Integration of Physical Design and Sequential Optimization

Philip Chong

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2006-21
http://www.eecs.berkeley.edu/Pubs/TechRpts/2006/EECS-2006-21.html

March 6, 2006

Integration of Physical Design and Sequential Optimization

by

Philip Chong

B.A.Sc. (University of Waterloo) 1996

A dissertation submitted in partial satisfaction of the
requirements for the degree of
Doctor of Philosophy

in

Engineering — Electrical Engineering and Computer Sciences

in the

GRADUATE DIVISION
of the
UNIVERSITY OF CALIFORNIA, BERKELEY

Committee in charge:
Professor Robert K. Brayton, Chair
Professor Richard Newton
Professor Robin Hartshorne

Spring 2006

The dissertation of Philip Chong is approved:

Chair Date

Date

Date

University of California, Berkeley

Spring 2006

Integration of Physical Design and Sequential Optimization

Abstract

Integration of Physical Design and Sequential Optimization

by

Philip Chong

Doctor of Philosophy in Engineering — Electrical Engineering and Computer Sciences

University of California, Berkeley

Professor Robert K. Brayton, Chair

This work examines the interaction between the physical design of digital integrated circuits and sequential optimization techniques used for performance enhancement. In particular, the integration of floorplanning and placement with retiming and clock skew scheduling is explored. A theoretical result is given which addresses the computational complexity of circuit partitioning under constraints derived from sequential optimization; this motivates the need for heuristic approaches to the related placement problem. Another theoretical result provides a characterization of the feasible retimings of a sequential circuit; this result is used to motivate an effective method for floorplanning integrated with sequential optimization. Practical techniques for using sequential slack to drive standard-cell placement are shown here; experiments demonstrate significant improvement in final design performance using these methods. Another part of this work examines how the role of sequential optimization and physical design changes when the design allows for asynchronous or latency-insensitive communication between modules. A theoretical result relating to the problem of clock tree implementation for clock skew scheduling under process variation is given. Finally an experimental technique for floorplanning using nonlinear programming is demonstrated.

Professor Robert K. Brayton
Dissertation Committee Chair

To Mom and Dad
And All My Friends

Contents

List of Figures

List of Tables

Acknowledgments

The Maximum Mean Cycle Partitioning problem presented in Chapter 2 was posed by Christoph Albrecht. I am indebted to Christoph for showing me this delightful problem and for all the insightful discussions on this and other material.

Chapter 4 was the result of a collaboration with Aaron Hurst and Andreas Kuehlmann. Thanks go to Aaron for the shared code and work on this project, and to Andreas for providing test cases and infrastructure support making this work possible.

Chapter 6 was part of the result of a larger collaboration with Christoph Albrecht, Aaron Hurst and Andreas Kuehlmann.

The work presented in this dissertation was supported in part by the MARCO/DARPA Gigascale Silicon Research Center (GSRC), the MARCO Focus Center for Circuit & System Solutions (C2S2), and the National Sciences and Engineering Research Council of Canada (NSERC).

Chapter 1

Introduction

1.1 Motivation

The progress of semiconductor fabrication technology is changing the impact of interconnect delay on the performance of digital integrated circuits. In the past, such delays were often ignored during the design process, as they were considered negligible compared to the delays of the gates in the circuit. However, today interconnect delay forms a substantial portion of the total circuit delay. [SK99] suggests between 24 and 36 percent of total circuit delay comes from interconnects in fabrication processes typically in use today.

Looking beyond today's processes shows an expected trend of increasing impact of interconnect delay. Table 1.1 shows the predicted evolution of interconnect delay based on projections from [ITR03]. The first row indicates the year of projection. The next three rows indicate the anticipated RC delay associated with 1mm lengths of Metal 1 wire, intermediate-length wire and global wire, respectively.

Of course, looking only at delay for a 1mm length of wire can be misleading, as this does not account for any change in gate sizing. As gate sizes shrink, a fixed length of wire will span an increasing number of gates and thus represent a more substantial interconnect. Put another way, if a design is implemented in an smaller technology, wire lengths will shrink accordingly.

To put this in proper context, we use the personal digital assistant system-on-a-chip (PDA SOC) design driver (also presented in [ITR03]) to normalize these figures. The PDA represents a typical design one might wish to implement on an ASIC. The projected clock frequency and process technology (feature size) for this application is indicated in the table. The last three rows of Table 1.1 shows the product of the RC delay constant (taken as ps/mm) for each of the three wire types with

Year	2003	2006	2009	2012	2015	2018
RC, 1mm Metal 1 (ps)	191	355	595	963	1510	2679
RC, 1mm Intermed. Wire (ps)	105	224	358	552	908	1582
RC, 1mm Global Wire (ps)	42	87	139	220	354	618
PDA SOC Design Driver						
Clock Freq. (MHz)	300	450	600	900	1200	1500
Process Tech. (nm)	101	90	65	45	32	22
RC-Size Product, Metal 1	0.0193	0.0320	0.0387	0.0433	0.0483	0.0589
RC-Size Product, Intermed. Wire	0.0106	0.0202	0.0233	0.0248	0.0291	0.0348
RC-Size Product, Global Wire	0.00424	0.00783	0.00904	0.00990	0.0113	0.0136

Table 1.1: ITRS interconnect technology projections. From [ITR03].

the feature size (in nm). Taking this product effectively normalizes the RC delay values in terms of the feature size. Of course this is not exact, as wire delay is not exactly a linear function of wire length. However, such a normalization is useful to roughly account for the expected decrease in feature size. Here we see that this normalized delay is expected to increase roughly threefold, looking forward to 2018.

