PERFORMANCE OPTIMIZATION OF LARGE-SCALE INTEGRATED CIRCUITS

by

Arvind Srinivasan

Memorandum No. UCB/ERL M91/104

27 November 1991

PERFORMANCE OPTIMIZATION OF LARGE-SCALE INTEGRATED CIRCUITS

by

Arvind Srinivasan

Memorandum No. UCB/ERL M91/104

27 November 1991

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

Performance Optimization of Large-Scale Integrated Circuits

Arvind Srinivasan

University of California
Berkeley, California

Department of Electrical Engineering
and Computer Science

Abstract

This work explores new and challenging problems encountered during the physical design of very large scale high performance integrated circuits. The first problem considered is that of determining a placement for the logic cells of a design on a chip in such a way that performance constraints are met. The problem is formulated and analyzed and efficient algorithms are developed for solving the formulation, including a number of practical constraints. The effectiveness of the techniques is illustrated by a successful implementation of the algorithms, experimentation on real circuits and comparisons with other approaches. The techniques are shown to have achieved the most important goals of a computer-aided design tool which are: (1) to deliver circuits whose performance is consistently enhanced, (2) to produce results that are predictable and characterizable and (3) to be efficient in terms of computer time and memory, even for very large designs.

The next problem addressed is that of controlling the precise instants at which storage elements in a large design are clocked in such a way as to minimize the clock period of the system. New and heretofore unsolved problems are formulated and solved under a variety of practical and realistic constraints using efficient algorithms. One of the most interesting subproblems occur when the logic delays are random variables with characterizable probability distributions. Efficient solutions are presented even under these constraints. The applicability of the techniques is illustrated through experimentation on real designs.

Prof. Ernest S. Kuh
Thesis Committee Chairman

Acknowledgements

Professor Ernest Kuh was my mentor during the years of my study at Berkeley. He has a unique and effective style of providing guidance. While allowing me considerable freedom in pursuing research work, he also gently propeled me back on course whenever I was diverted from a straight path. Under his tutelage, I learned to research and to carry a task to completion regardless of its difficulty. I am certain that the knowledge acquired as his student will prove invaluable throughout my life.

Professor Robert Brayton provided many helpful suggestions and listened to me patiently whenever I approached him excitedly for advice on some new discovery. He always found time to discuss, correct, and comment upon my work despite his busy schedule.

Professor Shmuel Oren helped me get started when, after after a discussion with him he commented: "I think you are doing the right thing". His guidance on my qualifying examination and reading of this thesis are appreciated.

Tahani Sticpewich has helped me time and again to clear a number of administrative hurdles. Always patient and understanding of a students' problems with "those people downstairs", she went out of the way to assist me with paperwork, forms, supplies and innumerable sundries.

Dr. Kamal Chaudhary's support and assistance in turning theoretical ideas into a practical tool are sincerely acknowledged. Michael Jackson's early work on timing-driven placement was an inspiration and the discussions with him proved illuminating. I would like to thank Narasimha Bhat, Timothy Kam, Shen Lin, Sharad Malik, Massoud Pedram, Malgorzata Marek-Sadowska, Minshine Shih, Ren-Song Tsay and Deborah Wang for being part of a productive and friendly work environment. George "Gyorgy" Carvalho's "dotmania" and his aesthetic contributions to the workplace enlivened the work atmosphere.

Collaboration with David LaPotin during the summer of 1990 was a memorable and fruitful experience and led to some of the work on skew optimization. Professor George Shanthikumar and Sridhar Seshadri for helped me understand the stochastic optimization problem.

I am grateful to my parents and brother for their support and encouragement throughout my career at Berkeley.

Financial support for this work received in part from the Semiconductor Research Corporation is gratefully acknowledged.

This thesis is dedicated to the Lord Venkatesha.

Contents

List of Figures

List of Tables

Chapter 1

The Scenario

1.1 The Case for Computer Tools

The first microprocessor, Intel's 4004, introduced in 1971, contained 2,300 transistors, operated at 0.06 MIPS and addressed 640 bytes of memory. In 1991, the newest descendant of the 4004, the Intel 80486 contains close to 1 million transistors, "operates at upto 27 MIPS, has enough addressing capability for an eight-page history of every person on earth", and "can scan the Encyclopaedia Britannica in 2 seconds" [Com 91]. What has made this four hundred-fold speed increase and thousand-fold increase in complexity possible in a short span of 20 years? How is it possible to define a multi-million transistor circuit on a piece of silicon that is just two centimeters on each side and be confident that it will work? The explosive progress has been made possible through the use of sophisticated computer tools and computer-aided design (CAD) techniques. The central role of computer tools in designing Very Large Scale Integrated (VLSI) circuits is *compexity management.* The ability to manage multi-million transistor circuits efficiently makes it possible to develop increasingly complex integrated circuit functions in a short time. The result is unabated technological advancement as we have witnessed in the last two decades. As proof of the role of computer aided design in technological advancement is the surge of the electronic design automation business industry from nothing in the 70's to $2.5 billion at the beginning of the current decade [Hig 89].

The design of an integrated circuit system using currently available computer techniques progresses in a sequence of well-defined steps (see Figure 1.1):

1. High-level synthesis: during this phase of the design process the behavior of the system, its components and the functionality of each component is described in a symbolic description language and verified automatically by high-level simulation tools

2. Logic Synthesis: at this level the interacting components are synthesized (generated) by means of tools that convert high-level symbolic descriptions of behavior into low-level logic implementations using *logic cells* - the basic building blocks of digital logic. The cell level description is usually called a *netlist*

3. Physical Synthesis: the netlist is converted into a layout and a precise geometric shape, size and location on several two-dimensional surfaces are defined for each cell

The reason for the division of the design into these three steps is to make the design process tractable. However, we are already beginning to witness a blurring at the boundaries between these steps and most researchers believe that future computer aided design systems will unify the steps into "system synthesis".

1.2 The Motivation for This Work

During the inception of the field of CAD designers were preoccupied with packing many devices (transistors) into the smallest area possible. This increased the number of chips that could be fabricated on a silicon die leading to increased revenue. At the time I began this research there were many techniques for area optimization at all three levels of IC design and area optimization was accepted as a reasonably well understood topic. Techniques for performance optimization were limited to the high-level synthesis step due to two reasons: (1) the high levels of the design were most often done manually and the methods had achieved sufficient maturity that some of the manual techniques were being translated into computer algorithms and (2) performance optimization has been a much harder problem primarily because the goal is to optimize performance with a minimum impact on the quality of other attributes.

As the first generation of commercial RISC chips gained acceptance in the computing world, designers began to realize (retrospectively) the need for computer tools that could assist in designing chips with predicable performance. The early RISC machines were almost an order of magnitude faster than the CISC computers primarily because of peformance-oriented high level design techniques. If performance optimization at the high

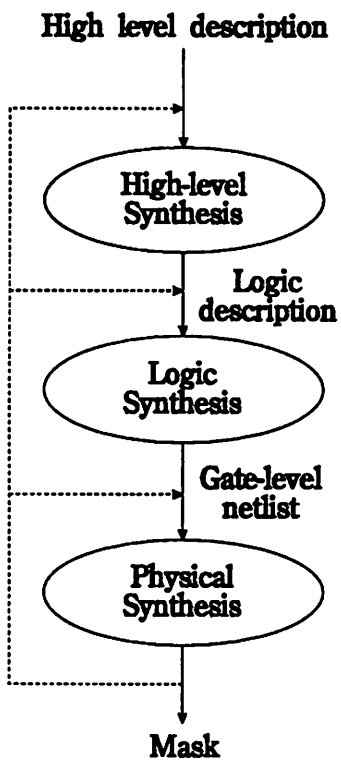


Figure 1.1: The flow of a typical IC design

level could bring about such a drastic improvement, what potential the other two steps of IC design held! It was in this setting that the work in this thesis evolved.

1.3 The Focus of This Work

There is potential for performance optimization at each level of the IC design process: high-level synthesis, logic synthesis and physical synthesis. Each area is unique in the problems it poses for the CAD algorithm developer and the problems are quite tightly coupled; one cannot truly perform optimization at the logic level without knowledge of the physical level for example, but the intractability of the whole synthesis problem often necessitates doing optimizations at each level independent of the other.

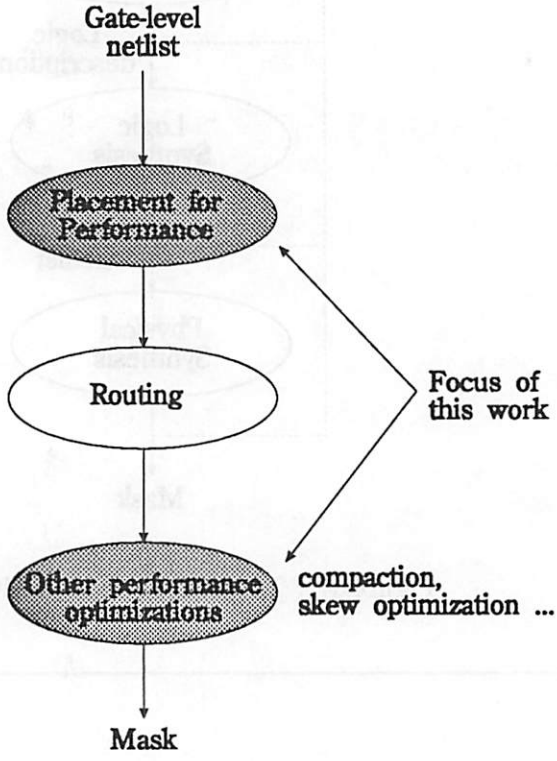


Figure 1.2: The focus of this thesis

The focus of this work is the area of *performance directed physical synthesis.* At the time of this work, the high-level designers were busy improving upon their successful performance improvement ideas and the logic designers were not far behind [Brayton 87]. Successful performance optimization techniques for physical design were woefully lacking and hence there existed a strong motivation for my work. The state of the art in commercial physical design tools during 1990 was such that Guide Arnout, the vice-president of engineering at Silvar-Lisco (a company that manufactures CAD software) [Hig 90] remarked: "for high-performance designs beyond 50,000 gates[1] ...designers should begin to consider performing full placement and routing", manually! What greater motivation could there be to look for new physical design techniques to break the 50,000 gate barrier?

The optimization techniques described herein are such that they do not modify the logic or perceived behavior of the system. One feature that is pervasive in all the optimization algorithms developed in this work is the exploitation of the structure in the problems. Just as the RISC designers were able to make a major leap forward by restructuring their designs, the techniques described here are able to operate effectively on designs much larger than before by taking a structured look at the problem. Two techniques for performance optimization are presented:

1. Performance Directed Placement
2. Clock Skew Optimization under realistic conditions

Neither technique involves modifying the logic circuitry or the high-level architecture of the design. They are intended to be applied after the behavior and logic description of the system are sufficiently crystallized to permit a description of the system in terms of logic cells and interconnections between cells.

1.3.1 The Performance Directed "Placement" Problem

The *placement* problem is that of determining exact locations for the logic cells of a digital system on a rectilinear piece of silicon such that performance, area, power and other electromagnetic requirements are met. It is not an easy problem to solve. Even a simplified form of the placement problem as defined in [Sahni 88] is considered intractable and belongs to the class of problems known as "NP-complete" [Garey 79].

[1] a cell is composed of *gates* or complementary pairs of CMOS transistors. A 50000 gate circuit would typically contain about 5000-10000 cells.

Therefore, to avoid compounding the difficulty of the problem, power and electromagnetic requirements are given secondary consideration and are usually dealt with after a layout that meets performance and area constraints is constructed.

Depending on the level at which placement is being done, the process can be broadly categorized into: (1) general-cell or (2) small-cell placement. The *small-cell* type of design is the target of this work. A *small-cell* IC is one in which the size of individual cells is relatively small compared to the dimension of the chip. Such designs are characterized by several thousand cells packed in an area measuring a few millimeters in length and width. Not only is the potential for performance optimization at the physical level of such designs the greatest as described below, but a *significant* number of application specific circuits manufactured today use cell-based IC technology. It is predicted [VLS 87] that in the long term, gate-arrays [2] and cell based ICs can account for upto 80% of the application specific IC (ASIC) market.

How Does Physical Layout Affect Performance?

One of the most important trends in silicon technology has been the scaling down of device and line geometries. The minimum feature width of devices that can be etched on silicon has decreased from about 8 microns in the late seventies to about 0.8 microns today. The speed of the metal-oxide semiconductor transistor which is the basic building block for cells, has increased dramatically by a factor of 20. Unfortunately, aggressive scaling has resulted in interconnect capacitance becoming the dominant determiner of performance in today's circuits. Informally, a *net* is the set of wire connections that link a cell to all of its output cells. A cell drives its outputs through interconnect wires belonging to the *output net* of that cell and as the wire capacitance increases, the time taken to charge and discharge the net increases. In fact, according to [El-Mansy 88] "the value of capacitance is increasing at a fast pace and promises to be the major performance limiter". In addition, the size of the chips manufactured today has increased, compounding the interconnect delay problem because signals have to travel longer lengths from input to output. It is clear that the layout of cells can significantly affect the performance. In particular, the placement of cells on the chip can have a significant impact on the performance of the IC. For small-cell ICs, performance driven placement is particularly important because of the large contribution

[2] gate arrays are large scale ICs with a regular array-like topology

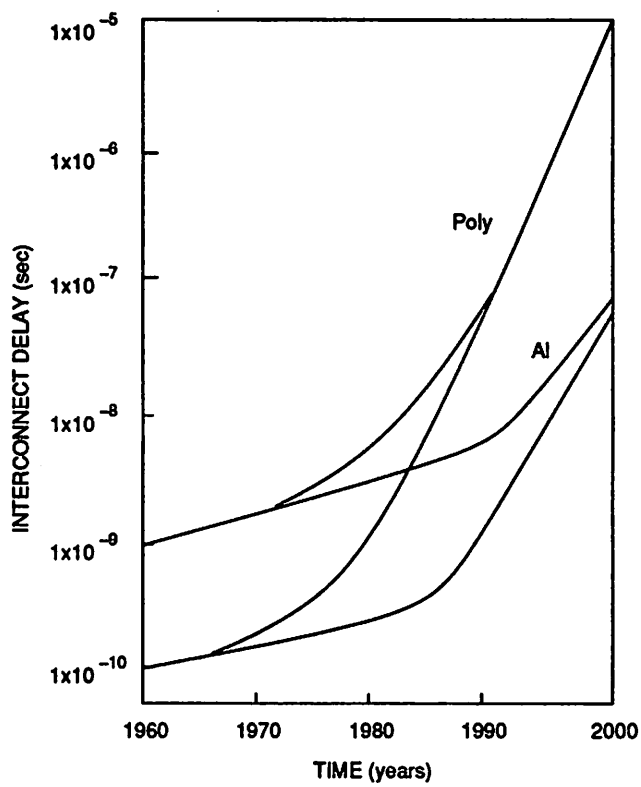


Figure 1.3: Graph illustrating the increase of global interconnect delays with time

of interconnect wire to the delays.

Figure 1.3 taken from [Bakoglu 90a] shows the increasing contribution of interconnect delay for long interconnect wires. Short interconnect wires have an equally significant contribution to delay as the following analysis illustrates. Consider a simple example to get an idea of the contribution of interconnect delay in today's ICs. The values in this analysis are derived from an industrial cell library. Let G_s be a cell as shown in Figure 1.4, driving

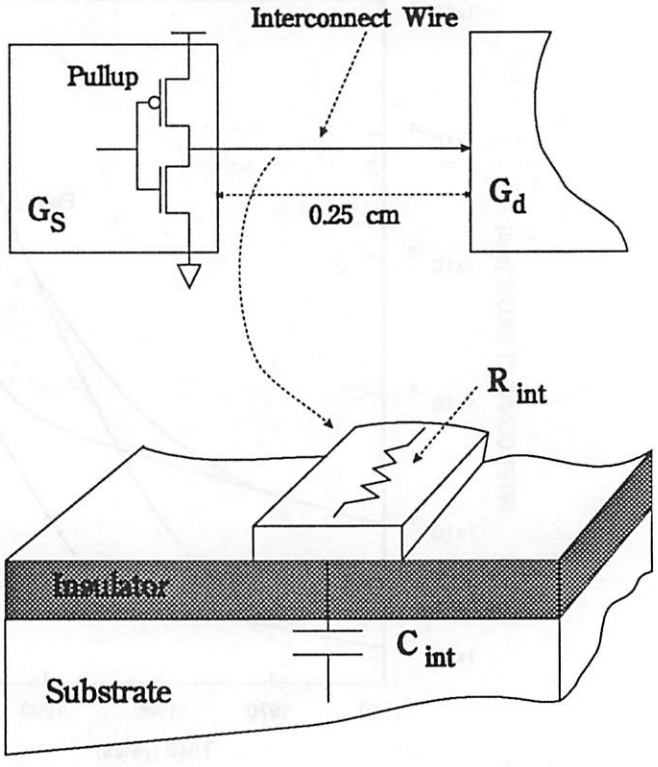


Figure 1.4: Example to quantify the contribution of wire delay

a length of interconnect wire that connects G_s to cell G_d. Typical cell delays for 0.8 micron technology are between 0.5 and 0.7 ns. Let us compute the RC delay contribution when the pullup transistor of G_s charges the interconnect wire. The average "on"-resistance of a pullup in performance optimized 0.8 micron CMOS technology is about 2.0 KΩ. The capacitance per unit length of 0.8 micron Aluminium wire is 2.0 pF/cm. Consider a chip

2.0 cm on a side. Assume that the wire connecting G_s to G_d travels across one eighths of the chip width (0.25 cm). The RC delay in such a wire is proportional to

$$2.0\text{pF/cm} \times 0.25\text{cm} \times 2000\Omega = 1.0\text{ns}$$

This value is already as much as the delay through a cell and the wire delay to cell delay ratio is expected to continue increasing in the future. When we consider the fact that the average interconnect length of a net on a 2.0 cm × 2.0 cm chip using Rent's rule is about 0.25 cm (see [Bakoglu 90c]) and there are about 15-30 levels of logic in a typical IC and thus 15-30 nets along a typical path, it is obvious that interconnect delay is a significant proportion of the total delay along a path in a circuit.

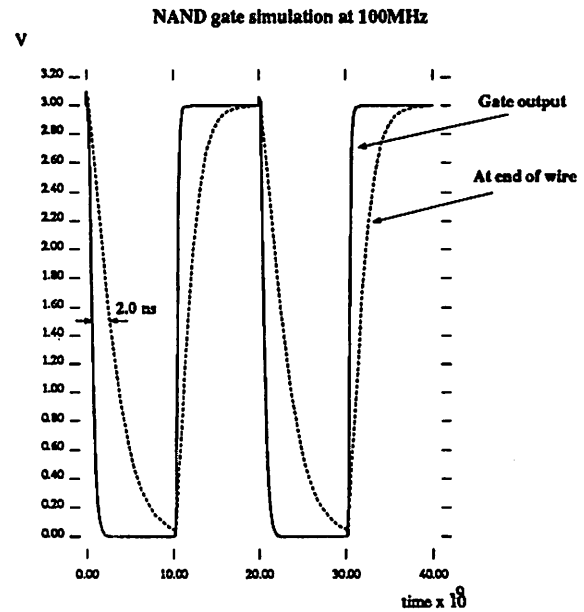


Figure 1.5: Simulated waveforms to illustrate wire delay

Figure 1.5 shows the simulated waveforms at the beginning and end of a 0.25cm line driven by a NAND cell in 1.25 micron CMOS technology. The commonly accepted measure of delay is the time between corresponding 50% points on the two waveforms. The figure shows the significant contribution of wire delay which is about 2.0 ns for the falling transition and about 1.0 ns for the rising transition.

1.3.2 The State of the Art

A performance driven computer tool should satisfy at least two goals in order to be useful:

1. It should deliver circuits with predictable performance

2. It should be efficient i.e., it should assist in designing faster chips in a short time.

The quest for such a tool in the area of physical layout started receiving attention in the early eighties. In one of the first documented attempts at performance optimization, [Wolff 78] developed techniques for optimizing the power and timing of LSI chips using ideas from physics to place the cells. Connections between cells were modeled as "springs" and a location was found for each cell that minimized its "potential energy". The model was a crude approximation of the wirability of the circuit at best. [Dunlop 84] did pioneering work in this area by designing a system that worked as follows: the circuit was initially laid out and then completely simulated on a computer to determine which input-to-output paths were limiting the performance. The layout was subsequently readjusted and then simulated again. The process was repeated several times till a layout that satisfied the performance and area requirements was obtained. The approach had several problems: (1) simulation was time-consuming, (2) it was not clear at the time how to modify the layout to ensure better performance, (3) sometimes the iterative process did not converge. Later, [Burstein 85] developed performance-driven circuit partitioning heuristics that resulted in some performance improvements with little loss in the wirability of the resulting circuit. However, the heuristic could not guarantee that the resulting chip met the performance requirements. In 1986, [Teig 86] described a method that interleaves timing analysis with placement and routing steps to successively refine net weights. The net weights are a measure of a net's criticality to timing and are used to bias layout tools.

An important development in the area of performance came from [Hauge 87, Nair 89b]. They developed a method of generating constraints on the sizes of nets that connect cells so that performance would be guaranteed. Any physical layout satisfying their bounds on the lengths of nets would also satisfy the timing constraints. However, no technique was given for positioning the cells that would guarantee the net bounds. [Jackson 89] developed a linear-programming approach for finding a layout that minimizes the *estimated* cycle time of a circuit. The true timing constraints and the measure of wirability of the

resulting circuit are modeled by approximations in this approach. The method works well on small examples, but on circuits of moderate size, it takes hours or even days to find a layout. The authors [Youssef 89] attempt to predict the critical path before placement of a circuit using correlation coefficients derived from historic data. The problem with predicting critical paths prior to placement is that the circuit performance cannot be guaranteed. [Marek-Sadowska 89] used ideas from *rectilinear distance facility location* and partitioning to minimize wire delay, but they too could not guarantee the performance of the resulting placement. [Prasitjutrakul 89] and [Ogawa 86] directed their efforts towards the general-cell style of physical layout.

Recently, Lin and Du [Lin 90] developed a constructive method of placing cells sequentially with a cost function that tries to capture timing behavior, but cannot guarantee satisfaction of the timing model. In [Sutanthavibul 90], the authors define regions in which cells that constrain the performance of the circuit must be placed and then attempt to meet these requirements by means of heuristic assignment of cells to regions on the chip. Donath and others in [Donath 90] use the technique of *simulated annealing* [Sechen 85, Vecci 83], which while being very effective, takes a long time to produce satisfactory results.

1.3.3 Contributions of This Work

In summary, many attempts at performance optimization have been heuristic and they make few guarantees regarding the resulting performance or area of the circuit. Those that have succeeded at achieving predictable performance, have often suffered from lack of computational efficiency.

One of the important contributions of this work [Srinivasan 90a, Srinivasan 91] is a technique that uses accurate and predictable approximations of wirability and timing of the circuit and finds a placement that satisfies the approximated timing constraints and minimizes the measure of wirability significantly faster than previous work. Thus, it is in keeping with the two basic goals of a performance oriented computer tool. The performance improvements are controllable and it is possible to make a choice between a range of circuits that tradeoff area and performance. Another key feature of this work is that its generality can accommodate a variety of wirability and timing models.

Many of today's VLSI circuits are *sequential* circuits, i.e., they contain memory or storage elements in addition to combinational or switching circuitry. These storage elements

are orchestrated by a *clocking* scheme that controls the precise instants at which the memory elements are active. In addition to optimizing the location of combinational and sequential elements, the model can take clock skew into consideration and simultaneously find optimal locations for the elements of a clock distribution tree.

The technique has been successfully applied to a wide range of circuits to generate final layouts. It yields layouts that are significantly faster than those produced by current area-optimizing placement techniques with little or no increase in chip area.

1.3.4 The Skew Optimization Problem

The clocking methodology of a digital system synchronizes the activity of storage elements. Researchers [Fishburn 81, Mijuskovic 87, Boon 89, Friedman 86] have developed a methods for controlling the exact instants at which each storage element is fired in such a manner that the *cycle time* of the system is minimized, where cycle time is defined as the time between successive clock pulses applied to a storage element. The smaller the cycle time, the faster the rate at which the digital system can operate. The approaches while effective, proved to be either unreliable on real systems or had to be conservatively applied because the delays through logic cells or blocks vary from chip-to-chip. For instance, when hundreds of copies of a system are constructed, there is variation in the delays through the blocks due to processing, varying temperature of operation and different environmental characteristics. The delays are functions of random variables whose distributions can be characterized by historic data. Traditionally, designers have dealt with varying delays either by performing worst-case analysis or by using rules-of-thumb. For a competitive design that pushes technology to the limits, an improvement in performance can be optimal, often at the risk of failure. Previous approaches have also neglected a number of practical issues. For example, it is not always possible to control every single synchronizing element on a chip. The number of control lines is limited by manufacturability considerations. The times at which the synchronizing elements may be fired are usually required to be multiples of some basic unit.

There has been very little work in the area of performance optimization in the presence of varying delays and practical constraints. [Fishburn 81] suggested an approach that considers several samples of the system being optimized and minimizes the worst deviation from the desired clock period over all samples. However, there is little generality

in the approach. The method uses cumbersome mathematical optimization techniques that could take a significant amount of computing resources for large-scale circuits.

In this work [Srinivasan 90b], a systematic approach for obtaining reliability-cycle time tradeoff curves is presented. The contribution of the work is (1) a general mathematical model that considers varying delays, (2) a technique that allows designers to obtain the smallest clock period such that a given reliability level for the system is achieved and (3) a formulation that enables a number of practical considerations to be resolved effectively. The mathematical model encompasses a broad class of systems ranging from ICs to multi-chip modules (MCMs) to printed circuit boards. Efficient algorithms have been developed for solving the *stochastic optimization* problem and obtaining the optimal clock period, where optimality is defined as the most reliable clock period that satisfies a given set of constraints. Another result of the work was the development of an efficient polynomial time algorithm for solving the problem when the delays are deterministic taking the graph structure of the problem into account. Tests on real examples indicate that useful reliability-cycle time tradeoff curves can be obtained efficiently. In the deterministic case, significant improvements in the cycle time can be made without restructuring the logic or redesigning the circuit.

1.4 Outline of the Thesis

The thesis can be roughly divided into two parts. The first describes the performance-directed placement problem and the second, the skew optimization problem. Within each part, the work is presented in chronological order. I believe that the purpose of a thesis should not only be to convey the ideas of the writer, but also his or her thinking process for that gives the reader greatest insight and stimulates the development of new thought. One way to give the reader a glimpse of my thought process was to describe the ideas as they evolved and that is how I have presented them in this thesis.

The second chapter lays the definitional foundation and describes the assumptions used in developing this work, and the third chapter called "The First Cut" describes early work on the performance-directed placement problem and the lessons learned. The fourth chapter describes how I attempted to improve the shortcomings of the first attempt. The fifth chapter called "Polishing the Cut" describes the evolution of the techniques into a reasonably satisfying form. The reader who is interested in quickly gleaning the key techniques

developed in this thesis may read the second and third chapters and then skip to the fifth (risking some loss of insight as a result). The sixth chapter describes the experiments conducted on the performance driven placement technique and draws conclusions from them. The seventh chapter describes the skew optimization techniques developed and conclusions and final thoughts follow that.

Chapter 2

Definitions and Assumptions

This chapter describes some of the early decisions taken during the course of the work and why they were made. Wherever possible, I make an attempt to point out the impact a certain decision had on subsequent work.

2.1 Introduction

Placement begins when the Logic Synthesis step of automatic IC design terminates. At this stage, a circuit is described in terms of logic cells, their functions, their delays and the interconnections between logic cells. As pointed out in the introduction, the focus of this work is on small-cell ICs and the number of cells is expected to be enormous - 10,000 is not uncommon. A performance directed placement algorithm for such a large number of logic cells must satisfy at least two basic criteria:

1. it should produce placements whose performance is predictable to a fair degree
2. it should be efficient in terms of computer memory and time

The first point needs no further elaboration. Efficiency is essential because the design process is iterative and during the design of a large scale IC the placement may have to be performed hundreds of times. A placement technique that takes minutes instead of hours has a significant impact in reducing the time to market an IC.

2.2 Definitions

After the logic synthesis step is complete an IC may be abstractly viewed as a collection of *modules* (or *cells*). The modules are interconnected by means of *nets* and a net is defined as the set of modules (or interchangeably, pins on modules) that it interconnects. Nets attach to the modules at *pins* (or *terminals*). Let $\mathcal{M} = \{m_1, m_2, \ldots, m_M\}$, $\mathcal{N} = \{n_1, n_2, \ldots, n_N\}$, and $\mathcal{P} = \{p_1, p_2, \ldots, p_P\}$ respectively denote the sets of modules, nets, and pins. The modules can be categorized by function as:

1. combinational: a module that computes a logic function based on its inputs and produces an output
2. synchronizing: a storage module that has data input, data output and clock signals. When the clock signal is active, the data input is sampled and stored internally and after some delay, the data output signal assumes the same value as the internally stored signal
3. primary input (PI): receives inputs from the external world outside the chip
4. primary output (PO): presents signals from the chip to the external world

To simplify the discussion, it is assumed that each cell computes only one function, each primary input receives only one signal from the outside world and each primary output presents only one signal to the outside world. Deviations from these assumptions can be handled by trivial modifications to the theory presented herein. Let f represent the number of primary inputs, and g represent the number of primary outputs; thus, there are $M - f - g$ *internal* modules where an internal module is one that does not receive any signals directly from the outside world. The chip is assumed to be a two dimensional region and hence we can assign a coordinate to the center of a cell m_j denoted by (x_j, y_j). In following discussion, the term *cell location* denotes the coordinate of the center of the cell. The coordinates of the pins on a cell can be derived from the coordinate of the cell itself since the pins are fixed on the cell. Let x_{p_i} and y_{p_i} denote the x and y coordinates of pin p_i on the chip. The locations of the cells of a net n can be indexed by

$$(x_i, y_i) \ \forall \ m_i \in n$$

The locations of pins of the net can be indexed by

$$(x_{p_i}, y_{p_i}) \ \forall \ p_i \in n$$

2.3 Wirelength Models

2.3.1 Why Use Models of Wirelength?

The placement problem (with or without performance constraints) using an exact model of wirability as a cost function has been shown to be NP-complete [Sahni 88]. Placement is typically an iterative process and ideally, during the course of placement optimization one would like to completely route every placement obtained to evaluate its quality. Most of the routing problems generated by cell-based placements have also been shown to be NP-complete. Needless to say, using an exact model for wirability and evaluating the cost of a placement using complete routing would be much too inefficient even on the most powerful computers available today. Hence, one must resort to efficient estimates of wirelength that make the problem tractable.

One of the first choices to be made was to decide on a model of chip wirability. I chose analytical continuous-space models over the discrete models for several reasons. The work on area-directed placement using analytical models had been extremely successful in satisfying the two basic criteria of predictable area and efficiency [Kleinhans 91] [Tsay 88]. Secondly, performance constraints can be compactly expressed as analytical constraints and such constraints have been used to accurately model the behavior of digital circuits [Horowitz 84]. I believed that analytical timing constraints would integrate well with an analytical wirelength model and make a unified solution technique possible. One of the features of this work is that the solution technique is generalizable to different measures of wirability and timing. I present here three models, in increasing order of complexity and accuracy.

2.3.2 Quadratic Wirelength Model

This model was originally introduced by Hall in 1971 [Hall 70] and later used very successfully for producing high quality area-directed layouts by placement systems like Gordian [Kleinhans 91] and PROUD [Tsay 88]. The quadratic wirelength model uses the following estimator for the length of net n

$$L_n = \sum_{p_i, p_j \in n} ((x_{p_i} - x_{p_j})^2 + (y_{p_i} - y_{p_j})^2) \tag{2.1}$$

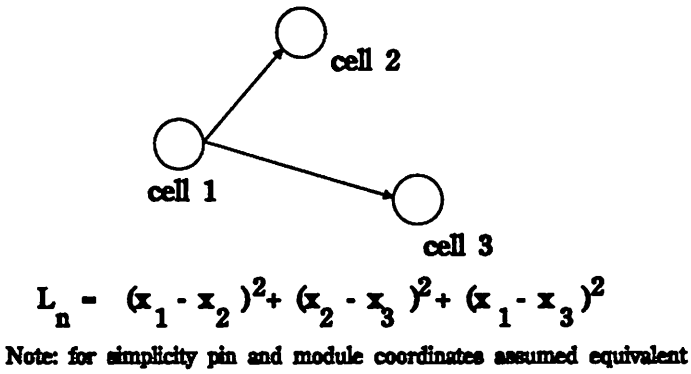


Figure 2.1: The Quadratic wirelength model

The estimate L_n is the square of the Euclidean distance between the pins on the net n. This approximation may be inaccurate for chips with large modules because it assumes that every pair of pins on a net is interconnected separately. However, for the type of ICs considered in this work (small-cell ASICs), the estimate has been shown to be accurate by [Hall 70], [Cheng 84] and [Tsay 89] and has been widely used in practice. Other quadratic measures have been introduced since the work of Hall. [Kleinhans 91] uses a model in which the wirelength of a net is modeled by the sum of the Euclidean squared distances of the cells from the mean coordinate location of the net. The difference between this and the model of Hall is minor and the analysis in this work remains unchanged.

The estimate of the cost of a placement can be written as

$$L = \frac{1}{2} \sum_{n \in \mathcal{N}} L_n \tag{2.2}$$

Note that since the pin locations can be expressed in terms of cell locations the function L is a function of cell coordinates. In further discussion, I will assume that the coordinate of a pin is the same as the coordinate of the cell to which it is attached. This does not detract from the generality of the techniques described since pin locations can be derived from cell locations. The effectiveness of the quadratic cost function has been researched in the past and [Hall 70], [Cheng 84], [Tsay 89] and [Kleinhans 91] show that it is a reliable model of the final routing.

2.3.3 Bounding-Box Model

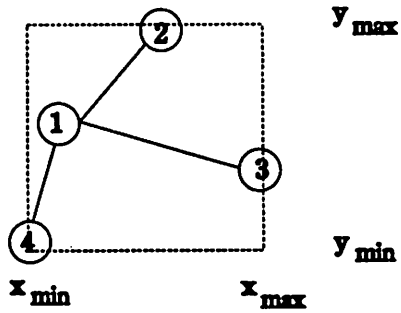


Figure 2.2: The net bounding-box model

Another successfully used wirelength measure [Sechen 88b] is the bounding-box model. Consider net n as shown in Figure 2.2. The estimator for the length of the net is:

$$L_n = x^n_{max} - x^n_{min} + y^n_{max} - y^n_{min} \tag{2.3}$$

where

$$x^n_{max} = \max_{p_i \in n}\{x_{p_i}\}$$
$$x^n_{min} = \min_{p_i \in n}\{x_{p_i}\}$$

and

$$y^n_{max} = \max_{p_i \in n}\{y_{p_i}\}$$
$$y^n_{min} = \min_{p_i \in n}\{y_{p_i}\}$$

The equation for total wirelength is identical to Equation 2.2.

