. In this section, we shall formulate the problem under the assumption that crosstalks only exist between adjacent wire segments. In next section, we shall extend the mixed ILP formulation to model crosstalks between wire segments which are not adjacent. It should be noted that such a formulation is far from being straight forward. Instead, we hope that one of the major contribution of this formulation is to demonstrate how similar problems can be formulated in a similar way.

Since crosstalks in a wire segment come from wire segments in adjacent tracks and adjacent columns, in order to compute the crosstalks between wire segments, the linear programming problem should be formulated in such a way that adjacency information between tracks is available readily. Since the vertical constraints are constraints on the relative positions of the tracks, the linear programming problem should be formulated in such a way that the relative positions between any two tracks can be checked easily. Even though these two kinds of information are closely related, it is not trivial to express them in an unified framework in the linear programming formulation.

Let t_1, t_2, ..., t_n denote the n tracks in the initial solution. To represent a permutation of the tracks, n 0-1 integer variables $T_{i1}, T_{i2}, ..., T_{in}$ are defined for each track t_i. $T_{ij} = 1$ if and only if t_i in the initial solution is permuted to t_j in the final solution, and $T_{ij} = 0$ otherwise.

These variables will be referred to as T-variables. There is a total of n^2 T-variables. Since each track in the initial solution is permuted to a distinct track in the final solution, and each track in the final solution is permuted from a distinct track in the initial solution, the following constraints are needed to enforce the condition that the T-variables represent a permutation of the tracks:

$$\sum_{j=1}^{n} T_{ij} = 1 \text{ for } 1 \leq i \leq n \tag{7.1}$$

$$\sum_{i=1}^{n} T_{ij} = 1 \text{ for } 1 \leq j \leq n \tag{7.2}$$

There is a total of $2n$ constraints on the T-variables.

To obtain adjacency information between tracks, for $1 \leq i < j \leq n$, a variable P_{ij} is defined for each pair of tracks t_i and t_j. $P_{ij} = 1$ if and only if t_{π_i} and t_{π_j} are adjacent in the final solution, and $P_{ij} = 0$ otherwise. These variables will be referred to as P-variables. There is a total of $\frac{n(n-1)}{2}$ P-variables. Obviously,

$$P_{ij} \geq 0 \text{ for } 1 \leq i < j \leq n \tag{7.3}$$

Since there are exactly $n-1$ pairs of adjacent tracks in the final solution,

$$\sum_{i<j} P_{ij} = n-1 \text{ for } 1 \leq i < j \leq n \tag{7.4}$$

So that the T-variables and the P-variables will be consistent in the sense that they represent the same permutation, the following constraints are needed. For two tracks t_i and t_j in the initial solution to be adjacent in the final solution, they must have been permuted to t_r and t_{r+1} in the final solution for some $1 \leq r \leq n-1$. Therefore, the following constraints are needed to force P_{ij} to be 1 if t_i and t_j in the initial solution are permuted to become adjacent in the final solution:

$$T_{i1} + T_{i2} + T_{j1} + T_{j2} \leq 1 + P_{ij} \tag{7.5}$$

$$T_{i2} + T_{i3} + T_{j2} + T_{j3} \leq 1 + P_{ij} \tag{7.6}$$

$$\ldots \tag{7.7}$$

$$T_{i(n-1)} + T_{in} + T_{j(n-1)} + T_{jn} \leq 1 + P_{ij} \tag{7.8}$$

There is a total of $\frac{n^3-n^2+2}{2}$ constraints on the P-variables. If t_{π_i} is adjacent to t_{π_j} in the final solution, the left hand side of one of the constraints for P_{ij} must be equal to 2, and P_{ij} is forced to be at least 1. On the other hand, if t_i is not adjacent to t_j in the final solution, the left hand sides of all the constraints for P_{ij} are smaller than 2, and the constraints do not force P_{ij} to assume any particular value greater than or equal to 0. Since there are exactly $n-1$ pairs of adjacent tracks in the final solution, $n-1$ P-variables are forced to be at least 1. Since the sum of all the P-variables is equal to $n-1$, and no P-variable may assume a negative value, these $n-1$ P-variables must assume value 1 and all other P-variables must assume value 0. Even though P-variables will assume only values of 0 or 1, they can be defined as continuous variables to reduce the number of integer variables in the mixed ILP formulation.

The value of t_{π_i} in the final solution can be expressed as $\sum_{k=1}^{n} kT_{ik}$. Therefore, the vertical constraint from t_i to t_j (that is, t_i must be permuted to a track which is above the track to which t_j is permuted) can be expressed as

$$\sum_{k=1}^{n} kT_{ik} \leq \sum_{k=1}^{n} kT_{jk} \tag{7.9}$$

In the worst case, there could be $\frac{n(n-1)}{2}$ vertical constraints. Therefore, $\frac{n(n-1)}{2}$ such linear constraints may be constructed.

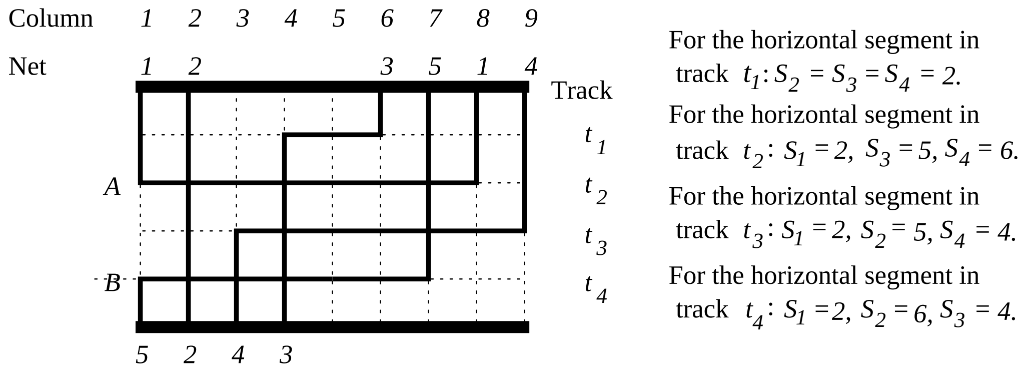
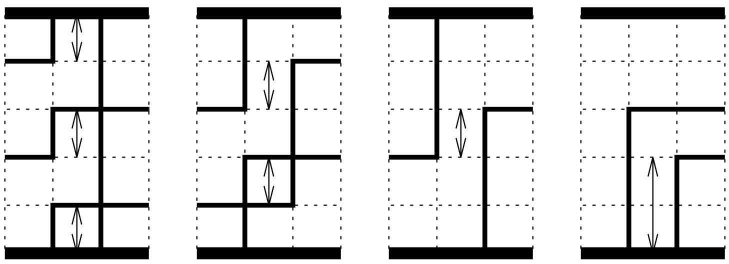


Figure 7.3. Horizontal coupling length

To compute the crosstalks in a net in the final solution, the crosstalk in each wire segment of the net needs to be computed. To compute the crosstalk in a horizontal wire segment in the final solution which was in track t_i in the initial solution, let S_j denote the coupling length between the wire segment and all the wire segments on track t_j in the initial solution (Figure 7.3). Since the locations of the wire segments on the tracks do not change after the permutation, S_j is a constant which can be computed from the initial solution. The crosstalk in a horizontal wire segment in the final solution can be expressed as

$$\sum_{all j<i} P_{ji}S_j + \sum_{all j>i} P_{ij}S_j \tag{7.10}$$

Crosstalk in a vertical wire segment is harder to compute in comparison with the crosstalk in a horizontal wire segment since the length of the vertical wire segment is determined by the permutation. A vertical wire segment can be either a feedthrough which connects a pin on the top boundary of the channel to a pin on the bottom boundary of the channel in the same column, a dogleg which connects two horizontal wire segments on two different tracks, or a vertical interconnection which connects a horizontal wire segment to a pin on the top or bottom boundary of the channel. Figure 7.4 shows how the crosstalks on vertical wire segments are computed. In Figure 7.4, only crosstalks from left adjacent wire segments are shown.



	(a)	(b)	(c)	(d)
Exact crosstalk:	*1+1+1=3*	*1+1=2*	*1*	*2*
Approximate crosstalk:	*1+1+1=3*	*3/2=1.5*	*1*	*(2+3)/2=2.5*

Figure 7.4. Crosstalk on vertical wire segments

Since a feedthrough spans the whole column and has a constant length of $n+1$, the crosstalk in it is the sum of the lengths of all the vertical wires in adjacent columns (Figure 7.4a). The final length of a vertical wire segment connecting a pin on the top boundary to a via in track t_i in the initial solution can be expressed as $\sum_{k=1}^{n} kT_{ik}$, and the final length of a vertical wire segment connecting a pin on the bottom boundary to a via in track t_j in the initial solution can be expressed as $n+1-\sum_{k=1}^{n} kT_{jk}$. To compute the length of a dogleg d with end points at tracks t_i and t_j in the initial solution, a variable D_d is defined. Since d will have its endpoints at tracks $\sum_{k=1}^{n} kT_{ik}$ and $\sum_{k=1}^{n} kT_{jk}$ in the final solution, the following constraints are introduced:

$$D_d \geq \sum_{k=1}^{n} kT_{ik} - \sum_{k=1}^{n} kT_{jk} \tag{7.11}$$

$$D_d \geq \sum_{k=1}^{n} kT_{jk} - \sum_{k=1}^{n} kT_{ik} \tag{7.12}$$

Note that these constraints only force D_d to be at least as large as the length of the dogleg in the final solution. Since the optimization process tries to maximize $minslack$, it will force the value of D_d to be sufficiently close to the length of d.

Since the locations of both endpoints of a dogleg are functions of the T-variables, it is not possible to express the exact crosstalk in the dogleg as a linear function of the T-variables unless the adjacent wire segment is a feedthrough. In our formulation, if an adjacent column is not empty and the wire segment(s) in the adjacent column is not a feedthrough, half of the length of the dogleg is used as an approximation of the crosstalk between the dogleg and the wire segment(s) in the adjacent column (Figure 7.4b). The error in the approximation is at most one half of the length of the dogleg in the final solution.

The crosstalks between vertical wire segments that connect pins on the boundary of the channel and horizontal wire segments come from either the configuration shown in Figure 7.4c or the configuration shown in Figure 7.4d. To compute the crosstalk between a wire segment which connects a pin on the top boundary to a horizontal wire segment in track t_i and a wire segment which connects a pin on the bottom boundary to a horizontal wire segment in track t_j (Figure 7.4c), a variable F which represents the crosstalk is defined. The following constraints on F are introduced:

$$F \geq 0 \tag{7.13}$$

$$F \geq \sum_{k=1}^{n} kT_{ik} - \sum_{k=1}^{n} kT_{jk} \tag{7.14}$$

Again, these constraints only force F to be at least as large as the crosstalk between the two wire segments, and the optimization process will force the value of F to be sufficiently close to the value of the crosstalk.

For the crosstalk between two vertical wire segments as shown in Figure 7.4d, it is not possible to express the crosstalk between them as a linear function of existing variables. In our formulation, the average wire length of the two wire segments is used to approximate the crosstalk between them as shown in Figure 7.4d. The error in the approximation is half of the difference between the lengths of the two wire segments.

After the crosstalks in different wire segments in the final solution are computed, the crosstalk in a net can be expressed as the sum of the crosstalks in all the segments of the net. The crosstalk slack of a net is the difference between the maximum tolerable crosstalk in the net and the crosstalk in the net. Since *minslack* is the minimum crosstalk slack among all the nets, the following constraints as introduced:

$$minslack \leq slack_i \text{ for every net } i \tag{7.15}$$

The objective of the optimization problem is to maximize *minslack*.

7.5 Crosstalks between Non-Adjacent Wire Segments

The mixed ILP formulation presented in the previous section considers only crosstalks between adjacent wire segments. However, the formulation can be extended to take into account crosstalks between non-adjacent wire segments. In this section, we shall describe how to formulate the crosstalk between two parallel wire segments which are k tracks or columns apart in the final solution.

