RITUAL: A PERFORMANCE DRIVEN PLACEMENT ALGORITHM

by

Arvind Srinivasan, Kamal Chaudhary, and E. S. Kuh

Memorandum No. UCB/ERL M91/103

19 November 1991

RITUAL: A PERFORMANCE DRIVEN PLACEMENT ALGORITHM

by

Arvind Srinivasan, Kamal Chaudhary, and E. S. Kuh

Memorandum No. UCB/ERL M91/103

19 November 1991

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

RITUAL: A PERFORMANCE DRIVEN PLACEMENT ALGORITHM

by

Arvind Srinivasan, Kamal Chaudhary, and E. S. Kuh

Memorandum No. UCB/ERL M91/103

19 November 1991

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

RITUAL: A Performance Driven Placement Algorithm

Arvind Srinivasan Kamal Chaudhary E. S. Kuh

Electronics Research Laboratory,
University of California, Berkeley

Abstract

In this paper we describe an algorithm for obtaining a placement of large scale cell-based ICs subject to performance constraints. The problem is formulated as a constrained programming problem and is solved in two phases: continuous and discrete. Constraints are placed on total path delays rather than nets and behaviour of all the paths is captured. A unified mathematical technique, based on Lagrangian Relaxation is used. The algorithm yields good results as we show on a set of real examples. On the average, we are able to make upto 15% improvement in the wire delay of these examples with little or no impact on chip area after routing. These improvements are obtained by modifying the placement alone. The acronym RITUAL represents the key idea of our technique: Residual Iterative Technique for Updating All Lagrange multipliers.

1 Introduction

1.1 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.

Interconnect wires have a 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 driving a length of interconnect wire that connects G_s to a receiving 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 $\times$ 2.0 cm chip using Rent's rule is about 0.25 cm (see [Bakoglu 90a]) 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.

1.2 Problem Definition

Given a sequential circuit composed of a large number of small cells the problem is to place the cells on a two-dimentional plane so as to minimize total wire length while satisfying user specified timing constraints and cell position constraints. The total wire length is a measure of routablity of the circuit. The cell position constraints are required to satisfy the design rules. They may include constraints such as requiring cells to lie on a grid, or to place them in rows, etc. Even simpler formulations of the problem without timing constraints are known to be NP-Complete.

1.3 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 89]. 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.

2 Overview

The strategy used in RITUAL is to perform timing constrained optimization in continuous space followed by a discrete-space optimization step to satisfy timing and cell position constraints. The first phase, called the global phase, formulates the problem as a constrained quadratic programming problem. The second phase, called the assignment phase, is formulated as a constrained assignment problem. A unified technique based on Lagrangian Relaxation is used in both phases.

The result of the first phase is a uniform distribution of cells on the chip that satisfies timing constraints. However, the solution may not satisfy all cell position constraints. This result is referred to as a global placement. The global placement is used as an initial solution for the second phase. In the assignment phase, cells are assigned to prespecified slot locations such that timing and spatial constraints are met. There is a smooth transition from the global to the assignment phase.

In addition to placing cells within the core of the chip, RITUAL can place input and output pads on the periphery of the chip so as to satisfy constraints and improve the quality of the placement. Pad placement can be done simultaneously with core placement or as an independent step.

The solution technique used in RITUAL is very powerful and allows us to handle a variety of constraints easily. Besides timing constraints, constraints such as, net length bounds, matching two or more net lengths, clock tree buffer placement and distances between cells can be incorporated. Any practical requirements that can be formulated as convex constraints (linear or non-linear) can be included.

3 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

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

4 Timing Models

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 1. 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 1 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 $T_{clk \to Q}$. This relationship is expressed as follows

$$CP \geq T_{long} + T_{skew} + T_{clk \to Q} + T_{su} \qquad (1)$$

If equation 1 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 and the synchronizing elements in the circuit are of the *level-sensitive* type. [Wakerly 90] has an excellent discussion of this problem.

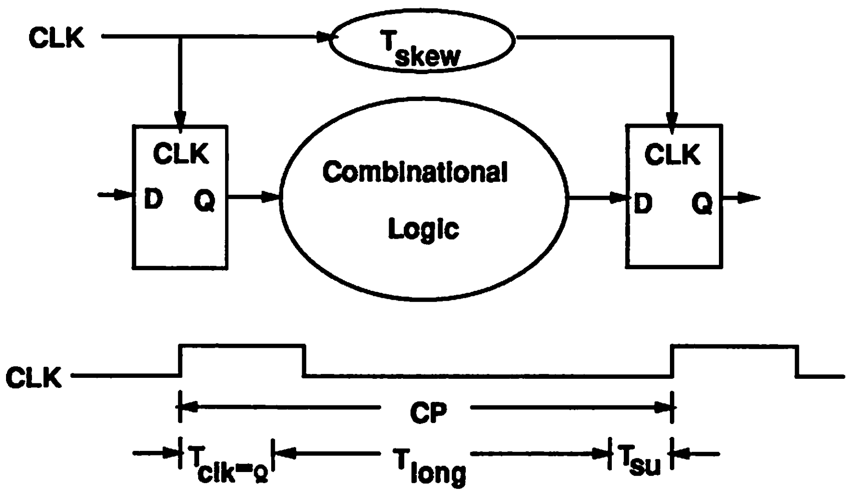

Figure 1: Figure illustrating clocking constraints

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.