2.3.4 Single-Trunk Steiner Tree Model

This model is the most complex and tends to estimate the final routed length of nets accurately. As experiments revealed (see Chapter 6), by using this model it was possible to achieve layouts with low area. As shown in Figure 2.3, in this model the pins of a net are assumed to connect to a single trunk that passes through the mean position

of the pins either vertically (Figure 2.3 (a)) or horizontally (Figure 2.3 (b)). In the figure, the expression for the wirelength estimate for one of the configurations is shown. To get an accurate estimate of the wirelength, the average net length of the two possible configurations is taken. Let

$$L_n^x = y_{max}^n - y_{min}^n + \sum_{p_i \in n} |x_{p_i} - \overline{x}_n|$$

$$L_n^y = x_{max}^n - x_{min}^n + \sum_{p_i \in n} |y_{p_i} - \overline{y}_n|$$

where $\overline{x}_n$ and $\overline{y}_n$ denote the mean x and y positions of the pins of net n. Then the single-trunk Steiner tree length of net n is defined as:

$$L_n = \frac{1}{2}(L_n^x + L_n^y) \tag{2.4}$$

While the techniques developed in this thesis are applicable to any of these wirelength

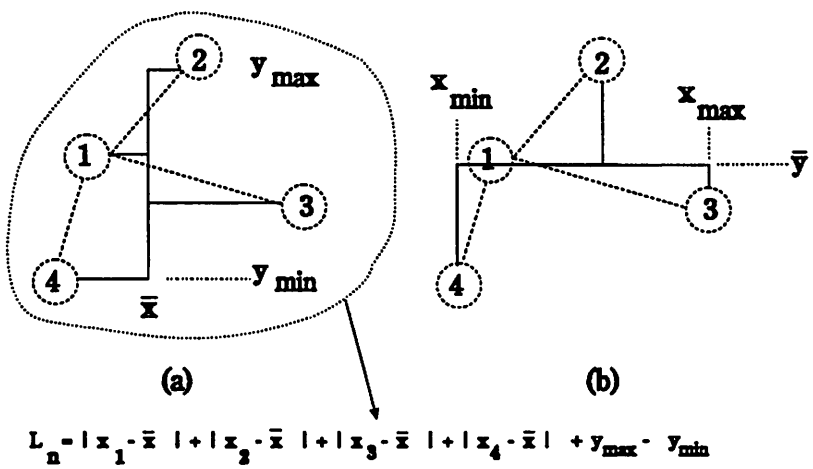


Figure 2.3: Single-trunk Steiner tree wirelength model

models, the analysis will be biased heavily in favor of the first model, the quadratic wirelength. It should be noted that any one of the models presented above could be used in the general solution framework presented in this thesis. Restricting attention to one model makes subsequent discussions much simpler. The following mathematical property of the wirelength models is fully exploited in succeeding chapters and is of key consequence:

Proposition 2.3.1 *The total wirelength cost function based on the quadratic, bounding-box or single-trunk Steiner tree wirelength model is a convex function of the cell locations.*

Proof. The convexity of the quadratic wirelength function is proved in 3.2.3. The Steiner tree wirelength consists of the sum of absolute valued linear terms, each of which is convex. Thus the total wirelength is a convex function of the cell positions.

The bounding-box wirelength function consists of the difference between the maximum of linear terms (for example $\max_{p_i \in n}\{x_{p_i}\}$) and the minimum of linear terms. The maximum of a set of linear terms and the negative of the minimum of a set of linear terms are both convex functions (see [Murty 83]). Hence, the bounding-box wirelength consists of the sum of convex functions and is itself convex. □

2.4 Timing Models

2.4.1 Timing Problems in Digital Logic

Consider a block of combinational logic receiving inputs from synchronizing elements and presenting outputs to synchronizing elements as shown in Figure 2.4. This is a general sequential machine model and in this work it is assumed that cycles of combinational logic do not exist. If signals are applied to the inputs of the combinational logic, then after some time T_{long} the circuit's outputs will settle to values that are a function of the circuit's inputs. If the outputs are sampled before T_{long} units of time have elapsed the circuit may not behave as designed. Thus, the longest path delay through the combinational logic constrains the earliest time that the output may be sampled. Figure 2.4 illustrates the relationship between the longest path delay T_{long}, the clock period CP, the skew to the synchronizing clock pins T_{skew}, the set-up time of the synchronizing elements T_{su}, and the synchronizing elements internal clock to output delay $T_{clk \rightarrow Q}$. This relationship is expressed as follows

$$CP \geq T_{long} + T_{skew} + T_{clk \rightarrow Q} + T_{su} \tag{2.5}$$

If equation 2.5 is not satisfied, then a long path timing problem exists in the design. The short path problem occurs when a signal arrives at the output too early and races around the circuit before the end of one clock cycle. This happens if the clock period is too large

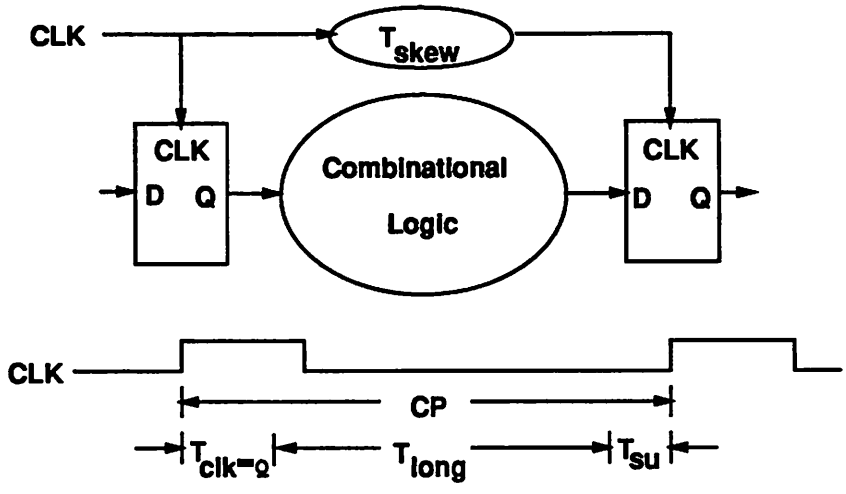


Figure 2.4: Figure illustrating clocking constraints

and the synchronizing elements in the circuit are of the *level-sensitive* type. [Wakerly 90] has an excellent discussion of this problem.

2.4.2 Assumptions

This work restricts attention to one specific timing problem: the long path problem and ignores the related short path problem. Most designers consider the long path problem to be the key timing problem in large scale digital ICs. Ad hoc methods (such as adding delay lines) to fix the short path problem usually work well. However, the long path problem is not usually amenable to ad hoc fixes.

It is assumed that cell signal flow is unidirectional for every input-output conducting path in a cell. Similarly, each net has a signal direction associated with its output pin. Associated with every signal flow is a rising and falling delay that is a function of the corresponding cell and interconnect delay models. A single delay value is calculated for each signal flow that is based on the rising and falling transitions. The methods to be discussed are generalizable to the case of separate rising and falling delays [Hitchcock 83]. Each synchronizing cell is assumed to have a clock pin, data-input pins, and a data-output

pin. Synchronizing cells may be allowed to move freely within the chip along with other combinational logic cells. For simplicity of discussion it is assumed that edge-triggered synchronizing elements are used. The methods described are generalizable to the case of level-sensitive latches.

The performance of a synchronous digital IC is inversely proportional to the circuit's cycle time or clock period. A *path* is defined to be a sequence of interconnected modules and nets with a well-defined starting point and ending point (the starting and ending points are represented by modules). A *critical path* is a path whose delay does not meet the timing requirements of chip.

2.4.3 Graph Representation of Chip Timing

Let the digraph $D_T(V, A)$ represent the integrated circuit in the physical/timing domain. Let the vertex set V be in one-to-one correspondence with the set of pins. Arc weights $d(v_i, v_j)$ denote the pin-to-pin signal propagation delays for all $(v_i, v_j) \in A$, and arc direction represents the direction of signal flow in the circuit. Also, let A^I and A^E model the signal behavior *internal* and *external* to all cells respectively; thus, internal signal arcs represent cell signal flow while external arcs represent net signal flow.

$$A = A^I \cup A^E \tag{2.6}$$

Let $\{v_1, \ldots, v_{M-g}\}$ represent the cell output pins in the circuit (it is assumed that each net is driven by a single-output pin and that primary inputs have no input pin and primary outputs have no output pin) and $\{v_{M-g+1}, \ldots, v_P\}$ correspond to the cell-input pins. Assume that p_i is the output pin of m_i and connects to n_i. In the event that a cell has more than one output pin, the cell may be replicated for each output with identical nets feeding each replicated cell and all the copies of the cell are constrained to a common location during physical design. A path Ψ, is defined by an unbroken sequence $(v_s, \ldots, v_e)$ of vertices that uniquely occur along the path. Delay in an integrated circuit may be viewed as consisting of two components: cell delay and net delay. Let the delay of module m_i be characterized by

$$d(v_j, v_i) \; \forall \, (v_j, v_i) \in A^I \tag{2.7}$$

and let the delay of net n_i be characterized by

$$d(v_i, v_j) \; \forall \, (v_i, v_j) \in A^E \tag{2.8}$$

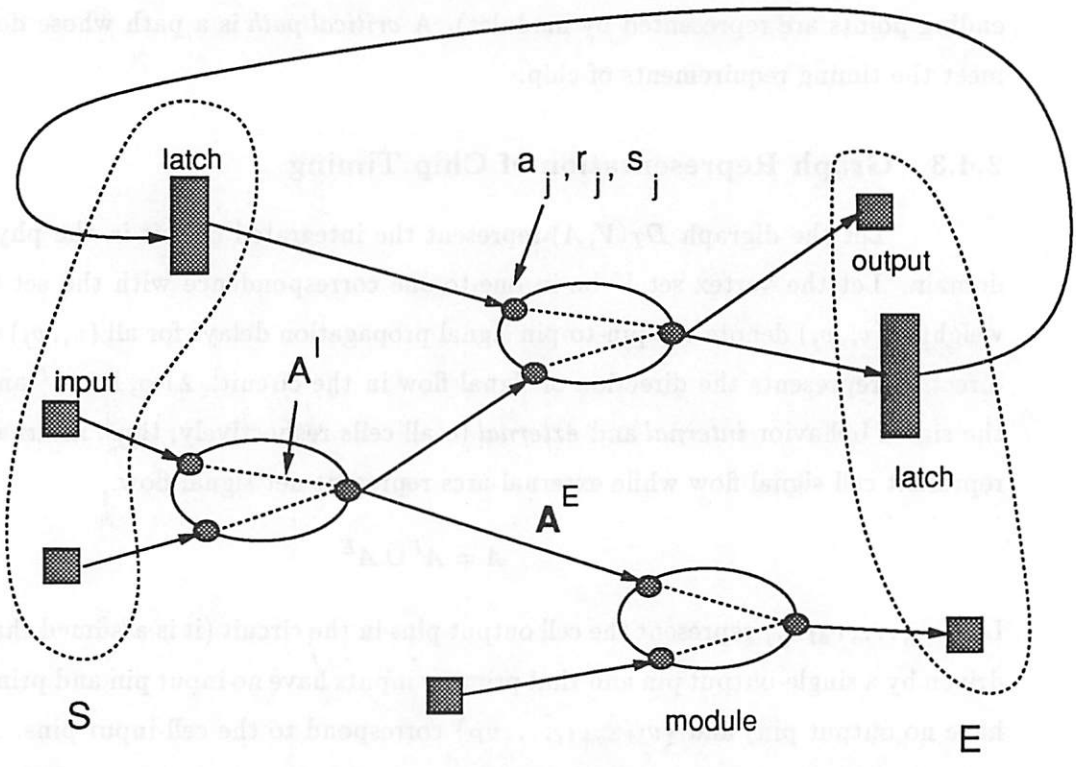


Figure 2.5: The timing graph D_T

The arcs and their meanings are illustrated in Figure 2.6. The greater flexibility of this multiple-arc cell and net model permits more accurate modeling than single cell and net delay models. This is particularly important when it becomes necessary to model different pin-to-pin net delays for aggressively scaled technologies where interconnect resistive contributions become significant. Let E denote the set of vertices representing path end

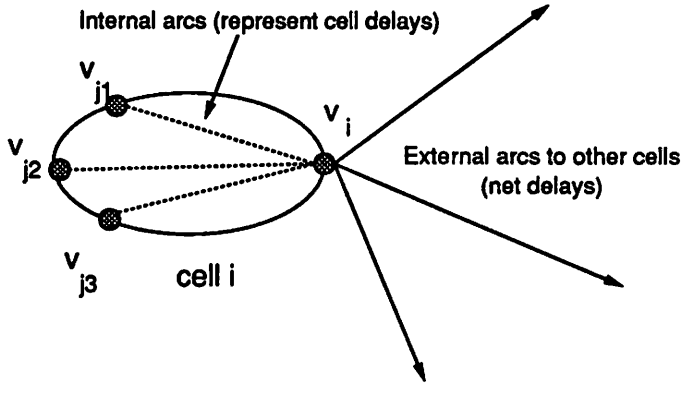


Figure 2.6: Internal and external arcs of a cell

points that correspond to the input pins of primary outputs and the data-input pins of the synchronizing modules. Associated with each path endpoint vertex is a *required arrival* time r_i specified by the designer of the circuit. In a similar manner, let S denote the set of vertices representing path starting points that correspond to the primary inputs and data-output pins of the synchronizing modules. Associated with each path starting point vertex is a designer specified *actual arrival* time a_i.

Path delay in the circuit is computed by a block-oriented search [Hitchcock 83]. Actual arrival times for cells not in S are determined in a breadth-first manner, beginning at the path starting points and terminating at the path ending points. The worst-case actual arrival time a_j and an arbitrary vertex is given by

$$a_j = \max\{a_i + d(v_i, v_j) \mid \forall(v_i, v_j) \in A\} \tag{2.9}$$

The required arrival times specified for the path end points may be propagated in a backward breadth-first manner through the circuit starting from vertices in E so that requirements

on the required arrival times for vertices not in E may be determined. The required arrival time r_i for an arbitrary vertex is defined to be

$$r_i = \min\{r_j - d(v_i, v_j) \mid \forall (v_i, v_j) \in A\} \tag{2.10}$$

Based on the calculation of actual arrival and required arrival times for all v_i, a *slack* s_i may respectively be defined as

$$s_i = r_i - a_i \tag{2.11}$$

Slack values are useful in characterizing the timing behavior of a circuit. A negative value of s_i for v_i indicates that a violation of a timing constraint has occurred.

Definition 2.4.1 *The timing of the chip is said to be feasible if and only if* $s_i \geq 0, \forall v_i \in V$.

A critical long path is defined as follows:

Definition 2.4.2 *A critical long path* Π *is a path* Ψ *in which the sequence of vertices* $(v_s, \ldots, v_e)$, $v_s \in S$ *and* $v_e \in E$ *comprising the path all have slack values less than zero.* $\Pi = \{v_i \mid s_i < 0 \ \forall \ v_i \in \Psi\}$

Thus, a necessary and sufficient condition for the non-existence of long paths is $s_i \geq 0, \forall v_i \in V$.

The arc weights $d(v_i, v_j)$ for all $(v_i, v_j) \in A^E$ are a function of the positions of the pins defining the cells. Let $X_i = (x_k)^T, \forall m_k \in n_i$ be the vector of x locations of pins on net n_i. Y_i is similarly defined.

Proposition 2.4.1 *Let* $d(v_i, v_j) = f(X_i, Y_i), \forall n_i \in \mathcal{N}$ *be any convex function corresponding to the arc* (v_i, v_j). *Then, the timing constraints form a convex set.*

Proof. For each non-empty path $\Pi_{se} = v_s \rightarrow v_e, v_s \in S, v_e \in E$, let

$$d(\Pi_{se}) = \sum_{(v_i, v_j) \in \Pi_{se}} d(v_i, v_j)$$

If there is no path from v_s to v_e, let $d(\Pi_{se}) = -\infty$. The timing constraints are equivalent to the following constraints:

$$d(\Pi_{se}) \leq T_e, \forall v_s \in S, \forall v_e \in E$$

But $d(\Pi_{se})$ is the sum of convex functions and is therefore a convex function. So, $d(\Pi_{se}) \leq T_e$ is a convex set. □

2.4.4 Interconnect Delay

The delay in a length of interconnect wire driven by a MOS transistor is an extremely complex function of the MOS transistor characteristics, interconnect line characteristics and the neighborhood of the interconnect wire [Bakoglu 90a]. Choosing a model that would be efficient and yet accurate was one of the first tasks. Efficiency is of concern for the following reason. During the process of placement, cell locations constantly change and as a result interconnect delays change. The interconnect delays may have to be updated several thousand times during the course of a placement and there may be several hundred thousand wires interconnecting a large design.

Bakoglu in [Bakoglu 90b] has presented an interconnect delay model that is the basis for the model chosen in this work. Consider a net n consisting of a driving cell G_s and $|n| - 1$ receivers. Cell G_s drives a length of interconnect wire connecting G_s to cell $G_1 \ldots G_{|n|-1}$ as shown in Figure 2.7. In the model, the cell is replaced by an effective source

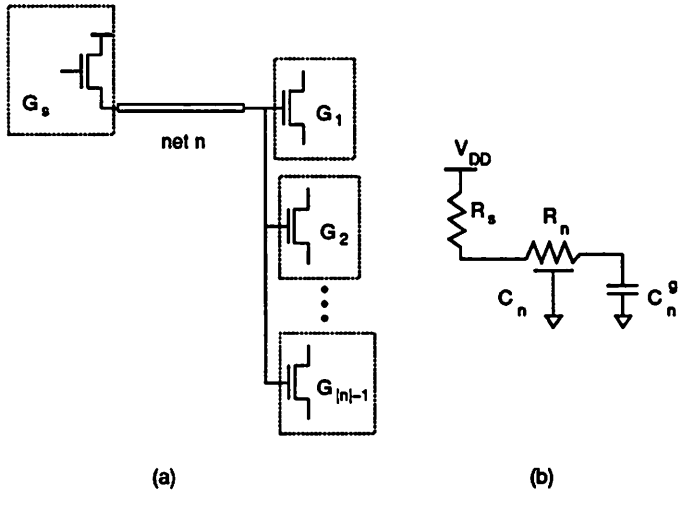


Figure 2.7: A net (a), and associated interconnect delay model (b)

resistance R_s which is computed from simulations. R_n represents the lumped resistance of the total length of interconnect wire of the net n. As shown by Bakoglu, an excellent approximation of the total wire delay of a sequence of cells on a path is obtained by taking

the products of resistive and capacitive terms and adding them. (The RC product is commonly referred to as the 50% delay). The delay in the interconnect is assumed to be composed of a "lumped" component d_l and a distributed component d_d:

$$d(s,d) = d_l + d_d$$

where

$$d_l = R_s \times (C_n + C_n^g) + R_n \times C_n^g$$

C_n represents an estimate of the total capacitance of the net n. C_n^g is the sum of the gate capacitances of all the output cells connected to G_s. d_d is the distributed RC delay due to the interconnect resistance and interconnect capacitance. This model has been shown in [Bakoglu 90a] and [Sakurai 83] to be accurate to within 4% for predicting the delay in aluminum interconnect wire for VLSI circuits.

2.4.5 Models for the Lumped Delay

In order to estimate d_l due to a net, we need to model the capacitance and resistance of the net during placement. Both these parameters are complex functions of the layout of the wires and neighboring nets. Once again, a choice has to be made that is efficient as well as accurate.

Analytical net capacitance models used in the past for large-scale circuits have relied on simplified net bounding-box estimates [Jackson 89] or similar techniques. However, the deviation of the final routed net length from the estimated value may be quite large. My goal was to be able to incorporate models of various complexities into a general framework. The techniques developed are capable of handling a variety of linear as well as non-linear delay models.

Bounding Box Model

In this model it is assumed that horizontal and vertical wires are routed on different layers and hence have different capacitance and resistance characteristics. Let C_h and R_h represent the capacitance and resistance per unit length respectively of horizontal wire and C_v and R_v the capacitance and resistance per unit length respectively of vertical wire. The

estimators for the capacitance and resistance of a net are:

$$\begin{aligned} C_n &= C_h(x^n_{max} - x^n_{min}) + C_v(y^n_{max} - y^n_{min}) \\ R_n &= R_h(x^n_{max} - x^n_{min}) + R_v(y^n_{max} - y^n_{min}) \end{aligned} \tag{2.12}$$

where

$$\begin{aligned} x^n_{max} &= \max_{p_i \in n}\{x_{p_i}\} \\ x^n_{min} &= \min_{p_i \in n}\{x_{p_i}\} \end{aligned}$$

and

$$\begin{aligned} y^n_{max} &= \max_{p_i \in n}\{y_{p_i}\} \\ y^n_{min} &= \min_{p_i \in n}\{y_{p_i}\} \end{aligned}$$

Single-Trunk Steiner Tree Model

This estimator uses a single-trunk steiner tree to model the length of a net. It is an accurate model and experiments on a number of chips yielded delay values close to delay estimates based on the final routing of the nets. The interested reader may wish to glance at Chapter 6.

$$\begin{aligned} C^x_n &= C_v(y^n_{max} - y^n_{min}) + C_h \textstyle\sum_{p_i \in n} |x_{p_i} - \overline{x_n}| \\ C^y_n &= C_h(x^n_{max} - x^n_{min}) + C_v \textstyle\sum_{p_i \in n} |y_{p_i} - \overline{y_n}| \\ C_n &= \tfrac{1}{2}(C^x_n + C^y_n) \end{aligned} \tag{2.13}$$

The resistance is modeled by similar equations.

Star Connected Net Model

Another model that was considered during the experimentation and yielded excellent results was the star-connected net. This model tends to overestimate the net length, but has the advantage of being simple and efficient to compute. Let (x_s, y_s) represent the location of the cell driving net n. The capacitance and resistance of the net are estimated as:

$$\begin{aligned} C_n &= \textstyle\sum_{m_i \in n} C_h|x_i - x_s| + C_v|y_i - y_n| \\ R_n &= \textstyle\sum_{m_i \in n} R_h|x_i - x_s| + R_v|y_i - y_n| \end{aligned} \tag{2.14}$$

2.4.6 The Distributed Delay Model

In the lumped part of the delay expression, the separation of the horizontal and vertical wire delay components is justifiable since the driver has to charge the vertical and

horizontal portions of the net. Unfortunately, with the distributed part it is not easy to determine how to separate the delay into individual components without performing a full routing for each net. The reason is that during the placement it is unclear whether the vertical section or the horizontal section of the net comes first along a path. The technique used in this work is to derive a "mean" resistance and capacitance for the distributed component. Let $C_w = \sqrt{C_h C_v}$ and $R_w = \sqrt{R_h R_v}$. R_w and C_w represent the geometric mean resistance per unit length and capacitance per unit length respectively. The geometric mean was chosen because the delay model consists of RC products and it yielded excellent results on simulations.

The distributed interconnect RC delay for the bounding-box model is estimated as:

$$d_d = 0.5 R_w C_w (x^n_{max} - x^n_{min} + y^n_{max} - y^n_{min})^2 \tag{2.15}$$

The factor of 0.5 which multiplies the expression arises because the resistive and capacitive terms are distributed (see [Bakoglu 90a]). For the single-trunk Steiner tree model, the equations are:

$$\begin{aligned} L^x_n &= (y^n_{max} - y^n_{min}) + \textstyle\sum_{p_i \in n} |x_{p_i} - \overline{x_n}| \\ L^y_n &= (x^n_{max} - x^n_{min}) + \textstyle\sum_{p_i \in n} |y_{p_i} - \overline{y_n}| \\ d_d &= \tfrac{1}{4} R_w C_w (L^x_n + L^y_n)^2 \end{aligned} \tag{2.16}$$

and for the star connected net model, we have:

$$d_d = 0.5 R_w C_w \sum_{m_i \in n} (|x_i - x_s| + |y_i - y_s|)^2 \tag{2.17}$$

2.4.7 Can interconnect resistance be ignored?

Assuming a constant value for R_n based on the average net length or neglecting it completely does not introduce significant error for current CMOS technology. Figure 2.8 compares the waveforms obtained using two models to a sophisticated lossy transmission line simulation based on [Roychowdhury 91]. The two models used are (1) the second order model that includes $d_l + d_d$ and (2) the first order model that includes only interconnect capacitance and neglects d_d and R_n. The length of the interconnect wire is 0.25 cm which is the typical length of a net in 1 micron technology on a 4.0 square cm chip. The simulation was performed using SPICE and the parameters for the driving cell as well as the interconnect wire were derived from a 1.0 micron process. The cell was driven by a 100MHz square

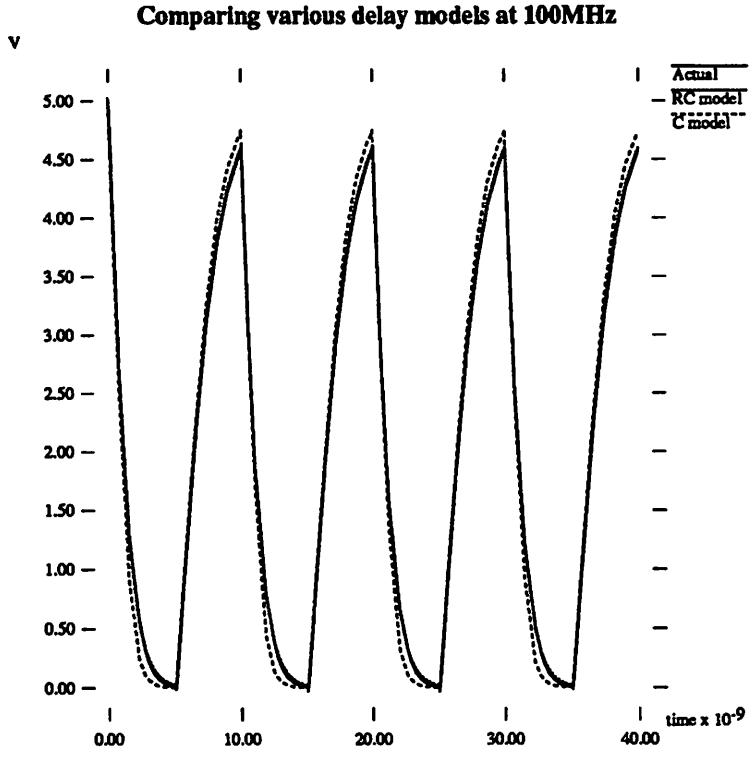


Figure 2.8: Simulation to compare various delay models

wave. As seen in the figures, the deviations for both the simplified models from the more sophisticated lossy transmission line model are minimal for this set of parameters The error introduced for this example by neglecting R_n completely is 5.3% for the falling transition and 5.5% for the rising transition of the output. The conclusion is that for current technology and for some time in the future, d_d and R_n may be safely ignored. However, for feature sizes below 0.6 micron, d_d begins to take on significant values and cannot be neglected.

2.4.8 Different Rising and Falling Signal Delays

In practice, different combinations of inputs to cells may have different delays to cell outputs. In addition, rising inputs may cause different cell delays and cell drive resistance values than falling signals. All of these generalizations can be dealt with in a similar manner to the analysis in the preceding sections. In order to keep the discussion simple, techniques will be presented for single delay values. However, the experimentation was conducted using a more complex delay model.

2.4.9 Putting it All Together

Having decided on models for the net capacitance and interconnect delay, we can now put them together. Consider cell m_i connected as shown in Figure 2.9. Let us compute the delay for the portion of a path defined by vertices (v_{j_1}, v_i, v_k). The quantity $d(v_{j_1}, v_i)$ is the intrinsic delay of the cell m_i from input vertex v_{j_1} to output vertex v_i and is a constant. Let the drive resistance of cell m_i be given by R_i. Suppose we are using the star connected net capacitance model. Let C_k and C_l represent the input gate capacitances of m_k and m_l respectively. The expression for the delay of arc (v_i, v_k) $(d(v_i, v_k))$ is

$$R_i \times (C_{n_i} + C_k + C_l) + R_{n_i} \times (C_k + C_l) + d_d$$

C_{n_i} and R_{n_i} are the capacitance and resistance respectively of the output net of m_i for the star connected net model and can be written as:

$$C_{n_i} = C_h(|x_k - x_i| + |x_l - x_i|) + C_v(|y_k - y_i| + |y_l - y_i|)$$
$$R_{n_i} = R_h(|x_k - x_i| + |x_l - x_i|) + R_v(|y_k - y_i| + |y_l - y_i|)$$

and the distributed delay component can be estimated as:

$$0.5 R_w C_w (|x_k - x_i| + |x_l - x_i| + |y_k - y_i| + |y_l - y_i|)^2$$

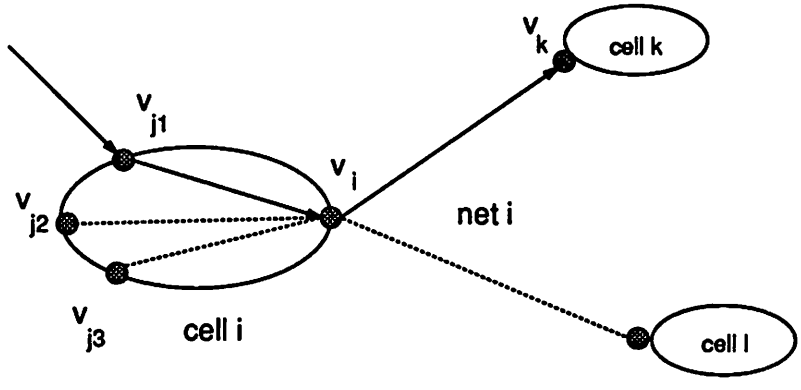


Figure 2.9: Example illustrating how to apply the delay models

Proposition 2.4.2 *The interconnect delay estimates based on the bounding box capacitance model, the single-trunk Steiner tree model or the star connected net model are convex functions of cell positions.*

Proof. It is easy to see that the lumped delay expressions are linear (absolute valued terms are considered linear) and hence convex. The distributed delay terms for all the three models consist of the square of a sum of absolute valued terms and can be represented by the general form:

$$f = [\sum_{i=1}^{k} |u_i|]^2$$

Let

$$g(u_1, \ldots, u_k) = \sum_{i=1}^{k} |u_i|$$

Let $\mathbf{u} = (u_1, \ldots, u_k)$. g is a convex function and $f(g)$ is an increasing convex function of g. In order to prove the convexity of f we need to show:

$$\alpha f(\mathbf{u}) + (1-\alpha) f(\mathbf{u}') \geq f(\alpha \mathbf{u} + (1-\alpha)\mathbf{u}')$$

for two distinct points $\mathbf{u}$ and $\mathbf{u}'$. We know that

$$\alpha g(\mathbf{u}) + (1-\alpha) g(\mathbf{u}') \geq g(\alpha \mathbf{u} + (1-\alpha)\mathbf{u}')$$

Since f is an increasing function of g, we have

$$f(\alpha g(\mathbf{u}) + (1-\alpha)g(\mathbf{u}')) \geq f(g(\alpha \mathbf{u} + (1-\alpha)\mathbf{u}'))$$

but, by convexity of $f(g)$ we have

$$\alpha f(g(\mathbf{u})) + (1-\alpha)f(g(\mathbf{u}')) \geq f(\alpha g(\mathbf{u}) + (1-\alpha)g(\mathbf{u}'))$$

□

2.5 Conclusions

To summarize, the key assumptions made in this work are listed below.

- The ICs considered have a large number of small cells. This enables estimates of wirability of the circuit to be made by simplified analytical wirelength functions.
- The circuit conforms to the model of a general sequential machine without combinational logic loops. It is possible to extend this work to the case where combinational logic loops exist provided the timing conditions that the loops must satisfy can be expressed by means of analytical equations.
- The circuit is operated at frequencies such that interconnect delays are convex functions of the pin locations that define the interconnect pattern.

Although the RC interconnect delay model described in this chapter is arguably a very basic one it is not the only model which can be used with the techniques in this thesis. More complex models like the Π ladder circuit and the T circuit can easily be incorporated should the need for greater accuracy arise. The gains obtained by using a more sophisticated model are limited since the delay functions for most of the models are *similar* in the mathematical sense, i.e, reducing the delay obtained by one expression also reduces the delay obtained by using the other.

Chapter 3

The First Cut

This chapter describes the first attempt at solving the performance-driven placement problem. A general recipe for solving the problem is described and then a specific solution technique is given for the the quadratic wirelength model and the bounding-box delay model. The general recipe follows a modified *active set* strategy. An active set strategy is a widely used mathematical programming technique for solving constrained optimization problems [Luenberger 84].

3.1 A General Recipe

3.1.1 Formulation

Let w denote the combined vector of x and y coordinates of cells in the circuit. The problem of minimizing wirelength subject to timing constraints can be stated as the following nonlinear programming problem.

$$\begin{array}{lll} \text{minimize} & \mathrm{L(w)} & \text{(GP)} \\ \text{subject to} & & \end{array}$$

$$\begin{array}{llll} a_j & \geq & a_i + d(v_i, v_j) & \forall (v_i, v_j) \in A \\ a_j & \leq & T_j & \forall v_j \in E \\ a_j & \geq & T_j & \forall v_j \in S \\ d(v_i, v_j) & = & f(X_i, Y_i) & \forall n_i \in \mathcal{N} \end{array} \qquad (3.1)$$

where T_j, $\forall j \in E$ and T_j, $\forall j \in S$ respectively represent the required arrival times at the path endpoints and the actual arrival times at the path starting points which are derived from the performance specifications and the clocking methodology. As before, L represents the wirelength measure which could be based on any one of the models presented earlier. X_i is the vector of x coordinates of the cells on net n_i and Y_i is similarly defined.

Thus, the performance-driven placement problem can stated very simply and it is tempting to assume at this point that it is amenable to solution by the use of widely available conventional techniques for mathematical programming. Unfortunately, conventional techniques fail on this problem even for circuits of small size, taking hours or even days to obtain a solution. For example, Jackson [Jackson 89] experimented with using the Simplex Method for solving the problem with the bounding-box model for wirelength and net capacitance and reported running times of the order of 16 hours for a circuit with about 1400 modules!

3.1.2 Why is the Problem Difficult to Solve?

Definition 3.1.1 *Active constraints at a point are defined to be those constraints that are satisfied with equality.*

Consider a simple example shown in Figure 3.1 (a). The delays through the cells are assumed to be the same for each input. Figure 3.1(b) shows the cell delays and net delays calculated as a function of cell positions for the current position of the cells. Let the required arrival time at all primary outputs be 7.4 units and the actual arrival time at all inputs be 0.0 units. The notation d_{i-j} is used as a short form of $d(v_i, v_j)$ and d_j denotes the intrinsic delay of module m_j which, for this particular example is the same for all inputs. The timing

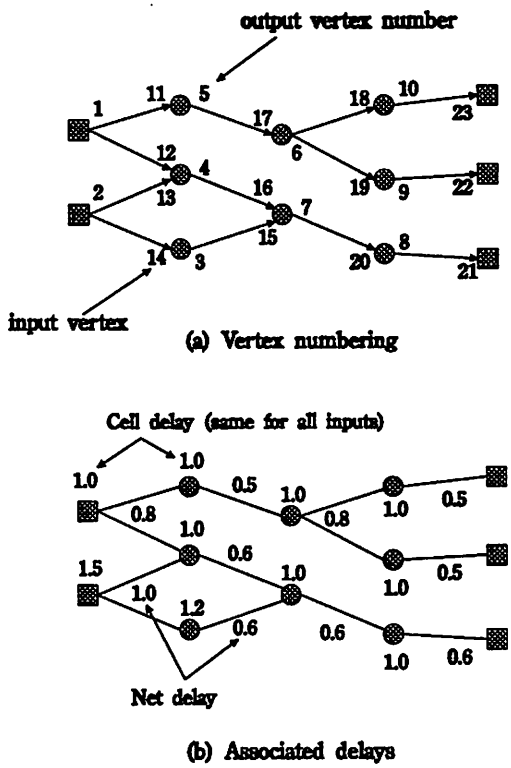


Figure 3.1: An example of a netlist, its vertex numbering and delays

constraints for this example can be written as follows:

$$\begin{aligned}
a_1 &= 0.0 + d_1 = 1.0 \\
a_2 &= 0.0 + d_2 = 1.5 \\
a_{12} &= a_1 + d_{1-12} \\
a_{13} &= a_2 + d_{2-13} \\
a_{14} &= a_2 + d_{2-14} \\
a_4 &\geq a_{12} + d_4 \\
a_4 &\geq a_{13} + d_4 \\
&\vdots \\
a_{23} &= a_{18} + d_{10} + d_{18-23} \\
d(1,11) &= f((x_1, x_4, x_5), (y_1, y_4, y_5)) \\
d(1,12) &= d(1,11) \\
d(2,13) &= f((x_2, x_3, x_4), (y_2, y_3, y_4)) \\
d(2,14) &= d(2,12) \\
&\vdots
\end{aligned} \tag{3.2}$$

Figure 3.2 (a) shows the values for the arrival times at all the vertices. The arcs corresponding to the active constraints form a *forest* and such a forest is shown for this example in Figure 3.2 (b). Let us perform a simple calculation to determine the number of active constraints and "active" variables associated with the constraints. Assume that the net delays are calculated by means of the bounding-box net capacitance and interconnect resistance is neglected. In order to use this model, four additional variables are needed for each net to represent the extents of the bounding box. In addition $4 \times |n_i|$ constraints are needed for each net so that the net delays may be expressed as a function of the bounding box of each net.