To compute the crosstalks between horizontal wire segments which are k tracks apart in the final solution, a variable Q_{ij} is defined for each pair of tracks t_i and t_j for $1 \leq i < j \leq n$. $Q_{ij} = 1$ if and only if t_{π_i} and t_{π_j} are k tracks apart in the final solution, and $Q_{ij} = 0$ otherwise. These variables will be referred to as Q-variables. Obviously,

$$Q_{ij} \geq 0 \text{ for } 1 \leq i < j \leq n \tag{7.16}$$

Since there are exactly $n-k$ pairs of tracks which are k tracks apart in the final solution,

$$\sum_{i<j} Q_{ij} = n - k \text{ for } 1 \le i < j \le n \tag{7.17}$$

So that the T-variables and the Q-variables will be consistent in the sense that they represent the same permutation, the following constraints similar to the ones for P-variables are needed. For two tracks t_i and t_j in the initial solution to be k tracks apart in the final solution, they must have been permuted to t_r and t_{r+k} for some $1 \le r \le n-k$. Therefore, the following constraints are needed to force Q_{ij} to be 1 if t_i and t_j in the initial solution are permuted to tracks that are k tracks apart in the final solution:

$$T_{i1} + T_{i(1+k)} + T_{j1} + T_{j(1+k)} \le 1 + Q_{ij} \tag{7.18}$$

$$T_{i2} + T_{i(2+k)} + T_{j2} + T_{j(2+k)} \le 1 + Q_{ij} \tag{7.19}$$

$$\ldots \tag{7.20}$$

$$T_{i(n-k)} + T_{in} + T_{j(n-k)} + T_{jn} \le 1 + Q_{ij} \tag{7.21}$$

There is a total of $\frac{n^3-kn^2+nk-n+2}{2}$ constraints on the Q-variables. If t_{π_i} is k tracks apart from t_{π_j} in the final solution, the left hand side of one of the constraints for Q_{ij} must be equal to 2, and Q_{ij} is forced to be at least 1. On the other hand, if t_i is not k tracks apart from t_j in the final solution, the left hand sides of all the constraints for Q_{ij} are smaller than 2, and the constraints do not force Q_{ij} to assume any particular value greater than or equal to 0. Since there are exactly $n-k$ pairs of adjacent tracks in the final solution, $n-k$ Q-variables are forced to be at least 1. Since the sum of all the Q-variables is equal to $n-k$, and no Q-variable may assume a negative value, these $n-k$ Q-variables must assume value 1 and all other Q-variables must assume value 0. Even though Q-variables will assume only values of 0 or 1, they can be defined as continuous variables to reduce the number of integer variables in the mixed ILP formulation.

We assume that the crosstalk between two wire segments is inversely proportional to their separating distance. Consequently, crosstalk in a horizontal wire segment from nets which are k tracks away in the final solution can be expressed as

$$\frac{1}{k}\left(\sum_{all j<i} Q_{ji}S_j + \sum_{all j>i} Q_{ij}S_j\right) \tag{7.22}$$

Since the separating distance between two vertical wire segments does not change after the permutation, crosstalk on a vertical wire segment from vertical wire segments which are k columns apart can be computed in the same way as described in the previous section except the computed value should be divided by k.

7.6 Reducing the Size of the Mixed ILP Formulation

For large problem instances, the problem size of the mixed ILP formulation might be very large. For example, for Deutsch difficult problem, there will be 1053 variables and 3681 constraints in the mixed ILP formulation. In order to solve the mixed ILP problems efficiently, the number of variables, constraints, and terms in the constraints need to be reduced. In this section, we first present techniques that can be used to reduce the number of variables. We then present techniques that can be used to reduce the number of constraints and the number of terms in the constraints. To reduce the number of variables, we try to determine the values for the variables before the mixed ILP problem is solved. Note that if the value of a variable is determined, the corresponding constraints on the variable can be deleted. Moreover, the corresponding terms of these variables in the undeleted constraints can be deleted.

As was mentioned in Section 7.2, a vertical constraint from t_i to t_k and a vertical constraint from t_k to t_j imply an extended vertical constraint from t_i to t_j. If there is an extended vertical constraint from t_i to t_j, there is at least one track between t_i and t_j in the final solution, which means t_i can not be adjacent to t_j in the final solution. Therefore, P_{ij} can be set to 0. Similar techniques can be applied to reduce the number of Q-variables.

From the vertical constraint graph, the minimum number of tracks above or below track t_i in the final solution can be determined. For any vertical constraint (either essential or extended) from t_i to t_j, t_i must be placed above t_j in the final solution. Therefore, the minimum number of tracks below t_i is the number of outgoing edges from vertex t_i, and the minimum number of tracks above t_i is the number of incoming edges to vertex t_i in the vertical constraint graph. If the minimum number of tracks above t_i is a, track t_i can not be permuted to any of the first a tracks (from the top) of the channel in the final solution. Therefore, all the variables T_{ij} with $j \leq a$ can be set to 0. Similarly, if the minimum number of tracks below t_i is b, all variables T_{ik} with $k \geq b$ can be set to 0.

Consider the variable F which represents the crosstalk between two vertical wire segments as shown in Figure 7.4c. If the minimum number of tracks below t_i plus the minimum number of tracks above t_j is larger than $n-2$, where n is the number of tracks, there will be no crosstalk between the two wire segments. In this case, F can be set to 0.

Even though there may be many vertical constraints in a routing solution, only a few of them are essential. Since the edges representing essential vertical constraints in the vertical constraint graph form a forest, the number of essential vertical constraints is smaller than the number of tracks in the channel. To satisfy all the vertical constraints, it is only necessary to satisfy all the essential vertical constraints. Consequently, only essential vertical constraints need to be included in the mixed ILP formulation.

Consider a constraint for variable P_{ij} of the form T_{ik} $+T_{i(k+1)}$ $+T_{jk}$ $+T_{j(k+1)} \leq 1 + P_{ij}$. Suppose that variable T_{ik} is set to 0 according to the techniques discussed above. Since t_i in the initial solution will not be permuted to t_k in the final solution, permuting t_j in the initial solution to t_{k+1} in the final solution does not force P_{ij} to be 1 in this constraint. Therefore, both T_{ik} and $T_{j(k+1)}$ can be deleted from the constraint. Similarly, if $t_{i(k+1)}$ is set to 0 according to the techniques discussed above, both T_{jk} and $T_{i(k+1)}$ can be deleted from the constraint. If all the terms on the left hand side of the constraint can be deleted, the constraint can be deleted completely. Similar techniques can be used to reduce the number of constraints and the number of terms in the constraints for Q-variables.

Reduction in the number of variables, constraints, and terms in the undeleted constraints depends on the problem instance. In all our experiments, large reductions were attained. For example, for Deutsch difficult problem, we were able to reduce the number of variables from

1053 in the initial formulation to 246 in the final formulation and the number of constraints from 3681 in the initial formulation to 423 in the final formulation.

7.7 Experimental Results

Our algorithm was implemented in C and tested on an IBM RISC6000 workstation. The Optimization Subroutine Library (OSL) distributed by IBM was used to solve the mixed integer linear programming problem. Since crosstalks are mostly caused by coupling capacitance between adjacent wires, only crosstalks between adjacent wire segments are formulated in our experiments. As was described in Section 7.5, crosstalks between non-adjacent wire segments can also be formulated in a similar fashion. Our algorithm was tested on three large benchmark circuits as well as many randomly generated circuits. The initial routing results for the three benchmark circuits and a small randomly generated circuit are summarized in Table 7.1. The circuit YK3C is an example from [49], the circuit D1 is from the GTE layout published in [47], and Deutsch is the well known Deutsch difficult problem [1]. The small randomly generated circuit is presented for illustration purpose.

cir. name	initial routing solution		
	no. of tracks	no. of columns	no. of nets
random	5	20	10
YK3C	18	79	54
D1	18	155	65
Deutsch	19	174	72

Table 7.1. Circuit specifications

The mixed ILP formulations of the circuits before and after variable and constraint reduction are summarized in Table 7.2. We were able to reduce the number of variables and constraints in the mixed ILP formulation significantly for all the circuits. Note that for Deutsch difficult problem, even though the initial problem formulation has the largest problem size, it has relatively small problem size after variable and constraint reduction. This is mainly because the initial routing solution for Deutsch difficult problem is a very compact one. Consequently, there are many vertical constraints in the initial solution and there is little freedom for moving the tracks around. The experimental results presented later in this section confirm this.

cir. name	before reduction		after reduction	
	no. of variables	no. of const.	no. of variables	no. of const.
random	94	104	31	54
YK3C	713	2910	347	821
D1	941	3062	409	1204
Deutsch	1053	3681	246	423

Table 7.2. Mixed ILP formulation

Since no maximum tolerable crosstalks in the nets were specified for any of the benchmarks circuits, we assign the maximum tolerable crosstalks in three different ways. In all the experimental results presented here, crosstalks in wire segments are computed as the exact coupling lengths of the immediate adjacent wires (not the approximation used in the linear formulation). In many cases, instead of computing the maximum tolerable crosstalk for each net, the designer specifies only one number as the maximum tolerable crosstalk for all the nets. Therefore, in our first experiment, uniform maximum tolerable crosstalks are assigned to all the nets. In our second experiment, the maximum tolerable crosstalk in each net is computed to be proportional to the total horizontal span of the net. The intuition behind this is that the nets with relatively longer horizontal spans usually tend to have relatively larger crosstalks in them, and the designer might design the circuit to accommodate the relatively larger crosstalks in these nets. In this approach, instead of using the crosstalk constraints to drive the routing process, the routing results are used to feedback to influence the circuit design. However, in most cases, the maximum tolerable crosstalks in nets are computed according to the electrical property of the circuits that drive the nets, which could be some random numbers. Therefore, in our last experiment, the maximum tolerable crosstalks in the nets are assigned randomly. The experimental results for the three experiments are summarized in Table 7.3, 7.4, and 7.5 respectively.

cir. name	initial solution		final solution			improvement	
	min-slack	*total-cross*	*min-slack*	*total-cross*	run time(min.)	*min-slack*	*total-cross*
random	-12.0	200.0	7.0	156.0	0.1	19.0	44.0
YK3C	-10.0	2902.0	11.0	2752.0	6.6	21.0	150.0
D1	2.0	3818.0	31.0	3518.0	5.1	29.0	300.0
Deutsch	-11.0	7584.0	5.0	7570.0	0.3	16.0	14.0

Table 7.3. Routing solutions for circuits with uniform maximum tolerable crosstalk

cir. name	initial solution		final solution			improvement	
	min-slack	*total-cross*	*min-slack*	*total-cross*	run time(min.)	*min-slack*	*total-cross*
random	-8.0	200.0	3.0	166.0	0.1	11.0	34.0
YK3C	-4.0	2902.0	4.0	2840.0	7.1	8.0	62.0
D1	-5.0	3818.0	6.0	3798.0	20.1	11.0	20.0
Deutsch	0.0	7584.0	3.0	7570.0	0.4	3.0	14.0

Table 7.4. Routing solutions for circuits with variable maximum tolerable crosstalk

cir. name	initial solution		final solution			improvement	
	min-slack	*total-cross*	*min-slack*	*total-cross*	run time(min.)	*min-slack*	*total-cross*
random	-17.0	200.0	7.0	156.0	0.1	24.0	44.0
YK3C	-16.0	2902.0	37.0	2840.0	5.1	53.0	62.0
D1	-38.0	3818.0	57.0	3592.0	6.0	95.0	226.0
Deutsch	2.0	7584.0	12.0	7570.0	0.4	10.0	14.0

Table 7.5. Routing solutions for circuits with random maximum tolerable crosstalk

The quality of a routing solution is measured by the value of *minslack* and the sum of the crosstalk slacks in all the nets (*totalcross*). Smaller *totalcross* in the circuit mean a more reliable circuit design and lower power dissipation. In all experiments, we were able to improve the initial routing solutions which do not satisfy all the crosstalk constraints ($minslack \leq 0$) to the final routing solutions which satisfy the crosstalk constraints with a large margin.