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.

$$A = A^I \cup A^E \tag{2}$$

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

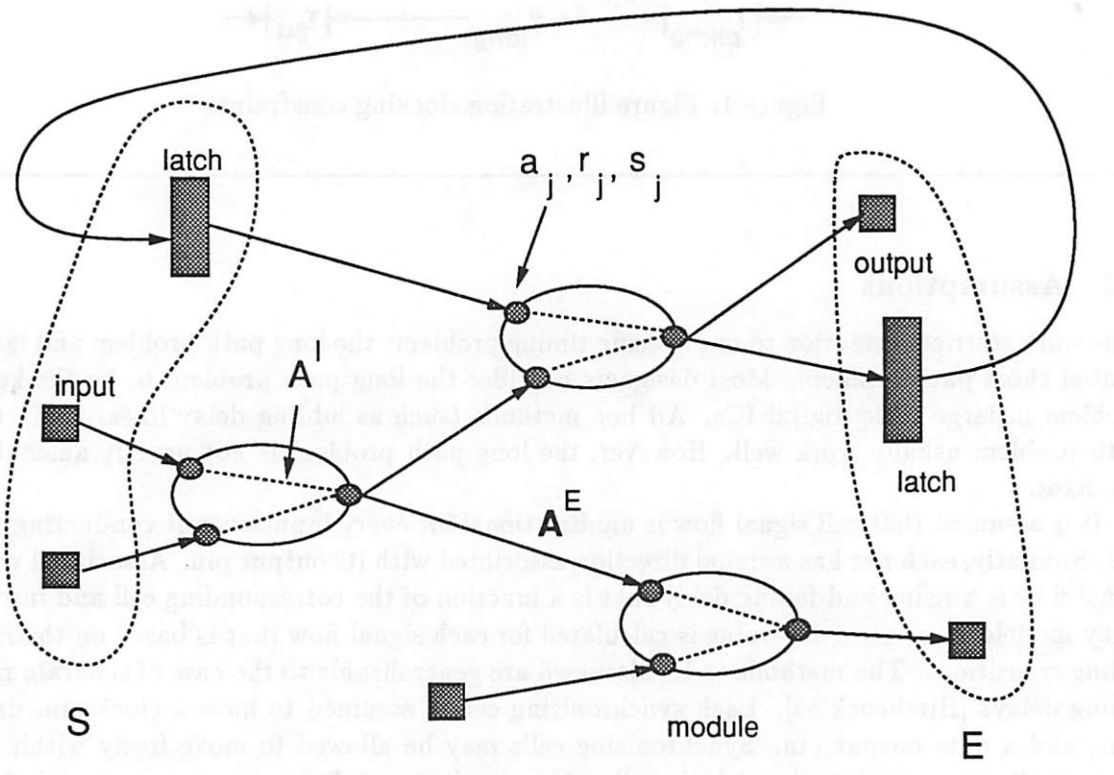

Figure 2: The timing graph D_T

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

$$d(v_j, v_i) \; \forall \, (v_j, v_i) \in A^I \tag{3}$$

and let the delay of net n_i be characterized by

$$d(v_i, v_j) \; \forall \, (v_i, v_j) \in A^E \tag{4}$$

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 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

$$a_j \;=\; \max\{a_i + d(v_i, v_j) \,|\, \forall (v_i, v_j) \in A\} \tag{5}$$

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

$$r_i \;=\; \min\{r_j - d(v_i, v_j) \,|\, \forall (v_i, v_j) \in A\} \tag{6}$$

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

$$s_i \;=\; r_i - a_i \tag{7}$$

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 4.1 *The timing of the chip is said to be* <u>*feasible*</u> *if and only if* $s_i \geq 0,\ \forall v_i \in V$.

A critical long path is defined as follows:

Definition 4.2 *A* <u>*critical long path*</u> 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.* $\Pi = \{v_i | s_i < 0 \,\forall\, v_i \in \Psi\}$

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 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. □

4.4 Interconnect Delay

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 3. Let

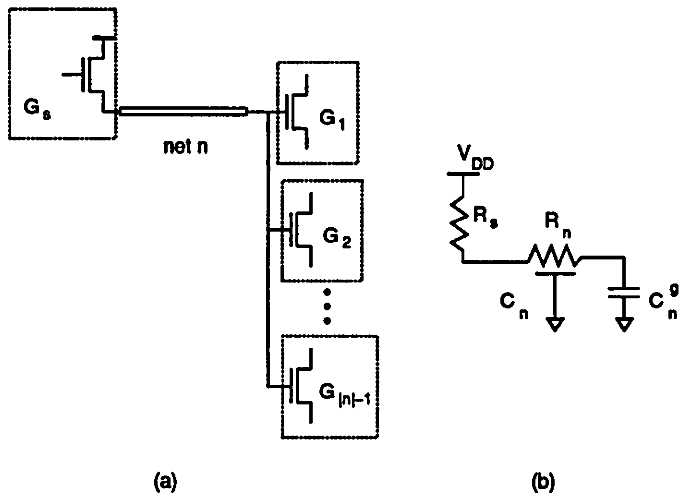

Figure 3: A net (a), and associated interconnect delay model (b)

R_s^{rise} represent the resistance of the driving cell during a rising transition on its output and R_s^{fall} be the resistance during a falling transition. The rising and falling waveform delays of the output net of G_s can be approximated by:

$$\begin{aligned} d_s^{rise} &= R_s^{rise}(C_{net} + C_{load}) + 0.5R_w(C_{net} + C_{load}) \\ d_s^{fall} &= R_s^{fall}(C_{net} + C_{load}) + 0.5R_w(C_{net} + C_{load}) \end{aligned} \tag{8}$$

C_{net} is the interconnect capacitance of the output net of G_s and C_{load} is the capacitance of the driven pins. R_w is the lumped interconnect resistance and for current technologies, its contribution is of the second order. The factor of 0.5 which multiplies R_w is based on the analysis of [Bakoglu 90b] to model distributed RC delay. Although RITUAL can be easily modified to include the second order delay effects due to R_w, in this paper we will neglect it for simplicity. For details see [Srinivasan 91].