On top of the purely physical effects of process scaling, designs are also being expected to run at increasingly faster rates (see the clock frequency row in Table 1.1). Furthermore, designs are growing larger and more complex as consumers demand more features in the products which contain these ICs. The combined effect of all these trends is to make it increasingly difficult for designers to achieve the circuit performance necessary.

This work focuses on two powerful techniques used by digital circuit designers for performance optimization. Retiming [LS83, LS91] and clock skew scheduling [Fis90, DS95] are methods which can be used to improve design performance. We call these *sequential optimization* techniques, as they require changing the nature of the sequential elements within the targeted design. Retiming involves changing the structural location of the sequential elements, while clock skew scheduling involves changing the relative clock skews between the sequential elements.

Of course, performance optimization requires accurate modeling and prediction of interconnect delays in the design. Since such delays are dependent on the length of the wires, this means that the problem of sequential optimization and the physical implementation of the design are closely interrelated. This indicates that an *integration* of physical design and sequential optimization is necessary to achieve good results with these techniques.

Our work focuses on the floorplanning and placement aspects of the physical design prob-

lem. Of course, other aspects of physical design, such as routing and clock tree implementation, can affect sequential optimization greatly. However, these are not the focus of this work. This thesis explores how sequential optimization can be integrated with floorplanning and placement tools, and presents new techniques for performance optimization of digital synchronous circuits using these ideas.

1.2 Existing Work

Placement and floorplanning are processes for determining non-overlapping locations for circuit elements on a silicon die while minimizing a given cost function. Placement is distinguished from floorplanning in that placement is the term generally used with objects at a fine-grained level (e.g. standard cells each representing a single gate of logic), while floorplanning involves much larger objects (e.g. macroblocks composed of hundreds of thousands of gates). A typical design flow might utilize both floorplanning and placement techniques, especially if the design is large and has been designed in a hierarchical fashion, or if the design is built from pre-existing macroblocks.

1.2.1 Placement

Placement is typically performed in two stages. The first, called *global placement*, concerns finding general locations for the standard cells on the overall die. The result from global placement may not have all overlaps between cells resolved, but the spreading of cells is sufficiently uniform to allow the next step, *detailed placement* or *legalization*, which transforms the nearly-legal result of global placement to a final legalized placement with no cell overlaps, to be performed with less cost of computation. However detailed placement generally focuses on optimizations localized to small areas of the die [DJS91, KMR04] or optimization with the objective to minimize total perturbation with respect to the global placement [BV04]. Therefore in this work we focus on global placement, as the greatest optimization potential lies in this step.

Placement has been well-studied in the past for several interesting cost functions. The most common cost function used in the literature is wirelength. Although wirelength by itself is not a very useful metric, it has the advantage of being relatively easy to optimize, and does represent a coarse measure of routing congestion [WS00]. Wirelength has also been observed to be roughly correlated with timing of the design [CKM$^+$99a].

Nearly all modern global placement techniques fall into two categories. The first category

contains what are known as *analytic placers*. These formulate the placement problem abstractly as a quadratic program as follows: Let $G = (V, E)$ be a graph representing the circuit, where the vertices represent the standard cells to be placed and the edges represent the interconnections between the cells. The goal is then to find locations $(x(v), y(v)) \in \mathbb{R}^2$ for all vertices $v \in V$ to minimize

$$\sum_{(u,v)\in E} (x(u) - x(v))^2 + (y(u) - y(v))^2$$

The cost function serves as a rough estimate for the wirelength. As the cell locations collapse onto a single point if all vertices are unconstrained, additional vertices with fixed locations are added to the problem. These extra vertices are typically taken to be I/O pins around the die boundary so that the movable cells will lie within the convex hull of the fixed pins.

The main difficulty with the analytic approach is that the placement result tends to be clustered in the center of the die and is insufficiently spread out for the subsequent detailed placement step. This has resulted in a focus in the literature on techniques for spreading the cells. Commonly this is done iteratively, either through the addition of "spreading forces" or extra fixed points [EJ98, HMS02, VC04] or using graph partitioning techniques to subdivide the die into disjoint areas and force spreading by assigning cells to these areas [TKH88, TK91a, KSJ88].