Note that in all of our test cases, as the values of *minslack* increase, *totalcross* also increases. To have a better understanding on how the crosstalks in the nets change as the tracks are permuted, we take a closer look at the crosstalk slacks of the nets in the initial and final solutions of the small circuit "random". The initial routing solution for "random" is shown in Figure 7.5a, and the crosstalk slacks of the nets in the initial routing solution are listed in the first row of Table 7.6. The final routing solution for "random" is shown in Figure 7.5b, and the crosstalk slacks of the nets in the final routing solution are listed in the second row of Table 7.6. As shown in Table 7.6, the optimization process increases *minslack* by permuting the tracks so that the crosstalk slacks of the nets with small crosstalk slacks such as nets 3, 5, 6, and 7 increase. However, since there are only limited routing space, the decrease in the crosstalk in some nets usually leads to the increase in the crosstalk in some other nets. In “random”, the crosstalk slacks of the nets with large slacks such as nets 1, 2, 4, 8, 9 and 10 decrease. This process redistributes the slacks of nets with excessive slacks to nets with relatively small slacks so that the crosstalk constraints can be satisfied. In all our case, we were also able to increase the crosstalk slack of the nets with small (or negative) crosstalk slacks more than the decrease in the crosstalk slacks of other nets.

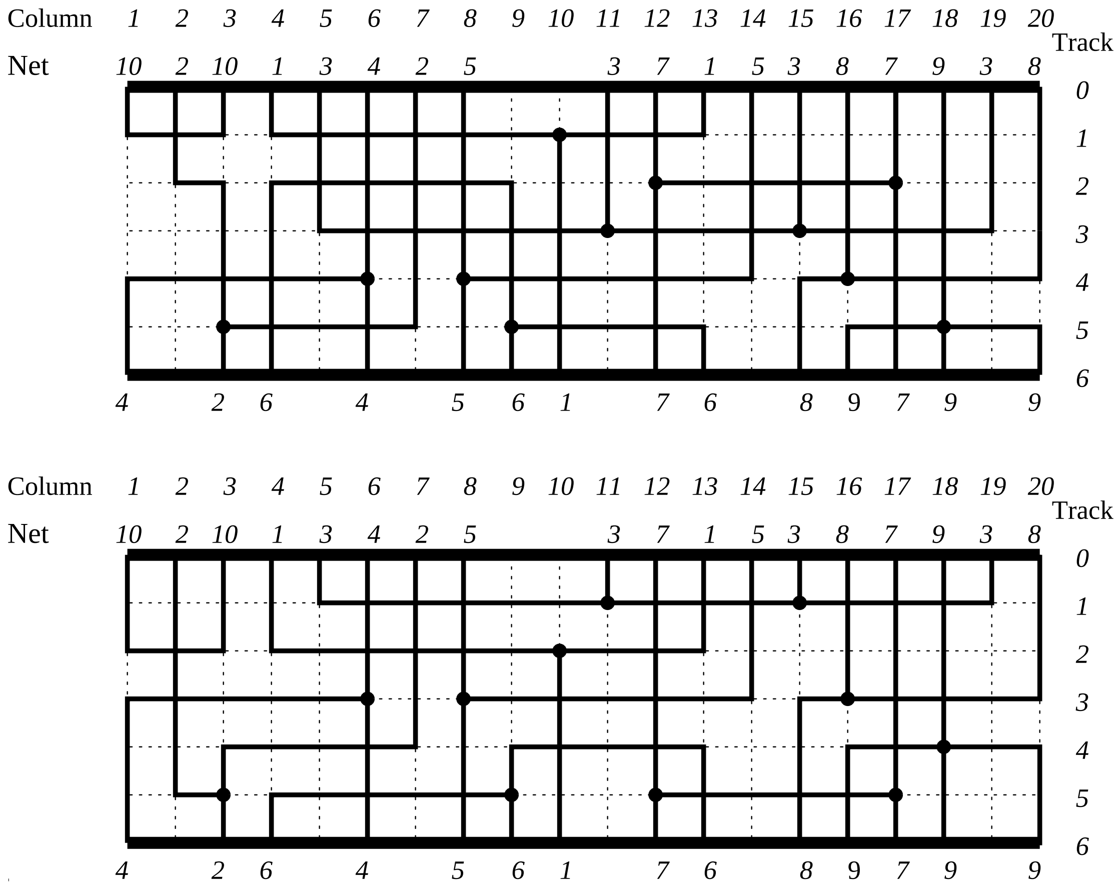


Figure 7.5. Permutation example

net no.	1	2	3	4	5	6	7	8	9	10
initial value	14.0	10.0	12.0	18.0	7.0	3.0	8.0	11.0	15.0	26.0
final value	7.0	9.0	15.0	16.0	13.0	17.0	12.0	19.0	14.0	22.0

Table 7.6. Crosstalk slacks of the nets in circuit "random"

Comparing the results in Table 7.3, 7.4, and 7.5, we observe that relatively smaller improvements were achieved in Table 7.4 where the maximum tolerable slacks of the nets are assigned proportional to net horizontal spans. This is mainly due to the fact that the maximum tolerable slacks of the nets are assigned in a way which confines the final solution largely to the initial routing solution, and it is hard to improve the crosstalks by altering the initial routing solutions. However, in most cases where the designer wants to control the routing process by using the crosstalk constraints, we were able to obtain significant improvements on the crosstalks as shown in Table 7.3 and 7.5.

Note that in all experiments, a relatively small amount of improvements on *minslack* were obtained for Deutsch difficult problem. As the permutation result in Table 7.7 where uniform maximum tolerable crosstalk is used shows, only three tracks in the initial solution were permuted to different tracks in the final solution. The reason for this is that the initial routing solution was very tightly packed, and there is little freedom to permute the tracks. This also explains the substantial reduction in the number of variables and the number of constraints in the mixed ILP formulation after applying the reduction techniques.

init. track	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
final track	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	17	18	16	19

Table 7.7. Permutation result for Deutsch difficult problem

7.8 Summary

In this chapter, we addressed the gridded channel routing problem with crosstalk constraints. We proposed a permutation approach to the solution of the gridded channel routing problem with crosstalk constraints by utilizing existing channel routing algorithms and permutation techniques. The permutation problem was proven to be NP-complete. A novel mixed ILP formulation and effective procedures for reducing the number of variables and constraints in the mixed ILP formulation were then presented. The experimental results are very promising.

CHAPTER 8

A Gridded Switchbox Routing Algorithm with Crosstalk Constraints

8.1 Introduction

In this chapter, we shall present a switchbox routing algorithm which is capable of satisfying crosstalk constraints and minimizing the total crosstalk among all the nets [22]. As was discussed in Chapter 6, it is difficult to consider the crosstalks between wires in a global way during the routing process. Therefore, in our algorithm, instead of developing heuristic routing algorithms which try to route the interconnections with crosstalk in mind, we propose a new approach to the switchbox and channel routing problems with crosstalk constraints. In this approach, a conventional routing algorithm is first used to generate an initial routing solution optimizing certain objectives such as minimizing the channel height for the channel routing problem. The wire segments in the initial routing solution are then re-assigned to satisfy the crosstalk constraint and to minimize the total crosstalk among all the nets. By re-assigning a wire segment, we mean to move a horizontal wire segment to another row (track) or a vertical wire segment to another column[1] while maintaining the validity of the routing solution.

For a channel routing problem with crosstalk constraints, a conventional channel routing algorithm is first used to generate a routing solution with the objective of minimizing the channel height. The horizontal wire segments in the initial routing solution are then re-assigned to satisfy the crosstalk constraints and to minimize the total crosstalk among all the nets. Re-assigning the horizontal wire segments changes the relative positions of the horizontal wire segments and the lengths of the vertical wire segments, which in turn, change the values of the crosstalks in the nets. The process of re-assigning horizontal (or vertical) wire segments will be referred to as the *assignment process*. For example, for the initial routing solution shown in Figure 8.1a, if the crosstalk in a net is computed as the total coupling length between the net and its adjacent wire segments, the crosstalk in net 2 is 7 and the total crosstalk among all the nets is 28. By re-assigning the horizontal wire segments in nets 1 and 5 to row 1, we obtain a new routing solution shown in Figure 8.1b in which the crosstalk in net 2 equal to 4 and the total crosstalk among all the nets equal to 20. Since the assignment process re-assign wire segments to existing rows, the assignment process does not increase the number of rows in the routing solution.

For a switchbox routing problem with crosstalk constraints, a conventional switchbox routing algorithm is first used to generate a valid routing solution. The assignment process is then carried out iteratively between the horizontal and vertical wire segments in order to satisfy

[1] including the possibility of not moving the wire segment at all.

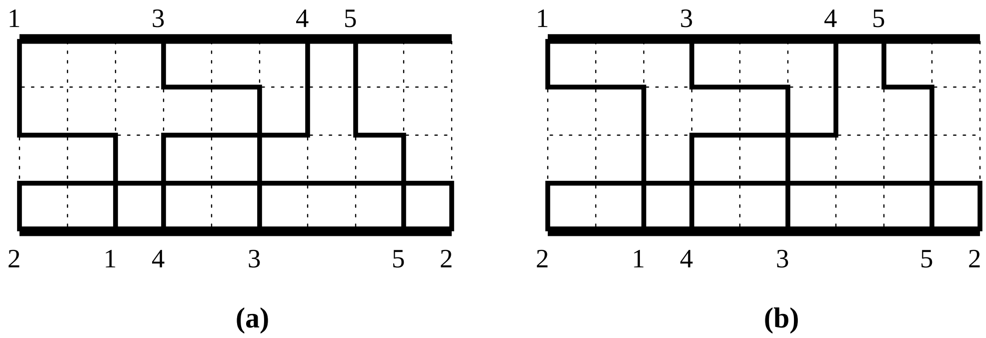


Figure 8.1. Assignment process for the channel routing problem

the crosstalk constraints and to minimize the total crosstalk among all the nets. The iterative assignment process continues until no more improvement can be obtained on the crosstalks. For example, for the initial routing solution shown in Figure 8.2a, the crosstalk in net 2 is 20 and the total crosstalk among all the nets is 66. By carrying out the assignment process iteratively between horizontal and vertical wire segments as shown in Figure 8.2, we obtain the final routing solution shown in Figure 8.2f in which the crosstalk in net 2 is 12 and the total crosstalk among all the nets is 50. Notice that the iterative assignment process does not change the number of rows and columns in the routing solution. Since each intermediate routing solution is a valid routing solution, the final routing solution is also a valid routing solution.