4.5 Models for the Lumped Capacitance

In order to estimate d_{rise} and d_{fall} due to a net, we need to model the capacitance 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. Our 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.

4.5.1 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:

$$\begin{aligned} C_n &= C_h(x^n_{max} - x^n_{min}) + C_v(y^n_{max} - y^n_{min}) \\ R_n &= R_h(x^n_{max} - x^n_{min}) + R_v(y^n_{max} - y^n_{min}) \end{aligned} \quad (9)$$

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

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

4.5.2 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.

$$\begin{aligned} C^x_n &= C_v(y^n_{max} - y^n_{min}) + C_h \sum_{p_i \in n} |x_{p_i} - \overline{x_n}| \\ C^y_n &= C_h(x^n_{max} - x^n_{min}) + C_v \sum_{p_i \in n} |y_{p_i} - \overline{y_n}| \\ C_n &= \tfrac{1}{2}(C^x_n + C^y_n) \end{aligned} \quad (10)$$

The resistance is modeled by similar equations.

4.5.3 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:

$$\begin{aligned} C_n &= \sum_{m_i \in n} C_h|x_i - x_s| + C_v|y_i - y_n| \\ R_n &= \sum_{m_i \in n} R_h|x_i - x_s| + R_v|y_i - y_n| \end{aligned} \quad (11)$$

Proposition 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.*

For the proof, see [Srinivasan 91].

5 Phase I: Continuous Space Optimization

5.1 Quadratic Objective Function

The quadratic objective function 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 variant of the quadratic wirelength model we use has the following representation for the length of net n

$$L_n = \sum_{p_i, p_j \in n} ((x_{p_i} - \overline{x_n})^2 + (y_{p_i} - \overline{y_n})^2) \tag{12}$$

where $\overline{x_n}$ and $\overline{y_n}$ are defined as:

$$\overline{x_n} = \frac{1}{|n|} \sum_{m_j \in n} x_j$$

$$\overline{y_n} = \frac{1}{|n|} \sum_{m_j \in n} y_j$$

Other quadratic measures have been introduced since the work of Hall. The differences between

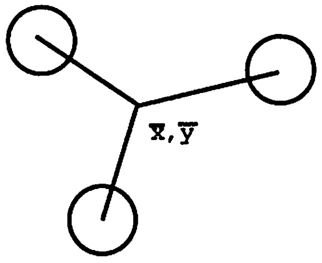

Figure 4: The Quadratic wirelength model

these models are minor and the analysis in this work remains unchanged for any convex model. The estimate of the cost of a placement can be written as

$$L = \sum_{n \in \mathcal{N}} L_n \tag{13}$$

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, we 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. of the final routing. The extension of this work to other models (for example the bounding-box model or the approximate Steiner tree model) of wirelength has been fully explored in [Srinivasan 91].

5.2 Formulation

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

$$\begin{array}{llll} a_j & \geq & a_i + d(v_i, v_j) & \forall (v_i, v_j) \in A \\ a_j & \leq & T_j & \forall v_j \in E \\ a_j & \geq & T_j & \forall v_j \in S \\ d(v_i, v_j) & = & f(X_i, Y_i) & \forall n_i \in \mathcal{N} \end{array} \tag{14}$$

where the function $f(X_i, Y_i)$ is the net delay equation for the output net of cell m_i based on the bounding-box, steiner or star connected net capacitance estimate presented earlier.

5.3 Optimality Conditions

The quadratic objective function can be written compactly in matrix notation as:

$$L(\mathbf{x}, \mathbf{y}) = \frac{1}{2}(\mathbf{x}^T\mathbf{B}\mathbf{x} + \mathbf{y}^T\mathbf{B}\mathbf{y}) + \mathbf{c}^T\mathbf{x} + \mathbf{d}^T\mathbf{y} \tag{15}$$

where **x** is a vector of the x-coordinates of the module locations and **y** is a vector of the y-coordinates. **c** and **d** are constant vectors arising from the fixed modules. **B** is a symmetric matrix, typically sparse and can be stored very efficiently using sparse matrix data structures as in [Bunch 76]. In addition, it is shown that **B** is positive-definite.

The total number of variables in the problem is $2M + P$. However, the P arrival time 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 $\mathbf{w}_{pin}$ denote the vertex arrival time variables. Then

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

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

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

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

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

Then, the cost function can be rewritten as

$$L = \frac{1}{2}\mathbf{w}^T\mathbf{Q}\mathbf{w} + \mathbf{b}^T\mathbf{w} \tag{16}$$

Theorem 5.1 *If there exists at least one module with fixed x coordinate and one module with fixed y coordinate and the modules do not form disconnected subsets, then GP has a unique solution.*

Proof. When there are more than one fixed module, the proof that a unique solution exists can be found in [Srinivasan 91]. When there is one fixed module the obvious unique optimal solution is the one in which all the cells occupy the same location as the fixed module. □

It is assumed in the following discussion that the wirelength models and the timing models used are one of those presented in in earlier. Under this assumption, 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] λ satisfying

$$\begin{aligned} \nabla \mathbf{L}(\mathbf{w}^*) + \lambda^T \nabla \mathbf{A}^* &= \mathbf{0} \\ \lambda^T \nabla \mathbf{A}^* &= \mathbf{0} \\ \lambda &\geq \mathbf{0} \end{aligned} \tag{17}$$

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].