The partitioning-based cell spreading techniques eventually developed into the second category of global placement techniques, those known as *partitioning-based placers*. These forgo the use of the quadratic program altogether, and instead make use of recursively partitioning the netlist graph and assignment of partitions to disjoint die areas to enforce adequate cell spreading. Typically some form of the *mincut partitioning* problem is used, which is as follows: Find a partition (A, B) of the vertices V such that $|A| = |B|$ and the size of the set of cut nets

$$|\{(u,v) \in E : (u \in A \wedge v \in B) \vee (u \in B \wedge v \in A)\}|$$

is minimized. Usually a variant of the Fiduccia-Mattheyses heuristic is used to solve the graph partitioning problem [FM82, KAKS97, CKM99b]. Partitioning-based placers have shown reasonably good correlation between the use of the mincut partitioning heuristic and the wirelength of the final result [A$^+$99, CKM00].

1.2.2 Floorplanning

Floorplanning is chiefly distinguished from placement by the need to deal with large macroblocks. Unlike standard cell gates, which for a particular design are all chosen from a library

where all cells have uniform height, macroblocks can vary significantly in size, from thousands to hundred of thousands of gates. Moreover, floorplanning tools must account for *soft macroblocks*, which represent a subdesign of known logical structure, but without layout information. Such soft blocks will have a fixed area, but their aspect ratio may vary.

The literature has mainly focused on floorplanning as an optimization problem where the cost function is a linear weighted sum of the total die area (i.e. the sum of the macroblock areas plus any wasted space due to packing inefficiencies) and total wirelength of the buses connecting the macroblocks.

Numerous techniques have been presented in the literature for floorplanning. The oldest of these use *slicing tree* structures to represent the relative positions of the macroblocks [Bre77, Ott82, WL86]. A slicing tree is a binary tree graph where the leaves represent the modules to be placed and the internal nodes represent horizontal and vertical *cutlines* at the appropriate level of the hierarchy. Some newer approaches use non-slicing structures of various types to represent the relative positions [MFNK96, NFMK96, GCY99, H$^+$00].

The key feature of these floorplan representations is that they represent the layout of the macroblocks in a compact, easily manipulated form. For all these floorplan representations, efficient algorithms exist to convert the compact representation into actual locations and aspect ratios for the macroblocks. Thus the search space of feasible floorplans is reduced. In the literature, the optimization technique typically used is simulated annealing based on manipulations of the underlying floorplan representation.

1.2.3 Sequential Optimization

We consider two techniques for sequential optimization. The first is *retiming*, which is based on the observation that replacing registers located at the output of a gate with registers located at the input of the gate does not change the functionality of the circuit [LS83, LS91]. By applying such register movement operations, various optimizations can be achieved, such as minimizing the total number of registers or minimizing the clock period of the design.

Clock skew scheduling is the second technique for sequential optimization which we consider. This involves adjusting the delay (skew) of the clock signals to the individual registers of the design, in order to improve the overall circuit performance [Fis90, DS95]. [Fis90] observes that clock skew scheduling can be considered equivalent to retiming; instead of physically moving registers across gates, clock skew scheduling moves registers virtually by delaying their clock signal

appropriately.

There have been some efforts to integrate retiming and placement. [CL00, Lim00] addresses the problem of "physical planning" (partitioning in the context of geometric layout) under the freedom to perform retiming on the final design. There the notion of *sequential slack* is introduced, and an iterative net weighting scheme is used during partitioning. [YMS03] avoids iterative techniques by using slack budgeting. [SB02] demonstrates an approach to field-programmable gate array design which accounts for retiming. These techniques have various difficulties associated with them, and we contrast these with our work in Chapter 4.

1.3 Outline

Here we outline the structure of the subsequent chapters of this thesis and highlight the contributions presented therein.

Chapter 2 shows the $\mathcal{NP}$-completeness of a simple partitioning problem under sequential timing constraints. This result provides theoretical justification for our heuristic approach taken through the rest of the thesis. That is, this proof motivates the need to attack sequentially-based physical design problems with heuristic techniques, rather than seek exact optimal solutions. This key motivation was not provided in any of the existing works in this area.

Chapter 3 presents a novel proof for a theorem which characterizes all feasible retimings of a circuit. While an equivalent theorem was previously proven in [SSBSV92], our proof is significantly different, and provides a practical constructive technique for transforming a given retiming into any retiming compatible with the given one. This chapter also presents a retiming-aware floorplanning application which takes advantage of this theorem.

Chapter 4 addresses some shortcomings of the technique given in Chapter 3. The concept of *sequential slack* is presented, and sequential timing-aware placement techniques using heuristics based on this metric are given.

Chapter 5 extends our ideas to the domain of asynchronously-communicating systems. Here, adding latency to communication paths has no effect on the correctness of a design (unlike with synchronous systems, where excessive latency leads to incorrect results). However, latency does affect performance, resulting in a different optimization problem. We present a technique for quickly estimating the performance impact for a given assignment of latencies along interconnect, and use this to create a performance-driven floorplanning algorithm for asynchronously-communicating systems.

Chapter 6 discusses the problem of realization of clock trees for synchronous digital circuits. A sufficient condition for the solution to the optimal clock scheduling problem in the face of process variations is given.

Chapter 7 presents a new experimental technique for floorplanning using a general-purpose nonlinear programming package. While the results are inconclusive, there is still hope that the general optimization framework of nonlinear programming can allow more sophisticated modeling of the flexibility introduced by sequential optimization.