In [21], Gao proposed a permutation approach to the gridded channel routing problem with crosstalk constraints. In this approach, a conventional channel routing algorithm is first used to generate a routing solution with the objective of minimizing the channel height. The rows in the routing solution are then permuted to produce a routing solution that satisfies the crosstalk constraints. In comparison with the permutation approach, the assignment approach proposed here explores a much larger solution space and produces better routing solutions. For example,

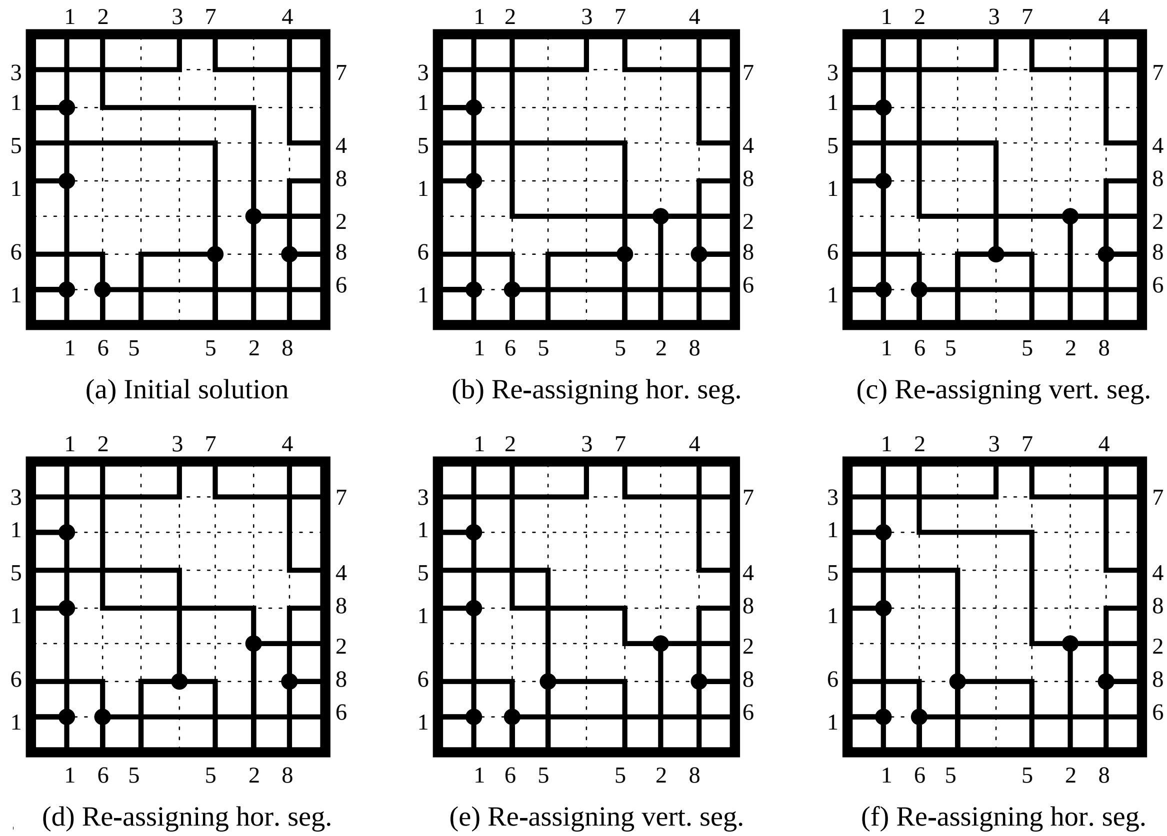


Figure 8.2. Iterative assignment process for the switchbox routing problem

for the initial routing solution shown in Figure 8.1a, while the assignment approach is capable of generating a new solution which satisfies the crosstalk constraints and minimizes the total crosstalk among all the nets, the permutation approach can not produce a new solution at all due to the vertical constraints between the rows. Also, for the initial routing solution shown in Figure 8.2a, while the assignment approach is capable of generating a new solution which satisfies the crosstalk constraints and minimizes the total crosstalk among all the nets, the permutation approach can not produce a new solution due to the fixed pins on the boundaries. Since switchbox routing problems usually have a large number of fixed pins on the boundaries, the permutation approach is quite ineffective in solving the switchbox routing problems with crosstalk constraints.

Since there are many good conventional switchbox and channel routing algorithms, we shall focus our attention on the assignment problem for a given initial routing solution. Since the assignment process for the vertical wire segments in a switchbox routing problem is similar to the assignment process for the horizontal wire segments, we shall only present our algorithm in terms of the assignment of the horizontal wire segments. The formal definition of the assignment problem is given in Section 8.2. In Section 8.3, we shall present a mixed integer linear programming (ILP) formulation for the assignment problem. In order to solve the mixed ILP problem effectively, in Section 8.4, we shall present some techniques for reducing the number of variables and constraints in the ILP formulation. Experimental results will be presented in Section 8.5. The novelty of the ILP formulation together with the effective techniques for reducing the number of variables and constraints in the formulation enable us to obtain excellent results for all the circuits we tested.

8.2 Problem Formulation

A routing area is a rectangular region with pins placed on the four boundaries. For a channel routing problem, the pins on the top and bottom boundaries are fixed. For a switchbox routing problem, the pins on all four boundaries are fixed. For a gridded routing problem, it is assumed that there is an orthogonal gird superimposed on the routing area. All pins on the boundaries are placed on the grid points, and all the interconnection wires are laid along grid edges (Figure 8.3). There are two layers available for routing, one for the horizontal wires and the other one for the vertical wires. In a valid routing solution, no wire segments of different nets may overlap.

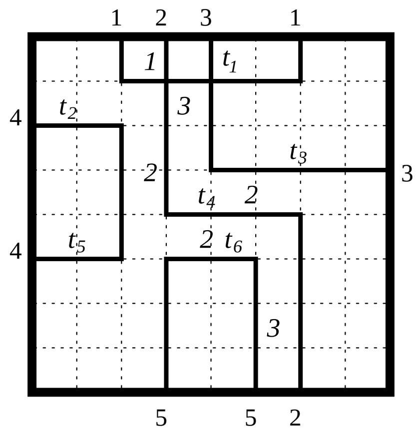


Figure 8.3. Crosstalks and constraints

Our algorithm is capable of considering crosstalks between adjacent wire segments and crosstalks between non-adjacent wire segments. Since crosstalk between two wire segments decreases as their separating distance increases, and since it is likely that crosstalk between two non-adjacent wire segments will be shielded by other wires between them, to simplify the computation, we shall assume that crosstalks only exist between adjacent wire segments.

Since the distance between two adjacent rows or columns is fixed in a gridded routing area, the crosstalk between two wire segments in adjacent rows or columns is proportional to their coupling length. Without loss of generality, we shall set the proportional constant to 1 and consider the coupling length the same as the crosstalk. The crosstalk C_k in a net N_k is the sum of the crosstalks in all the wire segments of N_k. As an example, the crosstalk in N_2 in Figure 8.3 is $1+2+3+2+2+3=13$. Notice that the coupling length between the upper vertical wire segment of N_2 and the vertical wire segments to its left are 1 and 2, respectively, the coupling length between the upper vertical wire segment and the vertical wire segment to its right is 3, the coupling length between the horizontal wire segment of N_2 and the horizontal wire segment above it is 2, the coupling length between the horizontal wire segment and the horizontal wire segment below it is 2, and the coupling length between the lower vertical wire segment of N_2 and the vertical wire segment to its left is 3. The maximum tolerable crosstalk M_i in N_i is a given constant derived by the designer according to the electrical property of the circuit components. The difference between M_k and C_k is called the *crosstalk slack* of N_k. The minimum value of crosstalk slacks among all the nets is called the *minimum slack*, denoted by $minslack$. In order to satisfy all the crosstalk constraints, it is necessary that $minslack \geq 0$. Since $minslack$ measures the safety margin on the crosstalk constraints, a good routing solution should have $minslack$ as large as possible. The sum of the crosstalks in all the nets is denoted $totalcross$. As was pointed out in Section 8.1, a good routing solution should also have $totalcross$ as small as possible. To be able to maximize $minslack$ and minimize $totalcross$ at the same time, we define the objective of the assignment problem as maximizing $w_s \times minslack - w_c \times totalcross$ where w_s and w_c are weight factors. By assigning a relatively large value to w_s, more emphasis will be placed on maximizing $minslack$. By assigning a relatively large value to w_c, more emphasis will be placed on minimizing $totalcross$.

For a given routing solution, horizontal wire segment t_i is *constrained* to be above horizontal wire segment t_j if t_i must be placed above t_j in a valid routing solution. Such a constraint will be referred to as a *vertical constraint* from t_i to t_j. For example, in Figure 8.3, since wire segment t_1 has a via which is connected to a pin on the top boundary that is directly above a via in t_4 which is connected to a pin on the bottom boundary, there is a vertical constraint from t_1 to t_4. Notice that vertical constraints are transitive in the sense that a vertical constraint from t_i to t_j and a vertical constraint from t_j to t_k *imply* a vertical constraint from t_i to t_k. In Figure 8.3, a vertical constraint from t_1 to t_4 and a vertical constraint from t_4 to t_6 imply a vertical constraint from t_1 to t_6. Notice also that a vertical constraint might be *implied* by the possible locations of the horizontal wire segments. In Figure 8.3, since t_2 is connected to a fixed pin in row 2 and t_5 is connected to a fixed pin in row 5, a vertical constraint is implies from t_2 to t_5. Vertical constraints that are not implied by other vertical constraints or the possible locations of the horizontal wire segments are called *essential vertical constraints*, and vertical constraints that are implied by other vertical constraints or the possible locations of the horizontal wire segments are called *implied vertical constraints*. Notice that the vertical constraints in a valid routing solution can not be circular. If all the essential vertical constraints are satisfied, all the implied vertical constraints are also satisfied.

For a given routing solution, there is a *horizontal constraint* between horizontal wire segment t_i and t_j if placing them on the same row causes them to overlap (or causes their endpoints to overlap and t_i and t_j belong to different nets). For example, in Figure 8.3, there is a horizontal constraint between t_3 and t_6.

For a pair of vertical wire segments t_k and t_l in the same column, if there is no vertical constraint between any horizontal wire segment connected to t_k and any horizontal wire segment connected to t_l, then t_k can be placed either above or below t_l (obviously, both t_k and t_l are doglegs). However, they can not overlap. Such constraints between doglegs in the same column will be referred to as the *non-overlapping constraints* between doglegs.

Given an initial routing solution, an *assignment* of the horizontal wire segments must satisfy the horizontal constraints, essential vertical constraints, and non-overlapping constraints in order to guarantee the validity of the routing solution. The objective of the assignment problem is to maximize $w_s \times minslack - w_c \times totalcross$.

8.3 A Mixed Integer Linear Programming Formulation

In this section, we shall formulate the assignment problem as a mixed integer linear programming (ILP) problem. It should be noted that such a formulation is far from being straight forward. Instead, we hope that a contribution of this formulation is to demonstrate how similar problems can be formulated in a similar way. It should also be noted that the wire assignment problem is significantly different from the permutation problem described in [21], and the mixed ILP formulation presented here is completely different from the one presented in [21]

Since crosstalks in a wire segment are due to the wire segments in adjacent rows or columns, we need to express adjacency information between horizontal wire segments in terms of linear variables and constraints in the mixed ILP formulation. Since the vertical constraints, the horizontal constraints, and the non-overlapping constraints are constraints on the relative positions of the wire segments, we also need to express information on the relative positions of two wire segments in terms of linear variables and constraints in the mixed ILP formulation. Even though these two kinds of information are closely related, it is not trivial to express them in an unified framework in the mixed ILP formulation. In [21], Gao presented one way to express these two kinds of information in an unified framework by using a large number of linear constraints. In this section, we shall present a novel way to express these two kinds of information in an unified framework without introducing a large number of linear constraints. To formulate the assignment problem as a mixed ILP problem, some effective techniques are used to transform disjunctions of linear constraints into conjunctions of linear constraints.

T-variables: We assume that the rows and columns are labeled in increasing order with the top most row being row 1 and the leftmost column being column 1 (the top boundary is labeled as row 0 and the bottom boundary is labeled as row $m+1$). For an initial routing solution with n horizontal wire segments and m rows, denote the horizontal wire segments by t_1, t_2, ..., t_n. For each horizontal wire segment t_i, there is a general integer variable T_i the value of which is the row in which t_i is placed in the final solution. These variables will be referred to as T-variables. Obviously, $1 \le T_i \le m$ for $1 \le i \le n$. There are n T-variables in the mixed ILP formulation.

Vertical constraints: Once the T-variables are defined, an essential vertical constraint from t_i to t_j can be expressed as $T_i < T_j$. Notice that since the satisfaction of the essential vertical constraints implies the satisfaction of the implied vertical constraints, it is not necessary to express the implied vertical constraint in the mixed ILP formulation.