Let $x_{max}^{n_i}$, $x_{min}^{n_i}$, $y_{max}^{n_i}$ and $y_{min}^{n_i}$ be the extents of the bounding-box of n_i. As before, let $C_{n_i}^g$ denote the total gate capacitance of all the inputs connected to net n_i. We can define these four variables in terms of inequalities as follows:

$$x_{max}^{n_i} \geq x_j, \forall m_j \in n_i \tag{3.3}$$

$$x_{min}^{n_i} \leq x_j, \forall m_j \in n_i \tag{3.4}$$

$$y_{max}^{n_i} \geq y_j, \forall m_j \in n_i \tag{3.5}$$

$$y_{min}^{n_i} \leq y_j, \forall m_j \in n_i \tag{3.6}$$

The bounding box based delay equation for an external arc (v_i, v_j) can be written as:

$$d(v_i, v_j) = R_i[C_h(x_{max}^{n_i} - x_{min}^{n_i}) + C_v(y_{max}^{n_i} - y_{min}^{n_i})] + C_{n_i}^g[R_h(x_{max}^{n_i} - x_{min}^{n_i}) + R_v(y_{max}^{n_i} - y_{min}^{n_i})] \tag{3.7}$$

R_i represents the driving resistance of cell m_i (whose output net is n_i).

For each net n_i, at least four of the bounding-box constraints will be active, corresponding to the cells that actually define the extents of the net. (There may be more if more than one cell defines any particular extent of a net). Thus, the number of active constraints contributed by the net delay model is at least $4 \times N$ and the number of variables is also $4 \times N$. Since under the assumptions of Chapter 2, each cell has only one output net and the primary outputs do not have output nets, there are $M - g$ nets. Hence, the minimum number of active constraints can be written as $4 \times (M - g)$.

For each vertex, the worst-case actual arrival time of the signal at that vertex comes from one of its inputs and there will be one active timing constraint of the form

$$a_i = a_j + d(v_i, v_j)$$

The number of active constraints contributed by the arrival time constraints is at least as many as the number of nets which equals $M - g$. Each constraint has two variables a_i and a_j, hence there are $2 \times (M - g)$ active arrival time variables. In all, there are *at least* $5 \times (M - g)$ active constraints and $6 \times (M - g)$ active variables.

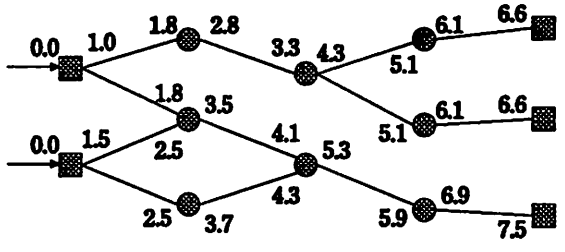
For the simple example with 13 vertices, the total number of active constraints is:

$$5 \times (M - g) = 5 \times (13 - 3) = 50$$

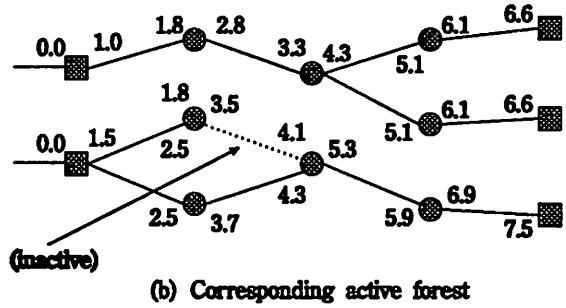
and the number of active variables is:

$$6 \times (M - g) = 6 \times (13 - 3) = 60$$

For a small sized circuit with 1000 internal cells, the number of active constraints are at least 5000 and the number of active variables are at least 6000!. By itself, the size of the problem vexes conventional techniques, but coupled with the degeneracy in the constraints, the problem becomes the bane of standard non-linear programming packages. Before we understand degeneracy and why it occurs, some preliminary notions need to be introduced.



(a) Vertex arrival times



(b) Corresponding active forest

Figure 3.2: Arrival time calculations and active forest derived therefrom

3.1.3 Reducing the Size of the Problem

It is assumed in the following discussion that the wirelength models and the timing models used are one of those presented in Chapter 2. Under this assumptions, due to the convexity of the objective function and the constraints, the general formulation GP is a convex programming problem. Let $\mathbf{A}^*$ denote the vector function (possibly non-linear) of the active constraints at a global minimum $\mathbf{w}^*$ and $\nabla\mathbf{A}^*$ denote the associated Jacobian matrix. Since the programming problem is convex, there exist Lagrange multipliers (one per constraint) [Luenberger 84] $\boldsymbol{\lambda}$ satisfying

$$\begin{aligned} \nabla\mathbf{L}(\mathbf{w}^*) + \boldsymbol{\lambda}^T\nabla\mathbf{A}^* &= \mathbf{0} \\ \boldsymbol{\lambda}^T\nabla\mathbf{A}^* &= \mathbf{0} \\ \boldsymbol{\lambda} &\geq \mathbf{0} \end{aligned} \tag{3.8}$$

provided $\mathbf{w}^*$ is a regular point of the constraints, i.e., at $\mathbf{w}^*$ the matrix $\nabla\mathbf{A}^*$ has full rank. The above conditions are popularly known as the Kuhn-Tucker first-order optimality conditions.

The Lagrange multiplier associated with a constraint at a given point $\mathbf{w}$ has a useful mathematical interptretation. It represents the sensitivity of the cost function to that constraint at that point. If the Lagrange multiplier is zero, then the constraint is inactive and has no effect locally. A positive value indicates that the constraint is binding and moving away from it towards the interior of the feasible region will increase the objective value. A negative Lagrange multiplier indicates that moving away from the constraint towards the interior of the feasible region will result in a decrease in the objective value and hence that constraint can be "dropped" provided all the other binding constraints are retained. An excellent treatment of Lagrange multipliers and their interpretation may be found in [Murty 83].

Proposition 3.1.1 *The active timing constraints and delay equations at the optimal solution* $\mathbf{w}^*$ *can be replaced by an equivalent set of active constraints which consists of equations for arcs on paths*[1] *in* D_T *from vertices in* S *to* E.

Proof. Let $E_c = \{v_i \in E | a_i = T_i\}$. In this discussion, consider external arcs. Internal arcs have constant delays and these delay values can be taken to the the right-hand side of

[1] the reader is referred to definition 2.4.2 for the definition of a path in this context

the extern arc equations. (In the example of Equation 3.2 the delays corresponding to the internal arcs are represented by the constants d_{i_j}.) If $E_c \neq \emptyset$ there exists a forest of arcs τ_A such that

$$a_j = a_i + d(v_i, v_j), \ \forall (v_i, v_j) \in \tau_A$$

For each arc $(v_i, v_j) \in \tau_A$, if $v_j \notin E_c$ and v_j has no arcs in τ_A directed out of it or $v_j \in E_c$ and $a_j < T_j$, we can increase a_j by a finite positive value ϵ without affecting the feasibility of the current solution. The solution remains optimal because a_j does not appear in the objective function. Thus, the equation corresponding to (v_i, v_j) can be deleted from the active set. If v_i has no other arcs in τ_A directed out of it, or all the arcs directed out of v_i were deleted by the above process, we can delete the delay equation for the output net associated with vertex v_j.

Similarly, if there is an arc $(v_i, v_j), v_i \notin S$ with no arcs in τ_A directed into v_i, we can decrease a_i by a finite positive constant ϵ and make the arc equations inactive without affecting the feasibility or optimality. Similar arguments can be made for the delay equations of the output net of a movable module with no arcs in τ_A directed into it. This process of deleting timing and delay equations can be repeated until the resulting forest τ'_A is rooted in S and all the arcs terminate in E_c. If E_c is empty, then all the constraints are removed by the process and the solution corresponds to the unconstrained optimal solution of the objective function. □

The forest resulting from this reduction is called a *reduced active forest* (RAF). The significance of this proposition is that in searching for an optimal solution, it suffices to look for solutions such that the set of active arcs form trees rooted in S and terminating in E. What benefits does this offer?

1. A vast reduction in the number of active variables and constraints is one effect. Consider the same small example presented earlier containing 1000 modules. In such an example, a typical number of cells on critical paths would be 10% or about 100. The number of active variables and constraints for this subset are about 400 each. However this saving alone is not enough to bring about a dramatic improvement in the time to solve the problem. To understand that, we need to delve into the solution process.

2. The numerical stability of the solution procedure is vastly improved. The reason is that degeneracy is avoided by restricting attention to only those constraints that are essential.

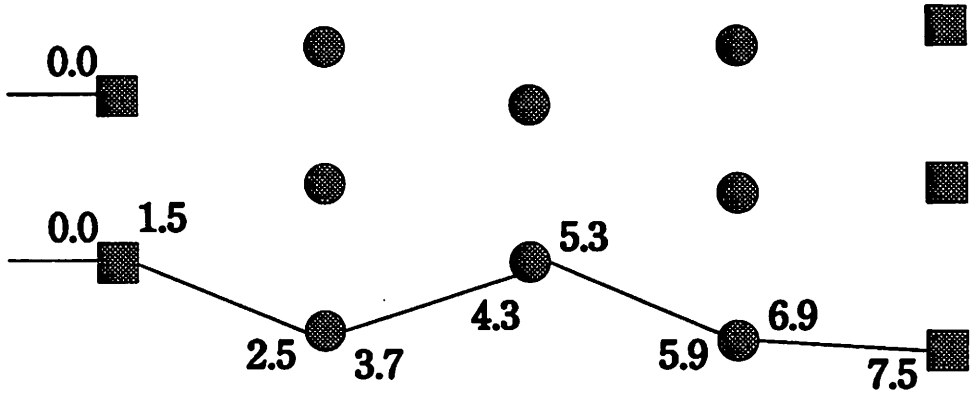


Figure 3.3: Reduced active forest for the simple example

3. A systematic and general solution technique can be developed for solving the problem, based on reduced active forest manipulation. The reduced active forest for the simple example is shown in Figure 3.3.

3.1.4 A General Algorithm

The generic primal active set algorithm is very simple and proceeds as follows:

1. Start with an arbitrary initial feasible solution and a current reduced active set τ_A. Let the current cell location vector be w.
2. Solve the equality-constrained non-linear programming problem corresponding to the current active forest. The solution yields a "step" or a direction from the current location to the new location obtained by the solution. Let this step be δw.
3. Move along δw until a new reduced forest becomes active.
4. Update the current RAF.
5. Check the current solution for optimality (using Equation 3.8. The Lagrange multipliers are obtained as a by-product of the solution process.) If it is optimal, stop.
6. Go to step 2.

Step 2, which involves solving an equality-constrained non-linear programming problem is as easy as unconstrained optimization for the wirelength and delay models presented in this work. It can be implemented very efficiently [Gill 89]. The key steps in the procedure are Step 3 and Step 4, which if performed efficiently, will make the entire algorithm practical. The remainder of this chapter is devoted to a discussion of how these steps can be implemented efficiently and the specific case of the quadratic wirelength model and bounding-box capacitance model is used to illustrate the ideas.

3.1.5 Degeneracy and How the Algorithm Avoids it

Definition 3.1.2 *The unique arcs of a path are those arcs that are not part of any other active critical path.*

Definition 3.1.3 *The unique arcs of a set of paths path τ with respect to a set of paths τ' are those arcs of τ that are not part of τ'.*

Definition 3.1.4 *The equations for an external arc (v_i, v_j) are defined as the following:*

$$\begin{aligned} a_j &= a_i + d(v_i, v_j) \\ d(v_i, v_j) &= f(X_i, Y_i) \end{aligned} \tag{3.9}$$

where $f(X_i, Y_i)$ is one of the interconnect delay estimates described in Chapter 2.

Definition 3.1.5 *A solution of GP associated with an RAF τ_A is the solution obtained by solving the equality constrained problem with all the constraints in τ_A taken as equality and all other constraints ignored.*

Proposition 3.1.2 *When a path becomes active, the equations for the unique arcs of the path can be added to the active set simultaneously, i.e., the step vector need not be computed after adding each unique arc on the path.*

Proof. Let the path be Ω. Let there be q unique arcs in the path. Suppose we have added timing and bounding-box constraints for $t < q$ arcs to the current active forest τ_A, obtaining a new active forest τ'_A. Then, at least one of the following are present in τ'_A: (1) an arc $(v_i, v_j) \in \Omega$ such that $v_i \notin S$ and v_i has no arcs in τ'_A directed into it, (2) an arc $(v'_i, v'_j) \in \Omega$ such that such that $v'_j \notin E$ and v'_j has no arcs in τ'_A directed outward from it. Informally, when we have activated $t < q$ unique constraints, Ω is disconnected because of

the uniqueness of the q arcs. By Proposition 3.1.1, the forest τ'_A can be reduced to τ_A, so the optimal solution for τ_A is also optimal for τ'_A. □

Corollary 3.1.1 *Let τ_A represent a reduced active forest and τ'_A represent an active forest that differs from τ_A in exactly one path, i.e., $\tau'_A - \tau_A$ is a single path from S to E. Let Ω_A be the unique arcs of τ'_A with respect to τ_A. Let ω_A be any proper subset of Ω_A. Then all the solutions of GP associated with the different active forests $\tau_A \cup \omega_A$ have the same objective function value.*

Proof. For any $\omega_A \subset \Omega_A$, $\tau_A \cup \omega_A$ can be reduced to τ_A and hence by Proposition 3.1.1 has the same objective function value as τ_A. □

Corollary 3.1.2 *The maximum number of paths in any reduced active forest is bounded by the number of arcs in the timing graph D_T.*

The significance of Proposition 3.1.2 is that when we encounter a new RAF during the algorithm, we need to add only a limited number of variables and constraints to the active set – those belonging to the unique arcs and more importantly, the addition itself can be done in one step. This method differs considerably from conventional mathematical programming techniques which, when moving from one RAF to the next, would add the variables for the arcs one at a time, in a random order, without making any progress (i.e., without improvement in the cost function) until *all* the unique arcs are added (by corollary 3.1.1). Until all of them are added the Lagrange multipliers for the unique arcs are zero (or very close to zero in practice due to numerical errors). It is possible for a conventional method to start *cycling*, i.e., to get stuck in an endless loop when there are numerical errors in Lagrange multiplier values. The above problems are often collectively (and informally) referred to as the *degeneracy* problem. Each degenerate addition step in a conventional method involves expensive matrix refactorizations and solving an equality-constrained problem which leads to inefficiency. On the other hand, Proposition 3.1.2 avoids these degenerate steps and obtains the next useful intermediate solution in one fell swoop. Corollary 3.1.2 is extremely powerful and is exploited fully in Chapter 5.

Informally, degeneracy in this context means that there are many active forests with the same objective value. The total number of degenerate active forests could be an

exponential function of the number of arcs in the timing graph. Conventional algorithms are unable to detect this problem.

3.1.6 Speedup Obtained

To get an idea of the magnitude of the speedup, a rough calculation may be illustative. Suppose a conventional algorithm encounters k active forests during the course of finding an optimum. Let the average number of arcs in a critical path be t. Let the fraction of all the modules that lie on critical paths be β. Assuming that about 30% of the arcs in a critical path are unique arcs, the conventional solution technique would take at least $O(tk)$ iterations, performing $O(M^2)$ work per iteration for the factorizations. Hence, the work done would be roughly $O(M^2tk)$ (one factorization per arc added). On the other hand, the modified algorithm would take $O(k)$ iterations performing about $O(\beta^2 M^2)$ work per iteration (this is a highly pessimistic estimate). The work done would then be about $O(\beta^2 M^2 k)$. For the small example with 1000 modules and with β approximately equal to 0.1 and with about 15 levels of logic ($t = 15$) the speedup would be $O(\frac{1}{\beta^2 0.3t})$ or about 450. Such a speedup was observed in practice. For larger examples, the speedup may be even greater.

3.1.7 Comment on the Generality of the Algorithm

Thus far, the algorithm discussed is completely general and may be applied to any combination of the wirelength and delay models presented in Chapter 2. The requirements are mild – convexity in the cost function and delay model. An interesting result is that the estimated speedup is independent of the models used.

3.2 Applying the Ideas

In order to illustrate the ideas more effectively, let us consider how the generic algorithm may be applied to a specific case. The quadratic wirelength model and bounding-box capacitance model are used as examples. In the delay model the distributed RC delay (the second-order term) of the interconnect wire is neglected.

3.2.1 Quadratic Cost Function

The modules can be partitioned into two sets, *fixed* and *movable*. Fixed modules are input/output pads or modules that have been assigned a location on the chip, for example, clock pads. Movable modules have variable x and y coordinates. The quadratic cost function of Equation 2.1 can be rewritten as

$$L = \frac{1}{2} \sum_{m_i, m_j, i \neq j} c_{ij}((x_i - x_j)^2 + (y_i - y_j)^2) \tag{3.10}$$

where c_{ij} represents the number of nets that modules m_i and m_j share. (x_i, y_i) and (x_j, y_j) represent the locations of m_i and m_j. Using matrix notation to compactly represent L, we get:

$$L(\mathbf{x}, \mathbf{y}) = \frac{1}{2}(\mathbf{x}^T\mathbf{B}\mathbf{x} + \mathbf{y}^T\mathbf{B}\mathbf{y}) + \mathbf{c}^T\mathbf{x} + \mathbf{d}^T\mathbf{y} \tag{3.11}$$

where $\mathbf{x}$ is a vector of the x-coordinates of the module locations and $\mathbf{y}$ is a vector of the y-coordinates. $\mathbf{c}$ and $\mathbf{d}$ are constant vectors arising from the fixed modules. $\mathbf{B}$ is a symmetric matrix with

$$\mathbf{B} = \mathbf{D} - \mathbf{C} \tag{3.12}$$

where $\mathbf{C} = [c_{ij}]$ and $\mathbf{D}$ is a diagonal matrix with $d_{ii} = \sum_{j=1}^{n} c_{ij}$. Typically, $\mathbf{B}$ is a highly sparse matrix and can be stored very efficiently using sparse matrix data structures as in [Bunch 76]. In addition, it is shown in a following section that $\mathbf{B}$ is positive-definite.

3.2.2 Bounding-Box Net Delay Model

Let $x_{max}^{n_i}$, $x_{min}^{n_i}$, $y_{max}^{n_i}$ and $y_{min}^{n_i}$ be the extents of the bounding-box of n_i. We can define these four variables in terms of inequalities as follows:

$$x_{max}^{n_i} \geq x_j, \forall m_j \in n_i \tag{3.13}$$

$$x_{min}^{n_i} \leq x_j, \forall m_j \in n_i \tag{3.14}$$

$$y_{max}^{n_i} \geq y_j, \forall m_j \in n_i \tag{3.15}$$

$$y_{min}^{n_i} \leq y_j, \forall m_j \in n_i \tag{3.16}$$

The bounding box based delay equation for an external arc (v_i, v_j) can be written as:

$$d(v_i, v_j) = R_i[C_h(x_{max}^{n_i} - x_{min}^{n_i}) + C_v(y_{max}^{n_i} - y_{min}^{n_i})] + C_{n_i}^g[R_h(x_{max}^{n_i} - x_{min}^{n_i}) + R_v(y_{max}^{n_i} - y_{min}^{n_i})] \tag{3.17}$$

where as before R_i represents the driving resistance of cell m_i (whose output net is n_i) and $C^g_{n_i}$ is the total capacitive load due to cell inputs connected to net n_i.

3.2.3 Specific Problem Formulation

Since variables representing bounds on each net and arrival times at the vertices on the timing graph have been introduced, the total number of variables in the problem is $2M + 4N + P$. However, the $(4N + P)$ arrival time and bounding-box variables do not enter into the cost function, so the value of the cost function at any point is unchanged and the sparsity of the matrix representing the cost function is retained. To simplify further discussion some notation is introduced. Let

$$\mathbf{w}_{cell} = \begin{bmatrix} \mathbf{x} \\ \mathbf{y} \end{bmatrix}$$

be the combined vector of x and y coordinates of cell positions. Let $\mathbf{w}_{net}$ denote the vector of net bounding-box variables and $\mathbf{w}_{pin}$ the vertex arrival time variables. Then

$$\mathbf{w} = \begin{bmatrix} \mathbf{w}_{cell} \\ \mathbf{w}_{net} \\ \mathbf{w}_{pin} \end{bmatrix}$$

is the $2M + 4N + P$ vector of all variables. Let

$$\mathbf{Q} = \begin{bmatrix} \mathbf{B} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{B} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \end{bmatrix}$$

be the combined $(2M + 4N + P) \times (2M + 4N + P)$ matrix for the cost function and let

$$\mathbf{b} = \begin{bmatrix} \mathbf{c} \\ \mathbf{d} \\ \mathbf{0} \\ \mathbf{0} \end{bmatrix}$$

Then, the cost function can be rewritten as

$$L = \frac{1}{2}\mathbf{w}^T\mathbf{Q}\mathbf{w} + \mathbf{b}^T\mathbf{w} \tag{3.18}$$

The problem of minimizing wirelength subject to timing constraints can now be stated as the following nonlinear program

$$
\begin{array}{lllll}
\text{minimize} & \mathrm{L} & & & \text{(NLP)} \\
\text{subject to} & x_{max}^{n_i} & \geq & x_j & \forall m_j \in n_i,\ \forall n_i \in \mathcal{N} \\
& x_{min}^{n_i} & \leq & x_j & \forall m_j \in n_i,\ \forall n_i \in \mathcal{N} \\
& y_{max}^{n_i} & \geq & y_j & \forall m_j \in n_i,\ \forall n_i \in \mathcal{N} \\
& y_{min}^{n_i} & \leq & y_j & \forall m_j \in n_i,\ \forall n_i \in \mathcal{N} \\
& a_j & \geq & a_i + d(v_i, v_j) & \forall (v_i, v_j) \in A \\
& a_j & \leq & T_j & \forall v_j \in E \\
& a_j & \geq & T_j & \forall v_j \in S
\end{array}
\tag{3.19}
$$

Theorem 3.2.1 *For all* $x \neq \alpha\mathbf{e}$ *where* $\mathbf{e} = (1, 1, \ldots, 1)$, $\mathbf{x}^T\mathbf{B}\mathbf{x} \geq 0$. *Similarly, for all* $y \neq \alpha\mathbf{e}$ *where* $\mathbf{e} = (1, 1, \ldots, 1)$, $\mathbf{y}^T\mathbf{B}\mathbf{y} \geq 0$.

Proof. The proof proceeds by assuming that there is a vector $\mathbf{x} \neq \alpha\mathbf{e}$ such that $\mathbf{x}^T\mathbf{B}\mathbf{x} = 0$ and showing that this assumption leads to a contradiction. (Note that since the objective function consists of the sum of squared terms, it can never be negative). Therefore, for some integers p and q the components of the vector $\mathbf{x}$ must satisfy $x_p \neq x_q$. Define $C(i) = \{m_j | c_{ij} \neq 0\}$. Consider a term in the objective function of the form $c_{ij}(x_i - x_j)^2, j \in C(i)$. This expression can be zero only if $x_i = x_j$. But this forces $x_j = x_i, \forall\, j \in C(i)$. Since the modules form a connected graph (by assumption), this in turn forces all modules connected to $C(i)$ to have the same x location as x_i. Proceeding in this manner, we have $x_1 = x_2 = \ldots = x_M$, which is a contradiction. That $\mathbf{y}^T\mathbf{B}\mathbf{y} > 0$ can be proved similarly. □

Corollary 3.2.1 *If there exist at least two fixed modules at distinct x and y locations and the modules do not form disconnected subsets, then* $\mathbf{x}^T\mathbf{B}\mathbf{x}$ *and* $\mathbf{y}^T\mathbf{B}\mathbf{y}$ *are strictly positive for all non-zero vectors* $\mathbf{x}$ *and* $\mathbf{y}$, *i.e., the matrix* $\mathbf{B}$ *is positive definite.*

Corollary 3.2.2 *Any relative minimum of L is also a global minimum.*

Proof. See [Luenberger 84, p459,p216,p180] □

Corollary 3.2.3 *The satisfaction of the Kuhn-Tucker first-order necessary conditions are both necessary and sufficient for a point to be a global minimizer of L.*

3.2.4 The Specific Algorithm

This section describes the algorithm used to solve the optimization problem. The algorithm follows a primal active set method, but departs from conventional techniques in the representation of constraints and activation of constraints.

Definition 3.2.1 *An active critical path is a path such that all of its vertices have zero slack.*

The algorithm proceeds as a sequence of major iterations. At major iteration k, a feasible point $\mathbf{w}^{(k)}$ is known, which satisifies the active constraints with equality. $\mathcal{A}$ denotes the set indices of constraints that are active. Let $\mathbf{A}^{(k)}$ denote the matrix of constraints that are active during iteration k, i.e., $\mathbf{A}^{(k)}$ denotes those constraints that are satisfied with equality. This includes both path constraints and bounding-box constraints for nets that lie on active critical paths. For linear constraints, $\nabla \mathbf{A} = \mathbf{A}$, so the Kuhn-Tucker conditions reduce to:

$$\begin{aligned} \nabla \mathbf{L}(\mathbf{w}^*) + \boldsymbol{\lambda}^T \mathbf{A}^* &= \mathbf{0} \\ \boldsymbol{\lambda}^T \mathbf{A}^* &= \mathbf{0} \\ \boldsymbol{\lambda} &\geq \mathbf{0} \end{aligned} \tag{3.20}$$

Let $\Gamma^{(k)} = \{\Pi_1, \ldots, \Pi_\ell\}$ be the set of active critical paths during iteration k. Let $\mathcal{N}_\Gamma \subseteq \mathcal{N}$ denote the set of nets that form the critical paths. Let $A_\Gamma \subseteq A$ denote the set of arcs in D_T that lie on the active critical paths. Let E_Γ and S_Γ denote the starting and ending points of the active critical paths. Each major iteration attempts to locate the solution to an equality constrained problem formed by deleting the inactive constraints from NLP. This is done by shifting the origin to $\mathbf{w}^{(k)}$ and looking for a "step vector" $\boldsymbol{\delta}$ that solves the following problem.

$$
\begin{array}{lrcll}
\text{minimize} & \multicolumn{3}{l}{\frac{1}{2}\delta^T \mathbf{Q}\delta + \delta^T \mathbf{g}^{(k)}} & \text{(EP)} \\
 & x^{n_i}_{max} & = & \{x_j | x_j = \max_k\{x_k,\ \forall m_k \in n_i\}\}, & \forall n_i \in \mathcal{N}_\Gamma \\
 & x^{n_i}_{min} & = & \{x_j | x_j = \min_k\{x_k,\ \forall m_k \in n_i\}\}, & \forall n_i \in \mathcal{N}_\Gamma \\
 & y^{n_i}_{max} & = & \{y_j | y_j = \max_k\{y_k,\ \forall m_k \in n_i\}\}, & \forall n_i \in \mathcal{N}_\Gamma \\
\text{subject to} & y^{n_i}_{min} & = & \{y_j | y_j = \min_k\{y_k,\ \forall m_k \in n_i\}\}, & \forall n_i \in \mathcal{N}_\Gamma \\
 & a_j & = & a_i + d(v_i, v_j) & \forall (v_i, v_j) \in A_\Gamma \\
 & a_j & = & T_e & \forall v_j \in E_\Gamma \\
 & a_j & = & T_s & \forall v_j \in S_\Gamma
\end{array}
\tag{3.21}
$$

$\mathbf{g}^{(k)}$ is the gradient vector at the current point, defined as

$$\mathbf{g}^{(k)} = \nabla \mathbf{L}(\mathbf{w}^{(k)}) = \mathbf{Q}\mathbf{w}^{(k)} + \mathbf{b} \tag{3.22}$$

The solution to the above problem (EP) can be found by solving the Kuhn-Tucker conditions for EP, which form a linear system of equations, corresponding to the Hessian matrix of the augmented Lagrangian.

$$\begin{bmatrix} \mathbf{Q} & \mathbf{A}^T \\ \mathbf{A} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \delta^{(k)} \\ \lambda \end{bmatrix} = \begin{bmatrix} \mathbf{g}^{(k)} \\ \mathbf{0} \end{bmatrix} \tag{3.23}$$

The step vector $\delta^{(k)}$ consists of three parts: δ_{cell}, the step vector corresponding to cell positions, δ_{net} corresponding to the net bound variables and δ_{pin} corresponding to the arrival time variables. The vector λ is the vector of Lagrange multipliers for the constraints in $\mathbf{A}$.

If $\delta^{(k)}$ is feasible with respect to the constraints not in $\mathbf{A}^{(k)}$, i.e., a timing verification on the graph yields a feasible timing graph, then the step is accepted and $\mathbf{w}^{(k+1)} = \mathbf{w}^{(k)} + \delta^{(k)}$. If not, then a line search is made in the direction of $\delta^{(k)}$to find the best feasible point. The line search procedures are explained in detail in following sections. At this point, it suffices to note that these procedures return a step length parameter $\alpha^{(k)}$ such that

$$\mathbf{w}^{(k+1)} = \mathbf{w}^{(k)} + \alpha^{(k)}\delta^{(k)} \tag{3.24}$$

minimizes L along the direction $\delta^{(k)}$ and $\mathbf{w}^{(k+1)}$ is feasible with respect to all the constraints. If $\alpha^{(k)} < 1.0$, a new constraint becomes active and this is added to the current active set $\mathcal{A}$.

If $\delta^{(k)} = \mathbf{0}$, and $\lambda_i^{(k)} \geq 0$, $i = 1, \ldots, |\mathcal{A}|$, then by Theorem 1, we are at the optimal solution. If

$$\lambda_q^{(k)} < 0 \tag{3.25}$$

for some constraint $q \in \mathcal{A}$, it is possible to drop constraint q from $\mathcal{A}$ according to the interpretation of Lagrange multipliers discussed in Section 3.1.3. After removing constraint q, the algorithm continues as before. If more than one constraint satisfies Equation 3.25, then select

$$q = \arg \min_i \lambda_i^{(k)} \tag{3.26}$$

The algorithm starts with a feasible point $\mathbf{w}^{(1)}$ and an initial active set of constraints whose matrix is $\mathbf{A}^{(1)}$. (Finding a feasible point is discussed in Section 3.2.9) The flow of the algorithm may be summarized as follows:

Algorithm

1. Given $\mathbf{w}^{(1)}$ and an active set $\mathcal{A}$, set the iteration index k to 1.
2. Solve (EP) for $\delta^{(k)}$.
3. If $\delta^{(k)} = \mathbf{0}$ and $\lambda_i^{(k)} \geq 0$, $\forall\, i \in \mathcal{A}$, stop. The optimal solution has been reached.
4. Find $\alpha^{(k)}$, a step length parameter and set $\mathbf{w}^{(k+1)} = \mathbf{w}^{(k)} + \alpha^{(k)}\delta^{(k)}$
5. If $\alpha^{(k)} < 1$ add some constraint(s) to $\mathcal{A}$ according to Section 3.2.7
6. If $\lambda_q^{(k)} < 0$ for some q, delete a constraint from $\mathcal{A}$
7. Set $k = k + 1$ and go to step 2

Proposition 3.2.1 *If the conditions of Theorem 3.2.1 are satisfied and at each non-terminal step $\alpha^{(k)} \neq 0$, the algorithm given above terminates in a finite number of steps at the optimal solution $\mathbf{w}^*$.*

Proof.

The proof of this proposition follows the proof of the active set theorem in [Luenberger 84]. After the solution corresponding to an active set is found, since the step length is positive, the step results in a strict decrease in the objective function. Thus, once the algorithm leaves an active set, it never returns to it. There are only a finite number of *reduced active*

forests. Associated with each RAF, there are only a finite number of active sets corresponding to different active bounding-box constraints. □

3.2.5 Line Search Procedure (1): Bounding-Box Constraints

Given δ, and the current active set of bounding box constraints, this procedure computes α_1, the maximum step length such that the cells defining the bounding-box for a net on an active critical path change. Let N_Γ denote the set of nets that lie on some active critical path. Let δx_j denote the computed step in the x direction for cell j and δy_j the step in the y direction. Let δx^n_{max} denote the step for the maximum x bound for net n and δx^n_{min} the step for the minimum x bound. δy^n_{max} and δy^n_{min} are similarly defined for the y direction.

For each net $n \in N_\Gamma$, the procedure involves computing the parameters

$$\alpha^X = \min_{m_j \in n} \left| \frac{x^n_{max} - x_j}{\delta x_j - \delta x^n_{max}} \right| \tag{3.27}$$

$$\alpha^x = \min_{m_j \in n} \left| \frac{x^n_{min} - x_j}{\delta x_j - \delta x^n_{min}} \right| \tag{3.28}$$

$$\alpha^Y = \min_{m_j \in n} \left| \frac{y^n_{max} - y_j}{\delta y_j - \delta y^n_{max}} \right| \tag{3.29}$$

$$\alpha^y = \min_{m_j \in n} \left| \frac{y^n_{min} - y_j}{\delta y_j - \delta y^n_{min}} \right| \tag{3.30}$$

$$\alpha_1 = \min_{n \in N_\Gamma} \{\min(1, \alpha^X, \alpha^x, \alpha^Y, \alpha^y)\} \tag{3.31}$$

3.2.6 Line Search Procedure (2): Timing Constraints

Given δ and α_1, this procedure computes the maximum step length $\alpha^{(k)}$ such that the chip's timing remains feasible. $\alpha^{(k)}$ is determined by performing a bisection timing verification (BTV). The bisection timing verification is a combination of bisection line search and timing verification and departs from conventional techniques in that the graph representation of the timing constraints is used to compute a maximum feasible step. The

procedure is very efficient and its run time depends linearly on the size of the timing graph (number of edges and cells). In the procedure, ϵ is a small positive constant.

1. $\alpha = \alpha_1$
2. Update the cell positions based on α
3. Update the net bounding-box positions
4. Update the delays for all external arcs
5. Calculate the minimum slack in D_T
6. If the minimum slack ≤ 0 (the step length is too long)
   (a) $\sigma = \alpha \ / \ 2$
   (b) $\alpha = \alpha - \sigma$
   (c) While the absolute value of the minimum slack in $D_T > 0$, do
      i. Update the cell positions based on α
      ii. Update the net bounding-box positions
      iii. Update the delays for all external arcs
      iv. Calculate the minimum slack in D_T
      v. $\sigma = \sigma \ / \ 2$
      vi. If the minimum slack $> \epsilon$ then set $\alpha = \alpha + \sigma$
      vii. If the minimum slack $< -\epsilon$ the set $\alpha = \alpha - \sigma$
7. STOP

The bisection search can be performed to any degree of accuracy. For any fixed degree of accuracy, the *while* loop is performed a constant number of times and each iteration involves a breadth first traversal of the timing graph which is $O(M)$. Hence, the work done in this procedure is $O(M)$. Since the error decreases at a quadratic rate, very few iterations are required in practice.

3.2.7 Activating a constraint

There are several conditions under which new constraints are added to the active set. These may be listed as follows:

1. $\alpha^{(k)} = 1$. No new constraint is added since the full step is feasible. The algorithm proceeds to the next major iteration.

2. $\alpha^{(k)} = \alpha_1 < 1$. In this case, a new bounding box constraint becomes active. Without loss of generality, assume that $\alpha_1 = \left| \frac{x^n_{max} - x_j}{\delta x_j - \delta x^n_{max}} \right|$, for some net n. The constraint that becomes active is $x^n_{max} \geq x_j$. This constraint is added as an equality to $\mathcal{A}$. If more than one net constraint gives $\alpha_1 = \left| \frac{x^n_{max} - x_j}{\delta x_j - \delta x^n_{max}} \right|$, all of them are added to the active set.

3. $\alpha^{(k)} < \alpha_1$. In this case, a new critical path becomes active. Several constraints corresponding to timing constraints for unique arcs of the critical path and bounding-box equations for each unique arc become active. By Proposition 3.1.2, we can add all the equations simultaneously to the active set.

   Let $\Omega = (v_s, \ldots, v_e)$ denote the new critical path. Let $\mathcal{N}_\Omega \subseteq \mathcal{N}$ denote the set of nets that form the new critical path. Let $A_\Omega \subseteq A$ denote the set of unique arcs in D_T that lie on the new critical path. For the critical path Ω, the constraints that become active are

$$\begin{aligned}
x^{n_i}_{max} &= \{x_j | x_j = \max_k\{x_k,\ \forall m_k \in n_i\}\}, \quad \forall n_i \in \mathcal{N}_\Omega \\
x^{n_i}_{min} &= \{x_j | x_j = \min_k\{x_k,\ \forall m_k \in n_i\}\}, \quad \forall n_i \in \mathcal{N}_\Omega \\
y^{n_i}_{max} &= \{y_j | y_j = \max_k\{y_k,\ \forall m_k \in n_i\}\}, \quad \forall n_i \in \mathcal{N}_\Omega \\
y^{n_i}_{min} &= \{y_j | y_j = \min_k\{y_k,\ \forall m_k \in n_i\}\}, \quad \forall n_i \in \mathcal{N}_\Omega \\
a_j &= a_i + d(v_i, v_j) \qquad \forall (v_i, v_j) \in A_\Omega \\
a_e &= T_e \qquad v_e \in E_\Omega \\
a_s &= T_s \qquad v_s \in S_\Omega
\end{aligned} \tag{3.32}$$

3.2.8 Deleting a constraint

The strategy used to delete a constraint is to select the constraint with the most negative Lagrange multiplier and remove it from the active set $\mathcal{A}$. This selection is not invariant to scaling of the variables in the problem, but since the Quadratic Program (QP)

for placement is naturally well scaled, the strategy works well. When deleting a bounding-box constraint from the active set, the algorithm follows the usual active set method. Since the algorithm adds only the unique arcs, when one timing constraint is deleted, following Proposition 3.1.2, equations for all the unique arcs associated with a path are deleted. Again, this results in an increase in efficiency over conventional active set methods and avoids degeneracy and the problems associated with it.