Corollary 5.1 *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. (For proof see [Srinivasan 91].)*

5.4 Why is the Problem Difficult to Solve?

The number of constraints and variables in GP can be enormous even for problems of moderate size. For a typical problem with 1000 cells and 3000 nets, the number of active variables could be upto 18,000 and the active constraints could number 15,000. However, what makes the problem even more difficult is that the constraint set is highly degenerate. The effect of degeneracy is that standard quadratic-programming algorithms flounder for many steps without improving the objective function. For a detailed description of the problems in using conventional quadratic programming methods, see [Srinivasan 91].

6 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 work 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.

6.1 A Simple Example

Consider the constrained optimization problem:

$$\begin{array}{ll} \min & (5x-14)^2 \\ \text{subject to} & x^2 \leq 5 \end{array} \tag{18}$$

The optimal solution to this problem is $x = \sqrt{5}$. From basic Lagrangian theory [Fisher 85] it can be shown that the problem stated above is equivalent to the following optimization problem:

$$\max_{\mu \geq 0} \min_{x} \; \mathrm{L}(x,\mu) = (5x-14)^2 + \mu(x^2-5) \tag{19}$$

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 19. 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:

   $$\begin{array}{rcl} t^{(0)} & = & \alpha \\ \mu^{(k+1)} & = & \max\{0, \mu^{(k)} + t^{(k)}({x^{(k)}}^2 - 5)\} \\ t^{(k+1)} & = & \beta t^{(k)} \end{array} \tag{20}$$

   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 $\alpha = 1.0, \beta = 1.0$. For a fixed value of μ the optimal solution is

$$\begin{array}{rcl} 2(5x-14) + \mu(2x) & = & 0 \\ x & = & \frac{14}{5+\mu} \end{array} \tag{21}$$

The values for x and μ are listed for each iteration.

1. Solve Equation 21 with $\mu^{(0)} = 0$. The optimal solution is $x^{(0)} = 2.8$.

2. Solve Equation 21 with $\mu^{(1)} = \mu^{(0)} + ({x^{(0)}}^2 - 5) = 2.84$. The initial value of μ was too low and this step increases it by an amount equal to the residue in the constraint.
3. Proceeding in a similar manner to Step 2, we obtain $x^{(1)} = 1.78$, $\mu^{(2)} = 1.03$.
4. $x^{(2)} = 2.32$, $\mu^{(3)} = 1.42$.
5. $x^{(3)} = 2.18$, $\mu^{(4)} = 1.17$.
6. $x^{(4)} = 2.27$, $\mu^{(5)} = 1.31$.
7. $x^{(4)} = 2.22$, $\mu^{(6)} = 1.23$....

In the limit x converges to the optimal value of 2.23... 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]). This example illustrates the power of the relaxation method in solving nonlinearly constrained problems by series of unconstrained optimizations.

6.2 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:

$$\begin{array}{lrcl} \min & \mathbf{f}(x) & & \\ \text{subject to} & \mathbf{g}(x) & \leq & c \\ & \mathbf{h}(x) & \leq & d \end{array} \tag{22}$$

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 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:

$$\begin{array}{ll} \max\limits_{\lambda \geq 0} \ \min\limits_{x} & \mathbf{f}(x) + \lambda^T(\mathbf{g}(x) - c) \\ \text{subject to} & \mathbf{h}(x) \leq d \end{array} \tag{23}$$

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

1. Select an initial value for λ.
2. Solve
$$\begin{array}{ll} \min\limits_{x} & \mathbf{f}(x) + \lambda^T(\mathbf{g}(x) - c) \\ \text{subject to} & \mathbf{h}(x) \leq c \end{array} \tag{24}$$
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 Section 4. 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.

7 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:

$$\begin{aligned} \frac{1}{M}\sum_{i\in\mathcal{M}} x_i &= r_x \\ \frac{1}{M}\sum_{i\in\mathcal{M}} y_i &= r_y \end{aligned} \tag{25}$$

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 into are partitioned 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 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).

$$\begin{aligned} \frac{1}{|S_j|}\sum_{i\in S_j} x_i &= r_j^x, \quad j = 1,...,4 \\ \frac{1}{|S_j|}\sum_{i\in S_j} y_i &= r_j^y, \quad j = 1,...,4 \end{aligned} \tag{26}$$

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.

8 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:

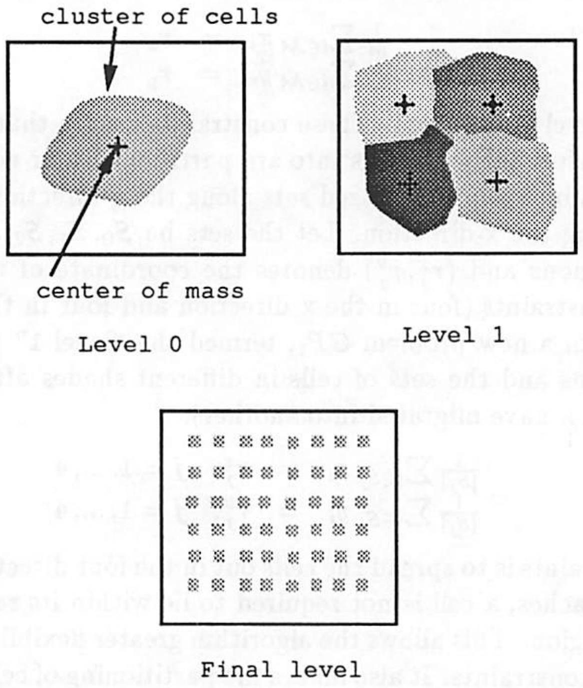

Figure 5: A sequence of showing application of spread constraints

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

8.1 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:

$$\begin{array}{lllll} \text{minimize} & \mathrm{L}(\mathbf{w}) & & & (GP_k) \\ \text{subject to} & a_j & \geq & a_i + d(v_i, v_j) & \forall (v_i, v_j) \in A \\ & a_j & \leq & T_j & \forall v_j \in E \\ & a_j & \geq & T_j & \forall v_j \in S \\ & d(v_i, v_j) & = & f(X_i, Y_i) & \forall n_i \in \mathcal{N} \\ & \frac{1}{|S_j|} \sum_{i \in S_j} x_i & = & r_j^x, & j = 1, \ldots, k \\ & \frac{1}{|S_j|} \sum_{i \in S_j} y_i & = & r_j^y, & j = 1, \ldots, k \end{array} \tag{27}$$

Let $\mathbf{w}$ represent the combined vector of cell x and y positions and vertex arrival time variables. Let $\mathbf{A}\mathbf{w} \leq \mathbf{c}$ 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:

$$\max_{\lambda \geq 0} \min_{x} \ \frac{1}{2} \mathbf{w}^T \mathbf{Q} \mathbf{w} + \mathbf{b}^T \mathbf{w} + \lambda^T (\mathbf{A}\mathbf{w} - \mathbf{c}) \tag{28}$$

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

$$\mathbf{w}^{(k+1)} = -\mathbf{Q}^{-1} [\lambda^{(k)} \mathbf{A} + \mathbf{b}] \tag{29}$$

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 29 is λ and the only product to be computed is $\lambda^{(k)}\mathbf{A}$. This is a linear-time computation since the maximum number of active equations is equal to the number of edges in the timing graph. In an efficient implementation, $\mathbf{Q}^{-1}$ is never computed. Since $\mathbf{Q}$ is positive definite, the equation Equation 29 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 29 is equivalent to solving a minimum quadratic cost flow problem.

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

8.2 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:

$$\lambda^{(k+1)} = \max\{0, \lambda^{(k)} + t^{(k)}(\mathbf{A}\mathbf{w}^{(k)} - \mathbf{c})\} \tag{30}$$

In this formula, $t^{(k)}$ is a scalar step size and $\mathbf{w}^{(k)}$ is the optimal solution for Equation 29 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].

8.3 Computing $t^{(k)}$

Recall that all the delay models described in Section 4 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 5.4. 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 edges, write the delay equations as linear equations, removing the absolute values from Equation 11, 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 6). Then, the right hand side of Equation 29 is updated based on this configuration

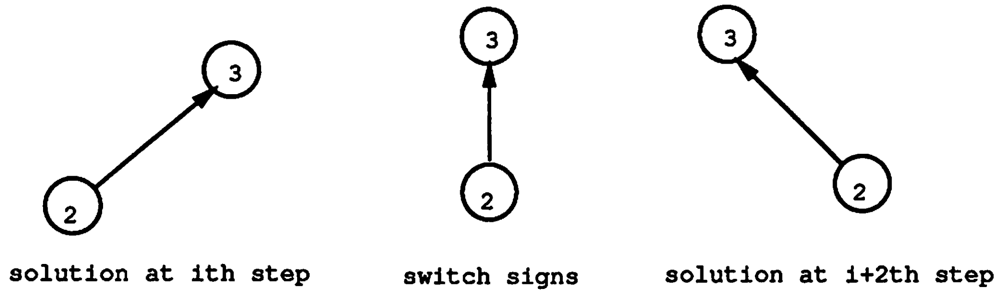

Figure 6: Linearizing absolute valued constraints

and the system of equations is solved for 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 6). $\mathbf{w}^{(k+1)}$ and $\lambda^{(k+1)}$ are updated according to this value of $t^{(k)}$.

8.4 Updating the critical arc set

The algorithm maintains a set of active critical arcs throughout the algorithm. Active arcs are those whose current Lagrange multipliers are positive. By maintaining the set of active arcs, 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 arcs are critical, this typically makes updating the right hand side a sub-linear procedure. In any event, the number of critical arcs at any time is linearly bounded. 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 **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 arcs .

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

8.5 Computational Complexity

The flow of the algorithm is described in Figure 7. 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 **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 edges, 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 arc 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 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.

8.6 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 29 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.