Horizontal constraints: To express a horizontal constraint between horizontal wire segments t_i and t_j, we need to express the condition "$T_i > T_j$ *or* $T_i < T_j$" in terms of linear constraints. To be able to express disjunctions of linear constraints, one extra 0-1 integer variable z_{ij} and the following linear constraints are introduced:

$$-mz_{ij} < T_i - T_j \tag{8.1}$$

$$T_i - T_j < m(1 - z_{ij}) \tag{8.2}$$

Since $1 \leq T_i \leq m$ and $1 \leq T_j \leq m$, the difference between T_i and T_j is less than m. If $z_{ij} = 0$, Constraint 8.2 is redundant and Constraint 8.1 is equivalent to $T_j < T_i$. If $z_{ij} = 1$, Constraint 8.1 is redundant and Constraint 8.2 is equivalent to $T_i < T_j$. Since z_{ij} can only assume value of either 0 or 1, and since there is no other constraint on z_{ij}, Constraints 8.1 and 8.2 are equivalent to "$T_i < T_j$ *or* $T_i > T_j$". In the worst case, there might be one horizontal constraint between each pair of horizontal wire segments, and $\frac{n(n-1)}{2}$ 0-1 integer variables and $n(n-1)$ linear constraints are needed to represent the horizontal constraints.

Non-overlapping constraints between doglegs: For a pair of doglegs t_i and t_j in the same column with non-overlapping constraint, we need to express the condition "$max_i < min_j$ *or* $min_j < max_i$" in terms of linear constraints, where max_i is the row number of the bottom endpoint of t_i, min_i is the row number of the top endpoint of t_i, max_j is the row number of the bottom endpoint of t_j, and min_j is the row number of the top endpoint of t_j. max_i and min_i are defined as continuous variables and the following constraints are introduced:

$$max_i \geq T_k \text{ for every horizontal wire segment } t_k \text{ connected to } t_i \tag{8.3}$$

$$min_i \leq T_k \text{ for every horizontal wire segment } t_k \text{ connected to } t_i \tag{8.4}$$

The constraints in 8.3 force the value of max_i to be at least as large as the row number of the bottom endpoint of t_i, and the constraints in 8.4 forces the value of min_i to be at least as small as the row number of the top endpoint of t_i. Because the optimization process tries to maximize $w_s \times minslack - w_c \times totalcross$, and because of the signs of the coefficient of max_i and min_i in the constraints as will be shown later, the optimization process will force the values of max_i and min_i to be sufficiently close to their true values so that the non-overlapping constraints are satisfied if and only if the constraints in 8.3 and 8.4 are satisfied. Moreover, the optimization process will force the values of max_i and min_i to be sufficiently close to their true values so that the value of the objective function is computed correctly. Notice that it is possible to force the value of max_i and min_i to be exact by using extra variables and constraints. However, our formulation enables us to reduce the number of variables and constraints significantly. To reduce the computation time needed in solving the mixed ILP problem, even though the row numbers are integers, max_i and min_i are defined as continuous variables.

To express the condition "$max_i < min_j$ *or* $min_i > max_j$" in terms of linear constraints, one extra 0-1 integer variable z_{ij} and two linear constraints are introduced:

$$max_i < min_j + mz_{ij} \tag{8.5}$$

$$max_j < min_i + m(1 - z_{ij}) \tag{8.6}$$

Since t_i and t_j are both doglegs, $max_i - min_j < m$ and $max_j - min_i < m$. If $z_{ij} = 0$, Constraint 8.6 is redundant, and Constraint 8.5 is equivalent to $max_i < min_j$. If $z_{ij} = 1$, Constraint 8.5 is redundant, and Constraint 8.6 is equivalent to $max_j < min_i$. Since z_{ij} can only assume value of either 0 or 1, and since there is no other constraint on z_{ij}, Constraints 8.5 and 8.6 are equivalent to "$max_i < min_j$ *or* $max_j < min_i$". It is obvious that if t_i and t_j do not overlap in the final solution, the optimization process will force the values of max_i, max_j, min_i, and min_j to be close enough to their true values so that Constraints 8.5 and 8.6 are satisfied.

Crosstalks between adjacent vertical wire segments: Denote the crosstalk between vertical wire segments t_k and t_l in adjacent columns by C. Obviously, $C \geq 0$. Without knowing the relative positions of the endpoints of t_k and t_l in the final solution, two extra 0-1 integer variables $z1_{kl}$ and $z2_{kl}$ and the following constraints are introduced which will enable us to compute the value of C:

$$C \geq max_l - min_k - (m+1)z1_{kl} - (m+1)z2_{kl} \tag{8.7}$$

$$C \geq max_l - min_l - (m+1)z1_{kl} - (m+1)(1 - z2_{kl}) \tag{8.8}$$

$$C \geq max_k - min_k - (m+1)(1 - z1_{kl}) - (m+1)z2_{kl} \tag{8.9}$$

$$C \geq max_k - min_l - (m+1)(1 - z1_{kl}) - (m+1)(1 - z2_{kl}) \tag{8.10}$$

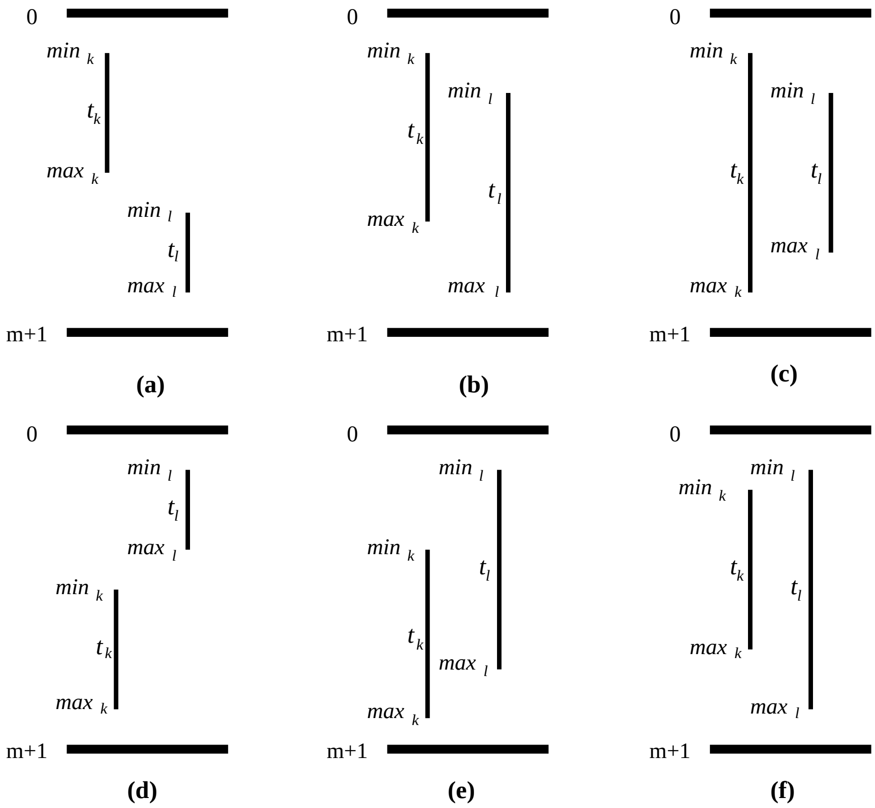


Figure 8.4. Possible relative positions of t_k and t_l

To show that the constraints above force the value of C to be the exact crosstalk between t_k and t_l, we need to consider the six possible relative positions of the endpoints of t_k and t_l as shown in Figure 8.4.

Since $max_k \leq m+1$, $max_l \leq m+1$, $min_k \geq 0$, and $min_l \leq 0$, it can be easily derived that $max_l - min_k \leq m+1$, $max_l - min_l \leq m+1$, $max_k - min_k \leq m+1$, and $max_k - min_l \leq m+1$. If $z1_{kl} = 0$ and $z2_{kl} = 0$, Constraint 8.7 is equivalent to $C \geq max_l - min_k$ and Constraints 8.8, 8.9, and 8.10 are redundant (the right hand side of the constraints are less than or equal to 0). If $z1_{kl} = 0$ and $z2_{kl} = 1$, Constraint 8.8 is equivalent to $C \geq max_l - min_l$ and Constraints 8.7, 8.9, and 8.10 are redundant. If $z1_{kl} = 1$ and $z2_{kl} = 0$, Constraint 8.9 is equivalent to $C \geq max_k - min_k$ and Constraints 8.7, 8.8, and 8.10 are redundant. Finally, if $z1_{kl} = 1$ and $z2_{kl} = 1$, Constraint 8.10 is equivalent to $C \geq max_k - min_l$ and Constraints 8.7, 8.8, and 8.9 are redundant. If the relative positions of the endpoints of t_k and t_l in the final solution are as shown in Figure 8.4a, among the four possible combinations of the values of $z1_{kl}$ and $z2_{kl}$, the combination with $z1_{kl} = 1$ and $z2_{kl} = 1$ gives the smallest possible value for C which is 0. Since our objective is to maximize $w_s \times minslack - w_c \times totalcross$ where $totalcross$ is the total crosstalk among all the nets, and since there is no other constraints on $z1_{kl}$ and $z2_{kl}$, the optimization process will force the value of $z1_{kl}$ to 1, the value of $z2_{kl}$ to be 1, and the value of C to be 0. If the relative positions of the endpoints of t_k and t_l in the final solution are as shown in Figure 8.4b, the combination with $z1_{kl} = 1$ and $z2_{kl} = 1$ gives the smallest possible value for C which is the actual crosstalk $max_k - min_l$. To maximize $w_s \times minslack - w_c \times totalcross$, the optimization process will force the value of $z1_{kl}$ to 1, the value of $z2_{kl}$ to be 1, and the value of C to be $max_k - min_l$. Moreover, the value of max_k will be forced to be the row number of the bottom endpoint of t_k, and the value of min_l will be forced to be the row number of the top endpoint of t_l. It can be shown in a similar way that the optimization process will force the value of C to be the exact crosstalk between t_k and t_l if the relative positions of the endpoints of t_k and t_l in the final solution are as shown in Figure 8.4c, 8.4d, 8.4e, or 8.4f.

Crosstalks between horizontal wire segments: To compute the crosstalks between horizontal wire segments, we need to express adjacency information between horizontal wire segments in the final solution. A 0-1 integer variable P_{ij} is defined for each pair of horizontal wire segments t_i and t_j if their coupling length is non-zero and they could become adjacent in the final solution. $P_{ij} = 1$ if and only if t_i and t_j are adjacent in the final solution, and $P_{ij} = 0$ otherwise. These variables will be referred to as P-variables. In the worst case, there might be $\frac{n(n-1)}{2}$ P-variables where n is the number of horizontal wire segments. So that the T-variables and the P-variables will be consistent in the sense that they will represent the same assignment, for each P_{ij}, an extra 0-1 integer variable z_{ij} and the following constraints are introduced:

$$1 - P_{ij} - (m+1)z_{ij} \leq T_i - T_j - 1 \tag{8.11}$$

$$1 - P_{ij} - (m+1)(1 - z_{ij}) \leq T_j - T_i - 1 \tag{8.12}$$

$$T_i - T_j - 1 \leq (1 - P_{ij})(m-1) \tag{8.13}$$

$$T_j - T_i - 1 \leq (1 - P_{ij})(m-1) \tag{8.14}$$

Notice that $T_i - T_j - 1 < m - 1$ and $T_j - T_i - 1 < m - 1$. If t_i is assigned to a row above t_j in the final solution ($T_i < T_j$), the right hand side of Constraint 8.11 is smaller than 0, and the value of z_{ij} is forced to be 1 by Constraint 8.11. Similarly, if t_i is below t_j in the final solution, the value of z_{ij} is forced to be 0 by Constraint 8.12. If $T_i - T_j = 1$ (t_i is adjacent to t_j), the right hand side of Constraint 8.11 is equal to 0. Since the value of z_{ij} is forced to be 0 by Constraint 8.12, the value of P_{ij} is forced to be 1 by Constraint 8.11. If $T_j - T_i = 1$, the right hand side of Constraint 8.12 is equal to 0. Since the value of z_{ij} is forced to be 1 by Constraint 8.11, the value of P_{ij} is forced to be 1 by Constraint 8.12. If $T_i - T_j > 1$ (t_i is not adjacent to t_j), the left hand side of Constraint 8.13 is greater than 0, and Constraint 8.13 forces the value of P_{ij} to 0. Similarly, if $T_j - T_i > 1$, Constraint 8.14 forces the value of P_{ij} to 0. Notice that Constraints 8.11, 8.12, 8.13, and 8.14 can be satisfied in all possible cases, and P_{ij} has value of 1 if and only if t_i is adjacent to t_j in the final solution.