Chapter 8 summarizes the contributions presented in this thesis.

Chapter 2

On The Complexity Of Partitioning Under Sequential Timing Constraints

2.1 Introduction

Graph partitioning techniques form a critical core for many placement algorithms currently in use today. Some algorithms rely on partitioning alone [CKM00, Lim00, A$^+$99], others use partitioning as a subproblem [KSJA91, TKH88]. In all cases, partitioning is used to subdivide large intractable problems into smaller ones which can be more readily solved.

Partitioning is a graph-theoretic problem. In a circuit context, circuit elements (e.g. gates) are represented by vertices of the graph, and interconnections (e.g. wires) between the elements are represented by edges. Traditionally, partitioning has been applied in placement with minimization of the number of *cut edges* in the graph as the cost function. *Min-cut* partitioning has been found to be an effective heuristic for placement where the minimization of total wire length is the primary metric used to evaluate the final placed design [CKM00, A$^+$99]. Intuitively, this is due to how min-cut partitioning separates a network into localized subnetworks. Interconnects which are cut by the partition tend to become long global wires in the final design, whereas interconnects contained within a single subpartition tend to be shorter local wires. Thus there is generally a good correlation between the number of cut edges during partitioning and the total wire length in the final placement.

In practice, other considerations besides wirelength must be considered. Most notably, *performance* of the final placed design is often a critical concern. Historically, much of the research in performance-driven partitioning has been focused on combinational-timing driven placement.

However, recent work has dealt with partitioning for performance in a sequential setting, allowing for retiming and clock skew scheduling to take place [Lim00, PKL98].

A common approach for performance-driven partitioning is to use *net weighting* together with the usual min-cut partitioning techniques, in order to prevent critical nets whose delays have a large impact on performance from being cut by the partition. This heuristic net weighting technique was developed for combinational timing-driven partitioning, but has been adopted by researchers looking at sequential-timing driven partitioning as well [Lim00, PKL98]. However, the complexity of partitioning under sequential flexibility was previously not made clear, and the justification for adopting such a heuristic technique was not based on theoretical grounds.

In this chapter, we present a proof for the $\mathcal{NP}$-completeness of partitioning under a sequential performance metric. This shows that the adoption of such heuristic techniques is well-justified.

2.1.1 Definitions

In the following, let $G = (V, E)$ be a finite directed graph.

Definition 2.1 (Partition). A *partition* (A, B) of the vertices V is a pair of subsets $A \subseteq V, B \subseteq V$ such that $A \cup B = V$ and $A \cap B = \emptyset$.

Definition 2.2 (Cut Edge). Edge $(u, v) \in E$ is *cut* by partition (A, B) if either $u \in A$ and $v \in B$, or $u \in B$ and $v \in A$.

Definition 2.3 (Cycle). A *cycle* ℓ in G is a nonempty ordered list of edges

$$\ell = \langle (u_1, v_1), (u_2, v_2), \ldots, (u_{|\ell|}, v_{|\ell|}) \rangle \qquad (u_i, v_i) \in E, i \in \{1, \ldots, |\ell|\}$$

such that $v_i = u_{i+1}, i \in \{1, \ldots, |\ell| - 1\}$, $v_{|\ell|} = u_1$, and all the u_i are unique.

For simplicity, here the term cycle is used to refer to *simple cycles* (with unique vertices) only. Define $\mathcal{C}(G)$ to be the set of all cycles in G.

Definition 2.4 (n-Cycle). An *n-cycle* is a cycle with n edges (equivalently, n vertices).

Let $d : E \to \mathbb{R}$ be a labeling of the edges of G with real numbers. d represents the *delay* for signals which travel across the edges.

Definition 2.5 (Maximum Mean Cycle). The *maximum mean cycle* (MMC) of graph G under labeling d is

$$\mathrm{MMC}(G,d) = \max_{\ell \in C(G)} \frac{\sum_{(u,v)\in\ell} d(u,v)}{|\ell|}$$

Several algorithms are known for the efficient computation of the MMC in polynomial-time [DIG98, Kar78].

2.2 Maximum Mean Cycle Partitioning

Consider the following model of a gate-level network: let $G = (V,E)$ be a directed graph where V represents the registers of the network, and E the paths connecting these registers (possibly through combinational gates). The delay labeling d then represents the maximum combinational delay between the registers of the design.

Under this model, the maximum mean cycle $\mathrm{MMC}(G,d)$ represents the minimum clock cycle which can be achieved using clock skew scheduling techniques [Fis90, SBSV92, Szy92, DS95]. That is, the MMC becomes the performance metric to be minimized for the design.

The impact of partitioning on the MMC is modeled here in a simple fashion. A fixed extra delay is added to the the delay for every edge which is cut by the partition, while the delays for the remaining edges are not changed. This is somewhat analogous to the use of the min-cut metric for partitioning for wirelength; intuitively, the cut edges become global interconnects, and thus will incur extra delay, whereas uncut edges will tend to be more localized and thus faster.