9 Phase II: Discrete Space Optimization

Although it is possible to add slot constraints as described in Section 7 till 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 **C** is set up with as many rows as the number of the cells in the region and as many columns

```
1. Initialize the level $l = 0$.
2. Initialize the current active set to $\emptyset$.
3. Add the spread constraints for level $l$ to the active set. There are no Lagrange
   multpliers for the timing constraints initially since $\tau_A$ is empty. (Multipliers
   exist for the spread constraints for level 0.)
4. Update $\lambda^T A$ and solve Equation 29.
5. Perform a timing verification on the timing graph $D_T$ and update the current active
   arc set.
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.
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 7: Outline of Lagrangian Relaxation

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:

$$\begin{array}{llll} \min \sum_{ij} \mathbf{C}_{ij} z_{ij} & & & DP \\ \text{subject to} & & & \\ \sum_{j=1}^{N} z_{ij} & = & 1, & i = 1,\ldots,M \\ \sum_{i=1}^{M} z_{ij} & \leq & 1, & j = 1,\ldots,N \\ \mathbf{x} & = & \mathbf{ZX} & \\ \mathbf{y} & = & \mathbf{ZY} & \\ \mathbf{Aw} & \leq & \mathbf{c} & \\ z_{ij} & \in & \{0,1\}, & \forall i,j \end{array} \tag{31}$$

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

9.1 Solving the Constrained Problem

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.

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 **W** denote the combined vector of slot locations:

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

Using this notation, we can rewrite 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:

$$\begin{array}{llll} \min & \sum_{ij} \mathbf{C}_{ij} z_{ij} & & DP \\ \text{subject to} & & & \\ \sum_{j=1}^{N} z_{ij} & = & 1, & i = 1, \ldots, M \\ \sum_{i=1}^{M} z_{ij} & \leq & 1, & j = 1, \ldots, N \\ \mathbf{A}\mathbf{Z}_{xy}\mathbf{W} & \leq & \mathbf{c} & \\ z_{ij} & \in & \{0,1\}, & \forall i,j \end{array} \tag{32}$$

The relaxed problem is:

$$\begin{array}{llll} \max\limits_{\nu \geq 0} \min & \sum_{i=1}^{M} \sum_{j=1}^{N} \mathbf{C}_{ij} z_{ij} + \nu^T(\mathbf{A}\mathbf{Z}_{xy}\mathbf{W} - \mathbf{c}) & & RP \\ \text{subject to} & & & \\ \sum_{j=1}^{N} z_{ij} & = \ 1, & i = 1, \ldots, M & \\ \sum_{i=1}^{M} z_{ij} & \leq \ 1, & j = 1, \ldots, N & \\ z_{ij} & \in \ \{0,1\}, & \forall i,j & \end{array} \tag{33}$$

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:

$$\begin{array}{llll} \min & \sum_{i=1}^{M} \sum_{j=1}^{N} \mathbf{C}_{ij} z_{ij} + \nu^T(\mathbf{A}\mathbf{Z}_{xy}\mathbf{w}) & & RP \\ \text{subject to} & & & \\ \sum_{j=1}^{N} z_{ij} & = \ 1, & i = 1, \ldots, M & \\ \sum_{i=1}^{M} z_{ij} & \leq \ 1, & j = 1, \ldots, N & \end{array} \tag{34}$$

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

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

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

$$\begin{array}{llll} \text{min} & \sum_{ij}[\mathbf{C}_{ij} + \sum_{k=1}^{p}\nu_k(\mathbf{A}_{ki}\mathbf{X}_j + \mathbf{A}_{k,i+M}\mathbf{Y}_j)]z_{ij} & LA & \\ \text{subject to} & & & \\ \sum_{j=1}^{N} z_{ij} & = 1, & i = 1,\ldots,M & \\ \sum_{i=1}^{M} z_{ij} & \leq 1, & j = 1,\ldots,N & \end{array} \qquad (35)$$

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 arcs (i.e. A, the matrix representation of the current timing constraints) and ν according to $\nu^{(k+1)} = \max(0, \nu^{(k)} + (\mathbf{Aw} - \mathbf{c}))$.
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 arcs. 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 arcs with non-zero multipliers passing through the cell. As the reader may recall, only a linear number of arcs 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 minimze 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 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 8 and repeating the process until the improvement is small. Note that the absolute valued delay

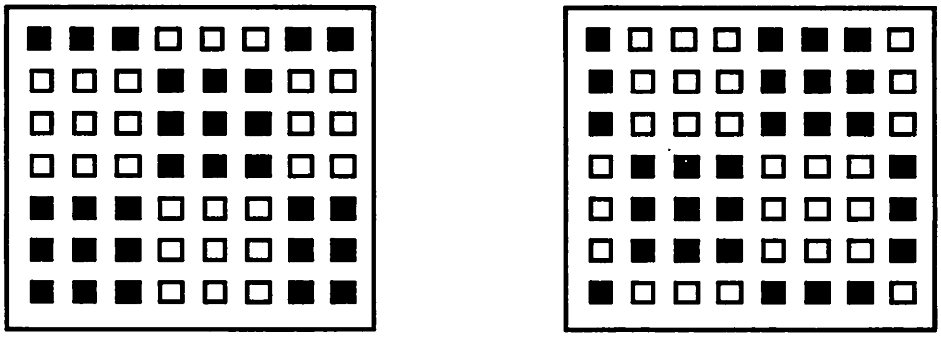
Figure 8: Regions of optimization are shifted at alternate iterations

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 9.

9.2 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

```
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 $\nu$.
4. Update the set of critical arcs and $\nu$ according to $\nu^{(k+1)} = \max(0, \nu^{(k)} + (\mathbf{Aw} - \mathbf{c}))$.
5. Shift the regions of optimization. (See Figure 8)
6. Repeat steps 2-5 until the constraints are satisfied to a desired accuracy.
```

Figure 9: The Slot Assignment Algorithm