With the P-variables, crosstalks in horizontal wire segment t_i can be computed as $\sum S_{ij}P_{ij}$ for all t_j with P_{ij} defined where S_{ij} is the coupling length between t_i and t_j. Notice that S_{ij} are constants which can be determined before the assignment problem is solved. For example, for the routing solution shown in Figure 8.5, $S_{13} = 1$, $S_{23} = 2$, and $S_{34} = 2$. Once the value of the P-variables are determined, the value of the crosstalk can be computed explicitly. For the routing solution shown in Figure 8.5, $P_{13} = 0$, $P_{23} = 1$, $P_{34} = 1$, and the value of the crosstalk on t_3 can be computed as $1 \times 0 + 2 \times 1 + 2 \times 1 = 4$.

Minimum crosstalk slack: The crosstalk in a net is the sum of the crosstalks in all wire segments of the net. To obtain the value of *minslack*, it is defined as a continuous variable and the following constraints are introduced:

$$minslack \leq M_i - C_i \text{ for every net } N_i \tag{8.15}$$

where M_i is the maximum tolerable crosstalk in N_i, and C_i is the total crosstalk in N_i. Above constraints force *minslack* to be at least as small as the smallest crosstalk slack among all the nets. Since the objective of the mixed ILP problem is to maximize $w_s \times minslack - w_c \times totalcross$, the optimization process will force the value of *minslack* to be exactly the smallest crosstalk slack among all the nets.

8.4 Reducing the Size of the Mixed ILP Formulation

We presented in Section 8.3 a mixed ILP formulation for the assignment problem. For a large problem instance, the number of variables and constraints in the formulation might become very large. For example, for the assignment problem for Burstein's Difficult problem, there are 829 variables and 1371 constraints in the mixed ILP formulation. In this section, we shall present some techniques for reducing the number of variables and constraints in the mixed ILP formulation.

We first introduce the concept of a constraint graph. A *constraint graph* is a graph with both directed and undirected edges in which the vertices represent the horizontal wire segments, solid directed edges represent the essential vertical constraints, dotted directed edges represent the implied vertical constraints, and undirected edges represent the horizontal constraints (Figure 8.6). Given an initial routing solution, by scanning the routing area column by column, a set of vertical constraints which includes all the essential vertical constraints (and maybe some implied vertical constraints) can be found. By checking the coupling length between horizontal wire segments, a set of horizontal constraints can be found (some of them might be implied). After the set of horizontal and vertical constraints is found, a constraint graph can be constructed. Without knowing whether a vertical constraint is essential or implied, it is conservatively assumed that all the vertical constraints are essential. For example, for the routing solution shown in Figure 8.6a, the constraint graph shown in Figure 8.6b can be constructed. Since a vertical constraint from t_i to t_j implies a horizontal constraint between t_i and t_j, if there is a vertical constraint from t_i to t_j, the horizontal constraint between t_i and

t_j can be removed from consideration in the mixed ILP formulation. To minimize the number of horizontal and essential vertical constraints that need to be formulated in the mixed ILP formulation, it is desirable to find as many implied vertical constraints as possible. Here, we will demonstrate the techniques to find the implied vertical constraints by using the example shown in Figure 8.6a.

First, a vertical constraint from t_i to t_j and a vertical constraint from t_j to t_k implies a vertical constraint from t_i to t_k. In Figure 8.6a, the vertical constraint from t_1 to t_4 and the vertical constraint from t_4 to t_6 imply a vertical constraint from t_1 to t_6. Because of this implied vertical constraint, the horizontal constraint between t_1 and t_6 can be deleted from the constraint graph (Figure 8.6c).

Next, the top and bottom most row that each horizontal wire segment may occupy are computed. The top and bottom most row that t_i may occupy gives the range of the value of T_i in the mixed ILP formulation. Smaller ranges of the T-variables lead to reduction in the number of essential vertical constraints and horizontal constraints. As will be discussed later, smaller ranges of the T-variables also lead to reduction in the number of variables and constraints needed to compute the crosstalks on the wire segments.

To reduce the range of T_i, we first check if t_i is connected to a fixed pin on the boundaries. In Figure 8.6a, since t_2 is connected to a fixed pin on row 2, the top and bottom most row that t_2 may occupy are both 2. Similarly, the top and bottom most row that t_3 may occupy are both 3, and the top and bottom most row that t_5 may occupy are both 5. Since the rows that t_2, t_3, and t_5 may occupy are fixed, the values of T_2, T_3, and T_5 are also fixed. Moreover, from the top and bottom most row that t_2, t_3, and t_5 may occupy, we can derive immediately the three implied vertical constraints from t_2 to t_3, from t_3 to t_5, and from t_2 to t_5. The implied vertical constraint from t_1 to t_5 can also be derived (Figure 8.6d).

The range of T_i can be further reduced by checking the vertical constraints (both essential and implied) from t_i to other wire segments and from other wire segments to t_i. In Figure 8.6, since the bottom most row that t_2 may occupy is row 2, the vertical constraint from t_1 to t_2 implies that the bottom most row that t_1 may occupy is $2 - 1 = 1$. Since the top most row that t_1 may occupy is also 1, the value of T_1 is fixed. Since the bottom most row that t_1 may occupy (row 1) is higher than the top most row that t_2 may occupy (row 2), the vertical constraint from t_1 to t_2 can be treated as an implied vertical constraint (Figure 8.6e). Similarly, because of the vertical constraint from t_1 to t_4 and the vertical constraint from t_4 to t_6, the top most row that t_6 may occupy is at least 3. Because t_3 is fixed in row 3, the top most row that t_6 may occupy is 4, and there is an implied vertical constraint from t_3 to t_6 (Figure 8.6f).

It should be noted that reductions in the ranges of the T-variables may lead to an increase in the number of implied vertical constraints, which in turn, may lead to reductions in the ranges of the T-variables. Therefore, the reduction procedure should be executed iteratively until the ranges of the T-variables can not be reduced.

The vertical constraints and the reduction in the ranges of the T-variables can also be used to reduce the number of variables and constraints needed to compute the crosstalks between wire segments.

As was discussed in Section 8.3, without knowing the relative positions of the endpoints of the vertical wire segments t_k and t_l in adjacent columns, two 0-1 integer variables $z1_{kl}$ and $z2_{kl}$ and Constraints 8.7, 8.8, 8.9, and 8.10 are needed to compute the value of the crosstalk C between t_k and t_l. From the range of the T-variables, the maximum and minimum possible values of max_k, min_k, max_l, and min_l can be computed. If the minimum possible value of max_k is greater than the maximum possible value of max_l, only one 0-1 integer variable $z2_{kl}$ and two constraints are needed to compute the value of C:

$$C \geq max_l - min_k - (m+1)z2_{kl} \tag{8.16}$$

$$C \geq max_l - min_l - (m+1)(1 - z2_{kl}) \tag{8.17}$$

If the maximum possible value of max_k is smaller than the minimum possible value of max_l, only one 0-1 integer variable $z2_{kl}$ and two constraints are needed to compute the value of C:

$$C \geq max_k - min_k - (m+1)z2_{kl} \tag{8.18}$$

$$C \geq max_k - min_l - (m+1)(1 - z2_{kl}) \tag{8.19}$$

Similarly, if the relative values of min_k and min_l are known, only one extra 0-1 integer variable $z1_{ij}$ and two constraints are needed to compute the value of C. If both the relative values of max_k and max_l and the relative values of min_k and min_l are known, only one constraint is needed to compute the value of C.

As was discussed in Section 8.3, for an P-variable defined between t_i and t_j where t_i was in a row above t_j in the initial routing solution, one 0-1 integer variable z_{ij} and Constraints 8.11, 8.12, 8.13, and 8.14 are needed to force the value of P_{ij} to be consistent with the values of T_i and

T_j. Since either Constraint 8.11 or Constraint 8.12 will be violated if $T_i = T_j$, Constraints 8.11 and 8.12 imply a horizontal constraint between t_i and t_j. Therefore, if P_{ij} is defined, the horizontal constraint between T_i and T_j can be excluded from consideration in the mixed ILP formulation. Moreover, if there exists a vertical constraint from t_i to t_j, only two constraints are needed to force the value of P_{ij} to be consistent with the values of T_i and T_j:

$$1 - P_{ij} \leq T_j - T_i - 1 \tag{8.20}$$

$$T_j - T_i - 1 \leq (1 - P_{ij})(m - 1) \tag{8.21}$$

If the difference between the top most row that t_j may occupy and the bottom most row that t_i may occupy is greater than 1, t_i can not be adjacent to t_j in any routing solution. Therefore, P_{ij} and the corresponding constraints can be deleted from the mixed ILP formulation.

Clearly, the reduction in the number of variables and constraints depends on the problem instance. In all our experiments, as will be shown in Section 8.5, large reductions were achieved on both the number of variables and constraints. For example, for Burstein's Difficult problem, we were able to reduce the number of variables from 829 in the initial formulation to 50 in the final formulation and reduce the number of constraints from 1371 in the initial formulation to 105 in the final formulation.

8.5 Experimental Results

Our algorithm was implemented in C and executed on an IBM RISC6000 workstation. The Optimization Subroutine Library (OSL) distributed by IBM was used to solve the mixed ILP problems. Our algorithm was tested on several benchmark circuits and many randomly generated circuits. Here, we shall present the results of running our algorithm on two randomly generated switchbox routing circuits, two well known switchbox benchmark circuits, and one channel routing circuit. The circuit specifications for the switchbox routing circuits are summarized in Table 8.1, and the initial routing solution of the Extended Burstein's Difficult problem is shown in Figure 8.7a. The circuit Burstein (Burstein's Difficult problem) was taken from [5], and the circuit Ext-Burstein (the Extended Burstein's Difficult problem) was taken from [10].

cir. name	initial solution		
	no. of rows	no. of columns	no. of nets
Example1	18	18	18
Example2	21	21	27
Burstein	15	23	24
Ext-Burstein	21	22	24

Table 8.1. Circuit specifications

The numbers of variables and constraints in the ILP formulations for the first iteration of the assignment process before and after the reduction techniques are applied are summarized in Table 8.2. As shown in Table 8.2, we were able to reduce the number of variables and constraints significantly for all the circuits. Notice the relatively large reduction in the number of variables and constraints for Burstein's Difficult problem and the Extended Burstein's Difficult problem. The reasons for this are as follows. First, a large number of the horizontal wire segments in the initial routing solutions of these two circuits are connected to fixed pins on the left and right boundaries. Due to these fixed pins, the rows that these horizontal wire segments may occupy are fixed. Also, for these two problems, there is a large number of implied vertical constraints in the initial routing solutions. These constraints lead to a substantial reduction in the number of variables and constraints in the ILP formulation.

cir. name	before reduction		after reduction	
	no. of variables	no. of constr.	no. of variables	no. of constr.
Example1	536	822	128	239
Example2	854	1339	143	279
Burstein	829	1371	50	105
Ext-Burstein	1044	1650	62	128

Table 8.2. Mixed ILP formulation

Since no maximum tolerable crosstalks in the nets were specified for any of the circuits we tested, we assign the maximum tolerable crosstalks for the nets in two different ways. In our first test, a uniform maximum tolerable crosstalk is specified for all the nets in each circuit. The values of *minslack* and *totalcross* of the routing solutions generated by our algorithm are summarized in Table 8.3. In our second test, the maximum tolerable crosstalks in the nets are assigned randomly. The values of *minslack* and *totalcross* of the routing solutions generated by our algorithm are summarized in Table 8.4. In all our experiments, w_s is set to 10 and w_c is set to 1.

cir. name name	initial solution *min-slack*	*total-cross*	final solution *min-slack*	*total-cross*	time(min.)	improvement *min-slack*	*total-cross*
Example1	-6	510	2	414	0.5	8	96
Example2	-9	772	9	710	7.1	18	62
Burstein	-1	778	5	768	0.1	6	10
Ext-Burstein	-20	1092	3	1012	0.4	23	80

Table 8.3. Routing solutions for circuits with uniform maximum tolerable crosstalks

cir. name name	initial solution *min-slack*	*total-cross*	final solution *min-slack*	*total-cross*	time(min.)	improvement *min-slack*	*total-cross*
Example1	-10	510	1	414	1.0	11	96
Example2	-9	772	8	720	2.8	17	62
Burstein	-3	778	4	762	0.1	7	16
Ext-Burstein	-20	1092	11	1008	1.4	31	84

Table 8.4. Routing solutions for circuits with random maximum tolerable crosstalks

As was shown in Table 8.3 and Table 8.4, *minslack* for all the circuits are negative before the iterative assignment process which means some crosstalk constraints are violated. After the assignment process, all the crosstalk constraints are satisfied with some margins. Moreover, the total crosstalk among all the nets is reduced significantly for every circuits.