3.2.9 Finding an initial feasible point

Theoretically, the problem of finding a feasible solution for the timing constraints is as difficult as solving the optimization problem. However a practical heuristic method that works for almost all chips proved effective in finding an initial feasible solution. The idea is to place all the movable cells at the same location, say the center of the chip. For cell-based ICs, the fixed cells are usually input or output pads. Typically, the input pads have a very low resistance (high drive capability). Likewise the cells whose outputs are connected to output pads are also capable of driving large wire loads since they have only one fanout. In the event that this method fails, one may resort to a Simplex-like algorithm or a similar feasible point routine that operates on RAF's. Fortunately, the development of better techniques during the course of this work rendered finding an initial solution unnecessary and these techniques are the subject of following chapters.

3.2.10 Solving the Linear System Efficiently

Most of the work done at each iteration of the algorithm involves solving the system

$$\begin{bmatrix} \mathbf{Q} & \mathbf{A}^T \\ \mathbf{A} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \delta \\ \lambda \end{bmatrix} = \begin{bmatrix} \mathbf{g}^{(k)} \\ \mathbf{0} \end{bmatrix}$$

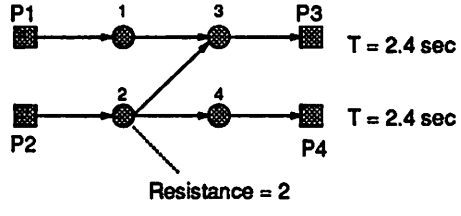
However, the standard efficient way of solving a positive definite QP is not to directly solve the above system. Typically, the problem is solved by projecting the Hessian and the gradient onto the nullspace of the current working set $\mathbf{A}^{(k)}$ [Gill 89]. Let $\mathbf{Z}^{(k)}$ denote a matrix whose columns form a basis for the set of vectors orthogonal to $\mathbf{A}^{(k)}$. The projected Hessian is $\mathbf{Z}^{(k)T}\mathbf{Q}\mathbf{Z}^{(k)}$ and the step at $\mathbf{w}^{(k)}$ in the reduced space ($\mathbf{w}_r^{(k)}$) is obtained by solving

$$\mathbf{Z}^{(k)T}\mathbf{Q}\mathbf{Z}^{(k)}\mathbf{w}_r^{(k+1)} = -\mathbf{Z}^{(k)T}g^{(k)} \tag{3.33}$$

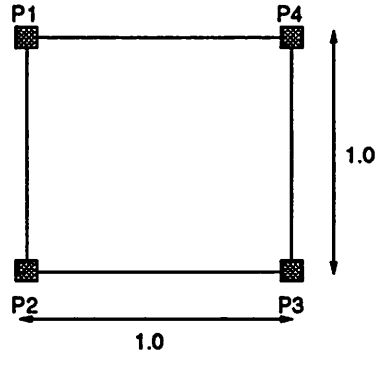
The structure of the positive definite quadratic objective function is exploited. Since the Hessian matrix is symmetric, usually a *Cholesky* factorization of the projected Hessian is maintained. Special techniques are applied to update the factorization with minimum effort when a constraint is added to the active set or deleted from it.

3.2.11 An Example

A simple example may help to clarify the key steps involved in the algorithm. Consider four cells connected as shown in Figure 3.4. All the cells except cell m_2 have



(a) Netlist



(b) IO pad location

Figure 3.4: Example to illustrate steps of Primal algorithm

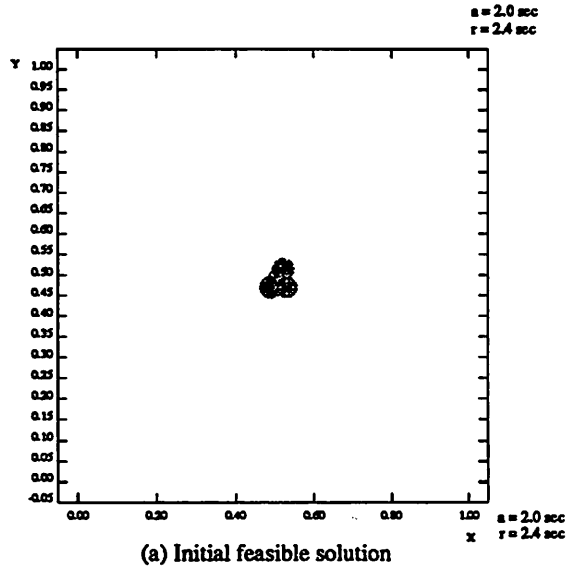
a driving resistance of 1 unit. Cell m_2 has a driving resistance of 2 units. The required time is 2.4 units for all primary outputs and there are no sequential cells in the circuit.

The initial feasible solution is shown in Figure 3.5(a). The initial RAF is the empty set. The algorithm proceeds by computing a step direction at this initial solution that takes it towards the optimal solution. However, before the entire step can be taken, a new RAF is encountered (shown in Figure 3.5(b)). The longest step that can be taken before the new RAF is encountered is detected by the bisection timing verification procedure described earlier. Next, the algorithm computes a step direction (Figure 3.5(c)) subject to the equality constraints associated with the current RAF. Before the entire step can be taken, the cell defining the maximum x extent (x^2_{max}) of the bounding box of net n_2 changes from m_4 to m_3. The constraint $x^2_{max} = x_3$ is added to the active set. In the figure, the step is displayed as a vector emerging from cells m_3 and m_4. Only the x component of the step is shown for simplicity. The algorithm steps upto the point where this change occurs (Figure 3.5(d)) and then computes a new direction. Once again, the step of Figure 3.5(d) is limited by the activation of a new bounding box constraint for net 2. The constraint $y^2_{max} = y_2$ is added to the active set. A new step is computed and the algorithm terminates at the optimal solution. Optimality is detected in Figure 3.6(e) because the Lagrange multipliers for all the active constraints are non-negative and the computed step has zero length.

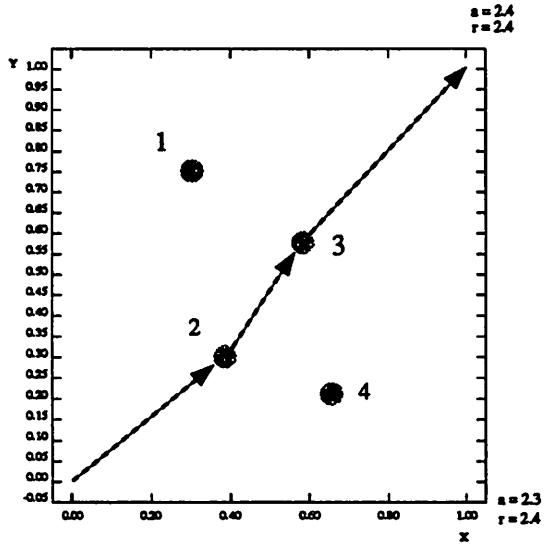
3.3 Using a More General Delay Model

So far the second order delay effects of interconnect, namely the distributed *RC* effects have been neglected. What happens to the solution technique when the second order effects are included?

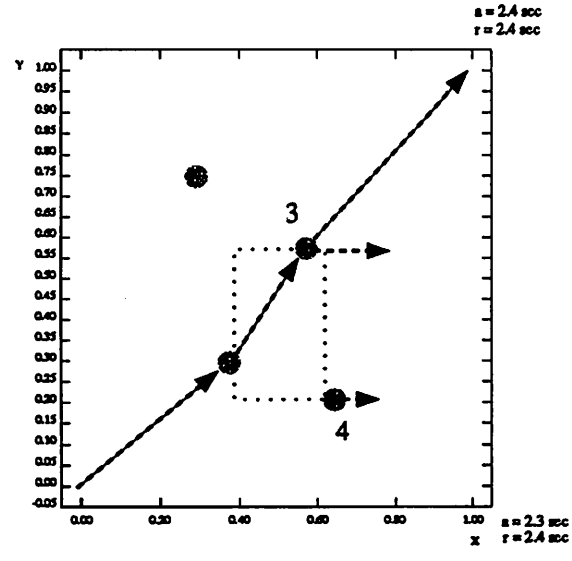
The general algorithm remains the same. However, at each iteration a quadratic function needs to be minimized, subject to second-order equality constraints. This problem can be converted to one of solving a nonlinear system of equations at each iteration. There are many commonly known and highly stable techniques for solving nonlinear systems composed of convex functions. One popularly used method that has been very successful for other large-scale circuit analysis methods such as SPICE [Nagel 75] is Newton's Method. The methods for determining the step length parameter α also need to be modified to deal with second order constraints and efficient procedures exist for such computations [Zangwill 69]. Other than increasing the computational effort, using a second order delay model has no significant impact on the general technique presented in this chapter.



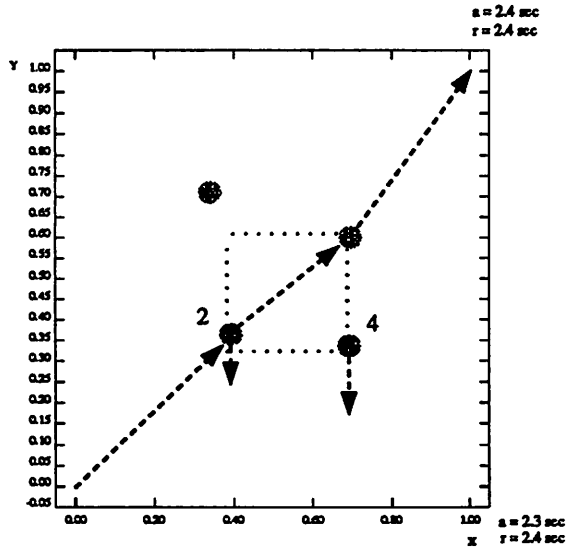
(a) Initial feasible solution



(b) Updated RAF

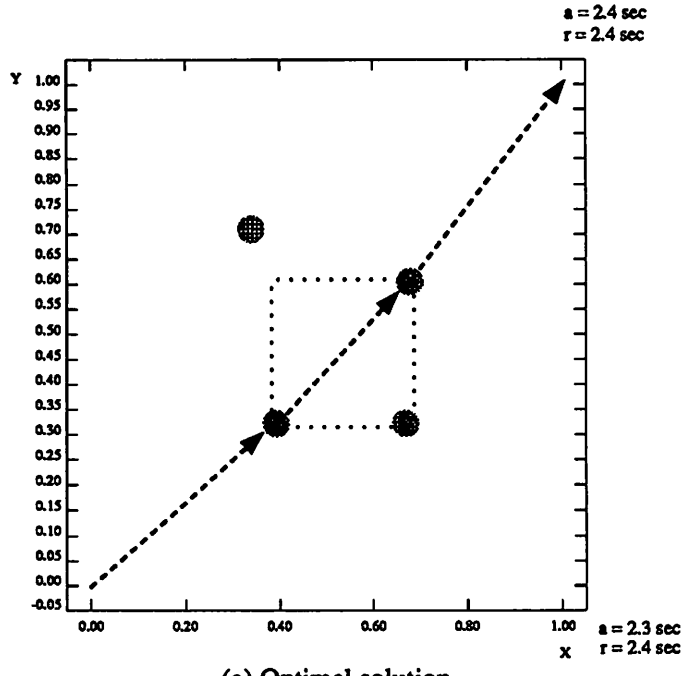


(c) Compute new step



(d) Compute new step

Figure 3.5: Steps of Primal algorithm on example



(e) Optimal solution

Figure 3.6: Steps of Primal algorithm on example

3.4 Simultaneous Clock Tree Placement

A simple example is used to illustrate how a clock distribution network may be placed simultaneously along with the other cells in the circuit. For the purposes of the

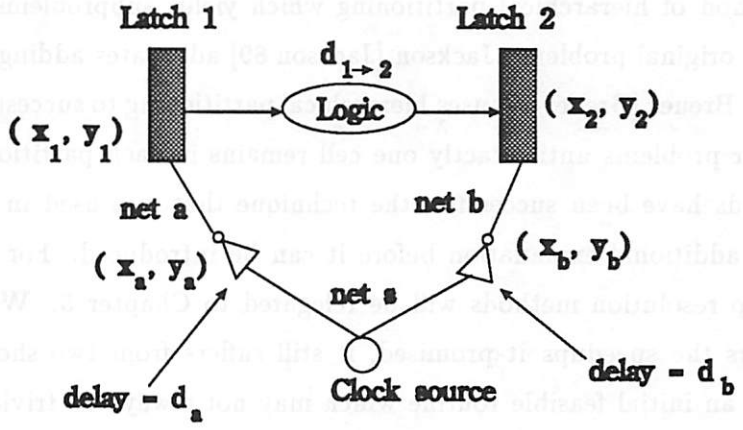


Figure 3.7: Clock tree placement example

objective function, the clock tree nets are treated like any other net. Consider the example shown in Figure 3.7. Let the cycle time be T for all latch-to-latch paths. In the figure, it is assumed that there is only one path from latch 1 to latch 2, with delay d_{1-2}. The intrinsic delays of the buffers are denoted by d_a and d_b. The equations that model the timing behavior of the paths in the figure can be written as:

$$\begin{aligned} T &\geq d_a + d(n_a) + d_{1-2} - d_b - d(n_b) \\ d(n_a) &= f(x_1, y_1, x_a, y_a) \\ d(n_b) &= f(x_2, y_2, x_b, y_b) \end{aligned} \tag{3.34}$$

The delay of net n_s does not appear in the expression because this delay is incurred along the path from the source to either latch and cancels out. It is the difference in the delay between the clock source to the two latches that is important. By writing equations in this manner, it is possible to model a general clock network and determine a placement for the elements of the tree.

3.5 Conclusions

In a large scale IC the cells need to be laid on a regular grid without any overlaps so that design rules are not violated. The algorithm described in this chapter will find a "global" solution, i.e., one in which cells may possibly overlap. There are many techniques that can be integrated with the primal algorithm. For example, Tsay [Tsay 89] uses a method of hierarchical partitioning which yields subproblems with the same structure as the original problem. Jackson [Jackson 89] advocates adding linear constraints successively. Breuer [Breuer 77] uses hierarchical partitioning to successively generate smaller and smaller problems until exactly one cell remains in each partition. While these and other methods have been successful, the technique that was used in this work is different and needs additional explanation before it can be introduced. For that reason, discussion of overlap resolution methods will be relegated to Chapter 5. While the Primal algorithm delivers the speedups it promised, it still suffers from two shortcomings. Firstly, it requires an initial feasible routine which may not always be trivial to find. Secondly, after experimentation I came to the conclusion that the speed up was not sufficient to be able to place extremely large circuits (larger than 2500 cells or about 10000 gates). The basis for this conclusion is explained in detail in Chapter 6. The problem lies in the memory requirements of the Primal active set method. As the algorithm proceeds, constraints are appended to the active set and each addition requires updating sparse matrix factors. The process of updating typically adds fillins to the matrix and some of the sparsity is lost. For large examples, it was observed that the matrix quickly became unwieldy and some of the advantages of the RAF based efficient algorithm were lost. This problem may be alleviated by reordering and refactorizing the matrix every few iterations. Unfortunately, that takes a significant amount of time for large circuits. The succeeding chapters describe how the techniques were substantially revised to avoid these problems.

Chapter 4

Refining the Cut

The Primal algorithm described in the previous chapter suffers from two shortcomings. To summarize the concluding section of that chapter:

1. It requires an initial feasible solution that may not be always available readily. Finding an initial feasible solution using conventional techniques like Phase-I of the Simplex method, a Big-M method [Murty 83] or even an interior point method [Karmarkar 84] takes a long time.

2. For large examples (2500 cells or more), the matrix manipulation techniques required to implement the Primal algorithm become a speed and memory bottleneck.

4.1 Some Observations

What led me to the algorithms described in this chapter? Experimentation with industrial circuits using the Primal algorithm raised some interesting questions. The timing-constrained solution was typically not drastically different (i.e., did not involve a drastic change in cell locations) compared to the unconstrained solution. Was it possible to use the unconstrained solution as a starting point?

Allied to this question was the fact that similar to the conventional Primal active set algorithm, mathematical programming researchers had successfully devised Dual active set methods that were proven to be more efficient. These algorithms started with the unconstrained solution and did not require a feasible point. Was it possible to extend the Dual active set methods to operate on RAF's and retain the efficiency and advantages of

RAF's? Would it be possible to prove convergence as easily as in the Primal algorithm?

4.2 The General Dual Algorithm

The work of Idnani on dual active set methods [Idnani 80] was an inspiration. A general outline of the dual active set method of Idnani is described briefly in this section. Following that, the extensions and modifications required for the algorithm to work on RAFs is presented.

Definition 4.2.1 *A point is said to be primal optimal with respect to the constraints of a constrained convex minimization problem if it satisfies the Kuhn-Tucker first order necessary conditions.*

Definition 4.2.2 *A point is said to be dual feasible for a constrained convex minimization problem if it is primal optimal with respect to any subset of the constraints of the problem.*

The dual active set method always maintains a "solved subproblem" which is *dual feasible* i.e., primal optimal with respect to a subset of the constraints. The active set for the solved problem is iteratively updated according to the following strategy.

1. Let the current solved subproblem be $SP = \emptyset$.
2. Select a violated constraint, if any, say p. If there is no violated constraint, terminate; the optimal solution has been found.
3. Let $VP = SP \cup \{p\}$.
4. Solve VP.
5. If VP has no solution, terminate because the feasible region is empty; othewise we have a new solved subproblem. Let $SP = VP$, go to step 2.

Let n^+ denote an added (violated) constraint. Let N denote the matrix of currently active constraints. The constraints are of the form:

$$\begin{aligned} Nx &\leq b \\ n^+x &\leq b^+ \end{aligned} \tag{4.1}$$

As shown in [Idnani 80], VP has no solution if the added constraint is linearly dependent on the constraints in the active set and the following condition is satisfied:

Condition 4.1: There exists an r such that $N^T r = n^{+^T}$ and $r \leq 0$.

Proof. Suppose z is a allowable step direction for the above constraints. It follows that z must satisfy the following condition.

$$\begin{aligned} Nz &\leq 0 \\ n^+ z &< 0 \end{aligned} \tag{4.2}$$

But $n^+ = r^T N$. Therefore,

$$n^+ z = r^T N z$$

But $r \leq 0$ and $Nz \leq 0$ imply that $n^+ z \geq 0$ which is a contradiction. Therefore, there is no allowable direction such that $Nz \leq 0$ and $n^+ z < 0$ and therefore, the subproblem is infeasible. □

The vector r is called the vector of *infeasibility multipliers.* By maintaining infeasibility multipliers as shown in [Goldfarb 83] Condition 4.1 can be checked very efficiently whenever a new constraint is added to the active set. Note that unlike the primal method, there is a choice of which constraint to add in the dual method.

4.2.1 Extending the Algorithm to RAF's

The general dual algorithm is very appealing for three reasons: (1) It uses the unconstrained solution as a starting point. For the performance-driven placement problem, it is likely to start out closer to the final solution than an arbitrary solution (like the Primal algorithm). (2) It does not require a feasible solution. (3) It can detect infeasibility very efficiently.

Fortunately, by modifying the general procedure, carefully selecting the added constraints and modifying the definition of a subproblem, it is possible to prove finite termination (without the degeneracy assumption) and solve the problem efficiently.

Definition 4.2.3 *A subproblem corresponding to a* **reduced forest** τ_A *(RFP_A) is defined as the subproblem of GP obtained by considering the path and arc delay constraints corresponding to τ_A and ignoring all the other constraints.*

Definition 4.2.4 *A solved subproblem corresponding to a* **reduced forest** τ_A *($RFSP_A$) is defined as the* **reduced active forest** τ_{SA} *and solution vector* $\overline{\mathbf{w}}$ *obtained by finding*

the optimal solution for the subproblem of GP subject to the path and arc delay constraints associated with τ_A, i.e., by solving RFP_A.

Definition 4.2.5 *In the following discussion, define a critical path Ω to be synonymous with the set of constraints consisting of timing constraints and arc delay equations for some path with nonpositive slack.*

The notation τ_A is used unambiguously to refer to both - a forest and the timing and delay equations for arcs in the forest. The modified dual active set algorithm proceeds as follows:

1. Solve the unconstrained problem corresponding to NLP. The current reduced forest is $\tau_A = \emptyset$.
2. Select a path Ω with negative slack, if any. If there is no such path, terminate; the optimal solution has been found.
3. Let $\tau'_A = \tau_A \cup \Omega$.
4. Solve the subproblem corresponding to τ'_A (RFP_A) to obtain τ_{SA}.
5. If RFP_A has no solution, terminate because the feasible region is empty; othewise we have a new solved subproblem. Let $\tau_A = \tau_{SA}$, go to step 2.

4.3 Specific Algorithm

In order to illustrate the details of the dual algorithm, the familiar case of the quadratic wirelength model and bounding-box based net delay model is considered. Let $\mathbf{A}^{(k)}$ denote the matrix of constraints that are active during iteration k, i.e., $\mathbf{A}^{(k)}$ denotes those constraints that are satisfied with equality. This includes both path constraints and bounding-box constraints for nets that lie on active critical paths. Let $\Gamma^{(k)} = \{\Pi_1, \ldots, \Pi_\ell\}$ be the set of active critical paths during iteration k, i.e., the *reduced active forest.* Let $\mathcal{N}_\Gamma \subseteq \mathcal{N}$ denote the set of nets that form the active critical paths. Let $A_\Gamma \subseteq A$ denote the set of arcs in D_T that lie on active critical paths. Let E_Γ and S_Γ denote the starting and ending points of active critical paths. Recall that the delay along an external arc is

expressed as a function of the bounding box of the net corresponding to that arc and the bounding box of a net n_i is represented by the following constraints:

$$x_{max}^{n_i} \geq x_j, \forall m_j \in n_i \tag{4.3}$$

$$x_{min}^{n_i} \leq x_j, \forall m_j \in n_i \tag{4.4}$$

$$y_{max}^{n_i} \geq y_j, \forall m_j \in n_i \tag{4.5}$$

$$y_{min}^{n_i} \leq y_j, \forall m_j \in n_i \tag{4.6}$$

Let $\boldsymbol{\lambda}^{(k)}$ denote the Lagrange multipliers at iteration k. Let $\delta\boldsymbol{\lambda}$ denote the computed step direction for the Lagrange multipliers.

The flow of the specific algorithm is first described and then the important steps are described in greater detail.

1. **Compute the unconstrained minimum.**

   $$\mathbf{w}^{(0)} = -\mathbf{Q}^{-1}\mathbf{b}$$

   $\mathbf{w}^{(0)}$ is a *dual feasible* point for NLP, i.e. it is primal optimal for the null set of constraints. Set the initial active set $\mathbf{A}^{(0)} = \emptyset$ and the initial reduced forest $\tau^{(0)} = \emptyset$. Set the initial Lagrange multipliers $\boldsymbol{\lambda}^{(0)} = 0$. Let the current subproblem $SP = \emptyset$. Let the iteration index $k = 1$.

2. **Determine an RFP.**

   Let s_Ω denote the slack of path Ω. Select a path Ω which satisfies:

   $$s_\Omega = \min_{v_i \in D_T} s_i$$

   If $\Omega = \emptyset$, terminate; the optimal solution has been found. Let $\tau^{(k)} = \tau^{(k-1)} \cup \Omega$.

3. **Solve the RFP.**

   Solve the subproblem associated with $\tau^{(k)}$ to obtain $RFSP_{\tau^{(k)}}$. Update $\mathbf{A}^{(k)}$, $\boldsymbol{\lambda}^{(k)}$ and $\mathbf{w}^{(k)}$. During the process of solving $RFP_{\tau^{(k)}}$, it is possible that some paths may leave $\tau^{(k)}$. There is considerable freedom in choosing a method to solve the subproblem. For example, one could use the primal algorithm of Chapter 3 or the general dual algorithm described in Section 4.2 could be used. The dual algorithm is preferred because it can detect infeasibility.

4. Let $k = k + 1$. Go to step 2.

Proposition 4.3.1 *If the dual method is used to solve the subproblem of step 3, the dual active set algorithm defined above will find the optimum point in a finite number of iterations or terminate when there is no feasible solution.*

Proof. Just before step 3 of the algorithm is executed, some subproblem of the original problem is solved because the algorithm always maintains dual feasibility. Every time that the step is executed, the objective value increases, since it corresponds to a RFP containing a violated critical path not in the previous RFP. Therefore, after solving an RFP we never return to the RFP again. There are only a finite number of RFPs. The proof that solving an RFP is a finite process is identical to the proof of finiteness of the dual algorithm in [Goldfarb 83]. □

4.3.1 Solving an RFP

The preferred method used to solve the subproblem associated with $\tau^{(k)}$ is the dual algorithm. Such a "dual algorithm within a dual algorithm" can detect infeasibility in the constraints very easily as described in the next section. The dual method for solving an RFP proceeds as follows: Given a reduced forest $\tau^{(k)}$, compute s_i the slack for each vertex of the reduced forest. Define

$$s_\tau(\mathbf{w}) = \min_{v_i \in \tau}\{s_i\}$$

Define

$$r(\mathbf{w}) = \min_{n_j \in N_\Gamma}\{r_{ij} | r_{ij} = x^j_{max} - x_i, \forall i \in n_j\}$$

$$l(\mathbf{w}) = \min_{n_j \in N_\Gamma}\{l_{ij} | l_{ij} = x_i - x^j_{min}, \forall i \in n_j\}$$

$$u(\mathbf{w}) = \min_{n_j \in N_\Gamma}\{u_{ij} | u_{ij} = y^j_{max} - y_i, \forall i \in n_j\}$$

$$b(\mathbf{w}) = \min_{n_j \in N_\Gamma}\{b_{ij} | b_{ij} = y_i - y^j_{min}, \forall i \in n_j\}$$

$s_\tau(\mathbf{w})$ represents the minimum slack in the current reduced forest. $r(\mathbf{w})$ represents the most violated right extent bounding-box constraint in the current reduced forest. Similarly $l(\mathbf{w})$

represents the most violated left extent bounding-box constraint. $u(\mathbf{w})$ and $b(\mathbf{w})$ can be interpreted similarly. The RFP is not solved until the following conditions are satisfied:

$$\begin{aligned} s_\tau(\mathbf{w}) &\geq 0 \\ \min[r(\mathbf{w}), l(\mathbf{w}), u(\mathbf{w}), b(\mathbf{w})] &\geq 0 \end{aligned} \tag{4.7}$$

The strategy is to continue adding violated constraints until a solution is found or infeasibility is detected.

1. If $s_\tau(\mathbf{w}) < 0$ then timing constraints in τ are violated. The constraints to be added to the active set are the timing and bounding-box constraints for unique arcs of the violated path. All the constraints can be added simultaneously as per Proposition 3.1.2 of Chapter 3.

2. If $s_\tau(\mathbf{w}) \geq 0$ and $\min[r(\mathbf{w}), l(\mathbf{w}), u(\mathbf{w}), b(\mathbf{w})] < 0$ then a bounding-box constraint for some net in τ_A is violated. The constraint to be added is the violated bounding-box constraint.

3. If $s_\tau(\mathbf{w}) \geq 0$ and $\min[r(\mathbf{w}), l(\mathbf{w}), u(\mathbf{w}), b(\mathbf{w})] \geq 0$ the RFP is solved.

4. Compute the step directions $\delta\mathbf{w}$ and $\delta\boldsymbol{\lambda}$ assuming the violated constraint(s) is(are) active.

5. Compute t_1 the maximum step before a Lagrange multiplier turns negative. Compute t_2, the maximum step such that the added constraint becomes active. Let $t = \min(t_1, t_2)$.

6. Apply the infeasibility test at this time. (see Section 4.3.2) and stop if the constraints are infeasible.

7. If $t = t_1$ drop the constraint corresponding to t_1. Such a step is called a *partial step* because the violated constraints still remain violated. The algorithm will try to add them again during the next iteration. Such a step is necessary because dual feasibility must be maintained at all times.

8. If $t = t_2$ a constraint(s) is(are) added to the active set.

9. Update $\mathbf{w}^{(k)}$, $\boldsymbol{\lambda}^{(k)}$, $\tau^{(k)}$ and $\mathbf{A}^{(k)}$ and go to step 1.

4.3.2 Testing Infeasibility Efficiently

Definition 4.3.1 *A path is said to be feasible if there exists an assignment of locations to the cells of all the nets along the path such that the required arrival time at the path endpoint is satisfied.*

Let Ω be a single path from S to E. It is trivial to test Ω for infeasibility. Let m_r denote the cell with the highest drive (smallest resistance) on Ω. Let $\Omega = \{v_s, v_1, \ldots, v_r, v_{r+1}, \ldots, v_e\}$ be the sequence of *external* vertices occurring along Ω. If at least one of v_s and v_e are movable vertices, then Ω is feasible because we can then collapse all the cells to one point. Suppose both v_s and v_e are fixed cells. Place all the cells of the output nets associated with $\{v_s, \ldots, v_r\}$ at the location of v_s. Place all the cells of the output nets associated with $\{v_{r+1}, \ldots, v_e\}$ at the location of v_e. Compute the delay of Ω in this configuration. If it exceeds the required arrival time of v_e then Ω is infeasible since no other configuration of cells along Ω will yield a smaller delay. When a new path Ω is added to the current reduced forest τ, the procedure for testing feasibility is as follows:

1. Apply the simple feasibility test to the single path Ω. If it is infeasible, stop.
2. If Ω is feasible by itself and $\Omega \cap \tau = \emptyset$ stop, because $\Omega \cap \tau$ is also feasible.
3. If $\Omega \cap \tau \neq \emptyset$, apply the second feasibility test to the constraints.

The second feasibility test is as follows. Let a^+ denote an added constraint. Let A denote the matrix of currently active constraints. The constraints are of the form:

$$\begin{aligned} \mathbf{A}x &\leq \mathbf{b} \\ \mathbf{a}^+x &\leq b^+ \end{aligned} \tag{4.8}$$

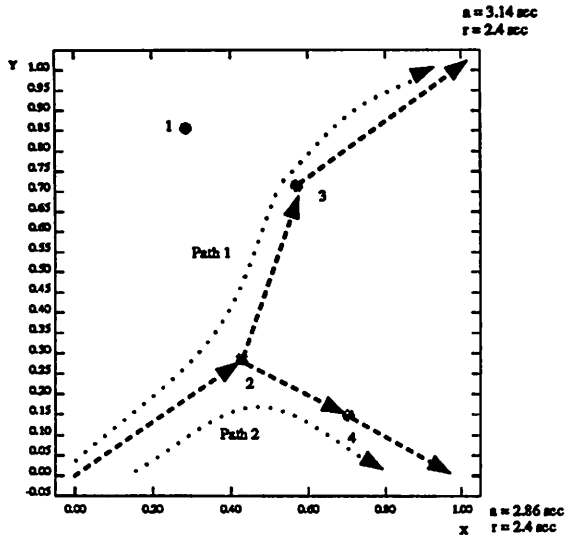
We need to test whether there exists a vector r such that $\mathbf{A}^T\mathbf{r} = \mathbf{a}^+$ and $\mathbf{r} \leq 0$. $\mathbf{A}$ has full rank (and hence so does $\mathbf{A}^T$) by construction. (If $\mathbf{A}$ does not have full rank, there is a redundant constraint in $\mathbf{A}$ which can be deleted during the course of the algorithm.) The method of determining $\mathbf{r}$ is to solve the linear system $\mathbf{A}^T\mathbf{r} = \mathbf{a}^+$ and test if $\mathbf{r} \leq 0$. The key to efficiency is to solve the linear system incrementally as constraints are added to the active set, rather than solving the entire system every time. This is done by maintaining sparse factors of $\mathbf{A}^T$ and updating the factors every time a new constraint is added. [Idnani 80] shows how to combine the sparse factors of $\mathbf{A}^T$ and $\mathbf{A}$ efficiently.

4.3.3 Speedup Obtained

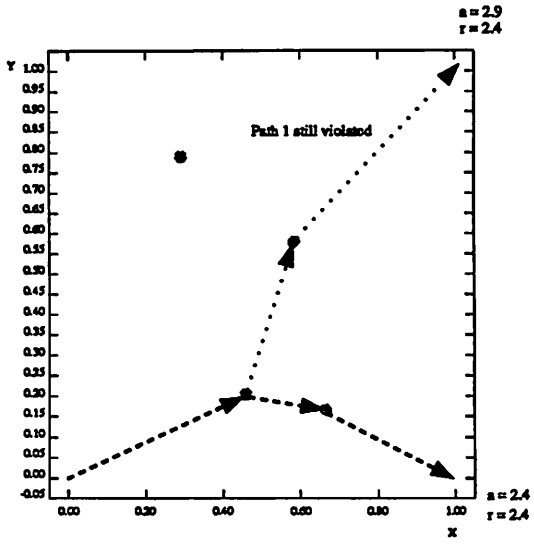
A similar calculation to the one used in Chapter 3 can be used to estimate the speedup due to operating on RAFs. Suppose a conventional dual algorithm encounters k active forests during the course of finding an optimal solution. Let the number of arcs in a critical path be t. Let the fraction of all the modules that lie on critical paths be β. Assuming that about 30% of the arcs in a critical path are unique arcs, the conventional solution technique would take at least $O(tk)$ iterations, performing $O(M^2)$ work per iteration for the factorizations of A and A^T. Hence, the work done would be roughly $O(M^2tk)$. On the other hand, the modified algorithm would take $O(k)$ iterations performing about $O(\beta^2 M^2)$ work per iteration. The work done would then be about $O(\beta^2 M^2 k)$ which is identical to the expression obtained earlier.

4.3.4 An Example

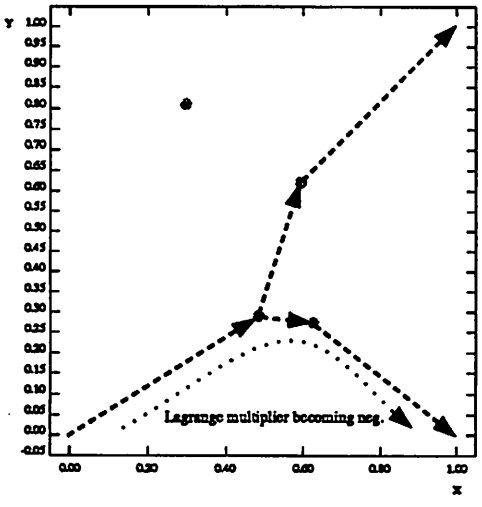
Consider the example of Chapter 3 with four movable cells and two fixed cells. The dual algorithm starts out with the unconstrained solution shown in Figure 4.1(a). The initial active set is empty. There are two violated critical paths as shown. Normally, the algorithm would add path 1 (the most violated) to the active set, in which case it would terminate in one step at the optimal solution! However, for the purpose of illustration, let us add path 2 instead. Following this, we solve the subproblem obtained by considering only the constraints associated with path 2 and ignoring all the other constraints. This yields the solution shown in Figure 4.1(b). At this point, the Lagrange multiplier of path 2 is positive. Note that path 1 is still violated and so we add the constraints associated with path 1 to the active set and solve the problem. This yields a step such that if the entire step is taken, the Lagrange multiplier for path 2 becomes negative (i.e., if the full step is taken, path 2 is no longer binding). So, we take a partial step upto the point where the Lagrange multiplier of path 2 is about to turn negative Figure 4.1(c). We cannot take the full step because *dual feasibility* must be maintained at all times. Then we drop all the constraints associated with path 2 and proceed to the optimal solution as shown in Figure 4.1(d). Even when we did not add the most violated path first, the Dual algorithm takes fewer steps than the Primal and does not involve any bounding box changes. In general too, it was observed that the Dual algorithm typically required fewer steps than the Primal and involved fewer bounding-box changes.