```
1. Determine number of levels of spread based on chip size/topology
2. Apply the algorithm of Figure 7
3. Apply the Algorithm of Figure 9
4. Optionally, perform input and output pad assignment on the chip boundary
```

Figure 10: The complete placement algorithm

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 $O(N^3)$ 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, first phase provides an excellent initial solution for the second phase.

10 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 [Pedram 91] have developed successful methods for pad placement which could be used in place of the basic procedure described here.

11 The Complete Algorithm

The flow of the algorithm is shown in Figure 10. The entire algorithm may be repeated several times if necessary.

12 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 10 will result in a placement into rows of cells with different row widths as shown in Figure 11. 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.

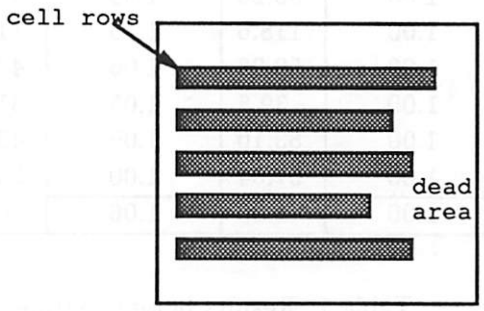

Figure 11: Uneven row widths in standard cells results in dead area

The constraints to be added are:

$$|r_i - \overline{r}| = 0, \; i = i, \ldots, n_r \tag{36}$$

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 written as:

$$\begin{array}{lll} \min & \sum_{ij}[\mathbf{C}_{ij} + \sum_{k=1}^{p} \nu_k(\mathbf{A}_{ki}\mathbf{X}_j + \mathbf{A}_{k,i+M}\mathbf{Y}_j) + \mu_j \mathbf{R_{ij}}]z_{ij} \quad LA & \\ \text{subject to} & & \\ \sum_{j=1}^{N} z_{ij} & = 1, & i = 1, \ldots, M \\ \sum_{i=1}^{M} z_{ij} & \leq 1, & j = 1, \ldots, N \end{array} \tag{37}$$

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.
- 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.

Example	# cells	RITUAL				TimberWolf5.6x	
		Wire Mode		Time Mode			
		Wirelength	Delay	Wirelength	Delay	Wirelength	Delay
C2	590	1.00	24.36	1.11	21.70	1.00	28.29
C3	1254	1.00	47.98	1.08	44.06	1.00	50.39
C5	1381	1.00	36.26	1.08	31.64	1.02	34.19
C6	1945	1.00	118.6	1.15	111.5	1.03	120.5
C7	2150	1.00	52.98	1.06	44.65	1.06	55.60
s9234	2748	1.00	39.8	1.05	31.31	0.94	37.6
s13207	4267	1.00	53.10	1.00	43.37	1.32	57.24
s15850	4981	1.00	57.64	1.00	50.96	0.94	62.40
Average		1.00	1.00	1.06	0.87	1.03	1.03

Table 1: Results before routing

Example	# cells	RITUAL				TimberWolf5.6x	
		Wire Mode		Time Mode			
		Area	Delay	Area	Delay	Area	Delay
C2	590	436	25.93	473	24.00	486	28.32
C3	1254	1010	50.70	1065	48.50	930	50.50
C5	1381	1215	35.43	1293	33.69	1248	35.95
C6	1945	1380	134.0	1529	128.2	1395	137.2
C7	2150	1648	54.41	1783	49.76	1771	55.13
s9234	2750	3088	45.30	3480	40.90	2950	44.20
Avg		1.00	1.00	1.09	0.93	1.02	1.02

Table 2: Results after routing

13 Experimental Results

We have implemented all the ideas described in this paper in C on a DEC3100 workstation. The package was tested on a set of ISCAS benchmarks ranging from 500 to 5000 cells. The examples were generated using SIS [Sentovich 91]. We ran RITUAL with timing and without timing constraints and compared the results with an Industry standard simulated annealing based placement package TimberWolf 5.6 [Sechen 88]. The results are tabulated in Table 1 and 2. Table 1 shows the estimated wirelength and delay. Results in Table 2 are after detailed routing. The examples were routed using TimberWolf's global router and the OCTTOOLS[Spickelmier 88] detailed router. No net criticality information was passed to the global or detailed router. Hence, the improvements are solely due to placement modifications. The delay values shown are the worst path delays in ns which include cell delays on the path. OCTTOOLS could not handle larger examples of Table 1 on our machine.