2.2.1 Problem Definition

Formally, we define the *maximum mean cycle partitioning* (MMCP) problem as follows:

Input: A directed graph $G = (V,E)$ with edge delays $d : E \to \mathbb{R}$, bin size $k \in \mathbb{Z}$, and cut set delay $\delta \in \mathbb{R}, \delta \neq 0$.

Output: A partition (A,B) of V, where $|A| \leq k, |B| \leq k$, which minimizes $\mathrm{MMC}(G,d')$, where $d' : E \to \mathbb{R}$ is defined as

$$d'(e) = \begin{cases} d(e) + \delta & \text{if } e \text{ is cut by } (A,B) \\ d(e) & \text{otherwise} \end{cases}$$

The equivalent decision problem is:

Input: A directed graph $G = (V, E)$ with edge delays $d : E \to \mathbb{R}$, bin size $k \in \mathbb{Z}$, cut set delay $\delta \in \mathbb{R}, \delta \neq 0$, and target $T \in \mathbb{R}$.

Output: 1 if there exists a partition (A, B) of the vertices where $|A| \leq k, |B| \leq k$, such that $\mathrm{MMC}(G, d') \leq T$ with d' defined as above; otherwise 0.

From a practical perspective, using negative values for δ may not make sense, because, as noted, the cut edges are generally physically interpreted as corresponding to longer global wires which incur more delay, not less delay. However, from a theoretical standpoint, while the complexity proof for the case where $\delta < 0$ takes a similar general approach with the the case where $\delta > 0$, there are notable differences in the details. Therefore, for completeness, we present both cases in the following section. As well, the case $\delta < 0$ is somewhat simpler than the case $\delta > 0$, so observing the proof for the former may help understanding the proof for the latter.

2.3 Theoretical Results

2.3.1 MMCP With Negative Cut Set Delay Is NP-Hard

Theorem 2.6. *For $\delta < 0$, MMCP is $\mathcal{NP}$-hard.*

Proof of Theorem 2.6. $\mathcal{NP}$-hardness can be shown by reduction from 3SAT. The 3SAT problem is [PS98]:

Input: A set

$$X = \{x_1, \ldots, x_{|X|}\}$$

of boolean variables and a set

$$W = \{w_1, \ldots, w_{|W|}\}$$

of boolean clauses, such that

$$w_i = (y_{i1} \vee y_{i2} \vee y_{i3}) \qquad i \in \{1, \ldots, |W|\}$$

where each y_{ij} is either a boolean literal $x_{k_{ij}} \in X$ or its negation $\overline{x_{k_{ij}}}$, and $\{k_{i1}, k_{i2}, k_{i3}\}$ are distinct for any given i.

Output: 1 if there exists an assignment of the binary values $\{0,1\}$ to the variables of X such that the boolean expression

$$w_1 \wedge \ldots \wedge w_{|W|}$$

evaluates to 1; otherwise 0.

Given an instance of 3SAT, an equivalent MMCP instance with $\delta < 0$ can be constructed as follows. Here vertices of the graph are identified with tuples of set elements to facilitate identification of the MMCP graph vertices with elements from the original 3SAT instance.

Let $X_P = X \cup \{\mathbf{1}\}$; $\mathbf{1}$ represents the constant boolean value 1. Let $X_N = \{\overline{x} : x \in X_P\}$ be the set of literals generated by taking the negations of the elements of X_P. The negation of $\mathbf{1}$ is the constant boolean value $\mathbf{0}$. Assume WLOG that $\mathbf{0}$ and $\mathbf{1}$ are distinct from all other elements of $X_P \cup X_N$. For the directed graph $G = (V,E)$, take $V = (X_P \cup X_N) \times W$. That is, there is a vertex in V associated with every combination of literal (either positive or negative) and clause from the 3SAT instance.

Create a set of edges

$$E^1 = \{(u,v) : u = (x_k, w_s) \in V, v = (\overline{x_k}, w_t) \in V\}$$

E^1 consists of all edges created by connecting each vertex u to all vertices v where the literal associated with v is the negation of the literal associated with u.

Create a second set of edges E^2 where

$$E^2 = E^2_1 \cup \ldots \cup E^2_{|W|}$$

where for all $i \in \{1, \ldots, |W|\}$

$$\begin{aligned} E^2_i &= \langle (r_{i1}, r_{i2}), (r_{i2}, r_{i3}), (r_{i3}, r_{i4}), (r_{i4}, r_{i1}) \rangle \\ r_{i1} &= (y_{i1}, w_i) \in V \\ r_{i2} &= (y_{i2}, w_i) \in V \\ r_{i3} &= (y_{i3}, w_i) \in V \\ r_{i4} &= (\mathbf{0}, w_i) \in V \end{aligned}$$

That is, each E^2_i consists of a single 4-cycle, each of which is associated with the clause w_i; the vertices are chosen to be among those which correspond to the 3 literals which appear in clause w_i along with a vertex corresponding to the literal 0. The order the vertices appear in the cycle is not relevant here; simply take them in order of their appearance in the given clause. E^2 is then the union of all E^2_i.

Now take $E = E^1 \cup E^2$. Take $d(e) = 1$ for all $e \in E$, take $k = \frac{|V|}{2}$, take $\delta = -1$, and take $T = 1 - \frac{1}{|V|}$. This is now an instance of the MMCP decision problem.