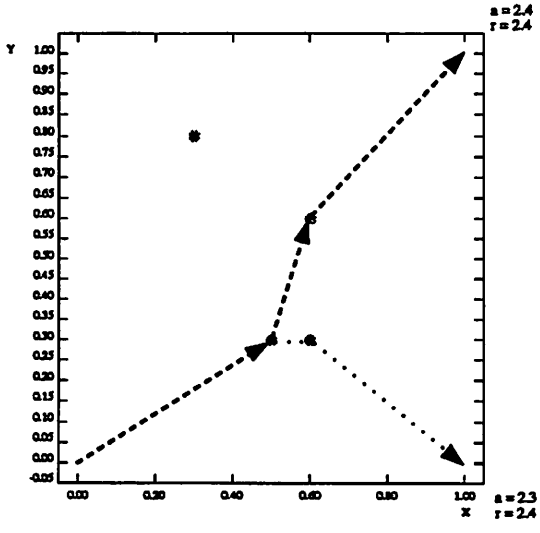
(a) Unconstrained solution, two violated paths



(b) Add path 2 to current RF and solve subproblem



(c) Add path 1 to RF and solve subproblem. Partial step is taken and path 2 drops out.



(d) Solve subproblem corresponding to Path 1. All constraints satisfied, solution is optimal.

Figure 4.1: Steps of Dual algorithm on example

4.3.5 A Comparison of the Dual and the Primal Algorithms

The significant advantages of the Dual Algorithm over the Primal are:

1. An initial feasible solution is not required.
2. Bisection Timing Verification is not required.
3. It can be implemented more efficiently [Goldfarb 83].
4. There is a choice of which constraint to add, unlike the primal.
5. Typically, experience indicates that the unconstrained optimal solution is "close" to the constrained one, and the dual converges rapidly to it.

One point to be noted here is that all the advantages of operating on RAFs apply to the dual algorithm too. Firstly, when a path becomes active, all the constraints for the unique arcs may be added simultaneously. When a path leaves the active set as result of a partial step, all the constraints for the unique arcs may be dropped simultaneously. Updating of sparse factors of the constraint matrix A or A^T can be done in one fell swoop for all constraints corresponding to the unique arcs. Another important point is that the dual algorithm converges without any assumptions on the step size. To summarize the differences between the primal and the dual algorithm, a few features are recapitulated here. The Primal algorithm:

- starts from a feasible solution
- produces a non-increasing sequence of objective values
- maintains feasibility at all times

while the Dual algorithm

- starts from the unconstrained optimal solution
- produces a non-decreasing sequence of objective values
- maintains dual feasibility at all times, i.e., it is infeasible with respect to the entire set of constraints until the final step
- detects infeasibility

1. Solve the unconstrained problem corresponding to NLP.
   Set the current active forest to $\tau_A = \emptyset$.
2. Select a path Ω with negative slack, if any. If there is no such path, terminate; the optimal solution has been reached. Let $\tau'_A = \tau_A \cup \Omega$.
3. Solve the subproblem corresponding to τ'_A (RFP_A).
4. If RFP_A has no solution, terminate because the feasible region is empty; otherwise we have a new solved subproblem. Let $\tau_A = \tau_{SA}$, go to step 2.

Figure 4.2: Outline of the Dual Algorithm

1. Start with an arbitrary initial feasible solution and a current reduced active set τ_A. Let the current cell location vector be w.
2. Solve the equality-constrained non-linear programming problem corresponding to the current active forest. The solution yields a ''step'' or a direction from the current location to the new location obtained by the solution. Let this step be δw.
3. Move along δw until a new reduced forest becomes active.
4. Update the current RAF.
5. Check the current solution for optimality (using Equation 3.8. The Lagrange multipliers are obtained as a by-product of the solution process.) If it is optimal, stop.
6. Go to step 2.

Figure 4.3: Outline of the Primal Algorithm

4.4 Conclusions and Comments

The dual algorithm avoids one of the key problems of the primal: the requirement of an initial feasible solution. In addition, it avoids the timing verifications that were necessary for the primal algorithm. It tends to start closer to the final solution than the primal algorithm and in experiments, it required fewer iterations to solve the problem. This corroborates the observations of Idnani.

Usually, fewer iterations are required and fewer fillins are created in the matrix factors. However, the improvement over the primal is at best a factor of two or three which is still not enough to solve very large problems. This led me to investigate methods for solving the problem without requiring matrix factorizations which is the subject of the following chapter.

Chapter 5

Polishing the Cut

The Primal method of Chapter 3 and the Dual method of Chapter 4 left many problems unsolved. The developments discussed in this chapter address and solve those issues. This chapter forms the bulk of the thesis and as a prelude to the contents, the key features are summarized in this section. The theoretical foundation of the techniques described here rests on an emerging method for solving very large-scale mathematical optimization problems that have structure inherent in them: *Lagrangian Relaxation.*

- The techniques are significantly faster than the Primal or Dual method. They keep the promise of being able to automatically place circuits of extremely large size (20K gates and more) in a short time on currently available computers.
- Degeneracy is no longer a problem.
- The slot constraints that were not addressed earlier can now be resolved in a unified manner.
- The method works in two phases. During the first phase, continuous constrained optimization is performed, and during the second phase, constrained discrete optimization techniques are applied. The transition from continuous to discrete space is gradual and smooth.
- No initial feasible solution is required and no unweildy matrix factorizations are necessary. In fact, the method demands only linear memory requirements.

- The method can handle convex nonlinear delay functions like those of the distributed model described in Section 2.4.6.

To illustate the method the star-connected delay model is used and interconnection resistance effects are neglected for simplicity. First, the original problem is restated for convenience:

$$\begin{array}{lll} \text{minimize} & \mathrm{L(w)} & \text{(GP)} \\ \text{subject to} & & \end{array}$$

$$\begin{array}{llll} a_j & \geq & a_i + d(v_i, v_j) & \forall (v_i, v_j) \in A \\ a_j & \leq & T_j & \forall v_j \in E \\ a_j & \geq & T_j & \forall v_j \in S \\ d(v_i, v_j) & = & f(X_i, Y_i) & \forall n_i \in \mathcal{N} \end{array} \tag{5.1}$$

For the star-connected net delay model (neglecting the interconnect resistance effects), the delay of external arc (v_i, v_j) can be written as:

$$d(v_i, v_j) = R_i \sum_{m_j \in n_i} [C_h |x_j - x_i| + C_v |y_j - y_i|] \tag{5.2}$$

Figure 5.1 shows a star connected net and the vertical and horizontal segments for the connections.

5.1 Lagrangian Relaxation

Lagrangian Relaxation has been used occassionally in the past by economists and operations researchers but has not found widespread use because of the difficulties involved in getting the method to converge to a solution on general problems. However, problems with special structure in the objective function and constraints respond magnificently to the technique [Fisher 85]. Unfortunately, finding special structure requires special insight in most cases. Luckily, the constrained optimization problem as stated in this thesis possesses some very useful properties that have been exploited fully in this work. In order to give the reader an idea of the method of Lagrangian Relaxation, a simple example is discussed.

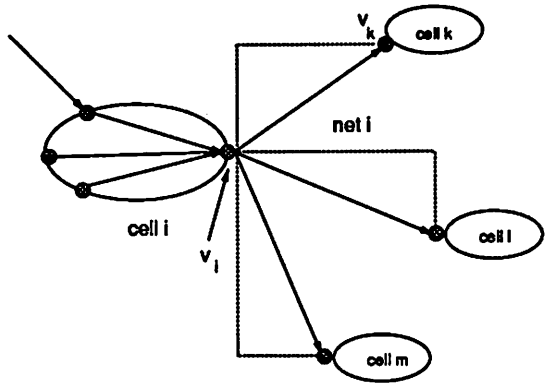


Figure 5.1: Example of a star connected net

5.1.1 A Simple Example

Consider the constrained optimization problem:

$$\begin{array}{ll} \min & (x-2)^2 \\ \text{subject to} & x \geq 5 \end{array} \tag{5.3}$$

The obvious optimal solution to this problem is $x = 5$, with optimum objective value 9. First, we rewrite the problem so that all the inequalities are of the form $a \leq 0$.

$$\begin{array}{ll} \min & (x-2)^2 \\ \text{subject to} & 5 - x \leq 0 \end{array} \tag{5.4}$$

From basic Lagrangian theory [Fisher 85] it can be shown that the problem stated above is equivalent to the following optimization problem:

$$\max_{\mu \geq 0} \; min_x \; \mathrm{L}(x, \mu) = (x-2)^2 + \mu(5-x) \tag{5.5}$$

where as before μ is the Lagrange multiplier associated with the constraint (if there are multiple constraints, there is one multiplier associated with each constraint). The method of Lagrangian Relaxation as applied to this problem is now described skeletally. The description here is considerably simplified for ease of explanation and the reader should be cautioned that several complicating details have been omitted. A more comprehensive treatment can be found in [Shapiro 79]. The method proceeds iteratively as follows:

1. Start with an initial value for μ, say 0. (Usually one can start with an educated guess).

2. For a fixed value of μ, solve the problem of Equation 5.5. For fixed μ this is an unconstrained minimization problem (and for problems with special structure, it is easy to solve).

3. Update μ based on the solution obtained. Intuitively, μ acts like a penalty on the constraint. If the current value of μ results in a solution that violates the constraint, it needs to be increased. If the value of μ is too high, the solution will be far from optimal - the constraint is satisfied by a wide margin. Thus, it is possible to update μ based on the residue in the constraint. There are many possible update methods that will guarantee convergence of the method for convex programming problems. One that is widely used is:

$$\begin{aligned} t^{(0)} &= \alpha \\ \mu^{(k+1)} &= \max\{0, \mu^{(k)} + t^{(k)}(5 - x^{(k)})\} \\ t^{(k+1)} &= \beta t^{(k)} \end{aligned} \tag{5.6}$$

   where α is a positive constant and β is a positive constant ≤ 1.0.

4. Repeat steps 2 and 3 till convergence.

Let us apply this recipe to the example, with $\alpha = 1.0, \beta = 1.0$. The values for x and μ are listed for each iteration.

1. Solve Equation 5.5 for $\mu^{(0)} = 0$. The optimal solution is $x^{(0)} = 2$.

2. Minimize $L(x, \mu)$ with respect to x for fixed $\mu^{(1)} = \mu^{(0)} + (5 - x^{(0)}) = 3$. The initial value of μ was too low and this step increases it by an amount equal to the residue in the constraint. The solution is $x^{(1)} = 3.5$.

3. Proceeding in a similar manner to Step 2, we obtain $\mu^{(2)} = 4.5$, $x^{(2)} = 4.25$.

4. $\mu^{(3)} = 5.25$, $x^{(3)} = 4.625$.

5. $\mu^{(4)} = 5.625$, $x^{(4)} = 4.8125$.

In the limit x converges to the optimal value of 5. In practice, there are several methods of accelerating the convergence and for well structured problems, typically only a few iterations are required (see further references in [Fisher 75]). For example, if we had chosen $\alpha = 2.0$ the algorithm would have converged to the optimal solution in one step.

5.1.2 Detailed Recipe for Lagrangian Relaxation

Let us now consider a detailed description of the method of Lagrangian Relaxation for a general convex problem of the form:

$$\begin{array}{lll} \min & \mathbf{f}(x) & \\ \text{subject to} & \mathbf{g}(x) & \leq c \\ & \mathbf{h}(x) & \leq d \end{array} \tag{5.7}$$

where $\mathbf{g}(x), \mathbf{h}(x)$ are convex vector functions of x. The constraint set is partitioned into $\mathbf{g}(x)$ and $\mathbf{h}(x)$. It is assumed that $\mathbf{g}(x)$ consists of constaints that complicate the problem and they are termed "complicating" constraints. It is also assumed that the problem is easy to solve in the absence of $\mathbf{g}(x)$. As an aside and a preview of the techniques in this chapter, note that the wirelength optimization problem GP of Chapter 3 is very easy to solve without the timing constraints. Hall [Hall 70] first solved it for the quadratic wirelength model and showed that the solution corresponds to solving a linear system of equations. Later, Tsay [Tsay 88] exploited the structure of the unconstrained problem to solve very large scale wirelength minimization problems. The corresponding Lagrangian problem is:

$$\begin{array}{ll} \max\limits_{\lambda \geq 0} \min\limits_{x} & \mathbf{f}(x) + \lambda^T(\mathbf{g}(x) - c) \\ \text{subject to} & \mathbf{h}(x) \leq d \end{array} \tag{5.8}$$

where λ is a vector of multipliers. The most general method proceeds as follows:

1. Select an initial value for λ.
2. Solve
$$\begin{array}{ll} \min\limits_{x} & \mathbf{f}(x) + \lambda^T(\mathbf{g}(x) - c) \\ \text{subject to} & \mathbf{h}(x) \leq c \end{array} \tag{5.9}$$
for a fixed value of λ.
3. Update λ.
4. Repeat steps 2-3 until convergence.

Step 3 is critical to the convergence of the algorithm, particularly for discontinuous absolute-valued constraints like the timing constraints of Chapter 2. Step 2 is critical to the efficiency of the algorithm. The key contributions of this chapter are efficient methods for performing steps 2 and 3.

5.1.3 Lagrangian Relaxation for Discrete Optimization Problems

The method of Lagrangian Relaxation is not restricted to continuous space optimization. It has been applied successfully by Held et. al. [Held 70], for solving the Traveling Salesman problem. Again, exploitation of the structure of the problem is key to successful application. Constrained discrete optimization is used to resolve slot constraints during the second phase of the algorithm described in this chapter.

5.2 Resolving Slot Constraints

A common requirement in most cell-based ICs is that the cells lie in slots or regular arrays. A solution of NLP will yield a "placement" that usually does not satisfy the slot requirements. Such a placement has been called an "initial" or "global" placement. Several techniques have been proposed to refine the global placement to produce a slotted final result [Breuer 77, Kleinhans 91, Tsay 88, Cheng 84, Jackson 89]. The technique that is used in this work is generalization of that proposed in [Kleinhans 91]. The key feature of the technique is the highly efficient method used to solve the problem with slot resolution constraints in the presence of timing constraints. The solution technique will be presented in a later section. At this point, it suffices to assume that an efficient solution method exists for the wirelength optimization problem in the presence of constraints.

Let r_x and r_y represent the coordinates of the center of the chip. First the global timing-constrained placement problem is solved with two additional constraints:

$$\begin{aligned} \frac{1}{M}\sum_{i\in\mathcal{M}} x_i &= r_x \\ \frac{1}{M}\sum_{i\in\mathcal{M}} y_i &= r_y \end{aligned} \tag{5.10}$$

This is termed as the "level 0" problem. These constraints ensure that the cells are spread around the center of the chip. After this, the cells are partitioned into four equal sized sets. This is done by first dividing the cells into two equal sized sets along the y-direction and then subdividing each set into two subsets along the x-direction. Let the sets be S_0, S_1, S_2 and S_3. The chip is divided into four equal-sized regions and (r_j^x, r_j^y) denotes the coordinate of the center of the ith region. Now, eight centering constraints (four in the x direction and four in the y direction) are added to the constraint set to form a new problem GP_1, termed the "level 1" problem. Figure 5.2 shows an example with four regions and the sets of cells in different shades after the solution. (Note that some cells from one region have migrated

into another). The eight constraints are:

$$\begin{aligned} \frac{1}{|S_j|}\sum_{i\in S_j} x_i &= r_j^x, \quad j = 1,...,4 \\ \frac{1}{|S_j|}\sum_{i\in S_j} y_i &= r_j^y, \quad j = 1,...,4 \end{aligned} \tag{5.11}$$

The effect of these constraints is to spread the cells out in the four directions. Note that unlike many other partitioning approaches, a cell is not required to lie within its region. A cell has freedom to migrate into any other region. This allows the algorithm greater flexibility in minimizing wirelength while still satisfying slot constraints. It also makes the partitioning of cells more flexible in that a cell may change partitions later in order to reduce wirelength or satisfy timing constraints. Following the solution of GP_1, the cells are repartitioned into sixteen sets, giving thirty two center of mass equations, and the new problem GP_2 is solved (with the timing constraints included). During the repartitioning, the old partition information is not considered and new partitions are generated based on current cell locations. The solution and repartitioning process can be repeated to a level of granularity such that one cell remains within each region. This technique can effectively resolve the slotted array requirements of cell-based ICs. However, following this method with a new constrained discrete optimization technique yielded significantly better results than using this method alone. The discrete optimization will be described in a following section. Note that the values for r_j^x and r_j^y need not be restricted to uniformly distributed centers of mass, but can be derived from the structure and location of slots on the chip.

5.3 The Continuous Optimization Algorithm

In this section, the details of the first phase are discussed. For illustration purposes, the quadratic wirelength and the star net delay models are used. Some of the features of the method are:

- Memory requirements are linear in the size of the problem
- The technique is iterative and very fast
- The problem can be solved to any desired accuracy
- It is generalizable to arbitrary convex delay functions
- All critical paths are considered in a very efficient manner

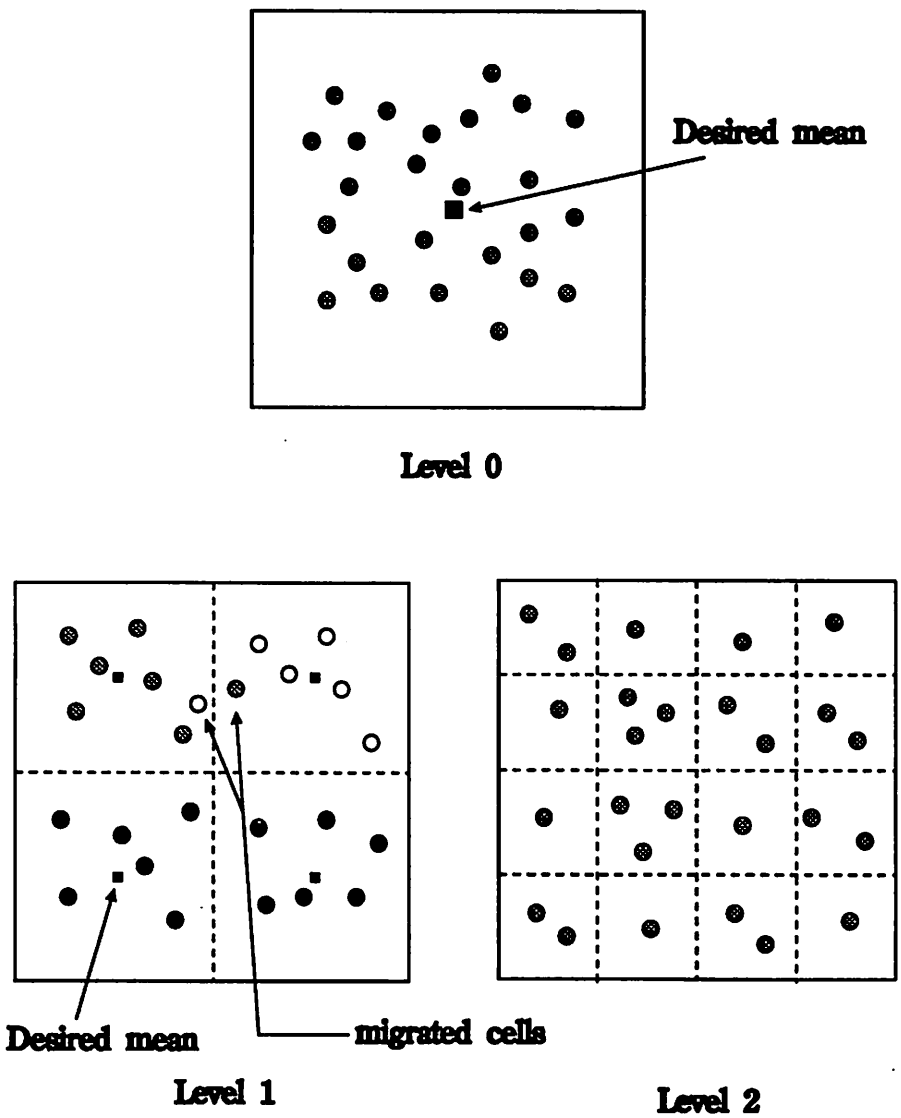


Figure 5.2: A sequence of showing application of spread constraints

- Slot resolution constraints are integrated in an efficient and consistent manner with the timing constraints
- The problem is solved optimally at every level

5.3.1 Solving the Lagrangian

The specific problem for the quadratic wirelength model and the star net delay model (neglecting interconnect resistance) with k centers of mass is restated below:

$$
\begin{array}{lllll}
\text{minimize} & \mathrm{L(w)} & & & (GP_k) \\
\text{subject to} & & & & \\
 & a_j & \geq & a_i + d(v_i, v_j) & \forall (v_i, v_j) \in A \\
 & a_j & \leq & T_j & \forall v_j \in E \\
 & a_j & \geq & T_j & \forall v_j \in S \\
 & d(v_i, v_j) & = & f(X_i, Y_i) & \forall n_i \in \mathcal{N} \\
 & \frac{1}{|S_j|} \sum_{i \in S_j} x_i & = & r_j^x, & j = 1, ..., k \\
 & \frac{1}{|S_j|} \sum_{i \in S_j} y_i & = & r_j^y, & j = 1, ..., k
\end{array}
\tag{5.12}
$$

Following the notation of Chapter 3, let $\mathbf{w}$ represent the combined vector of cell x and y positions and vertex arrival time variables. Let $\mathbf{Aw} \leq \mathbf{c}$ be the matrix representation of the timing and center of mass constraints. For the performance driven placement problem, with the quadratic wirelength model, the Lagrangian equation can be written as:

$$\max_{\lambda \geq 0} \min_{x} \frac{1}{2} \mathbf{w}^T \mathbf{Q} \mathbf{w} + \mathbf{b}^T \mathbf{w} + \lambda^T (\mathbf{Aw} - \mathbf{c}) \tag{5.13}$$

where λ is a vector of multipliers. For any fixed value of λ, say λ^k the problem has a very simple solution [1].

$$\mathbf{w}^{(k+1)} = -\mathbf{Q}^{-1}[\lambda^{(k)} \mathbf{A} + \mathbf{b}] \tag{5.14}$$

Note that $\mathbf{Q}$ is independent of cell locations. Thus, at every iteration, the only component that changes in the right-hand side of Equation 5.14 is λ and the only product to be computed is $\lambda^{(k)}\mathbf{A}$. This is a linear-time computation since by Corollary 3.1.2 the maximum number of active paths is linear in the size of the timing graph. In an efficient implementation, $\mathbf{Q}^{-1}$ is never computed. Since $\mathbf{Q}$ is positive definite, the equation Equation 5.14

[1] To keep the notation simple, it is assumed that $\lambda^{(k)}$ refers to a row vector, i.e., the transposition symbol is dropped.

can be solved iteratively using an algorithm like the accelerated Gauss-Seidel method for solving linear systems of equations[Golub 89]. If the Gauss-Seidel method is used, at each iteration k the previous solution $\mathbf{w}^{(k-1)}$ can be used as an initial solution for rapid convergence. It is interesting to note that Rockafellar [Rockafellar 84] shows solving Equation 5.14 is equivalent to solving a minimum quadratic cost flow problem.

5.3.2 Updating the Lagrange Multipliers

The method used to update Lagrange multipliers from iteration to iteration is based on the subgradient method for setting dual variables [Held 74]. This technique starts with an initial value λ^k and iteratively applies the formula:

$$\lambda^{(k+1)} = \max\{0, \lambda^{(k)} + t^{(k)}(\mathbf{A}\mathbf{w}^{(k)} - \mathbf{c})\} \tag{5.15}$$

In this formula, $t^{(k)}$ is a scalar step size and $\mathbf{w}^{(k)}$ is the optimal solution for Equation 5.14 for $\lambda = \lambda^{(k)}$. The components of $\mathbf{A}\mathbf{w}^{(k)} - \mathbf{c}$ are the slacks in the constraints. For the timing constraints the components are none other than the vertex slacks for the cells on critical paths. For the spread constraints, they are the differences between the desired centers of mass of the various groups and the actual centers of mass. The choice of $t^{(k)}$ is critical to the success of the algorithm for two reasons: (1) it is closely tied to the linearization of the absolute valued delay constraints and (2) it affects the convergence of the algorithm. The procedure for computing $t^{(k)}$ is explained in the following subsection. The convergence properties of such a method for updating λ are described in detail in [Held 74].

5.3.3 Computing $t^{(k)}$

Recall that the delay models described in Chapter 2 all have equations with absolute valued terms in them. It is possible to convert these delay constraints to linear constraints by using additional variables as in Section 3.1.2. However, there is a more efficient method that avoids introducing variables. Suppose the solution is $\mathbf{w}^{(k)}$. At this solution, for all the critical paths, write the delay equations as linear equations, removing the absolute values from Equation 5.2, switching signs wherever necessary to ensure that all the terms are non-negative. For example if currently $x_3 > x_2$ and there is a critical path passing from cell m_2 to cell m_3, write $x_3 - x_2$, otherwise, we write $x_2 - x_3$ (see Figure 5.3). Then, the right hand side of Equation 5.14 is updated based on this configuration and the

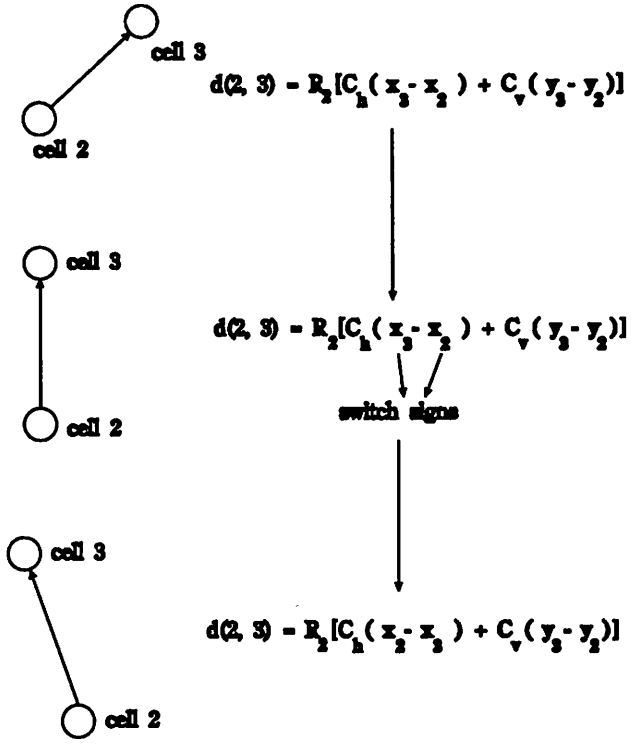


Figure 5.3: Linearizing absolute valued constraints

system of equations is solved for $\mathbf{w}^{(k+1)}$. Next, the largest value of $t^{(k)}$ is chosen such that a term in one of the delay equations just changes sign (i.e., is about to change from its current configuration to the opposite one as shown in Figure 5.3). $\mathbf{w}^{(k+1)}$ and $\lambda^{(k+1)}$ are updated according to this value of $t^{(k)}$. This method is equivalent to the method of introducing additional variables and produces identical behavior without the expense of more variables.

5.3.4 Updating the critical path set

The algorithm maintains a set of active critical paths (i.e. RAF) throughout the algorithm. Active paths are those whose current Lagrange multipliers are positive. By maintaining the set of active paths, further efficiency can be achieved since the components of $\lambda^T\mathbf{A}$ need not be computed when some component of λ, say $\lambda_i = 0$. Since not all paths are critical, this typically makes updating the right hand side a sub-linear procedure. In any event, the number of critical paths required to represent the problem at any time is linearly bounded as shown in Corollary 3.1.2. Thus, the maximum number of multipliers that are active at any time is linear in the size of the timing graph. Note that although the matrix $\mathbf{A}$ contains all the arc equations, in a practical implementation they are never explicitly computed or written unless they are required. The only arcs that actually participate in updating the right hand side are those which belong to the current set of critical paths (RAF).

After solving for $\mathbf{w}^{(k+1)}$, a fast timing analysis on the timing graph is performed to determine the paths that have become critical since the previous iteration. These paths are then added to the critical set with zero initial-valued Lagrange multipliers.

5.3.5 Computational Complexity

The flow of the algorithm is described in Figure 5.4. The work done per inner-loop iteration of the algorithm is very little since it involves a right-hand side update which is $O(M)$, one step of matrix solution (assuming a direct solution method) to solve for the new value of $\mathbf{w}$ which is $O(M^2)$, computing t, which is $O(E)$, where E is the number of edges in D_T, and updating the critical paths, which can be done in $O(M + E)$. Therefore, the work done per inner-loop iteration is bounded by $O(M^2)$. Note that the critical path set is continuously updated as new paths become critical. For the linearized delay equations, this procedure converges according to [Held 74, Shapiro 79]. There is no theoretical bound on

1. Initialize the level $l = 0$.
2. Initialize the current RAF $\tau_A = \emptyset$.
3. Add the spread constraints for level l to the active set. There are no Lagrange multpliers for the timing constraints initially since τ_A is empty. (Multipliers exist for the spread constraints for level 0.)
4. Update $\lambda^T \mathbf{A}$ and solve Equation 5.14.
5. Perform a timing verification on the timing graph D_T and update the current RAF.
6. Update the Lagrange multipliers for the active constraints (timing and spread). In the formulation, there is one constraint per edge in the timing graph. Although multiple edges in the timing graph will be added to the RAF, independent Lagrange multipliers are not necessary for each edge. By Corollary 3.1.1 there is only one effective Lagrange multiplier per path, since the equations for all the unique arcs of a path have the same slack.
7. If the spread constraints for the current level are satisfied and there are more levels, increase the level. and go to step 3.
8. If the constraints are satisfied to the desired accuracy, and the current level is at the maximum level for the chip, STOP.
9. Go to step 4.

Figure 5.4: Outline of Lagrangian Relaxation

the number of iterations required for convergence of the inner loop, however, in practice the number of iterations required per level was very low - 200-400 even for the largest examples tested.

5.3.6 Extension to Nonlinear Delay Models

It is straightforward to extend the Lagrangian Relaxation algorithm described above to a convex nonlinear delay model like that of the star-connected net with interconnect resistance effects. Assume that an iterative method like Gauss-Seidel relaxation is used to solve the system of equations generated at each iteration. In the case of linear delay equations, only the right-hand side of Equation 5.14 had to be updated every iteration. When the timing constraints are nonlinear however, some matrix entries may also need to be updated. The work done per iteration remains the same, although the number of iterations to convergence usually increases.

5.4 Resolving Slot Constraints in Practice

Although it is possible to add slot constraints as described in Section 5.2 until exactly one cell remains in each region, using a modification of that technique results in further improvement in timing and wirelength. The star connected net delay model is used to illustrate the ideas.

The idea is to perform hierarchical partitioning until a few (10-20) cells remain in each region. Following this, constrained Assignment (weighted bipartite matching) is used to assign slot positions to the cells within each region. Consider a region S_k containing M cells and $N \geq M$ slots. A cost matrix $\mathbf{C}$ is set up with as many rows as the number of the cells in the region and as many columns as the number of slots in the region. An entry $\mathbf{C}_{ij}$ in the cost matrix is the cost of assigning the cell m_i to slot s_j within region S_k. Let z_{ij} be an integer variable $\in \{0,1\}$ that is 1 if cell m_i occupies slot s_j and 0 otherwise. Let $\mathbf{X}_j$ be the x position of slot s_j and $\mathbf{Y}_j$ denote the y position of slot s_j. $\mathbf{X}$ is the constant vector of slot x positions and $\mathbf{Y}$ is the constant vector of slot y positions. The combined vector of cell x and y positions is denoted by $\mathbf{w}_{cell}$ as before. The first M entries in $\mathbf{w}_{cell}$ correspond to cell x positions and the subsequent M entries correspond to cell y positions. For the sake of brevity the subscript will be dropped from $\mathbf{w}_{cell}$ in the following analysis. Let $\mathbf{Z} = [z_{ij}]$ denote the $M \times N$ 0-1 matrix constructed from the assignment variables z_{ij}. For each cell i, the assignment variables corresponding to that cell are in row i of the matrix $\mathbf{Z}$. There is exactly one non-zero entry per row. Thus, for example $\mathbf{ZX}$ is the vector of cell x positions at the current solution.

For each region, the problem can be written as:

$$
\begin{array}{lcll}
\min \sum_{ij} \mathbf{C}_{ij} z_{ij} & & & (DP) \\
\text{subject to} & & & \\
\sum_{j=1}^{N} z_{ij} & = & 1, & i = 1, \ldots, M \\
\sum_{i=1}^{M} z_{ij} & \leq & 1, & j = 1, \ldots, N \\
\mathbf{x} & = & \mathbf{ZX} & \\
\mathbf{y} & = & \mathbf{ZY} & \\
\mathbf{Aw} & \leq & \mathbf{c} & \\
z_{ij} & \in & \{0,1\}, & \forall i, j
\end{array}
\tag{5.16}
$$

The additional constraints $\mathbf{Aw} \leq \mathbf{c}$ are the familiar timing (path) constraints written in terms of cell locations. The cost of assigning a cell to a slot is the total wirelength of all the

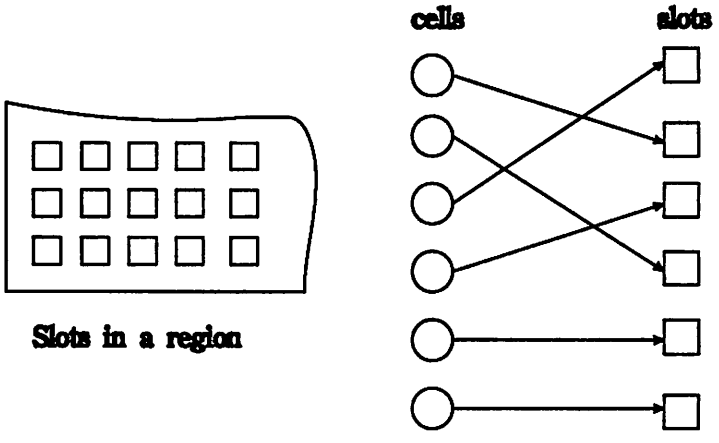


Figure 5.5: An illustration of the assignment formulation

nets that attach to the cell. Any wirelength model can be used here and no assumptions are made on the mathematical properties of the model. Obviously, one would use the most accurate model available. The problem as stated above is an integer program and falls in the class NP-complete.

5.4.1 Solving the Constrained Problem

Once again, the structure of the problem comes to the rescue. Observe that if the timing constraints are dropped, the problem reduces to Linear Assignment (bipartite matching). Algorithms of polynomial complexity exist for solving Linear assignment problems [Lawler 76]. Hence, we may view the timing constraints as "complicating constraints" and apply Lagrangian Relaxation to the problem. Let $\mathbf{W}$ denote the combined vector of slot locations:

$$\mathbf{W} = \begin{bmatrix} \mathbf{X} \\ \mathbf{Y} \end{bmatrix}$$

Using this notation, we can rewrite $\mathbf{w}$ in terms of the assignment variables as:

$$\mathbf{w} = \begin{bmatrix} \mathbf{Z} & \mathbf{0} \\ \mathbf{0} & \mathbf{Z} \end{bmatrix} \mathbf{W}$$

Let

$$\mathbf{Z}_{xy} = \begin{bmatrix} \mathbf{Z} & \mathbf{0} \\ \mathbf{0} & \mathbf{Z} \end{bmatrix}$$

The problem can be rewritten as:

$$\begin{array}{lcll} \min & \sum_{ij} \mathbf{C}_{ij} z_{ij} & & (DP) \\ \text{subject to} & & & \\ \sum_{j=1}^{N} z_{ij} & = & 1, & i = 1, \ldots, M \\ \sum_{i=1}^{M} z_{ij} & \leq & 1, & j = 1, \ldots, N \\ \mathbf{A}\mathbf{Z}_{xy}\mathbf{W} & \leq & \mathbf{c} & \\ z_{ij} & \in & \{0,1\}, & \forall i,j \end{array} \tag{5.17}$$

The relaxed problem is:

$$\begin{array}{lll} \max\limits_{\nu \geq 0} \min & \sum_{i=1}^{M} \sum_{j=1}^{N} \mathbf{C}_{ij} z_{ij} + \nu^T (\mathbf{A}\mathbf{Z}_{xy}\mathbf{W} - \mathbf{c}) & \\ \text{subject to} & & \\ \sum_{j=1}^{N} z_{ij} = 1, & & i = 1, \ldots, M \\ \sum_{i=1}^{M} z_{ij} \leq 1, & & j = 1, \ldots, N \\ z_{ij} \in \{0,1\}, & & \forall i,j \end{array} \tag{5.18}$$

This problem remains NP complete for variable ν. However, by relaxing it further a tractable formulation can be obtained. When ν is fixed, the term $\nu\mathbf{c}$ becomes a constant and can be dropped from the problem. For a fixed ν consider the problem:

$$\begin{array}{lll} \min & \sum_{i=1}^{M} \sum_{j=1}^{N} \mathbf{C}_{ij} z_{ij} + \nu^T (\mathbf{A}\mathbf{Z}_{xy}\mathbf{W}) & \\ \text{subject to} & & \\ \sum_{j=1}^{N} z_{ij} = 1, & & i = 1, \ldots, M \\ \sum_{i=1}^{M} z_{ij} \leq 1, & & j = 1, \ldots, N \end{array} \tag{5.19}$$

Note that the integer constraints on z_{ij} have been dropped too. Let $\mathbf{A}$ have p constraints. Then $\mathbf{A}\mathbf{Z}_{xy}\mathbf{W}$ can be rewritten as:

$$\mathbf{A}\mathbf{Z}_{xy}\mathbf{W} = \sum_{k=1}^{p} \sum_{i=1}^{M} \sum_{j=1}^{N} (\mathbf{A}_{ki}\mathbf{X}_j + \mathbf{A}_{k,i+M}\mathbf{Y}_j) z_{ij}$$

Substituting this expression in $\nu^T \mathbf{A}\mathbf{Z}_{xy}\mathbf{W}$, we can state the problem more intuitively as:

$$\begin{array}{lll} \min & \sum_{ij} [\mathbf{C}_{ij} + \sum_{k=1}^{p} \nu_k (\mathbf{A}_{ki}\mathbf{X}_j + \mathbf{A}_{k,i+M}\mathbf{Y}_j)] z_{ij} & \\ \text{subject to} & & \\ \sum_{j=1}^{N} z_{ij} = 1, & & i = 1, \ldots, M \\ \sum_{i=1}^{M} z_{ij} \leq 1, & & j = 1, \ldots, N \end{array} \tag{5.20}$$

Thus, for fixed ν, the formulation is a linear assignment problem (or minimum cost flow or weighted bipartite matching) which has an integer solution if it is feasible and it is to this problem that Lagrangian Relaxation is applied. The flow of the algorithm is:

1. Start with an initial value of $\nu = \nu^{(0)}$.
2. Solve LA for the current value of ν.
3. Update the set of critical paths (i.e. A, the matrix representation of the current RAF) and ν according to $\nu^{(k+1)} = \max(0, \nu^{(k)} + (\mathbf{A}\mathbf{w} - \mathbf{c}))$.