Notice that in comparison with the other circuits, relatively small improvements were obtained for Burstein's Difficult problem. This is due to the fact that many nets need to be connected within a small area, and the routing solution will be tightly packed no matter how the interconnection wires are routed. In other words, crosstalks in the wire segments will be large no matter how the nets are routed. The only way to improve upon the situation will be to increase the number of rows and columns in the routing area.

To illustrate the iterative assignment steps, the intermediate routing solutions for the Extended Burstein's Difficult problem with uniform maximum tolerable crosstalks on the nets are shown in Figure 8.7b, Figure 8.7c, and Figure 8.7d. For the initial routing solution shown in Figure 8.7a, *minslack* is equal to -20 and *totalcross* is equal to 1092. After the horizontal wire segments are re-assigned (Figure 8.7b), *minslack* is equal to 0 and *totalcross* is equal to 1024. After the vertical wire segments are re-assigned (Figure 8.7c), *minslack* is equal to 0 and *totalcross* is equal to 1022. And finally, after the horizontal wire segments are re-assigned again (Figure 8.7d), *minslack* is equal to 3 and *totalcross* is equal to 1012. The iterative assignment process converges in a few iterations for all our test circuits.

Our algorithm was also run on the four channel routing problems presented in [21]. The circuit specifications and the initial routing solutions for the channel routing circuits are summarized in Table 8.5. The circuit YK3C is an example from [49], the circuit D1 is from the GTE layout published in [47], and Deutsch is the well known Deutsch difficult problem [1]. "Random" is a small randomly generated circuit.

cir. name	initial routing solution		
	no. of tracks	no. of columns	no. of nets
Random	5	20	10
YK3C	18	79	54
D1	18	155	65
Deutsch	19	174	72

Table 8.5. Input circuit specifications

Since the channel routing benchmark circuits are large, the mixed ILP problems formulated for these the circuits are also large. To solve the mixed ILP problems effectively, the routing channels are broken into parts from left to right. For each part, a mixed ILP problem is formulated and solved. By breaking the routing channel into different parts, routing quality is traded with running time. However, even with the routing channel broken into parts, our algorithm still generates better routing solutions in comparison with the routing solutions generated by the permutation algorithm presented in [21].

For each channel routing problem, a uniform maximum tolerable crosstalk is specified for all the nets. The routing solutions generated by the permutation algorithm which was presented in [21] are summarized in Table 8.6, and the routing solutions generated by our algorithm are summarized in Table 8.7.

cir. name	initial solution		final solution			improvement	
	minslack	*totalcross*	*minslack*	*totalcross*	time(min.)	*minslack*	*totalcross*
Random	-12	200	7	156	0.1	19	44
YK3C	-10	2902	11	2752	6.6	21	150
D1	-8	3752	21	3518	5.1	29	234
Deutsch	-11	7570	5	7556	0.3	16	14

Table 8.6. Channel routing solutions generated by the permutation algorithm

As were shown in the tables, we were able to obtain significant amount of improvements in both *minslack* and *totalslack* for all circuits using our algorithm in comparison with the

cir. name	initial solution		final solution			improvement	
	minslack	*totalcross*	*minslack*	*totalcross*	time(min.)	*minslack*	*totalcross*
Random	-12	200	11	134	0.4	23	66
YK3C	-10	2902	24	2732	20.7	34	170
D1	-8	3752	75	3092	37.9	83	660
Deutsch	-11	7570	6	7474	117.1	17	96

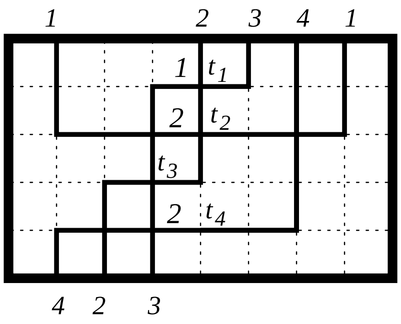
Table 8.7. Channel routing solutions generated by our algorithm

results generated by the permutation algorithm. Due to the relatively large problem size in our formulation, the mixed ILP problems formulated in our algorithm take longer time to solve. We believe that this extra time is well worthy since we were able be generate significantly better solutions.

Notice that a relatively small amount of improvements were obtained for Deutsch's Difficult Problem. Similar to Burstein's Difficult Problem, this is due to the fact that many nets are connected within a small area, and the routing solution will be tightly packed no matter how the interconnection wires are routed. In other words, crosstalks in the wire segments will be large no matter how the nets are routed.

8.6 Summary

In this chapter, we studied the gridded switchbox and channel routing problem with crosstalk constraints. We proposed a new approach to the problem which utilizes existing switchbox and channel routing algorithms and improves upon the routing solution by re-assigning the wire segments in the initial routing solution to rows and columns. A novel mixed integer linear programming formulation and effective procedures for reducing the number of variables and constraints in the ILP formulation were then presented. The experimental results are very promising.



Coupling lengths: $S_{13} = 1,\ S_{23} = 2,\ S_{34} = 2$

P-varaibles: $P_{13} = 0,\ P_{23} = 1,\ P_{34} = 1$

Crosstalk in t_3: $S_{13}P_{13} + S_{23}P_{23} + S_{34}P_{34} = 4$

Figure 8.5. Crosstalk in horizontal wire segment t_3

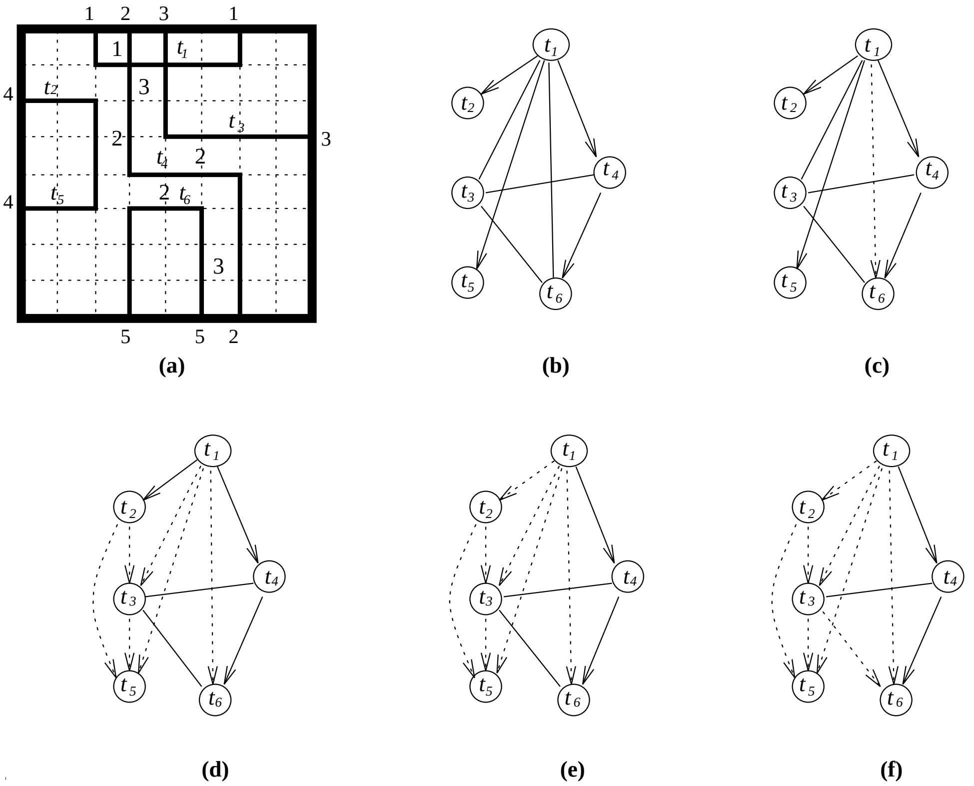
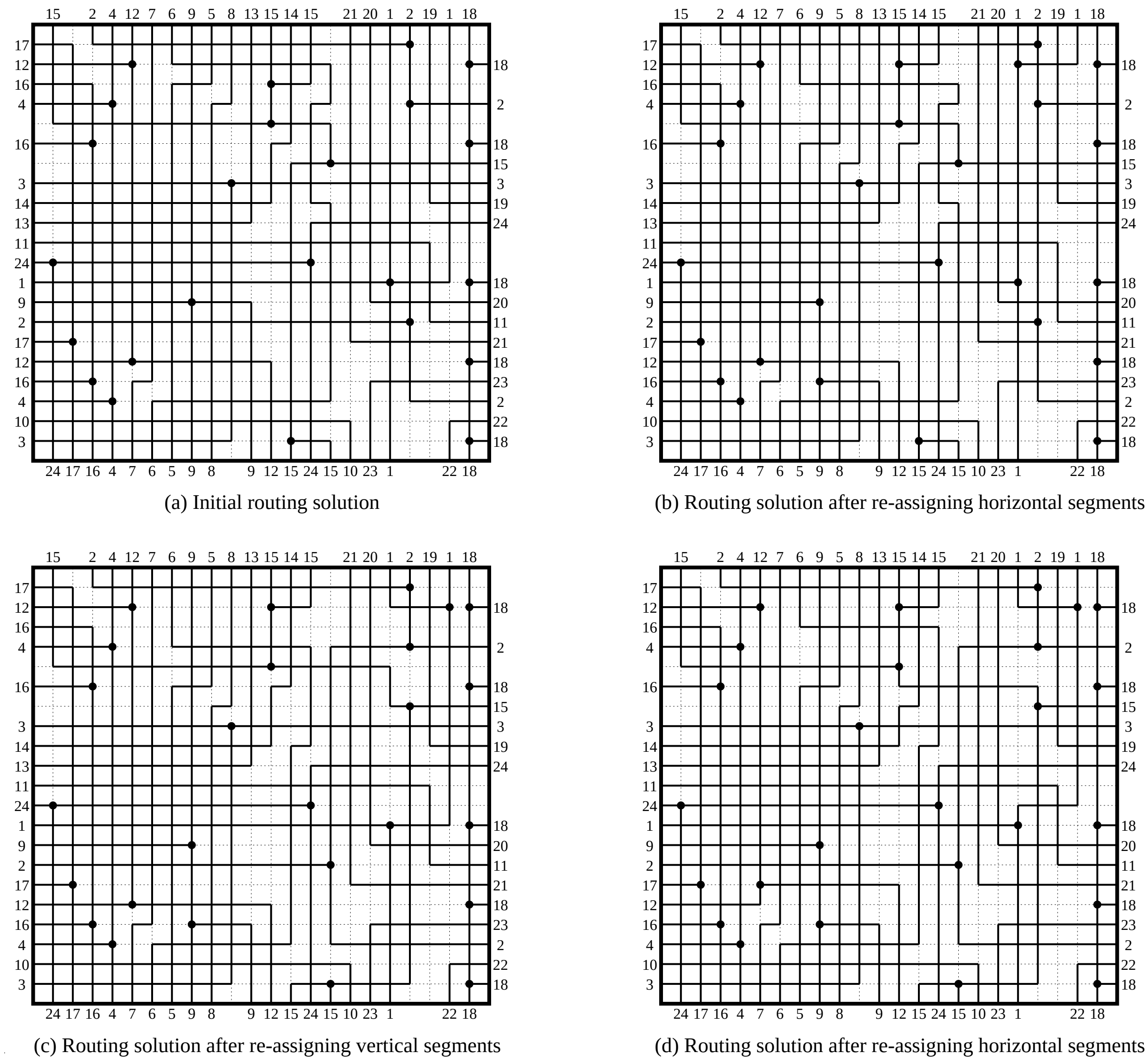


Figure 8.6. Constraint graph



(a) Initial routing solution

(b) Routing solution after re-assigning horizontal segments

(c) Routing solution after re-assigning vertical segments

(d) Routing solution after re-assigning horizontal segments

Figure 8.7. Intermediate routing solutions for the Extended Burstein's Difficult problem

BIBLIOGRAPHY

[1] Burstein, M. and Pelavin, R. Hierarchical Channel Router. *Integration, the VLSI Journal*, 1:21–38, 1983.