An example is shown in Figure 2.1. Here the 3SAT formula $(x_1 \vee \overline{x_2} \vee \overline{x_3}) \wedge (x_1 \vee \overline{x_2} \vee x_3) \wedge (\overline{x_1} \vee x_2 \vee \overline{x_3})$ has been converted to an MMCP instance graph using the above procedure. To

simplify the figure, double-headed arrows indicate pairs of vertices which are mutually connected by edges (2-cycles) in E^1. Solid lines indicate edges from E^1 while dashed lines indicate edges from E^2. The reader may find it instructional to identify the cycles in E^2 in the figure which correspond to the clauses in the 3SAT instance.

Note that the above reduction of 3SAT to MMCP is polynomial time, since the size of the MMCP input graph G is polynomially bounded by the size of the 3SAT instance; $|V| = (2|X| + 2)|W|$ and $|E| = |E^1| + |E^2| = (2|X| + 2)|W| + 4|W|$. Therefore it only remains to be shown that the reduction is valid; that is, it must be shown that the MMCP instance gives a result of 1 iff the 3SAT instance gives a result of 1.

Suppose the 3SAT instance has a satisfying assignment of variables; that is, there is an assignment so that the output of the 3SAT problem is 1. Construct a partition (A,B) of V where A contains those vertices whose corresponding literal has boolean value 1 under the given assignment, and B contains those vertices whose corresponding literal has boolean value 0. Note that $|A| = |B| = \frac{|V|}{2} = k$.

Lemma 2.7. *Every cycle in G either contains an edge from E^1 or consists solely of edges from a single cycle E^2_i for some $i \in \{1, \ldots, |W|\}$.*

Proof of Lemma 2.7. If a cycle in G does not contain any edge from E^1, then it can only contain edges from a single E^2_i, as the vertices of E^2_i form disjoint sets. □

Lemma 2.8. *Suppose (A,B) is a partition derived from a satisfying 3SAT assignment as described above. For every cycle $\ell \in \mathcal{C}(G)$, the partition (A,B) cuts some edge in ℓ.*

Proof of Lemma 2.8. All edges of E^1 are cut by (A,B), since all such edges connect vertices associated with a literal to another vertex which is associated with the negation of that literal. Also, each E^2_i contains at least one cut edge; if this were not the case, then all vertices of E^2_i would lie in B, a contradiction, as at least one literal from the 3SAT clause w_i must have boolean value 1. Using Lemma 2.7, every cycle must have at least one cut edge. □