4. Repeat steps 2 and 3 until the constraints are satisfied to a desired accuracy.

The relaxed linear assignment problem has an intuitively appealing interpretation. Consider the cost of assigning a cell m_i to a slot s_j. The cost of this assignment is:

$$\mathbf{C}'_{ij} = \mathbf{C}_{ij} + \sum_{k=1}^{P} \nu_k (\mathbf{A}_{ki}\mathbf{X}_j + \mathbf{A}_{k,i+M}\mathbf{Y}_j)$$

At every iteration, the cost of assigning a cell to a slot is modified by adding a term that is derived by looking at the Lagrange multipliers of the paths. The multipliers themselves are derived from the path slacks. Thus, the additional cost of assigning a cell to a slot can be derived by looking only at the paths with non-zero multipliers passing through the cell. As the reader may recall, Corollary 3.1.2 assures us that only a linear number of paths need to be considered in the worst case.

Unlike the continuous case, it may not be possible to obtain the exact optimum by solving the relaxed problem in this manner. However, a solution that is close to the optimal can usually be obtained very quickly. The term "near-optimal" deserves some explanation. What it means in this context is that the timing constraints may be violated by a marginal amount depending on the level of discretization. The reason is that since the cells are required to lie on discrete locations, there may not be a solution where the constraints are satisfied exactly – there are only a discrete number of possibilities for the left-hand side of each constraint. The amount of error in the constraints is usually extremely small and the larger the problem size, the smaller the granularity in the left-hand side and the smaller the violation. Note that the constraints are not always violated. They may be satisfied by an equally marginal amount (i.e., the error due to discretization could go either way). In practice, the observed violation was of the order of 1-2%.

Note that any solution obtained is locally near-optimal for a region only with respect to the connections outside the region, and does not guarantee to minimize the cost of connections within the region. (The timing constraints however, do represent the correct behavior inside and outside the region). The problem could have been formulated as a Quadratic Assignment to handle the connections within the region properly. However, the large run time of Quadratic Assignment makes it impractical for regions with even a moderate number of cells. For regions with fewer cells, the effect of interconnection within the region is small, and by repeating the Linear Assignment a few times an improved solution can be obtained. A further improvement in wirelength and timing can be obtained

by allowing cells to migrate outside their region. This is achieved by shifting the regions in x and y directions by half the region size at alternate iterations as shown in Figure 5.6 and repeating the process until the improvement is small. Note that the absolute valued delay

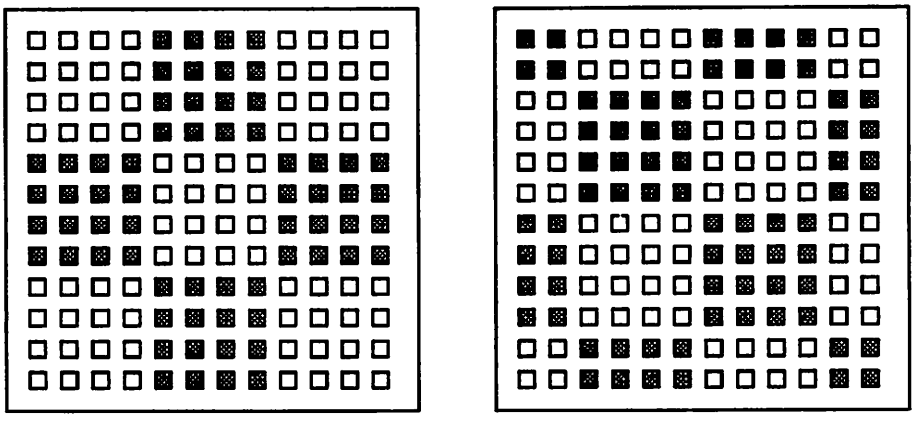


Figure 5.6: Regions of optimization are shifted at alternate iterations

equations can be dealt with in a manner similar to the continuous case, i.e., by ensuring that the equations are always written with the correct signs. The flow of the slot assignment algorithm is shown in Figure 5.7.

1. Divide the chip into a number of regions containing a few cells.
2. Start with an initial value of $\nu = \nu^{(0)}$.
3. Solve LA for the current value of ν.
4. Update the set of critical paths and ν according to $\nu^{(k+1)} = \max(0, \nu^{(k)} + (\mathbf{A}\mathbf{w} - \mathbf{c}))$.
5. Shift the regions of optimization. (See Figure 5.6)
6. Repeat steps 2-5 until the constraints are satisfied to a desired accuracy.

Figure 5.7: The Slot Assignment Algorithm

5.4.2 Some Comments on the Formulation

Experiments proved the method of constrained assignment to be extremely effective in reducing the wirelength while satisfying the timing constraints. The ability to use any assignment cost, regardless of the mathematical properties of the cost function makes it possible to include routability and congestion factors. A question that may arise in the mind of the reader at this time is: Why not perform assignment alone and drop the continuous space optimization altogether?

The reason is that the discrete constrained assignment is a local optimization. Linear Assignment is $O(N^2)$ and it would be prohibitively expensive to perform it on the entire chip. Additionally, since the assignment formulation does not model interconnection costs within the region, it would not work well for large sized regions. It works best for small regions and does local cell placement effectively. Thus, the first phase provides an excellent initial solution for the second phase.

5.5 Input/Output Pad Assignment

Further improvements in delay and wirelength can be obtained by reassigning the input and output pad locations on the boundary of the chip. This assignment can be performed by using the Linear Assignment technique described in the previous section to the inputs and output cells. However, the problem may be solved in a way that makes constrained assignment unnecessary. The idea is to keep the core of the chip (cells within the boundary) fixed during this procedure. First, the primary outputs are assigned to slots on the boundary, setting the cost of a slot to ∞ if assigning the output pad to that slot violates a timing constraint. If the slot is feasible, the cost is the wirelength of the nets attaching to that pad. Following this, the primary inputs are assigned to the remaining slots in a similar manner. This procedure may be iterated a few times to improve the solution obtained. Recently, Chaudhary and others [Chaudhary 91] have developed successful methods for pad placement which could be used in place of the basic procedure described here.

5.6 The Complete Algorithm

The flow of the algorithm is shown in Figure 5.8. The entire algorithm may be repeated several times if necessary.

1. Determine number of levels of spread based on chip size/topology
2. Apply the algorithm of Figure 5.4
3. Apply the Algorithm of Figure 5.7
4. Optionally, perform input and output pad assignment on the chip boundary

Figure 5.8: The complete placement algorithm

5.7 Practical Considerations

So far, the analysis has assumed that all the cells are of a similar size. Most cell-based ICs (standard cell, gate array and sea-of-gates) are composed of cells whose sizes vary in both the x and y dimensions. It is possible to accomodate cells of different sizes during the assignment phase of the algorithm. To illustrate the flexibility of including a variety of constraints in the algorithm, let us consider the specific example of placing standard cells. Standard cells are characterized by their uniform height but non-uniform width. If used unmodified, the algorithm of Figure 5.8 will result in a placement into rows of cells with different row widths as shown in Figure 5.9. This results in considerable "dead area" or wasted space. The problem can be solved by adding a row width constraint on each row. Let n_r be the number of rows and r_i be the width of row i. Let v_i represent the width of cell m_i.

The constraints to be added are:

$$|r_i - \overline{r}| = 0, \ i = i, \ldots, n_r \tag{5.21}$$

where $\overline{r} = \frac{\sum_{i=1}^{M} v_i}{n_r}$ is the average row width. The current width of a row is the sum of all the widths of all cells currently in that row. These constraints may be dealt with in a similar manner to the timing constraints during the discrete assignment phase of the algorithm. A multiplier μ_i is associated with each constraint (or conceptually, with each row). These multipliers can be transferred to slots, i.e., the multiplier for a slot s_j is denoted by μ_j and is the multiplier for the row in which the slot is located. The modified Lagrangian can be

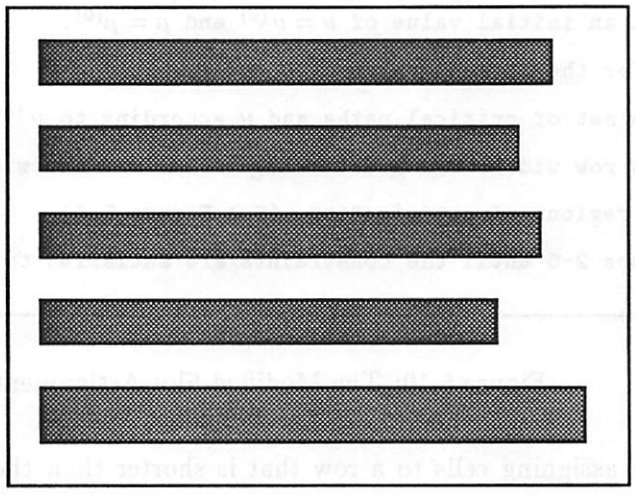


Figure 5.9: Uneven row widths in standard cells results in dead area

written as:

$$
\begin{array}{lll}
\min & \sum_{ij}[\mathbf{C}_{ij} + \sum_{k=1}^{p} \nu_k(\mathbf{A}_{ki}\mathbf{X}_j + \mathbf{A}_{k,i+M}\mathbf{Y}_j) + \mu_j\mathbf{R_{ij}}]z_{ij} & LA \\
\text{subject to} & & \\
\sum_{j=1}^{N} z_{ij} & = 1, & i = 1,\ldots,M \\
\sum_{i=1}^{M} z_{ij} & \leq 1, & j = 1,\ldots,N
\end{array}
\tag{5.22}
$$

where $\mathbf{R}_{ij} = v_i$, i.e., the width of cell m_i. Conceptually, to the wirelength and timing cost of assigning a cell to a slot, one must add the product of the multiplier of the row in which the slot is located and the width of cell. Intuitively, this tends to have the following effect:

- Rows that exceed the target width have a positive value of μ_i and cells are penalized for being assigned to that row.
- Rows that are shorter than the target width have a negative value of μ_i and the cost of assigning cells to those rows is reduced.
- When assigning cells to a row that exceeds the requirement, a wider cell is penalized more than a narrower one.

1. Divide the chip into a number of regions containing a few cells.
2. Start with an initial value of $\nu = \nu^{(0)}$ and $\mu = \mu^{(0)}$.
3. Solve LA for the current value of ν and μ.
4. Update the set of critical paths and ν according to $\nu^{(k+1)} = \max(0, \nu^{(k)} + (\mathbf{A}\mathbf{w} - \mathbf{c}))$.
5. Update the row widths and μ according to the new row widths.
6. Shift the regions of optimization. (See Figure 5.6)
7. Repeat steps 2-6 until the constraints are satisfied to a desired accuracy.

Figure 5.10: The Modified Slot Assignment Algorithm

- When assigning cells to a row that is shorter than the requirement, a wider cell is preferred over a narrower one.

Hence, the mathematical theory has a intuitive and natural interpretation behind it. The modified assignment algorithm is shown in Figure 5.10.

5.8 Conclusions

The techniques in this chapter brought the search for an effective performance-driven placement algorithm to a close. Lagrangian Relaxation offers a powerful method for controlling the tradeoff between the system cycle time and wirelength. Armed with this tool and assisted by some insight into the problem structure, several theoretical and practical issues are resolved in this chapter. Firstly, the techniques are capable of solving extremely large problems, and as experimentation showed, the problems could be solved in a short time. Secondly, the elusive slot constraints are effectively integrated into the solution technique. The next chapter discusses the results of implementing these techniques in a software package for performance-driven placement called RITUAL.

Chapter 6

Experimental Results

This chapter describes the results of a series of experiments conducted on the algorithms described in this work. The experiments culminated in a software system for performance-driven placement of large-scale ICs called RITUAL.

6.1 The Primal and Dual Algorithms

A program that implements the primal and dual algorithms for the solution of GP has been developed. It uses complete LU factorization [Golub 89] at every iteration to solve the linear system rather than complex and efficient schemes ([Gill 89]) for updating the factors. To illustrate the speed-up in computation time obtained even with a simplified implementation, the program was tested on a 210 cell industrial example that is part of the control logic for a 4-bit microprocessor and a 1418 cell gate-array. The results are compared against an industrial Quadratic Programming (QP) package and the published results of Jackson in [Jackson 89]. For the primal algorithm, an initial feasible point was found using the Simplex method. The initial feasible solution took 400 sec for ex1 and 9 hours for ex2 using Phase I of the Simplex method. (Note that the dual algorithm does not require an initial feasible solution.) In the 210 cell example the speed-up obtained by the dual algorithm over standard QP is a factor of 10, while in the 1418 cell example the speed-up is substantial. After 10 hours of running, the standard QP package had not found the optimal solution for the 1418 cell array and it was terminated, while the dual algorithm found the optimal solution in about 7 minutes of CPU time. The increase in speed over the method of [Jackson 89] is about 20 times.

Ex	cells	nets	nets on CP	Primal	Dual	QP
ex1	210	170	59	430s	8s	850s
ex2	1418	1161	133	9hrs	7min	>10hrs

Table 6.1: Results of a prototype implementation of the primal and dual methods

Example	%improvement in wire delay	%increase in Quad. wirelen
ex1	50%	3%
ex2	5.5%	0.07%

Table 6.2: Improvements in quadratic wirelength

The improvement in wire delay is 50% for the 210 cell example with only 3% increase in quadratic wirelength over the unconstrained placement while for the 1418 cell example, the dual and primal methods were able to obtain a 5.5% improvement in wire delay with an increase of only 0.07% in quadratic wirelength. The results are shown in Table 6.1. Note that the primal and the dual algorithm make the same improvements as expected, since the problem GP has a unique global minimum.

While these increases in speed are substantial, I concluded that they are insufficient to break the 20K cell barrier. The reason is that the matrix factorizations can be conservatively estimated to be $O(M^2)$ (see Chapter 3). Thus, a rough estimate of the time for obtaining a solution of GP using the dual algorithm for a 20K cell problem is 1372 minutes or about 23 hours! Upon considering that a solution of GP does not resolve cell overlaps, it is clear that the primal and dual methods are inadequate. This discouraging conclusion spurred the quest for better methods – those of Chapter 5.

6.2 RITUAL

6.2.1 Overview

RITUAL is a software package that incorporates the techniques of Chapter 5. The inputs to the package are:

- A sequential or combinational circuit in the form of a netlist,

- A library that defines the timing and physical characteristics of the cells,
- A description of the topology of the chip and the layout of the slots and,
- Specifications of the required arrival times at the primary outputs and latches and actual arrival times at the primary inputs.

RITUAL operates in two phases. The first phase uses Lagrangian Relaxation to solve the problem GP, including "spread" or center of mass constraints. The center of mass constraints are derived from the locations of slots on the chip and the topology of the chip. The first phase is applied until a few cells (15-30) remain in each region of the hierarchy. Following this, the second phase of placement is applied. The second phase implements constrained linear assignment in buckets of regions as described in Section 5.4. Finally, if desired, the placement of input and output pads can be performed on the boundary of the chip. RITUAL can be operated in two modes: wirelength and timing. In the wirelength mode, wirelength optimization is performed without regard to the timing constraints, while the timing mode attempts to satisfy the timing requirements while minimizing wirelength. A sequence of placements obtained during the course of RITUAL is shown in Figures 6.1 – 6.5.

6.2.2 Models Used

During the first phase, the quadratic wirelength model of Section 2.3.2 is used. During the second phase, the more accurate single-trunk Steiner tree model of Section 2.3.4 is used.

The timing model used in RITUAL is a first order estimate that neglects the interconnect resistance effects. Provisions are built into the software package to enable the use of a non-linear model if necessary. The timing model takes cell logic function (inverting/non inverting) into account and for each path, it computes the delay times of rising and falling waveforms. During the first phase, the star-connected net delay model of Section 2.4.5 is used and during the second phase, the single-trunk Steiner tree delay model of Section 2.4.5 is used. The matrix solution technique used in RITUAL is the Gauss-Seidel iterative method as described in [Golub 89, page 507].

In order to quantify the results, RITUAL was compared to an industrial quality placement package – TimberWolf (Version 5.6x) [Sechen 88b]. TimberWolf is a sophisticated

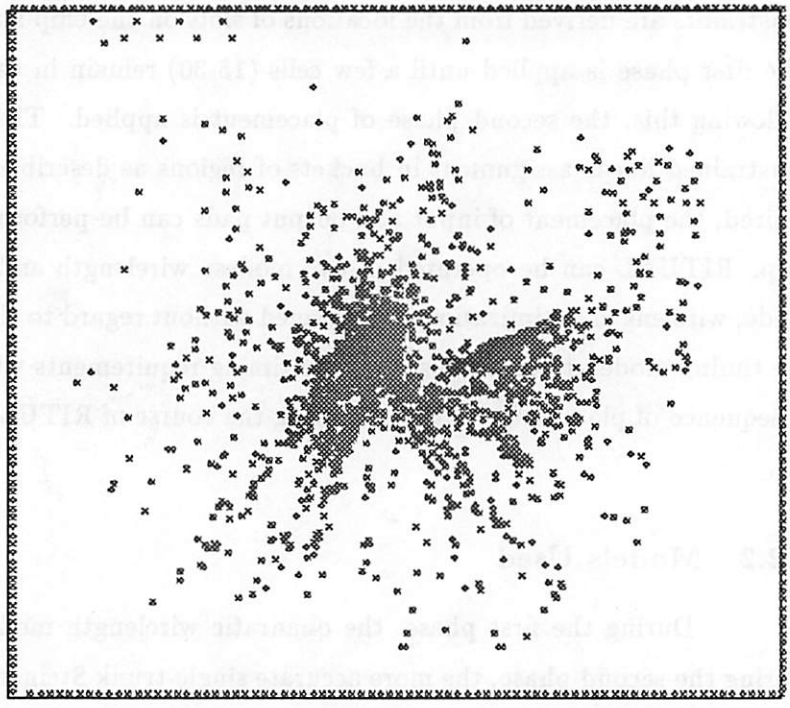


Figure 6.1: Level 0

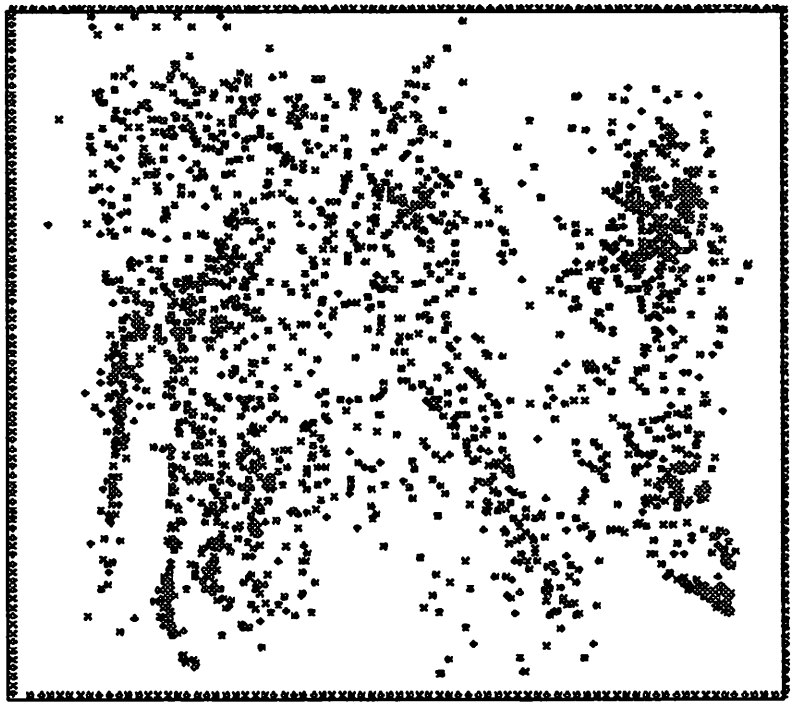


Figure 6.2: Level 1

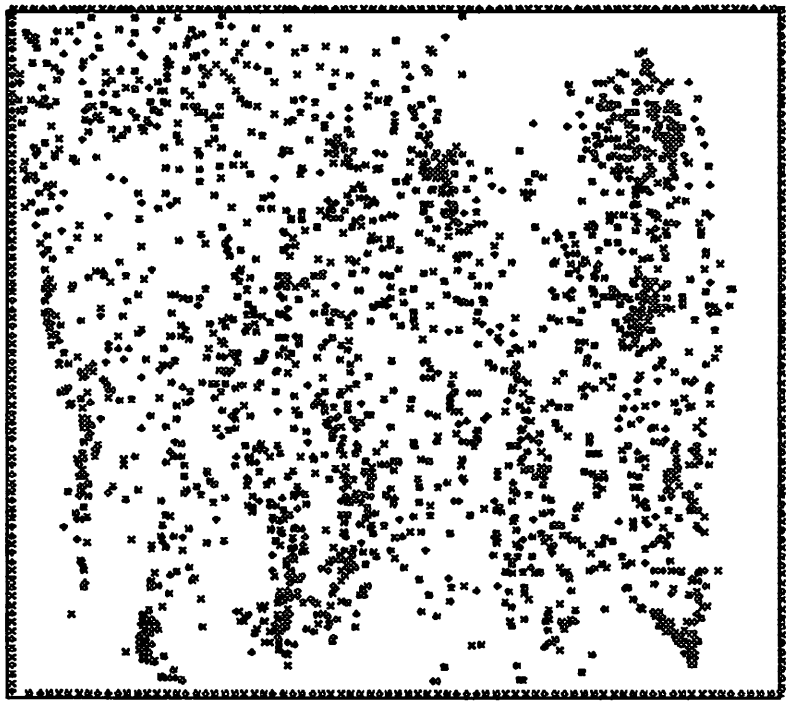


Figure 6.3: Level 2

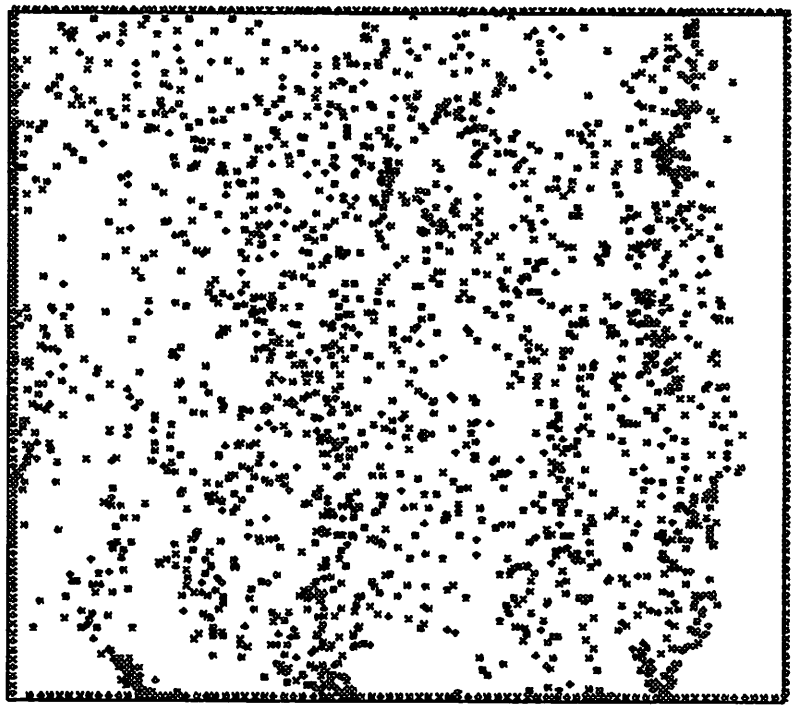


Figure 6.4: Level 3

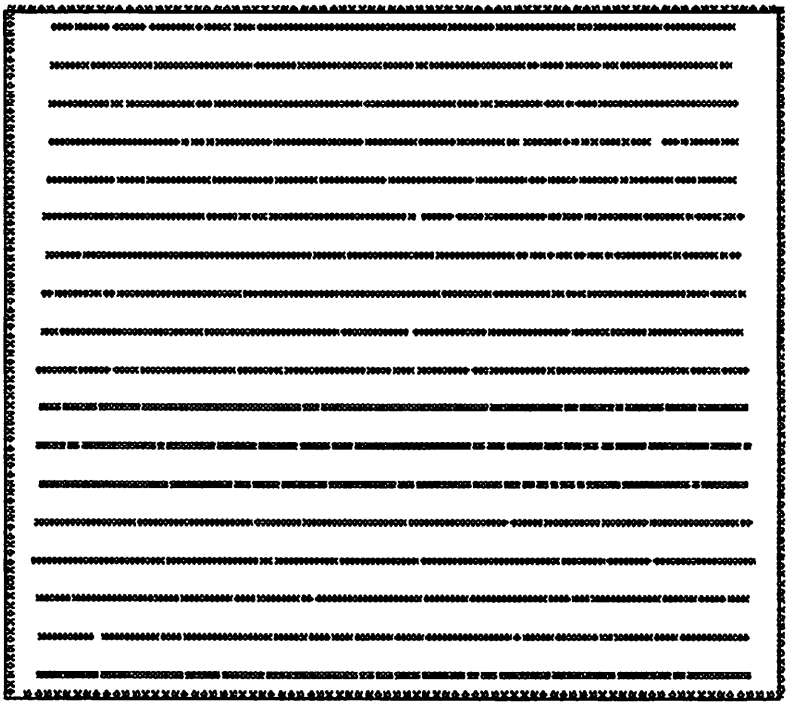


Figure 6.5: Final placement into rows

placement tool that uses simulated annealing for the placement of cell-based ICs and has been proven to yield layouts of excellent quality in the CAD industry.

6.2.3 Results

A popular set of sequential and combinational logic benchmarks from the Microelectronics Research Center of North Carolina [MCN 88] was used to rate the program. The logic functions of the benchmarks were optimized using the systems **misII** [Brayton 87] and **sis** [Sentovich 91] and mapped to the MSU standard-cell library. There were compelling reasons for using the logic benchmarks:

- Detailed timing information was available for the library cells
- The logic information which is essential to model the timing behavior correctly was readily available for the logic benchmarks
- It was possible to experiment with various modes of logic synthesis and investigate the effects of high level optimizations on performance-driven placement
- The benchmarks include combinational and sequential circuits and are accepted in the research community

OCTTOOLS packages [Spickelmier 88] were used to perform the global and detailed routing where possible. Due to limitations on the size of the examples that OCTTOOLS could handle on the available computers, only some of the examples could be completely routed. Standard area-based global and detailed routing tools were used and no net criticality information was passed to the routers. This ensures that any performance improvements are solely due to the placement of the circuits. No IO pad optimization was done and both RITUAL and TimberWolf were given identical IO pad locations.

Table 6.3 compares the two modes of RITUAL with TimberWolf. The column labeled WL denotes the single-trunk Steiner tree wirelength normalized to the wirelength mode of RITUAL (i.e., taking the wirelength mode of RITUAL as 1.00). The delay numbers are in nano-seconds and the delay for the longest path is shown (i.e., ***all paths*** on the chip have a delay less than the number shown in the delay column). The delays are based on a single-trunk Steiner tree estimation of net sizes. The wirelength mode of RITUAL is comparable to TimberWolf and the timing mode consistently improves on the delay in every

example. The average delay improvement is 13% over the RITUAL wirelength mode as well as over TimberWolf at the average cost of 7% in wirelength over RITUAL in wirelength mode and 3% over TimberWolf. Note that the improvement in the wire delay component is of the order of 20-40% because for these examples, the cell delays account for about 50% of the total delay.

The timing improvements shown in Table 6.3 are not the best possible improvements. Even further delay improvements are possible on most examples at a greater expense in wirelength. The results shown are what I believe represent a fair tradeoff between wirelength and delay. Any further increase in wirelength would possibly offset the gains made in delay.

Figure 6.6 compares the CPU time of RITUAL in wirelength and timing mode to TimberWolf5.6. In the figure, second order "best-fit" curves are plotted. The run time of RITUAL increases linearly with example size while TimberWolf shows a superlinear behavior. The speed increase over TimberWolf on the larger examples is about 10-15 times. Figures 6.7 and 6.8 are the placements produced by RITUAL on the example C2670 in the wirelength mode and timing mode respectively. In the figures, the longest path is shown by means of a thick line and the dotted lines represent fanout cells of nets along the longest path. A closer look at the figures will reveal that RITUAL has two effects on the path:

1. It tends to "straighten" the path
2. Cells belonging to nets on the path are brought closer to each other in order to reduce the net capacitance

The display of other paths in the circuit has been supressed for clarity.

6.2.4 Results After Routing

Table 6.4 shows the results for some chips after complete routing using the OCTTOOLS routing packages. The average delay improvement over RITUAL in wirelength mode is 5%, while over TimberWolf, it is about 7%. The chip area increases by only 4% over RITUAL in wirelength mode and 2% over TimberWolf. Why are the results not as substantial as those before routing?

- No special global and detailed routing tools were used

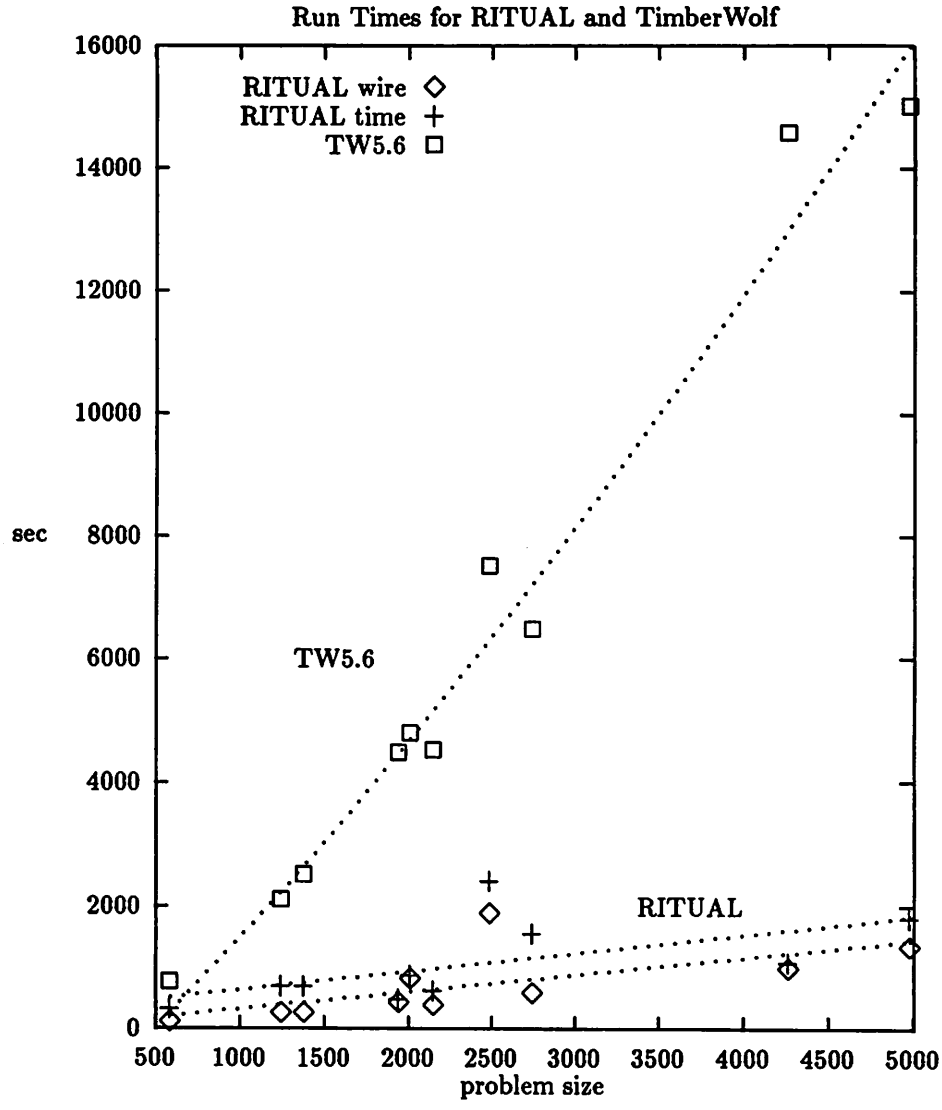


Figure 6.6: Figure showing the run time of RITUAL and TW5.6

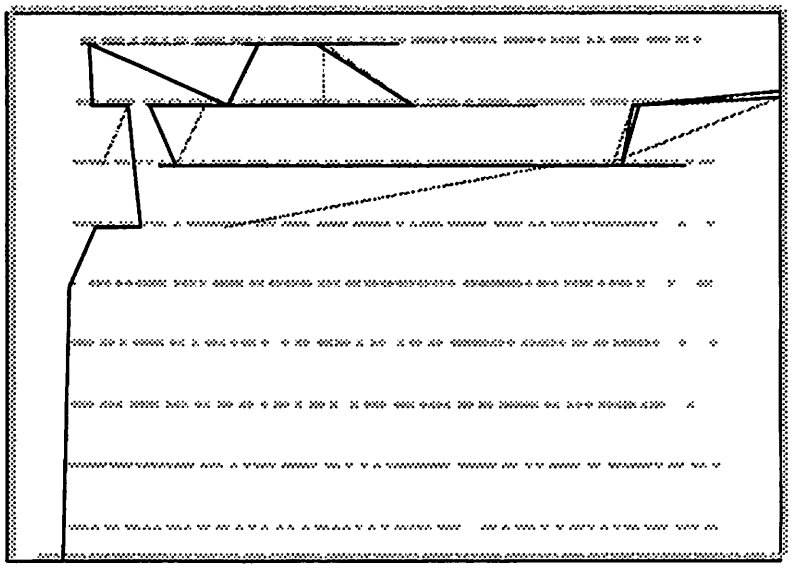


Figure 6.7: C2670 after placement by RITUAL in wirelength mode

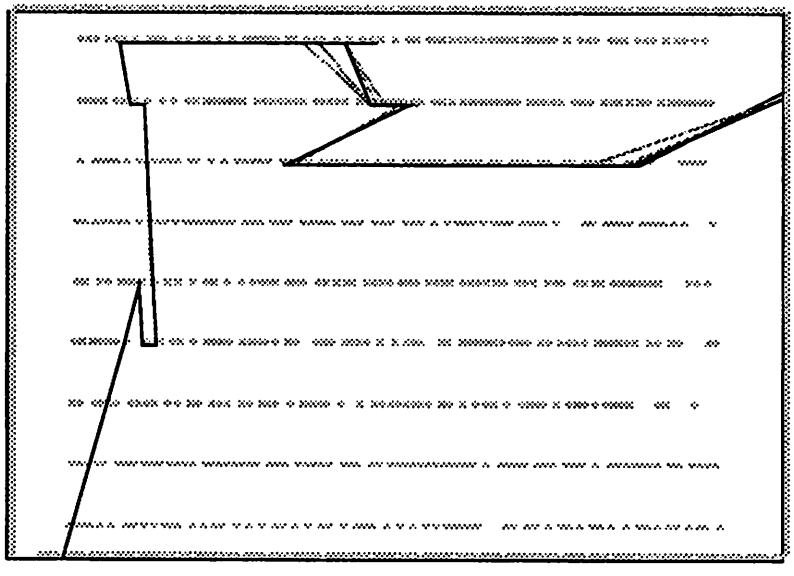


Figure 6.8: C2670 after placement by RITUAL in timing mode

Chip	cells	RITUAL				TW5.6	
		Wire		Time			
		WL	Delay	WL	Delay	WL	Delay
C2670	590	1.00	24.36	1.11	21.70	1.00	24.72
C3540	1254	1.00	47.98	1.08	44.06	1.00	50.39
C5315	1381	1.00	36.26	1.08	31.64	1.02	34.19
C6288	1945	1.00	118.60	1.15	111.50	1.07	124.5
C7552	2150	1.00	52.98	1.06	44.65	1.00	53.51
s9234	2748	1.00	39.80	1.05	31.31	0.94	35.87
s13207	4267	1.00	53.10	1.00	43.37	1.32	51.46
s15850	4981	1.00	57.64	1.00	50.96	0.94	62.40
Average	–	1.00	1.00	1.07	0.87	1.04	1.00

Table 6.3: Results of RITUAL before routing

Chip	cells	RITUAL				TW5.6	
		Wire		Time			
		Area	Delay	Area	Delay	Area	Delay
C2670	590	1.00	26.80	1.01	25.20	1.06	26.50
C3540	1254	1.00	50.70	1.05	48.50	0.94	50.50
C5315	1381	1.00	37.20	1.06	35.50	0.97	38.10
C6288	1945	1.00	132.70	1.00	130.00	1.00	135.50
C7552	2150	1.00	55.00	1.00	52.10	1.07	58.70
Average	–	1.00	1.00	1.04	0.95	1.01	1.02

Table 6.4: Results on some chips after routing

- The OCTTOOLS packages ([Rudell 88]) tend to move the IO pads to improve the routability and this changes the timing behavior of the resulting circuit
- The examples are the smaller ones. Large examples have a greater improvement before routing and are expected to retain the improvement after routing

6.3 Conclusions

The experiments demonstrate that it is possible to make significant improvements in delay at little or no expense in chip area by modifying the placement alone!. The improvements are expected to be even better if timing-driven routing tools are used. The

CPU time for obtaining these results is small. Even the largest example with about 5000 cells takes 15 minutes using RITUAL in wirelength mode and 30 minutes in timing mode. The memory requirements of RITUAL are linear in the size of the circuit and the CPU time also appears to be linear as shown in Figure 6.6. The wirelength quality of the results are comparable or better than those obtained using simulated annealing and the delay results are consistently better. These results can be obtained in a fraction of the time taken by simulated annealing.