[2] Burstein, M. and Youssef, M. N. Timing Influenced Layout Design. In *Proc. 22nd Design Automation Conference*, pages 124–130, 1985.

[3] Chen, H. H. and Kuh, E. S. Glitter: A Gridless Variable-Width Channel Router. *IEEE Trans. on Computer-Aided Design of ICs and System*, CAD-5(4):459–465, 1986.

[4] Chen, H. H. and Wong, C. K. Wiring and Crosstalk Avoidance in Multi-Chip Module Design. In *Proc. Custom Integrated Circuits Conference*, pages 28.6.1–28.6.4, 1992.

[5] Cho, T. W., Pyo, S. S., and Heath, J. R. A New Conflict Resolving Switchbox Router. In *Proceedings of the Second Great Lakes Symposium on VLSI*, pages 95–102, 1992.

[6] Cong, J. and Ding, Y. An Optimal Technology Mapping Algorithm for Delay OPtimization in Lookup-Table Based FPGA Designs. In *Proc. International Conference on Computer-Aided Design*, pages 48–53, 1992.

[7] Cong, J., Ding, Y., Gao, T., and Chen, K. C. An optimal performance-driven technology mapping algorithm for LUT based FPGAs under arbitrary net-delay modules. In *Proc. International Conference on Computer-Aided Design & Computer Graphics*, 1993. 599-604.

[8] Cong, J. and Liu, C. L. Over-the-Cell Channel Routing. *IEEE Trans. on Computer-Aided Design of ICs and System*, 9(4):408–418, April 1990.

[9] Cong, J., Wong, D. F., and Liu, C. L. A New Approach to Three or Four-Layer Channel Routing. *IEEE Trans. on Computer-Aided Design of ICs and System*, 7:1094–1104, October 1988.

[10] Curatelli, F. Switchbox Routing with Rerouting Capabilities in VLSI Design. *IEE Proceedings F [Circuits, Devices and Systems]*, 137(3):210–218, June 1990.

[11] Dai, W. M., Chen, H. H., Dutta, R., Jackson, M., Kuh, E. S., Marek-Sadowska, M., Sato, M., Wang, D., and Xiong, X. M. BEAR: A New Building-Block Layout Systems. In *Proc. International Conference on Computer-Aided Design*, pages 34–37, 1987.

[12] Dai, W. M., Kong, R., Jue, J., and Sato, M. Rubber Band Routing and Dynamic Data Representation. In *Proc. International Conference on Computer-Aided Design*, pages 52–55, 1990.

[13] Donath, W. E., Norman, R. J., Agrawal, B. K., Bello, S. E., Han, S. Y., Kurtzberg, J. M., Lowy, P., and McMillan, R. I. Timing Driven Placement Using Complete Path Delays. In *Proc. 27th Design Automation Conference*, pages 84–89, 1990.

[14] Dunlop, A. E., Agrawal, V. D., Deutsch, D. N., Jukl, M. F., Kozak, P., and Wiesel, M. Chip Layout Optimization Using Critical Path Weighting. In *Proc. 21st Design Automation Conference*, pages 133–136, 1984.

[15] Dunlop, A. E. and Kernighan, B. W. A Procedure for Placement of Standard-Cell VLSI Circuits. *IEEE Trans. on Computer-Aided Design of ICs and System*, CAD-4(1):92–98, January 1985.

[16] Fiduccia, C. M. and Matteyses, R. M. A Linear-Time Heuristic for Improving Network Partitions. In *Proc. 19th Design Automation Conference*, pages 175–181, 1982.

[17] Francis, R. J., Rose, J., and Vranesic, Z. Technology Mapping for Lookup Table-Based FPGAs for Performance. In *Proc. International Conference on Computer-Aided Design*, pages 568–571, 1991.

[18] Gao, T., Chen, K. C., Cong, J., Ding, Y., and Liu, C. L. Placement and placement driven technology mapping for FPGA synthesis. In *Proc. ASIC Conference*, pages 91–94, 1993.

[19] Gao, T., Chen, K. C., and Liu, C. L. A Performance Driven Hierarchical Partitioning Placement Algorithm. In *Proc. European Design Automation Conference*, pages 33–38, 1993.

[20] Gao, T. and Liu, C. L. Minimum Crosstalk Channel Routing. Submitted to *IEEE Trans. on Computer-Aided Design of ICs and System.*

[21] Gao, T. and Liu, C. L. Minimum Crosstalk Channel Routing. In *Proc. International Conference on Computer-Aided Design*, pages 692–696, 1993.

[22] Gao, T. and Liu, C. L. Minimum Crosstalk Switchbox Routing. Submitted to *Proc. International Conference on Computer-Aided Design*, 1993.

[23] Gao, T., Vaidya, P. M., and Liu, C. L. A Performance Driven Placement Algorithm. Submitted to *IEEE Trans. on Computer-Aided Design of ICs and System.*

[24] Gao, T., Vaidya, P. M., and Liu, C. L. A New Performance Driven Placement Algorithm. In *Proc. International Conference on Computer-Aided Design*, pages 44–47, 1991.

[25] Gao, T., Vaidya, P. M., and Liu, C. L. A Performance-Driven Macro-Cell Placement Algorithm. In *Proc. 29th Design Automation Conference*, pages 147–152, 1992.

[26] Garey, M. R. and Johnson, D. S. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, New York, 1979.

[27] Hauge, P. S., Nair, R., and Yoffa, E. J. Circuit Placement for Predictable Performance. In *Proc. International Conference on Computer-Aided Design*, pages 88–91, 1987.

[28] Hitchcock, R. B., Smith, G. L., and Cheng, D. D. Timing analysis of computer hardware. *IBM Journal of Research and Development*, 26(1):100–105, 1985.

[29] Jackson, M. A. B. and Kuh, E. S. Performance-Driven Placement of Cell Based IC's. In *Proc. 26th Design Automation Conference*, pages 370–375, 1989.

[30] Jackson, M. A. B., Srinivasan, A., and Kuh, E. S. A Fast Algorithm for Performance-Driven Placement. In *Proc. International Conference on Computer-Aided Design*, pages 328–331, 1990.

[31] Khachian, L. G. A polynomial algorithm in linear programming. *Soviet Mathematics Doklady*, pages 191–194, 1979.

[32] Marek-Sadowska, M. and Lin, S. P. Timing Driven Placement. In *Proc. International Conference on Computer-Aided Design*, pages 94–97, 1989.

[33] Nair, R., Berman, C. L., Hauge, P. S., and Yoffa, E. J. Generation of Performance Constraints for Layout. *IEEE Trans. on Computer-Aided Design of ICs and System*, 8(8):860–874, August 1989.

[34] Ogawa, Y., Ishii, T., Shiraishi, Y., Terai, H., Kozawa, T., Yuyama, K., and Chiba, K. Efficient Placement Algorithms Optimizing Delay for High-Speed ECL Masterslice LSI's. In *Proc. 23rd Design Automation Conference*, pages 404–410, 1986.

[35] Pedram, M. and Bhat. Layout Driven Technology Mapping. In *Proc. 28th Design Automation Conference*, pages 99–105, 1991.

[36] Rivest, R. L. and Fiduccia, C. M. A Greedy Channel Router. In *Proc. 19th Design Automation Conference*, pages 418–424, 1982.

[37] Shragowitz, E., Youssef, H., and Bening, L. C. Predictive Tools in VLSI System Design: Timing Aspect. In *Proc. of COMPEURO*, pages 48–55, 1988.

[38] Srinivasan, A. An Algorithm for Performance-Driven Initial Placement of Small-Cell ICs. In *Proc. 28th Design Automation Conference*, pages 636–639, 1991.

[39] Srinivasan, A., Chaudhary, K., and Kuh, E. S. RITUAL: A Performance Driven Placement Algorithm for Small Cell ICs. In *Proc. International Conference on Computer-Aided Design*, pages 48–51, 1991.

[40] Sutanthavibul, S. and Shragowitz, E. An Adaptive Timing-Driven Layout for High Speed VLSI. In *Proc. 27th Design Automation Conference*, pages 90–95, 1990.

[41] Sutanthavibul, S. and Shragowitz, E. Dynamic Prediction of Critical Paths and Nets for Constructive Timing-Driven Placement. In *Proc. 28th Design Automation Conference*, pages 632–635, 1991.

[42] Sutanthavibul, S., Youssef, H., and Shragowitz, E. Cell-Based Physical Design Under Timing Constraints. In *Proc. International Symposium on Circuits and Systems*, pages 877–880, 1990.

[43] Teig, S., Smith, R. L., and Seaton, J. Timing-Driven Layout of Cell-Based ICs. *VLSI Systems Design*, pages 63–73, May 1986.

[44] Terai, M., Takahashi, K., and Sato, K. A New Min-Cut Placement Algorithm for Timing Assurance Layout Design Meeting Net Length Constraint. In *Proc. 27th Design Automation Conference*, pages 96–102, 1990.

[45] Tsay, R. S. and Koehl, J. An Analytic Net Weighting Approach for Performance Optimization in Circuit Placement. In *Proc. 28th Design Automation Conference*, pages 620–625, 1991.

[46] Vaidya, P. M. A New Algorithm for Minimizing Convex Functions Over Convex Sets. In *Proc. 30th Annual IEEE Symposium on Foundations of Computer Science*, pages 338–343, 1989.

[47] Wong, D. F. and Liu, C. L. Compacted Channel Routing with Via Placement Restriction. *Integration, the VLSI Journal*, 4:267–307, 1986.

[48] Xilinx Inc. *The Programmable Gate Array Data Book.* Xilinx, San Jose, California, 1992.

[49] Yoshimura, T. and Kuh, E. S. Efficient Algorithms for Channel Routing. *IEEE Trans. on Computer-Aided Design of ICs and System*, 1:25–35, January 1982.

[50] Youssef, H., Lin, R. B., and Shragowitz, E. Bounds on Net Delays for Physical Design of Fast Circuits. In *Proceedings of the IFIP International Conference VLSI*, pages 111–118, 1991.

[51] Youssef, H., Shragowitz, E., and Bening, L. C. Critical Path Issue in VLSI Designs. In *Proc. International Conference on Computer-Aided Design*, pages 520–523, 1989.

VITA

Tong Gao was born on September 9, 1966 in Jiangsu, China. He received his B. S. degree and M. S. degree in Computer Science Department from University of Illinois at Urbana-Champaign in 1989 and 1992, respectively. Upon completion of his doctoral study, he will join Silicon Graphics Computer Systems Inc. in Mountain View, California.