Lemma 2.9. *For every cycle $\ell \in \mathcal{C}(G)$,*

$$\frac{\sum_{(u,v)\in\ell} d'(u,v)}{|\ell|} \leq T$$

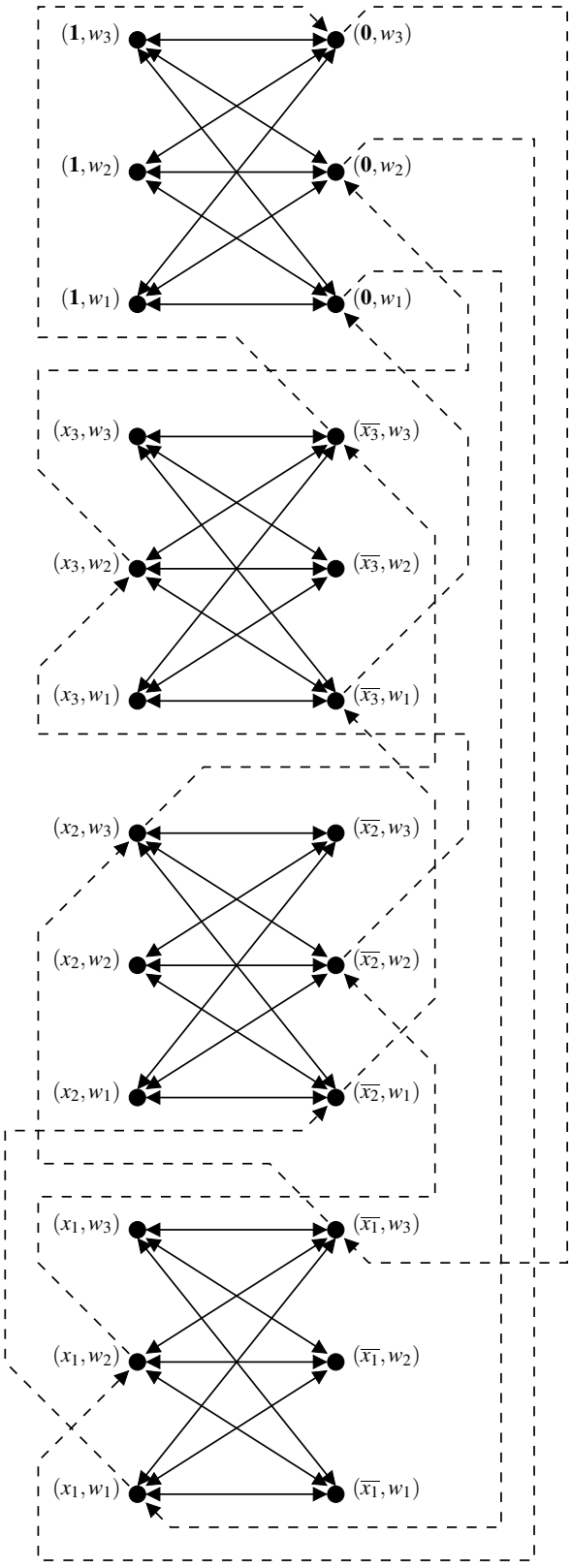


Figure 2.1: Example MMCP (negative cut set delay) instance from 3SAT reduction $(x_1 \vee \overline{x_2} \vee \overline{x_3}) \wedge (x_1 \vee \overline{x_2} \vee x_3) \wedge (\overline{x_1} \vee x_2 \vee \overline{x_3})$. Double-headed arrows indicate pairs of vertices mutually connected with edges (2-cycles). Solid lines indicate edges from E^1; dashed lines indicate edges from E^2.

Proof of Lemma 2.9. From the choices of d and δ,

$$d'(e) = \begin{cases} 0 & \text{if } e \text{ is cut by } (A,B) \\ 1 & \text{otherwise} \end{cases}$$

Then

$$\frac{\sum_{(u,v)\in\ell} d'(u,v)}{|\ell|} = 1 - \frac{c(\ell)}{|\ell|}$$

where $c(\ell)$ is the number of edges of ℓ which are cut. The result then falls from Lemma 2.8. □

By Lemma 2.9, $\mathrm{MMC}(G,d') \leq T$, so the partition (A,B) satisfies the the MMCP instance, and the MMCP instance generated from the 3SAT instance gives a result of 1.

Now for the converse; that is, if some partition (A,B) of V satisfies the constructed MMCP instance, then a satisfying assignment exists for the 3SAT instance.

Lemma 2.10. *Suppose (A,B) is a partition which satisfies all the conditions of the MMCP problem. For every cycle $\ell \in \mathcal{C}(G)$, the partition (A,B) cuts some edge in ℓ.*

Proof of Lemma 2.10. Again

$$d'(e) = \begin{cases} 0 & \text{if } e \text{ is cut by } (A,B) \\ 1 & \text{otherwise} \end{cases}$$

Now since (A,B) satisfies the MMCP instance,

$$\frac{\sum_{(u,v)\in\ell} d'(u,v)}{|\ell|} \leq T < 1$$

so some edge in ℓ must be cut by (A,B). □

Lemma 2.11. *For all $y \in X_P, w_a \in W, w_b \in W$, the vertex $u = (y, w_a)$ lies in A iff the vertex $v = (\overline{y}, w_b)$ lies in B.*

Proof of Lemma 2.11. From Lemma 2.10 and the fact that the edges

$$(u,v) \in E^1 \qquad \text{and} \qquad (v,u) \in E^1$$

form a cycle. □

Lemma 2.11 shows how to use the partition (A,B) to derive a satisfying assignment for the 3SAT instance. WLOG, assume the vertex $(\mathbf{1},w_1) \in A$. Then for every vertex in A associated with some positive literal, set the corresponding boolean variable to 1, and for every vertex in A associated with some negative literal, set the corresponding boolean variable to 0. By Lemma 2.11, this assignment is consistent: that is, if a variable is set to 1, all vertices associated with that variable in its positive sense must lie in A, and all vertices associated with that variable in its negative sense must lie in B. Likewise, for variables assigned to 0, all vertices associated with that variable in its positive sense must lie in B, and all vertices associated with that variable in its negative sense must lie in A.

Lemma 2.12. *The given assignment of variables satisfies the 3SAT instance.*

Proof of Lemma 2.12. Consider the cycle E_i^2. By Lemma 2.10, this cycle contains some cut edge. But by construction E_i^2 contains the vertex $(\mathbf{0},w_i)$, which lies in B (since $(\mathbf{1},w_1) \in A$, and Lemma 2.11 holds). Thus some vertex v in E_i^2 must lie in A, so the given 3SAT assignment must give the literal corresponding to v the boolean value 1. Therefore the 3SAT clause w_i is satisfied. □

From Lemma 2.9 and Lemma 2.12, the reduction from 3SAT to MMCP with $\delta < 0$ is shown to be valid. 3SAT is known to be $\mathcal{NP}$-complete [CLR90]. Therefore, under the restriction $\delta < 0$, MMCP is $\mathcal{NP}$-hard. □

2.3.2 MMCP With Positive Cut Set Delay Is NP-Hard

Theorem 2.13. *For $\delta > 0$, MMCP is $\mathcal{NP}$-hard.*

Proof of Theorem 2.13. Again by reduction from 3SAT. The construction of the MMCP instance differs in this case, however. Let $Y = X \cup \{\overline{x} : x \in X\}$ be the set of literals (both positive and negative) for all variables in X. Now define sets

$$W^1 = W \cup \{n_a, n_b, n_y, n_z\} \qquad \text{and} \qquad W^2 = (W \times \{1,2,3\}) \cup \{n_a, n_b\}$$

where $\{n_a, n_b, n_y, n_z\}$ are symbols distinct from the elements of $W \cup (W \times \{1,2,3\})$. These new symbols are used to introduce vertices which are not associated with any clause from the 3SAT instance. Also note the Cartesian product $W \times \{1,2,3\}$ is used to create a vertex associated with the individual literals in every 3SAT clause.