Chapter 7

Skew Optimization

7.1 Introduction

The key determinant of performance is the *cycle time* of a system. Keeping all other factors constant, a decrease in the cycle time results in an increase in the throughput, i.e., a decrease in the time taken to perform a computation. This chapter presents a practical approach for optimizing the clock period subject to physical and electrical constraints. Given a set of interconnected logic blocks that comprise a system, block delays, interconnect delays between blocks, latch delays, locations of the blocks on some package and locations of synchronizing elements (latches) that define the temporal boundaries between blocks, the formulation allows the designer to determine the smallest clock period and simultaneously constructs a clock tree. The clock tree is hierarchical and contains programmable delay elements.

In reality, the logic block delays are random variables whose distributions can be characterized using historic data. The commonly used approach to deal with variable delays is to increase the cycle time by some fixed percentage according to some rules of thumb or perform worst-case analysis while constructing a clock tree [Mijuskovic 87, Boon 89, Friedman 86, Kung 82, Fisher 82, Dhar 84, Gura 87, Fishburn 81]. However, this may not be the best approach in terms of the reliability-cycle time tradeoff and in some sense, may be overly conservative. For a competitive design that pushes technology to the limits, an improvement in the performance can be optimal often at the risk of system failure. The formulation can take these variations into account and determine the smallest clock period that satisfies a specified reliability value for the system. It is also capable of determining

the most reliable system configuration for a given clock period. In addition, it is possible to add a variety of practical constraints for example, constraints that limit the number of clock pins and still solve the problem efficiently.

Typical systems which are amenable to such optimizations involve multi-chip modules and high-performance packaging configurations such as thermal-conduction modules. The optimization techinques discussed here can be applied hierarchically at various levels, i.e., at the chip level, within packages comprised of chips and within cards comprised of packages.

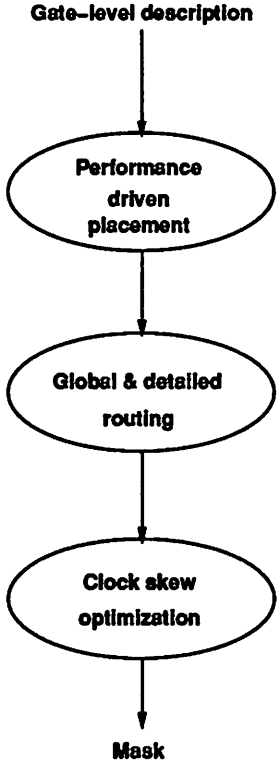


Figure 7.1: Figure showing the role of clock skew optimization

7.2 Skew Optimization

7.2.1 Physical Model

For explanation purposes, consider the physical model of a multi-chip module; the techniques can be applied to other hierarchical systems. The system is assumed to be composed of interconnected chips. The input specification consists of *chip description netlists* that specify for each chip, the netlist internal to the chip, delays though the cells internal to the chip, delays in the interconnect between cells, synchronizing element setup and hold times and wiring delays between chip clock pins and each synchronizing element. In addition, the interconnections across chips and their associated delays are also described by means of inter-chip netlists. An example of a multi-chip module is shown in Figure 7.2. For the purposes of simplicity, a single phase clock is assumed. The techniques are easily extensible to the case of multi-phase clocking schemes.

7.2.2 Timing Model

A model for detecting clock hazards was presented in [Kogge 81] and a model for optimizing skew was presented in [Fishburn 81]. The synchronous digital system used in these models consists of blocks of combinational elements separated by edge-triggered latches. The set of latches is denoted by $L = \{l_1, \ldots, l_n\}$. Let T denote the clock period. The circuit can be modeled as a *synchronous communication graph* G containing n vertices, one for each synchronizing element (henceforth also called a latch). There is a directed edge between vertex i and j if at some time during the clock cycle, there is a combinational logic path from latch l_i to l_j. Each edge has weights D_{ij} and d_{ij} which denote the largest and smallest combinational logic delays between latch l_i and l_j respectively. The setup and hold times of a latch, l_i, are denoted by $t_{SETUP\,i}$ and $t_{HOLD\,i}$ respectively. The programmable delay from the clock source to the latch l_i is denoted by δ_i. This delay is known as the *clock offset* for that latch.

Such an abstract communication graph may be easily constructed by a *timing analysis* on the physical model presented in the previous section, tracing the longest and shortest paths in a depth first manner.

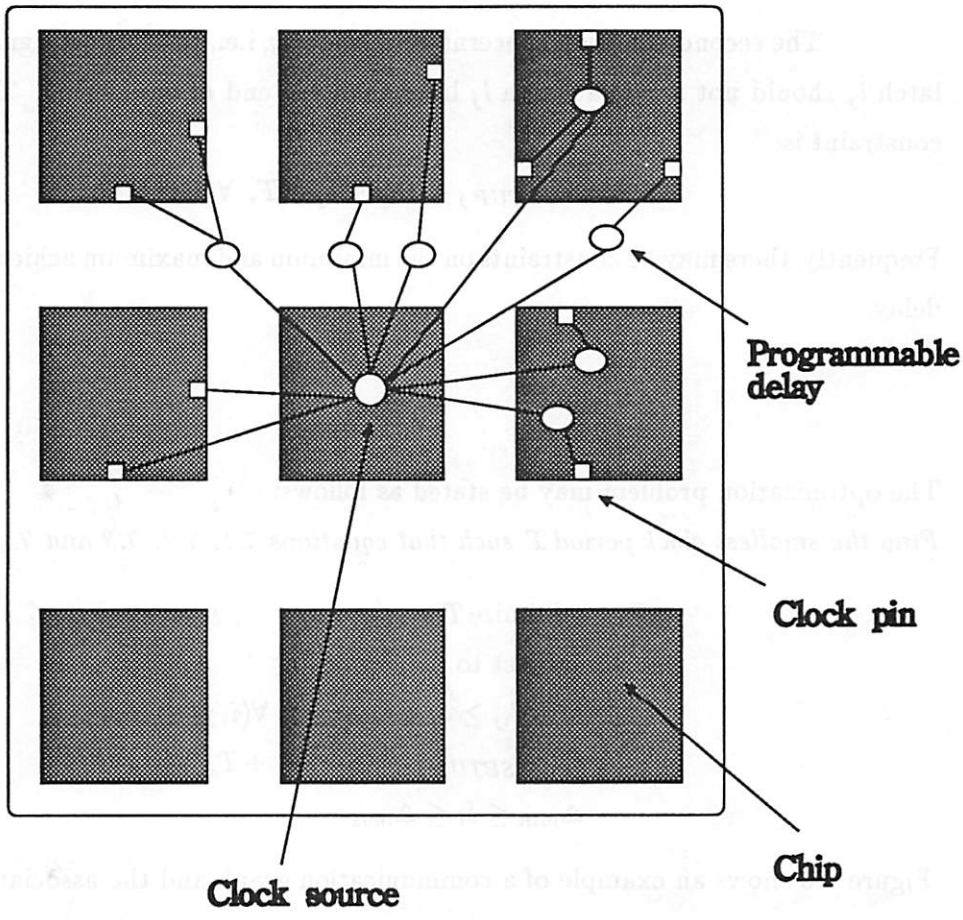


Figure 7.2: An example of a clock distribution network on a MCM

7.2.3 Formulation

The optimization problem can now be formulated. The first constraint concerns *double clocking.* It states that the signal from latch l_i to latch l_j through the fastest path should not race through the circuit before the end of one clock period. The equation can be written as follows:

$$\delta_i + d_{ij} \geq \delta_j + t_{HOLD\,j},\ \forall (i,j) \in G. \tag{7.1}$$

The second equation concerns *zero clocking*, i.e., the slowest signal from latch l_i to latch l_j should not arrive at latch l_j later than the end of one period. The corresponding constraint is:

$$\delta_i + t_{SETUP\,j} + D_{ij} \leq \delta_j + T,\ \forall (i,j) \in G. \tag{7.2}$$

Frequently, there may be constraints on the minimum and maximum achievable programmable delay.

$$\delta_i \geq \Delta_{min} \tag{7.3}$$

$$\delta_i \leq \Delta_{max} \tag{7.4}$$

The optimization problem may be stated as follows:
Find the smallest clock period T such that equations 7.1, 7.2, 7.3 and 7.4 are satisfied.

$$\begin{aligned} \mathcal{P}: \quad & \text{minimize } T \\ & \text{subject to} \\ & \delta_i + d_{ij} \geq \delta_j + t_{HOLD\,j},\ \forall (i,j) \in G \\ & \delta_i + t_{SETUP\,j} + D_{ij} \leq \delta_j + T,\ \forall (i,j) \in G \\ & \Delta_{min} \leq \delta_i \leq \Delta_{max} \end{aligned} \tag{7.5}$$

Figure 7.3 shows an example of a communication graph and the associated variables.

7.3 Practical Considerations

The above basic formulation is impractical because of the following issues:

- It requires every latch in the system to have a programmable delay line from the clock source.
- All values of offsets may not be achievable in practice.

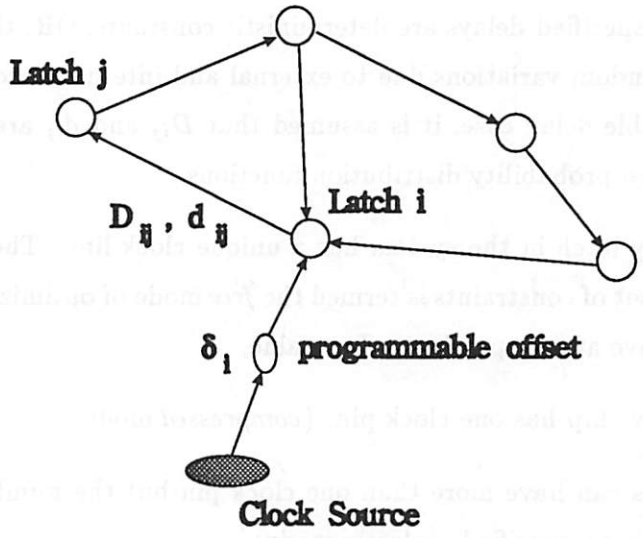


Figure 7.3: A synchronous communication graph

- There may be constraints that limit the total number of different clock offsets, on a per-chip basis and/or across the whole system.
- The specified delays for the cells within chips, inter-cell and inter-chip connections may be subject to random variations among different instances of the multi-chip modules due to manufacturing differences and uncontrollable operating conditions.

In contrast to [Fishburn 81] and previous work, this work presents a new approach to clock skew optimization that considers all of the above constraints. Specifically, by the application of the techniques described in this chapter, cycle time optimization can be performed under one or more of the following constraints:

1. The specified delays are deterministic constants, OR, the specified delays are subject to random variations due to external and internal uncontrollable influences. For the variable delay case, it is assumed that D_{ij} and d_{ij} are random variables with some known probability distribution functions.
2. Every latch in the system has a unique clock line. The optimization problem under this set of constraints is termed the *free* mode of optimization since each latch is "free" to have an independent offset value.
3. Every chip has one clock pin. (*compressed* mode).
4. Chips can have more than one clock pin but the number of clock pins available per chip is prespecified. (*cluster* mode).
5. The total number of different clock offsets in the system is prespecified. Some chips may have a limited number of pins while others may have no restriction. The technique will automatically distribute the total number of available clock offsets while satisfying the requirements of the chips with limited clock pins and minimizing the clock period.
6. A hierarchical clock tree with adjustable offset elements at various levels in the tree may be given. The technique will find assignments for the offsets at all the levels of the tree subject to any of the above constraints. (*2-Ply* mode.)
7. Discrete values of clock offset which are multiples of some basic offset unit may be required. The technique can be applied to this case subject to any of the above constraints, including a hierarchical clock tree. (*integer* mode.)

8. In many systems, copies of a chip are used in several subsystems. The constraint on the *copy chips* is that they must have identical internal clock trees and number of clock pins. The algorithm can satisfy this requirement.

9. Within a chip, because of physical layout reasons, it may not be possible for some latches to have the same value of clock offset. The technique is capable of solving the cycle time optimization problem subject to this constraint.

A significant feature of the approach is that the modes described above are not mutually exclusive, i.e., the constraints can be simultaneously applied.

7.4 Algorithms

In this section, several new algorithms that are used to implement the techniques are presented.

7.4.1 An Efficient Algorithm for Solving $\mathcal{P}$ when Delays are Deterministic

$\mathcal{P}$ may be solved efficiently using graph-based algorithms as follows. A lower bound on T is easily obtained as:

$$T_{min} = \max\{0, \min_{ij}(D_{ij} - \Delta_{max})\}$$

An upper bound on T is:

$$T_{max} = \max_{ij}\{D_{ij}\}$$

The algorithm is:

1. Perform a bisection search on the values of T between T_{max} and T_{min} to find the minimum feasible period. This is possible because if a clock period T_1 is feasible (infeasible) then any clock period $T_2 > T_1$ ($T_2 < T_1$) is also feasible (infeasible).

2. Test each T_i obtained during the bisection search for feasibility using the Bellman-Ford algorithm [Dreyfus 69].

Proposition 7.4.1 *The problem $\mathcal{P}$ can be solved in using $O(\log(T_{max}-T_{min})n^3)$ operations*

Proof. Let $\mathcal{F}$ denote the following problem:
For a given fixed constant T_0, find a feasible solution to

$$\begin{aligned} &\delta_i + d_{ij} \geq \delta_j + t_{HOLD\,j},\ \forall (i,j) \in \mathrm{G} \\ &\delta_i + t_{SETUP\,j} + D_{ij} \leq \delta_j + T_0,\ \forall (i,j) \in \mathrm{G} \\ &\Delta_{min} \leq \delta_i \leq \Delta_{max} \end{aligned} \tag{7.6}$$

Let $\mathcal{F}'$ be the following problem: Find a feasible solution to

$$\begin{aligned} &\delta_i + d_{ij} \geq \delta_j + t_{HOLD\,j},\ \forall (i,j) \in \mathrm{G} \\ &\delta_i + t_{SETUP\,j} + D_{ij} \leq \delta_j + T_0,\ \forall (i,j) \in \mathrm{G} \\ &\delta_i - \delta_j \leq \Delta_{max} - \Delta_{min}, \forall i,j \end{aligned} \tag{7.7}$$

Let $S' = \{\delta_1, \ldots, \delta_n\}$ be any feasible solution to $\mathcal{F}'$. Let

$$\delta_{min} = \min_i \{\delta_i\}$$

Clearly, $\delta_j - \delta_{min} \leq \Delta_{max} - \Delta_{min}, \forall j$. Define $S = \{\delta_1 - \delta_{min}, \ldots, \delta_n - \delta_{min}\}$. It can be easily verified that S satisfies all the constraints of $\mathcal{F}$. Also, any solution to $\mathcal{F}$ trivially satisfies $\mathcal{F}'$. Therefore, $\mathcal{F}$ and $\mathcal{F}'$ are equivalent. Note that for a given fixed clock period T_0, all the inequalities in $\mathcal{F}'$ have the form:

$$\delta_i - \delta_j \leq c_{ij}$$

where c_{ij} are real numbers. A system of inequalities in which all the equations have this form can be checked for feasibility and solved very efficiently in $O(mn)$ time, where m is the number of inequalities and n is the number of variables, using the Bellman-Ford algorithm. Thus, the complexity of the above algorithm is $O(\log(T_{max} - T_{min})n^3)$. Leiserson and Saxe have shown in [Leiserson 88] that even the mixed-integer version of the above form can be solved efficiently. □

7.4.2 Stochastic Delays

For simplicity of explanation, in this and following sections, it is assumed that the hold and setup times t_{HOLD} and t_{SETUP} are included in the delays D_{ij}. In a practical system, the delays D_{ij} are not fixed values for that particular system. There may be

variations in D_{ij} and d_{ij} due to processing, temperature, signal variations and other sources. D_{ij} and d_{ij} are assumed to be random variables with some known probability distribution functions.

The stochastic optimization problem may be stated as:

$$\begin{aligned} \mathcal{P}1: \quad & \text{minimize } T \\ & \text{subject to} \\ & Prob[\textstyle\bigcap_{(i,j)\in G}\{(\delta_i + d_{ij} \geq \delta_j) \bigcap (\delta_i + D_{ij} \leq \delta_j + T)\}] \geq \alpha \\ & \Delta_{min} \leq \delta_i \leq \Delta_{max} \text{ for } i = 1, \ldots, n \end{aligned} \tag{7.8}$$

where $\alpha \in (0,1)$ is the desired reliability level for the system.

For the purposes of discussion, let us introduce additional variables in the problem as follows:

$$\begin{aligned} \mathcal{P}1: \quad & \text{minimize } T \\ & \text{subject to} \\ & Prob[\textstyle\bigcap_{(i,j)\in G}\{(x_{ij} + d_{ij} \geq 0) \bigcap (x_{ij} + D_{ij} \leq T)\}] \geq \alpha \\ & x_{ij} = \delta_i - \delta_j \ \forall (i,j) \in \mathrm{G} \\ & \Delta_{min} \leq \delta_i \leq \Delta_{max} \text{ for } i = 1, \ldots, n \end{aligned} \tag{7.9}$$

7.4.3 Computational Strategies

There are several options available for solving the problem $\mathcal{P}1$. The first is to impose penalties on the violation of the probabilistic constraints. For example, instead of minimizing T subject to the timing constraints, one can minimize the following objective function:

$$T + \{\sum_{i,j} \log[1 - F^1_{ij}(-x_{ij})] + \log[F^2_{ij}(T - x_{ij})]\}^2$$

subject to

$$x_{ij} = \delta_i - \delta_j, \ \forall (i,j) \in G$$

$$\Delta_{min} \leq \delta_i \leq \Delta_{max} \text{ for } i = 1, \ldots, n$$

where F^1_{ij} is the probability distribution function associated with d_{ij} and F^2_{ij} is the distribution function associated with D_{ij}. d_{ij} and D_{ij} are assumed to be independent random

variables. This objective function is nonlinear and may not be easy to optimize, since it is not separable.

Converting the probabilistic constraints to deterministic ones is the second option considered. Since D_{ij} and d_{ij} involve random variables on the same arc, we make the following simplifying assumption:

$$D_{ij} - d_{ij} = \gamma_{ij} + \mu_{ij} D_{ij}$$

where γ_{ij} and μ_{ij} are non-negative constants. As suggested by Prekopa [Prekopa 88], one technique is to convert the chance constraints into equivalent deterministic ones. The deterministic constraints would have the form:

$$\prod Prob[D_{ij} \leq T - x_{ij} \bigcap D_{ij} \leq (T - \gamma_{ij})/\mu_{ij}] \geq \alpha$$

Prekopa shows that the left hand side of this constraint is logconcave under the assumption A: D_{ij} has a logarithmic concave distribution. This can be used to demonstrate that the constraint set is convex. The above constraint could be replaced by:

$$\sum_{(i,j)\in G} \min\{\log(Prob[D_{ij} \leq T - x_{ij}]), \log(Prob[D_{ij} \leq (T - \gamma_{ij})/\mu_{ij}])\} \geq \log \alpha$$

The well known sufficient condition for the concavity of a function of the form:

$$\sum_{i\in I} \log F_i(z_i)$$

where F_i is a distribution function is that there exists $0 \leq p \leq 1$ such that $f_i(x)$, the density function, is nonincreasing for $x \geq x(p_i)$. $x(p_i)$ is the p_i'th fractile. Define the set of functions

$$\wp = \{F(x) | \exists\ 0 \leq p \leq 1, F(x) \text{ is nonincreasing for } x \geq x(p)\}$$

Let

$$p_0 = \max_{i\in I}(p_i)$$

Then, $\sum_{i\in I} \log F_i(z_i)$ is concave for all z_i when

$$\prod_{i\in I} F_i(z_i) \geq p_0$$

For all symmetric unimodal distribution functions $F_i \in \wp$, $p_0 = 0.5$ [Kambo 84]. However, there is no known suitable linearization strategy for dealing with such a constraint.

The actual strategy chosen to efficiently solve the stochastic optimization problem involves maximizing the reliability and is discussed in the next section.

7.4.4 The Solution Approach

The strategy used is to maximize the probability of satisfying all the constraints for a given fixed clock period T_0. After conversion, the optimization problem takes form:

$$\begin{aligned}
&\mathcal{P}2: \text{maximize } \textstyle\sum_{(i,j)\in G} \log(\min(Prob[D_{ij} \le (T_0 - \gamma_{ij})/\mu_{ij}], Prob[D_{ij} \le T_0 - x_{ij}])) \\
&\text{subject to} \\
&x_{ij} = \delta_i - \delta_j \ \forall (i,j) \in G \\
&\Delta_{min} \le \delta_i \le \Delta_{max} \text{ for } i = 1, \ldots, n
\end{aligned} \tag{7.10}$$

Problem $\mathcal{P}2$ is shown to be equivalent to $\mathcal{P}1$. Because of the simple structure of the problem, the maximization problem turns out to be a separable convex optimization problem subject to totally unimodular constraints under the assumption A. Thus, efficient polynomial time algorithms exist for solving such a problem [Hochbaum 89].

Proposition 7.4.2 *$\mathcal{P}1 \Leftrightarrow \mathcal{P}2$ when the distribution functions of the logic delays are increasing in T*

Proof. Instances of $\mathcal{P}1$ will be written as $P_1(\alpha)$ and of $\mathcal{P}2$ as $P_2(T)$. For simplicity of presentation the optimal solutions as written as:

$$P_1(\alpha) = T^*$$

$$P_2(T^*) = \alpha^*$$

Let the realized probability of satisfying the constraints in $\mathcal{P}1$ at T^* be α_1. If $\alpha_1 > \alpha^*$, α^* cannot be optimal for $\mathcal{P}2$, which is a contradiction. If $\alpha_1 < \alpha^*$, we can reduce T^* in $\mathcal{P}2$ until $P_2(T^*) = \alpha_1$ under the assumption that the distribution functions are continuous. This leads to a contradiction, because in this case T^* cannot be optimal for $\mathcal{P}1$. So, $\alpha_1 = \alpha^*$.

Can

$$P_1(\alpha^*) = T^*$$

$$P_2(T) = \alpha^*, \text{ and } T \neq T^*?$$

If $T < T^*$ we have a contradiction. If $T > T^*$, let (x^1_{ij}) be the solution to $\mathcal{P}1$. Then, if we substitute the values (x^1_{ij}) into the objective function of $\mathcal{P}2$, we get

$$\prod_{(i,j)\in G} Prob[D_{ij} \le \min(T - x^1_{ij}, (T - \gamma_{ij})/\mu_{ij})] > \alpha^*$$

since the distribution functions are non-decreasing and continuous. This leads to a contradiction, unless the reliability achievable has reached its upper limit. So in the region of interest $T = T^*$. □

Proposition 7.4.3 *The constraint set of $\mathcal{P}2$ is totally unimodular*

Proof. Consider

$$x_{ij} - \delta_i + \delta_j = 0, \ \forall (i,j) \in G$$

The submatrix of the constraints corresponding to all the x_{ij}'s is an identity matrix. The columns corresponding to δ_i are that of a node-arc incidence matrix. So the constraint set is totally unimodular (see [Lawler 76]). □

Now, we have two algorithms for solving the cycle time optimization problem:

- Graph based approach for deterministic delays. (Method A)
- Stochastic optimization approach for non-deterministic delays. (Method B)

7.4.5 The Clustering Algorithm

This clustering algorithm finds natural groupings of latches within a chip in a manner that attempts to minimize the increase in the cycle time as a result of the grouping. The algorithm may be applied iteratively to effect a reduction in the number of distinct clock lines that are required for each chip. Note that global paths throughout the system are considered simultaneously when solving the optimization problem. The pseudo-code for the clustering algorithm for the case when every chip has a limited number of pins is as follows:

1. Solve the optimization problem using Method A or B to determine the values of offsets for each individual synchonizing element in the system
2. For each chip, group latches that have the same value of clock offset
3. repeat
   (a) For each chip that exceeds the pin requirements

i. Find a pair of latches (or pair of groups of latches) whose grouping affects the *estimated* cycle time the least. Let δ_1 and δ_2 be the offsets of the two latches. An estimate of the effect on the cycle time as a result of the grouping may be found by examining the edges incident to both the latches and computing the cycle time resulting from assigning the offset δ_1 to both latches and the cycle time resulting from assigning the offset δ_2 to both latches and taking the smallest of the two cycle times.

ii. Assign a common clock offset variable to all latches in a group

(b) Solve the optimization problem in the reduced number of variables

(c) Until the pin constraints are satisfied

The algorithm terminates because at every iteration, the number of distinct clock offsets per chip is reduced by at least one. In the case when a total number of allowable distinct clock offsets is given and some chips have a limited number of clock pins, the following algorithm is applied:

1. Solve the optimization problem (using Method A if the delays are deterministic and Method B if they vary) to determine the values of offsets for each individual synchonizing element in the system

2. For each chip, group latches that have the same value of clock offset

3. Let N_c be the total number of chips. Let M be the total number of available clock offsets. Let p_i denote the requirement on the number of pins for chip i. If a chip j does not have any specifications, $p_j = 0$. Let M_a be the total number of offsets actually available, i.e., $M_a = M - \sum_{i=0}^{N_c} p_i$.

4. repeat

   (a) For each chip i such that $p_i > 0$ and i exceeds the pin requirements

      i. Find a pair of latches (or groups) whose grouping affects the *estimated* cycle time the least. Let δ_1 and δ_2 be the offsets of the two latches. An estimate of the effect on the cycle time as a result of the grouping may be found be examining the edges incident to both the latches and computing the cycle time resulting from assigning the offset δ_1 to both latches and the cycle time

resulting from assigning the offset δ_2 to both latches and taking the smallest of the two cycle times.

ii. Assign a common clock offset variable to all latches in a group

(b) For each chip i such that $p_i = 0$

i. Define the target number of pins to be allocated to chip i as t_i. Define r_i as the number of distinct clock offsets actually required by chip i after solving the current problem. Let $S_a = \sum_{i|p_i=0} r_i$

ii. $t_i = \lfloor \frac{r_i}{S_a} M_a \rfloor$

iii. If r_i exceeds t_i find a pair of latches (or groups) in chip i whose grouping affects the estimated clock period the least and assign a common offset variable to the group

(c) Solve the problem in the reduced number of variables

(d) Until the pin constraints are satisfied

7.4.6 Discrete Clock Offsets

In the case when the clock offsets are restricted to be integer multiples of a unit offset δ, we can reformulate the problem as follows:

$$\begin{aligned} \mathcal{D}: \text{minimize } T \quad & \text{subject to} \\ & n_i\delta + d_{ij} \geq n_j\delta + t_{HOLD\,j}, \ \forall (i,j) \in \mathrm{G} \\ & n_i\delta + t_{SETUP\,j} + D_{ij} \leq n_j\delta + T, \ \forall (i,j) \in \mathrm{G} \\ & n_{min} \leq n_i \leq n_{max} \text{ for } i = 1, \ldots, n \end{aligned} \tag{7.11}$$

The optimization problem $\mathcal{D}$ is a mixed integer-linear program. It may be solved efficiently in the deterministic case by using the Leiserson-Saxe algorithm [Leiserson 88] instead of the Bellman-Ford algorithm. For the stochastic case, the problem is NP-complete and approach taken is to integerize the offsets to the nearest integer value.

7.4.7 Hierarchical Clock Tree

A hierarchical clock tree can be modeled with multiple levels of offset programmability by using linear equations similar to the ones in $\mathcal{P}$. The technique is illustrated for the simple case of a two-level tree in which each chip has a main programmable offset element

and the latches have a second level offset element. Extensions to other topologies are fairly straightforward and for the sake of simplicity, will not be discussed here. Let $c(j)$ denote the chip to which latch j belongs.

$$\begin{aligned}
\mathcal{H}: \text{minimize } T \quad & \text{subject to} \\
& s_{c(i)} + \delta_i + d_{ij} \geq \delta_j + s_{c(j)} + t_{HOLD\,j},\ \forall (i,j) \in \mathrm{G} \\
& s_{c(i)} \delta_i + t_{SETUP\,j} + D_{ij} \leq \delta_j + s_{c(j)} + T,\ \forall (i,j) \in \mathrm{G} \\
& \Delta_{min} \leq \delta_i \leq \Delta_{max} \text{ for } i = 1, \ldots, n \\
& S_{min} \leq s_k \leq S_{max}\ \forall k
\end{aligned} \tag{7.12}$$

The variables s_k denote the offset variables for each chip k. The problem may be reduced to one with the similar unimodular structure of $\mathcal{P}$ by a simple variable substitution. Let $v_i = s_{c(i)} + \delta_i$. Making the substitution, the problem becomes:

$$\begin{aligned}
\mathcal{H}1: \text{minimize } T \quad & \text{subject to} \\
& v_i + d_{ij} \geq v_j + t_{HOLD\,j},\ \forall (i,j) \in \mathrm{G} \\
& v_i + t_{SETUP\,j} + D_{ij} \leq v_j + T,\ \forall (i,j) \in \mathrm{G} \\
& \Delta_{min} \leq \delta_i \leq \Delta_{max} \text{ for } i = 1, \ldots, n \\
& S_{min} \leq s_k \leq S_{max}\ \forall k \\
& v_i = s_{c(i)} + \delta_i
\end{aligned} \tag{7.13}$$

This formulation has unimodular constraints but unfortunately does not possess the same structure of $\mathcal{P}$. The technique of [Hochbaum 89] can be used to solve the problem for both the deterministic and the stochastic problem.

7.4.8 Copy Chips

A chip of one type may be used at several places in a system. The constraint on such chips is that the internal clock offsets of such chips must be identical (for manufacturability). Constraints such as these can be easily incorporated in the above framework by introducing variables only for one chip and using the same variables for all the other copies. Thus, each copy generates a new set of constraints using a common set of offset variables.

7.5 Experimental Results

7.5.1 Deterministic Case

The algorithms were used to optimize a future proposed machine design for IBM Poughkeepsie. Figures 7.4 and 7.5 show the clustering technique where 8 iterations were used to satisfy constraints on the total number of clock offset pins in the system. Clock period units are in pico seconds. Offsets are limited to a maximum of half the longest path delay in the system. The figures show at each iteration the tradeoff between the clock period and the number of different clock offset pins. Without performing the optimization, the clock period is 23ns. In Figure 7.4, the optimized clock period varies from 17.6ns to 20ns depending on the pin constraints. Even in this tightly constrained problem, there is a minimum improvement of 13%.

Figure 7.6 shows the various modes of operation of the algorithm. (A) represents the unoptimized clock period (23ns). (B) is the clock period when each chip is constrained to have one clock pin and the offsets are integer (20.48ns). (C) represents the case in which each chip has one clock pin and the offsets are continuous (20ns). (D) is the integer clustered mode, in which each chip can have more than one clock pin but the offsets are discrete (18.8ns). (E) is the continuous clustered mode (18.2ns). (F) is the integer free mode, in which each latch in the system is allowed to have a discrete offset (18.0ns). (G) is the free continuous mode in which every latch can have a continuous offset. (H) is the clustered mode with a two-level tree and continuous offsets. (I) is the clustered two-level tree with discrete offsets. In the two-level tree, each chip had a single main offset and groups within a chip had an individual offset. Both the main and the group offsets were limited to 25% of the longest path delay in the system. In Figure 7.5 the maximum possible speedup in the clock is 23% and the minimum speedup is 12%. The case of 12% speedup occurs for the integer mode with one clock pin per chip which represents a very realistic and practical design point.