In wire mode our results are slightly better than TimberWolf and in timing mode we improve the total delay by about 20% over TimberWolf at the cost of very small (3%) penalty in terms of area. Please note that the improvement in wire component delay of the critical paths is about 40% as wire delay are about 50% of the total delay. Run times for the examples are plotted in Figure 12.

For larger examples the run time for RITUAL is more than 10 times lower than TImberWolf. It can be seen from the graph that the run time increases almost linearly with example size for RITUAL. To get some insight into what RITUAL does to the critical paths see Figure 13. The figure shows a placement for a small example in Wire and Timing mode. The solid lines in the figure show the most critical path and the dotted lines show the fanout cells of the critical cells. It is clear from the example that in the timing mode the path has been straightened to a large extent almost approaching the minimum mahattan distance between the PI and PO. Another effect is to bring the fanout cells closer to the driver, thus reducing the critical net delays. This effect is obtained at a small cost in wirelength.

14 Conclusions

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 paper. 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.

References

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

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

[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.

[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.

[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.

[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.

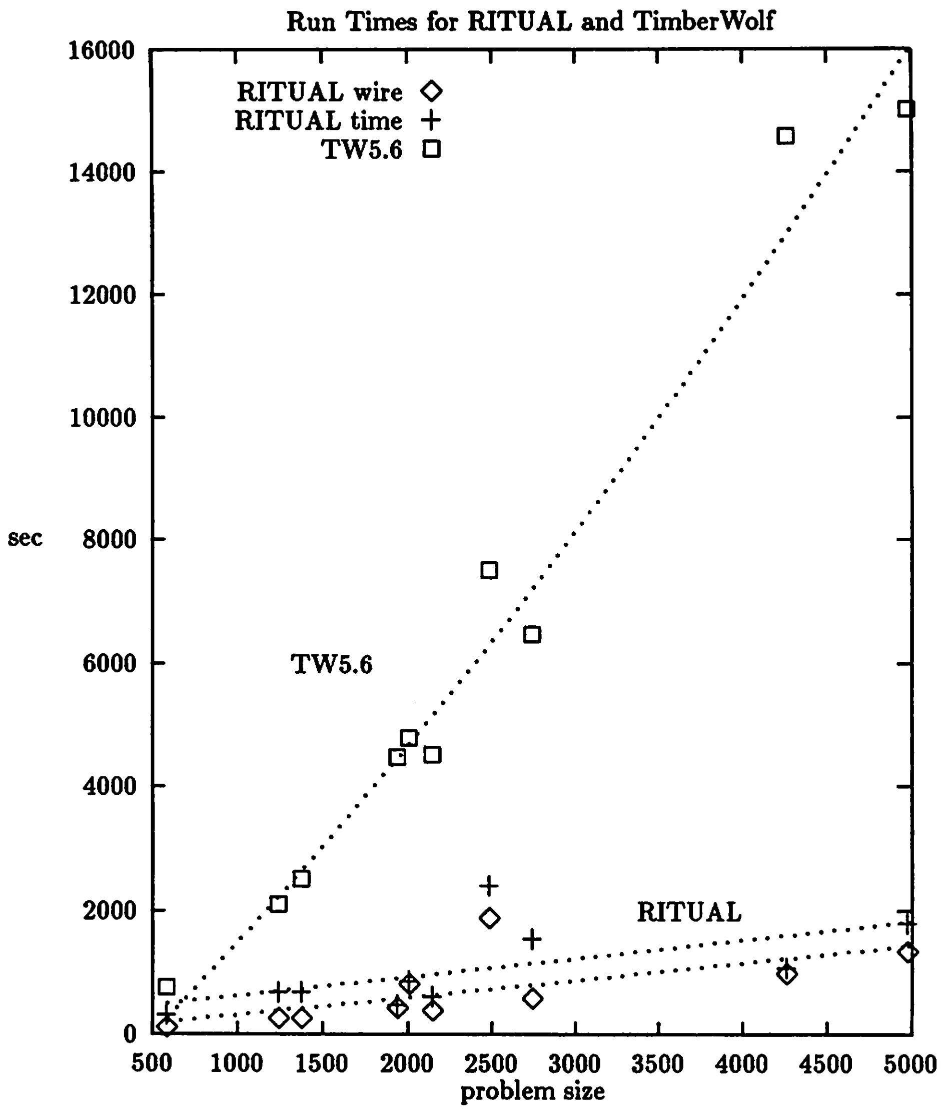

Figure 12: Figure showing the run time of RITUAL and TW5.6

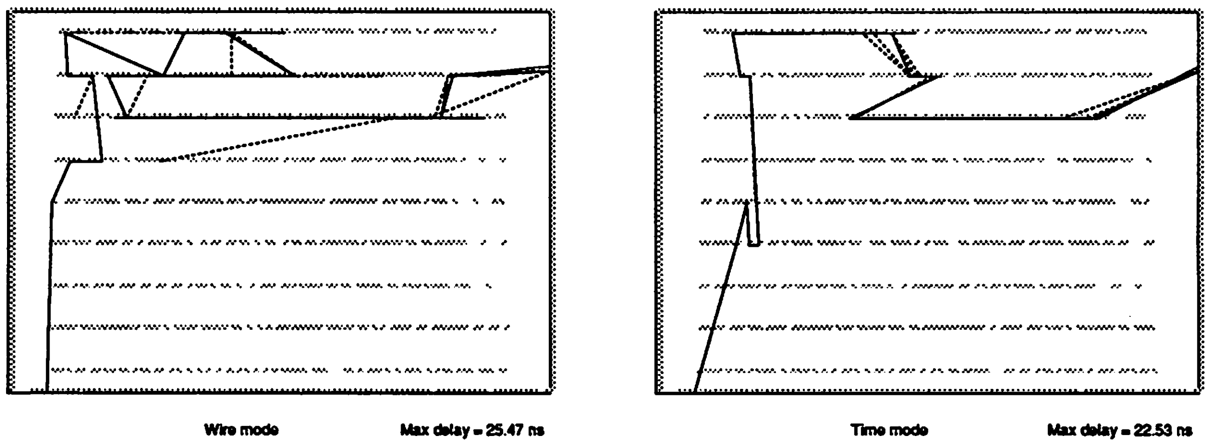

Figure 13: An example of a placement in Wire and Timing mode

[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.

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

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

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

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

[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.

[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.

[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.

[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.

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

[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.

[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.

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

[Nair 89] 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.

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

[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.

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

[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 88] 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.
[Spickelmier 88]	R. Spickelmier, editor. *Oct Tools Distribution 2.1.* University of California, Berkeley, March 1988.
[Srinivasan 91]	Arvind Srinivasan. *Performance Optimization of Large-Scale Integrated Circuits.* PhD thesis, University of California, Berkeley, 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.
[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.
[Vecci 83]	M. Vecci and S. Kirkpatrick. Global wiring by simulated annealing. *IEEE Trans. CAD*, CAD-2(4):215–222, October 1983.
[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.