7.5.2 Stochastic Case

A prototype program was implemented to obtain tradeoff curves between T and reliability and Δ_{max} and reliability and illustrate the feasibility and applicability of the ideas. Typical analysis of a digital system with about 100 nodes and 375 edges takes a few

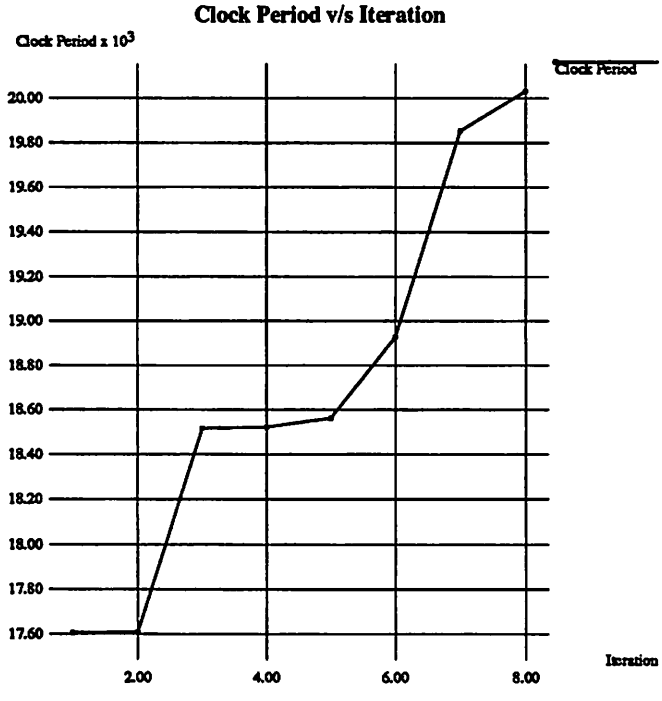


Figure 7.4: Clock Period v/s Iteration

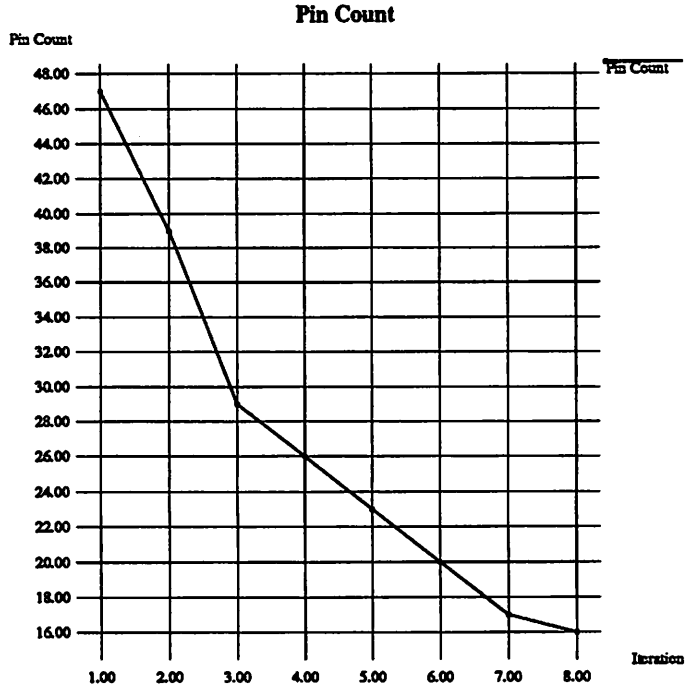


Figure 7.5: Pin Count v/s Iteration

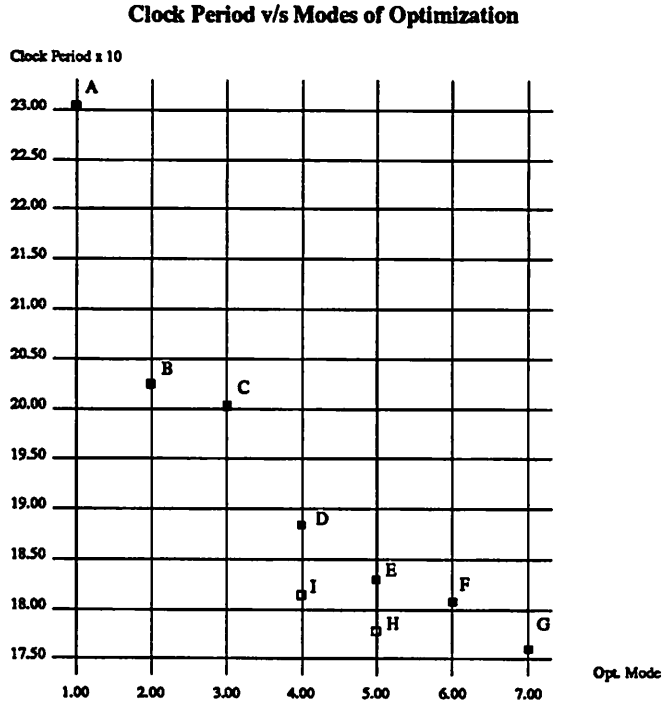


Figure 7.6: Various Modes of Optimization

seconds of CPU time. The program was tested on an industrial example and tradeoff curves were obtained for some sample distribution functions for D_{ij}. For a normal distribution, the curves are shown in Figures 7.7 and 7.8. The curves for a triangular distribution are shown in Figures 7.9 and 7.10. (Note: there may be small deviations from concavity in the plots due to the limited numerical resolution of the plotter).

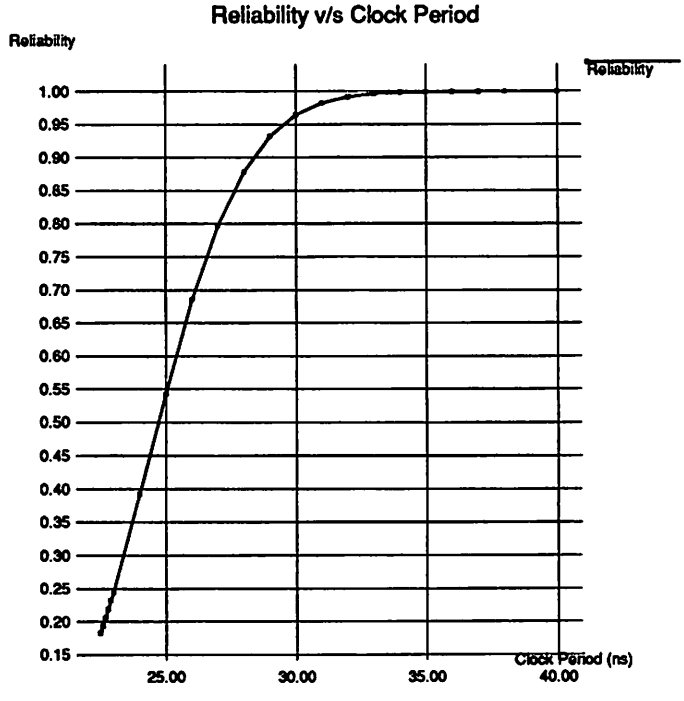


Figure 7.7: Reliability v/s Clock Period, normal distribution

7.6 Conclusions

In this chapter methods have been developed using graph based and linear programming techniques to optimally assign clock tree delay elements in a manner which minimizes the system clock period. The method considers all paths simultaneously and the

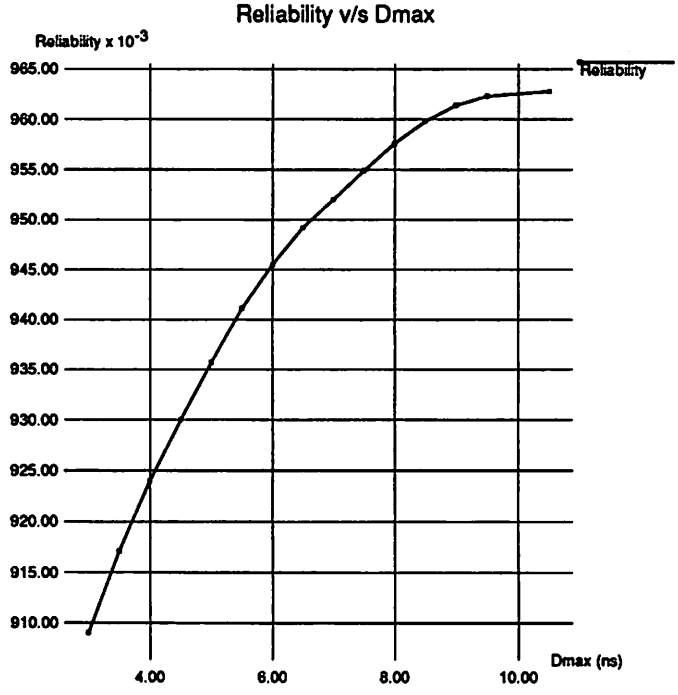


Figure 7.8: Reliability v/s Δ_{max}, normal distribution, $T_0 = 30ns$

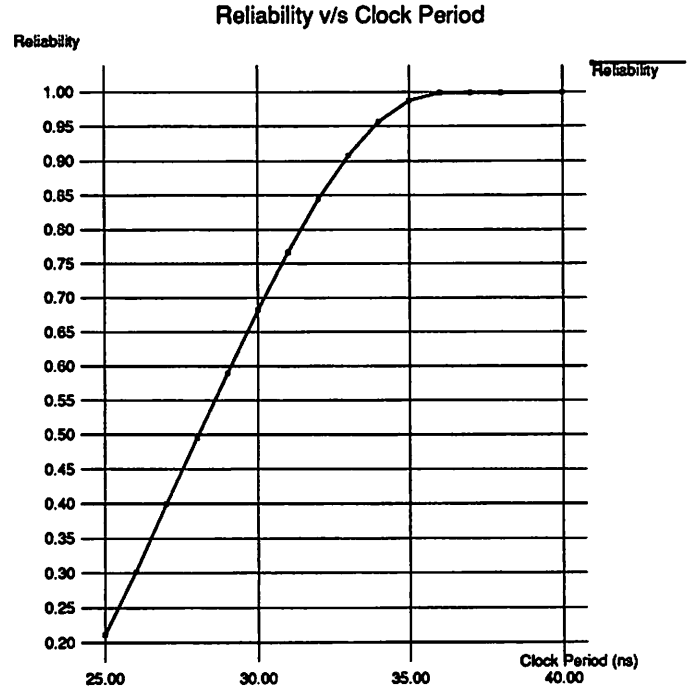


Figure 7.9: Reliability v/s Clock Period, triangular distribution

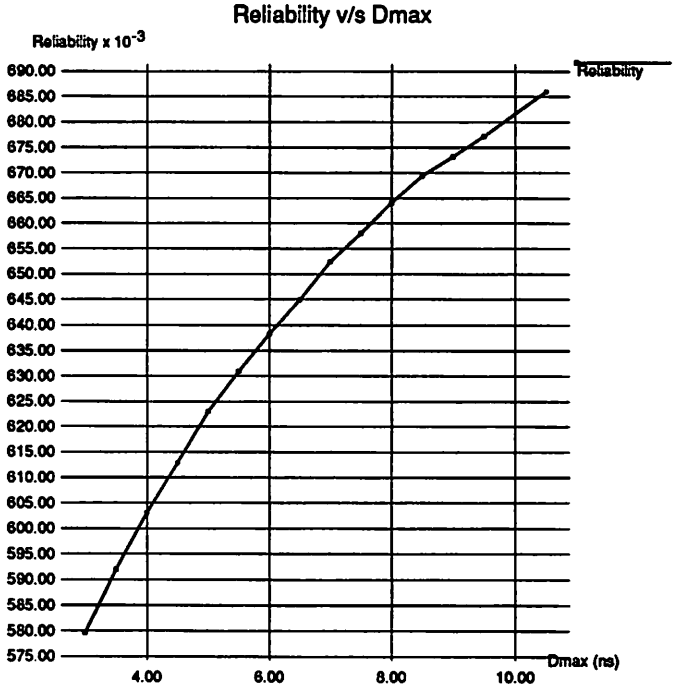


Figure 7.10: Reliability v/s Δ_{max}, triangular distribution, $T_0 = 30ns$

solution is globally optimal across an entire system. The formulation considers practical constraints such as clock pin counts per chip, clock pin counts over the entire system, copy chips constrained to be identical, physical latch placement, clock offset integrality requirements and variations in component delays due to manufacturing, temperature etc. The technique is hierarchical and the optimization can be applied at various levels within a system, i.e. chip level, package level, card level.

The approach is particularly useful for analyzing a system early in the design process. It allows designers to quickly evaluate various clocking topologies and provides insight as to the performance tradeoff associated with assigning clock pins, associating latches with clock pins and inserting delay offset elements. The complexity of approach supports an interactive design environment. A typical analysis for a 16 chip design containing 290 latches takes 30 cpu seconds on an RS6000/530 workstation.

Chapter 8

Final Thoughts

8.1 Looking Back

We started with an overview of the state of the art in performance optimization during the physical design of an integrated circuit. The focus of this work was then narrowed to two problems: (1) performance-driven placement problem for large-scale cell based ICs, and (2) skew optimization. Following this, wirelength and timing models were discussed. The first attempt at developing an efficient algorithm – the primal method, proved to be inadequate for large problems. An improvement was made and the dual algorithm developed. It resolved some of the issues that the primal algorithm could not, but was unable to break the large-scale circuit barrier. Finally, the method of Lagrangian Relaxation was fully exploited in Chapter 5 and an efficient algorithm devised. Experiments on the algorithm led to the implementation of a successful package for performance driven placement – RITUAL. The algorithm accomplished the goals that this work set forth:

- Predictability: the results of the performance optimization are predictable
- Efficiency

The skew optimization problem was discussed and new clocking problems that are of concern during high performance system design were formulated. The formulation involving stochastic delays in Chapter 7 is unique and hopefully opens up possibilities for research involving stochastic optimization in other areas of CAD. Efficient algorithms were developed for solving these problems and their effectiveness on real systems shown through prototype implementations.

8.2 What Lies Ahead?

Performance optimization during physical design is by no means a solved problem. The problem of relating performance optimizations at the logic level and the physical level still remains unsolved. Is it possible to integrate the logic level transformations like those of [Singh 88] with incremental performance driven placement so that placement and logic design go hand-in-hand? Is it possible to perform transistor and cell sizing during placement so that greater control on the wirelength-delay tradeoff is achieved? Can skew optimization, performance-driven placement and performance optimizations at the logic level be united in one theory?

Before these questions can be answered, we must develop a deeper understanding of the impact of logic optimizations on physical layout. Reducing the active area of gates certainly will reduce the total chip area for most moderately sized designs. But multi-million transistor designs are dominated by wiring area. In fact, a circuit with 10,000 cells may have upto 75% of its total area devoted to wiring! Logic systhesis tools today have at best a simplified view of the physical characteristics of the design. As designs become denser, it is increasingly important to develop a better and more detailed understanding of the relationship between the structure of a logic circuit and the resulting wiring area of the circuit.

8.3 Conclusions

As circuits become increasingly complex, new and challenging problems will arise in every aspect of computer-aided design. It is important not to lose focus amidst the blurring intricacies of chip design; the role of CAD has always been and will always be *complexity management.* This body of work has explored and developed some methods for managing the design during one part of the entire process of chip design. It is hoped that the methods described herein will lead to further discussions in the laboratory and the classroom and eventually engender new ideas.

Bibliography

[Aho 74] A. V. Aho, J. E. Hopcroft, and J. D. Ullman. *The Design and Analysis of Computer Algorithms.* Addison Wesley Publishing Company, Reading, Massachusetts, 1974.

[Bakoglu 86] H. B. Bakoglu, J. T. Walker, and J. D. Meindl. A symmetric clock-distribution tree and optimized high-speed interconnections for reduced clock skew in ulsi and wsi circuits. *IEEE Int. Conference on Computer Design: VLSI in Computers and Processors (ICCD-86)*, pages 118–122, October 1986.

[Bakoglu 90a] H. B. Bakoglu. *Circuits, Interconnections and Packaging for VLSI.* Addison Wesley, 1990.

[Bakoglu 90b] H. B. Bakoglu. *Circuits, Interconnections and Packaging for VLSI*, chapter 5, pages 202–211. Addison Wesley, 1990.

[Bakoglu 90c] H. B. Bakoglu. *Circuits, Interconnections and Packaging for VLSI*, chapter 9. Addison Wesley, 1990.

[Boon 89] S. Boon, S. Butler, R. Byrne, B. Setering, M. Casalanda, and Al Scherf. High performance clock distribution for cmos asic's. *IEEE Custom Integrated Circuit Conference*, pages 15.4.1–15.4.4, 1989.

[Brand 86] D. Brand and V. S. Iyengar. Timing analysis using functional relationships. *IEEE International Conference on Computer-Aided Design ICCAD-86*, pages 126–129, 1986.

[Brayton 87] R.K. Brayton, R. Rudell, A. Sangiovanni-Vincentelli, and A. Wang. Mis: A multiple-level logic optimization system. *IEEE Trans. on Computer-Aided Design*, CAD-6, No. 6:1062–1081, November 1987.

[Breuer 77] M. A. Breuer. Min cut placement. *Design Automation and Fault-Tolerant Computing*, 2, October 1977.

[Bunch 76] James R. Bunch and Donald J. Rose, editors. *Sparse Matrix Computations.* Academic Press, 1976.

[Burstein 85] Michael Burstein and Mary N. Youssef. Timing influenced layout design. In *IEEE Proceedings of the 22nd Design Automation Conference*, pages 124–130, 1985.

[Chaudhary 91] Kamal Chaudhary, Massoud Pedram, and E. S. Kuh. Io pad assignment based on circuit structure. *Proceedings of the International Conference on Computer Design*, page to appear, October 1991.

[Cheng 84] C. K. Cheng and E. S. Kuh. Module placement based on resistive network optimization. *IEEE Trans. Computer-Aided Design*, CAD-3:218–225, July 1984.

[Chung 79] F. K. Chung and F. K. Hwang. The largest minimal rectilinear steiner trees for a set of n points enclosed in a rectangle with given perimeter. *Networks*, 9(1):19–36, Spring 1979.

[Coleman 89] T. F. Coleman and L. A. Hulbert. A direct active set algorithm for large sparse quadratic programs with simple bounds. *Mathematical Programming*, 45:373–406, 1989.

[Com 91] *Computer Design*, pages 55–59, January 1991.

[Cook 86] W. Cook, A. M. H. Gerards, A. Schrijver, and E. Tardos. Sensitivity theorems in linear programming. *Mathematical Programming*, 34:251–264, 1986.

[Dang 80] R. L. M. Dang and N. Shigyo. A two-dimensional simulation of lsi interconnect capacitance. *IEEE Electron Device Letters*, EDL-2:196–197, August 1980.

[Dhar 84] S. Dhar, M. A. Franklin, and D. F. Wann. Reduction of clock delays in vlsi structures. *IEEE Int. Conference on Computer Design: VLSI in computers (ICCD)*, pages 778–783, 1984.

[Donath 90] W. Donath and R.J. Norman et. al. Timing driven placement using complete path delays. In *IEEE Proceedings of the 27st Design Automation Conference*, pages 84–89, 1990.

[Dreyfus 69] S.E. Dreyfus. An appraisal of some shortest path algorithms. *Operations Research*, 17, 1969.

[Duin 87] C. W. Duin and A. Volgenant. Some generalizations of the steiner problem in graphs. *Networks*, 17:353–364, 1987.

[Dunlop 84] A. E. Dunlop, V. D. Agrawal, D. N. Deutsch, M. F. Jukl, P. Kozak, and M. Wiesel. Chip layout optimization using critical path weighting. In *IEEE Proceedings of the 21st Design Automation Conference*, pages 133–136, 1984.

[El-Mansy 88] Youssef A. El-Mansy and William M. Siu. *In Handbook of Semiconductor Technology and Computer Systems, ed. Guy Rabbat.* Van Nostrand Reinhold Company, New York, 1988.

[Elmore 48] W. C. Elmore. The transient response of damped linear networks with particular regard to wideband amplifiers. *Journal of Applied Physics*, 19(1):55–63, January 1948.

[Fiduccia 82] C. M. Fiduccia and R. M. Mattheyses. A linear-time heuristic for improving network partitions. *Proc. 19th ACM/IEEE Design Automation Conference*, pages 175–181, 1982.

[Fishburn 81] J.P. Fishburn. Clock skew optimization. *AT&T Bell Laboratories. Murray Hill, NJ 07974*, 1981.

[Fisher 75] Marshall L. Fisher. Using duality to solve discrete optimization problems: Theory and computational experience. *Mathematical Programming Study*, 3:56–94, 1975.

[Fisher 81] Marshall L. Fisher. The lagrangian relaxation method for solving integer programming problems. *Management Science*, Vol 27, No. 1:1–18, January 1981.

[Fisher 82] A. L. Fisher and H. T. Kung. Synchronizing large systolic arrays. *Proceedings of SPIE*, 341:44–52, May 1982.

[Fisher 85] Marshall L. Fisher. An applications oriented guide to lagrangian relaxation. *INTERFACES*, 15:2:10–21, March-April 1985.

[Fletcher 87] R. Fletcher. *Practical Methods of Optimization.* Wiley, New York, New York, 2nd edition, 1987. Ch. 10.

[Florian. 75a] M. Florian. and J. Ferland. A method for computing equilibrium with elastic demands. *Transportation Science*, pages 321–333, 1975.

[Florian 75b] M. Florian, S. Nguyen, and J. Ferland. On the combined distribution-assignment of traffic. *Transportation Science*, pages 43–53, 1975.

[Friedman 86] E. G. Friedman and S. Powell. Design and analysis of a hierarchical clock distribution system for synchronous standard cell/macrocell vlsi. *IEEE Journal of Solid-State Circuits*, SC-21(2):240–246, 1986.

[Garey 79] Michael R. Garey. *Computers and Intractability: A guide to the theory of NP-Completeness.* W.H. Freeman and Company, 1979.

[Geoffrion 70a] A. M. Geoffrion. Elements of large-scale mathematical programming, part i: Concepts. *Management Science*, Vol. 11, No. 18:652–675, July 1970.

[Geoffrion 70b] A. M. Geoffrion. Elements of large-scale mathematical programming, part ii: Synthesis of algorithms and bibliography. *Management Science*, Vol. 11, No. 18:676–691, July 1970.

[Geoffrion 72] A. M. Geoffrion. Generalized benders decomposition. *Journal of Optimization Theory and Applications*, Vol 10, No. 4:236–250, 1972.

[Gill 84] P. E. Gill, W. Murray, M. A. Saunders, and M. H. Wright. Sparse matrix methods in optimization. *SIAM J. Sci. Stat. Comput.*, Vol 5., No. 3:562–572, September 1984.

[Gill 85] P. E. Gill, W. Murray, M. A. Saunders, and M. H. Wright. *Model Building and Practical Aspect of Nonlinear Programming*, pages 209–247. NATO ASI Series, Vol F15, 1985. in Computational Mathematical Programming.

[Gill 89] P. Gill, W. Murray, and M. Wright. *Practical Optimization.* Academic Press, New York, New York, 1989.

[Goldfarb 83] D. Goldfarb and A. Idnani. A numerically stable dual method for strictly convex quadratic programs. *Mathematical Programming*, 27:1–33, 1983.

[Golub 89] G. H. Golub and C. F. Van Loan. *Matrix Computations.* Johns Hopkins University Press, Baltimore, MaryLand, 1989.

[Gura 87] Carol Valentina Gura. Analysis of clock skew in distributed resistive-capacitive interconnects. Master's thesis, University of Illinois, June 1987.

[Hall 70] K. M. Hall. An r-dimensional quadratic placement program. *Management Science*, 17(3):219–229, November 1970.

[Hanan 66] M. Hanan. On steiner's problem with rectilinear distances. *SIAM Journal of Applied Math*, 14:255–265, 1966.

[Hatamian 87] M. Hatamian and G. L. Cash. Parallel bit-level pipelined designs for high speed signal processing. *Proceedings of the IEEE*, Vol 75, No. 9, September 1987.

[Hauge 87] P. S. Hauge, R. Nair, and E. J. Yoffa. Circuit placement for predictable performance. In *IEEE International Conference on Computer-Aided Design,ICCAD-87*, pages 88–91, 1987.

[Held 70] M. Held and R. M. Karp. The traveling-salesman problem and minimum spanning trees. *Operations Research*, 18:1138–1162, 1970.

[Held 74] M. Held, P. Wolfe, and H. P. Crowder. Validation of subgradient optimization. *Mathematical Programming*, 6:62–88, 1974.

[Hig 89] *High Performance Computer Systems*, pages 20–40, December 1989.

[Hig 90] Asic advances. *High Performance Computer Systems*, pages 16–29, May 1990.

[Hitchcock 83] R. B. Hitchcock, G.L. Smith, and D.D. Cheng. Timing analysis of computer hardware. *IBM Journal of Research and Development*, 26(1):100–105, 1983.

[Hochbaum 89] Dorit S. Hochbaum and J. George Shanthikumar. Convex separable optimization is not nuch harder than linear optimization. *Manuscript, University of California, Berkeley*, 1989.

[Horowitz 84] M. A. Horowitz. *Timing Models for MOS Circuits.* PhD thesis, Stanford University, 1984.

[Hsu 86] C. P. Hsu, R. Perry, S. Evans, J. Tang, and J. Lui. Automated layout of channelless gate-arrays. *IEEE Proceedings of the Custom Integrated Circuit Conference*, pages 281–284, 1986.

[Hwang 78] F. K. Hwang. The rectilinear steiner tree problem. *Design Automation and Fault Tolerant Computing*, pages 303–311, 1978.

[IBM 90] IBM. *Optimization Subroutine Library Reference and Guide.* IBM, 1990.

[Idnani 80] Ashok U. Idnani. *Numerically Stable Dual Projection Methods for Solving Positive Definite Quadratic Programs.* PhD thesis, City University of New York, November 1980.

[Jackson 87] M. A. B. Jackson, E. S. Kuh, and M. Marek-Sadowska. Timing-driven routing for building-block layout. In *IEEE International Symposium on Circuits and Systems*, pages 518–519, 1987.

[Jackson 89] M. A. B. Jackson and E. S. Kuh. Performance-driven placement of cell-based ic's. In *IEEE Proceedings of the 26th Design Automation Conference*, pages 370–375, 1989.

[Jackson 90a] M. Jackson, A. Srinivasan, and E.S. Kuh. A fast algorithm for performance driven placement. In *Int. Conf. on Computer-Aided Design*, volume 8, page to appear, 1990.

[Jackson 90b] Michael A.B. Jackson, Arvind Srinivasan, and E.S. Kuh. A fast algorithm for performance driven placement. *UCB ERL Memo, Electronics Research Laboratory, University of California, Berkeley.*, 1990.

[Jr. 71] Arthur F. Veinott Jr. Least d-majorized network flows with inventory applications. *Management Science*, Vol. 19, No. 9:457–567, May 1971.

[Kambo 84] N. S. Kambo. *Mathematical Programming Techniques.* Affiliated East-West Press, Madras, 1984.

[Karmarkar 84] N. Karmarkar. A new polynomial-time algorithm for linear programming. In *Proc. 16th Annual Symposium on the Theory of Computing*, 1984.

[Keyes 87] Robert W. Keyes. *The Physics of VLSI Systems.* Addison-Wesley, Reading, Massachusetts, 1987.

[Kleinhans 91] J.M. Kleinhans, G. Sigl, and K. J. Antreich. Gordian: Vlsi placement by quadratic programming and slicing optimization. *IEEE Trans. on Computer-Aided Design*, CAD-10, No. 3:356–365, 1991.

[Ko 89] Ping Keung Ko, 1989. Private Communication.

[Kogge 81] Peter M. Kogge. *The Architecture of Pipelined Computers.* Hemisphere Publishing Corporation, New York, 1981. McGraw-Hill Advanced Computer Science Series.

[Kung 82] S. Y. Kung and R. J. Gal-Ezer. Synchronous versus asynchronous computation in vlsi array processors. *Proceedings of SPIE*, pages 53–65, May 1982.

[Lawler 76] Eugene L. Lawler. *Combinatorial Optimization: Networks and Matroids.* Holt, Rinehart and Winston, 1976.

[Lee 88] K. W. Lee and C. Sechen. A new global router for row-based layout. *IEEE Intl. Conf. on Computer-Aided Design*, pages 180–183, November 1988.

[Leiserson 83] C. Leiserson, F. Rose, and J. Saxe. Optimizing synchronous circuitry by retiming. *Third Caltech Conf. on VLSI*, 1983.

[Leiserson 88] Charles E. Leiserson and James B. Saxe. A mixed-integer linear programming problem which is efficiently solvable. *Journal of Algorithms*, 9:114–128, 1988.

[Lin 84] T. M. Lin. *A Hierarchical Timing Simulation Model for Digital Integrated Circuits and Systems.* PhD thesis, California Institute of Technology, 1984.

[Lin 90] I. Lin and D. Du. Performance driven constructive placement. In *IEEE Proceedings of the 27st Design Automation Conference*, pages 103–105, 1990.

[Luenberger 84] D. G. Luenberger. *Linear and Nonlinear Programming.* Addison Wesley, Reading, Massachusetts, 2nd edition, 1984. Ch. 14.

[M. Burstein 83] S. J. Hong M. Burstein and R. Pelavin. Hierarchical vlsi layout: Simultaneous placement and wiring of gate-arrays. *Proc. VLSI*, pages 45–60, 1983.

[Marek-Sadowska 89] M. Marek-Sadowska and S. P. Lin. Timing-driven placement. *IEEE International Conference on Computer-Aided Design ICCAD-89*, pages 94–97, 1989.

[MCN 88] *Logic Synthesis and Optimization Benchmarks – User Guide.* Microelectronics Research Center of North Carolina, 1988.

[Mead 80] Carver A. Mead and Lynn A. Conway. *Introduction to VLSI Systems.* Addison-Wesley, Reading, Massachusetts, 1980.

[Micheli 86] Giovanni De Micheli. Performance-oriented synthesis in the yorktown silicon compiler. In *IEEE International Conference on Computer-Aided Design,ICCAD-86*, pages 138–141, 1986.

[Mijuskovic 87] D. Mijuskovic. Clock distribution in application specific integrated circuits. *Microelectronics Journal*, 18(4):15–27, 1987.

[Muller 86] Richard S. Muller and Theodore I. Kamins. *Device electronics for integrated circuits.* Wiley, New York, New York, 2nd edition, 1986. Ch. 10.

[Murtagh 87] Bruce A. Murtagh and Michael A. Saunders. *MINOS 5.1 Users's Guide.* Department of Operations Research, Stanford University, Stanford, CA, 1987.

[Murty 83] Katta G. Murty. *Linear Programming.* John Wiley and Sons, 1983.

[Nagel 75] W. Nagel. Spice2, a computer program to simulate semiconductor circuits. *University of California, Berkeley, Memo No. ERL-M520*, May 1975.

[Nair 89a] R. Nair, C. L. Berman, P.S. Hauge, and E. J. Yoffa. Generation of performance constraints for layout. *IEEE Trans. Computer-Aided Design*, CAD-8:860–874, August 1989.

[Nair 89b] R. Nair, C. Leonard Berman, P. S. Hauge, and E. J. Yoffa. Generation of performance constraints for layout. In *IEEE Transactions on Computer-Aided Design*, volume 8, pages 860–873, 1989.

[Ogawa 86] Yasushi Ogawa, Tatsuki Ishii, Yoichi Shiraishi, Hidekazu Terai, Tokinori Kozawa, Kyoji Yuyama, and Kyoji Chiba. Efficient placement algorithms optimizing delay for high-speed ecl masterslice lsi's. In *IEEE Proceedings of the 23rd Design Automation Conference*, pages 404–410, 1986.

[Ogier 88] Richard G. Ogier. Minimum-delay routing in continuous-time dynamic networks with piecewise-constant capacities. *Networks*, 18:303–318, 1988.

[Ohuchi 80] A. Ohuchi and I. Kaji. Algorithms for optimal allocation problems having quadratic objective functions. *J. Oper. Res. Soc. Japan*, 1:64–80, 1980.

[Ohuchi 84] A. Ohuchi and I. Kaji. Lagrangian dual coordinatewise maximization algorithm for network transportation problems with quadratic costs. *Networks*, 14:515–530, 1984.

[Penn 85] M. G. Penn. Impact and opportunities for application specific integrated circuits (asics). *IEE Proceedings*, 132(2):130–132, 1985.

[Prasitjutrakul 89] S. Prasitjutrakul and W. J. Kubitz. Path-delay constrained floorplanning: A mathematical programming approach for initial placement. In *IEEE Proceedings of the 26th Design Automation Conference*, pages 364–369, 1989.

[Prekopa 88] A. Prekopa, Yu Ermoliev, and R.J-B Wets (editors). *Numerical Techniques for Stochastic Optimization, Chapter: Numerical Solution of Chance Constrained Programming Problems.* Springer-Verlag, New York, 1988.

[Ravi 88] N. Ravi and R. E. Wendell. The tolerance approach to sensitivity analysis in network linear programming. *Networks*, Vol. 18:159–171, 1988.

[Reed 85] J. Reed. Yacr: Yet another channel router. Master's thesis, University of California, Berkeley, February 1985.

[Rockafellar 84] R. T. Rockafellar. *Network Flows and Monotropic Optimization.* John Wiley and Sons, New York, New York, 1984.

[Roychowdhury 91] Jaijeet Roychowdhury and Donald O. Pederson. Efficient transient simulation of lossy interconnect. In *IEEE Proceedings of the 28th Design Automation Conference*, pages 740–745, 1991.

[Rubinstein 83] Jorge Rubinstein, Paul Penfield, and Mark A. Horowitz. Signal delays in rc tree networks. *IEEE Trans. Computer-Aided Design*, CAD-2:202–211, July 1983.

[Rudell 88] R. Rudell. Wolfe – oct interface to the timberwolf standard cell placement program. In Rick Spickelmier, editor, *Oct Tools Distribution 2.1.* University of California, Berkeley, March 1988.

[Sahni 88] Sartaj Sahni, Atul Bhatt, and Raghunath Raghavan. *In Handbook of Semiconductor Technology and Computer Systems, ed. Guy Rabbat.* Van Nostrand Reinhold Company, New York, 1988.

[Sakurai 83] T. Sakurai. Approximation of wiring delays in mosfet lsi. In *IEEE Journal of Solid-State Circuits*, volume SC-18, No. 4, pages 418–426, August 1983.

[Saraswat 82] K. C. Saraswat and F. Mohammadi. Effect of scaling of interconnections on the time delay of vlsi circuits. *IEEE Transactions on Electron Devices*, ED-29:645–650, April 1982.

[Sechen 85] C. Sechen and A. Sangiovanni-Vincentelli. The timberwolf placement and routing package. *IEEE Journal of Solid-State Circuits*, 20:510–522, April 1985.

[Sechen 88a] C. Sechen. *VLSI Placement and Global Routing Using Simulated Annealing.* Kluwer Academic Publishers, 1988.

[Sechen 88b] C. Sechen and K. W. Lee. An improved simulated annealing algorithm for row-based placement. *Proc. IEEE Intl. Conference on Computer-Aided Design*, pages 478–481, November 1988.

[Sentovich 91] E. M. Sentovich. SIS: An interactive system for the synthesis of sequential logic circuits. 1991. Unpublished document.

[Shapiro 79] Jeremy F. Shapiro. *Mathematical Programming: Structures and Algorithms.* John Wiley and Sons, New York, 1979.

[Singh 88] Kanwar Jit Singh, Albert R. Wang, Robert K. Brayton, and Alberto L. Sangiovanni-Vincentelli. Timing optimization of combinational logic. In *Proceedings of the IEEE International Conference on Computer Aided Design*, 1988.

[Spickelmier 88] R. Spickelmier, editor. *Oct Tools Distribution 2.1.* University of California, Berkeley, March 1988.

[Srinivasan 90a] A. Srinivasan. Algorithms for performance-driven placement. *University of California, Berkeley, Technical Memo UCB/ERL M90/107*, 1990.

[Srinivasan 90b] A. Srinivasan, S. Seshadri, and J. G. Shanthikumar. Stochastic cycle-time optimization of sequential systems. *University of California, Berkeley, Technical Memo UCB/ERL M90/91*, 1990.

[Srinivasan 91] A. Srinivasan, Kamal Chaudhary, and E.S. Kuh. Ritual: An algorithm for performance-driven placement of cell-based ics. In *Int. Conf. on Computer-Aided Design*, page to appear, 1991.

[Sutanthavibul 90] S. Sutanthavibul and E. Shragowitz. An adaptive timing driven layout for high speed vlsi. In *IEEE Proceedings of the 27st Design Automation Conference*, pages 90–95, 1990.

[Teig 86] S. Teig, R. L. Smith, and J. Seaton. Timing-driven layout of cell-based ic's. *VLSI System's Design*, pages 63–73, May 1986.

[Terai 90] M. Terai, K. Takahashi, and K. Sato. A new mincut placement algorithm for timing assurance layout design. In *IEEE Proceedings of the 27st Design Automation Conference*, pages 96–102, 1990.

[Tsay 88] R. S. Tsay, E. S. Kuh, and C. P. Hsu. Proud: A sea-of-gates placement algorithm. *IEEE Design and Test of Computers*, pages 318–323, December 1988.

[Tsay 89] Ren-Song Tsay. *Partitioning, Placement, and Routing Algorithms for High Complexity Integrated Circuits.* PhD thesis, University of California, Berkeley, 1989.

[Vecci 83] M. Vecci and S. Kirkpatrick. Global wiring by simulated annealing. *IEEE Trans. CAD*, CAD-2(4):215–222, October 1983.

[VLS 87] *VLSI Systems Design*, pages 6–34, May 1987.

[Wakerly 90] John. F. Wakerly. *Digital Design: Principles and Practices.* Prentice Hall, Englewood Cliffs, New Jersey, 1990.

[Wolff 78] P.K. Wolff, A. E. Ruehli, B. J. Agule, J. D. Lesser, and G. Goertzel. Power/timing: Optimization and layout techniques for lsi chips. *Journal of Design Automation and Fault-Tolerant Computing*, pages 145–164, 1978.

[Youssef 89] H. Youssef, E. Shragowitz, and L. Bening. Critical path issue in vlsi design. *Proc. International Conf. on Computer-Aided Design*, pages 520–523, 1989.

[Zangwill 69] W. I. Zangwill. *Nonlinear Programming: A Unified Approach.* Prentice Hall, Englewood Cliffs, New Jersey, 1969.