Construct two sets of vertices

$$V^1 = Y \times W^1 \qquad \text{and} \qquad V^2 = \{\mathbf{0},\mathbf{1}\} \times W^2$$

where again **0** and **1** represent the boolean constants 0 and 1, respectively. Let $V = V^1 \cup V^2$.

Construct two sets of edges

$$E^1 = \{(u,v) : u = (y,w_s) \in V^1, v = (y,w_t) \in V^1, w_s, w_t \in W^1, w_s \neq w_t\}$$
$$E^2 = \{(u,v) : u = (y,w_s) \in V^2, v = (y,w_t) \in V^2, w_s, w_t \in W^2, w_s \neq w_t\}$$

E^1 joins every vertex $u \in V^1$ with every other vertex $v \in V^1$ which is associated with the same literal as u, and likewise E^2 joins every vertex $u \in V^2$ with every other vertex $v \in V^2$ which is associated with the same literal as u.

Now construct a third set of edges

$$E^3 = E^3_{r1} \cup \ldots \cup E^3_{r|X|} \cup E^3_{s1} \cup \ldots \cup E^3_{s|X|}$$

where

$$\begin{aligned}
E^3_{rj} &= \langle (r_{j1}, r_{j2}), (r_{j2}, r_{j3}), (r_{j3}, r_{j4}), (r_{j4}, r_{j1}) \rangle && j \in \{1, \ldots, |X|\} \\
r_{j1} &= (x_j, n_y) \in V^1 && j \in \{1, \ldots, |X|-1\} \\
r_{j2} &= (x_{j+1}, n_a) \in V^1 && j \in \{1, \ldots, |X|-1\} \\
r_{j3} &= (\overline{x_j}, n_y) \in V^1 && j \in \{1, \ldots, |X|-1\} \\
r_{j4} &= (x_{j+1}, n_b) \in V^1 && j \in \{1, \ldots, |X|-1\} \\
r_{|X|1} &= (x_{|X|}, n_y) \in V^1 \\
r_{|X|2} &= (\mathbf{1}, n_a) \in V^2 \\
r_{|X|3} &= (\overline{x_{|X|}}, n_y) \in V^1 \\
r_{|X|4} &= (\mathbf{1}, n_b) \in V^2
\end{aligned}$$

and

$$\begin{aligned}
E^3_{sj} &= \langle (s_{j1}, s_{j2}), (s_{j2}, s_{j3}), (s_{j3}, s_{j4}), (s_{j4}, s_{j1}) \rangle && j \in \{1, \ldots, |X|\} \\
s_{j1} &= (x_j, n_z) \in V^1 && j \in \{1, \ldots, |X|-1\} \\
s_{j2} &= (\overline{x_{j+1}}, n_a) \in V^1 && j \in \{1, \ldots, |X|-1\} \\
s_{j3} &= (\overline{x_j}, n_z) \in V^1 && j \in \{1, \ldots, |X|-1\} \\
s_{j4} &= (\overline{x_{j+1}}, n_b) \in V^1 && j \in \{1, \ldots, |X|-1\} \\
s_{|X|1} &= (x_{|X|}, n_z) \in V^1 \\
s_{|X|2} &= (\mathbf{0}, n_a) \in V^2 \\
s_{|X|3} &= (\overline{x_{|X|}}, n_z) \in V^1 \\
s_{|X|4} &= (\mathbf{0}, n_b) \in V^2
\end{aligned}$$

E^3 is the union of $2|X|$ disjoint 4-cycles of two types, E^3_{rj} and E^3_{sj}. Each cycle E^3_{rj} is either an alternation between particular vertices from V^1 associated with the variable x_j in both positive and negative literal form and particular vertices associated with the positive literal x_{j+1} (for $j \in \{1,\ldots,|X|-1\}$), or E^3_{rj} is an alternation between particular vertices from V^1 associated associated with the variable x_j in both positive and negative literal form and particular vertices from V^2 associated with boolean constant 1 (for $j = |X|$). The cycles E^3_{sj} are similar to the cycles E^3_{rj}, except that they contain vertices associated with the negative literal $\overline{x_{j+1}}$ instead of the positive literal x_{j+1} (for $j \in \{1,\ldots,|X|-1\}$), or constant 0 instead of 1 (for $j = |X|$). The vertices chosen for these cycles are associated with the elements n_a, n_b, n_y, n_z so that each 4-cycle is on a set of vertices disjoint from any other 4-cycle in E^3.

Construct a fourth set of edges

$$E^4 = E^4_1 \cup \ldots \cup E^4_{|W|}$$

where, for all $i \in \{1,\ldots,|W|\}$,

$$\begin{aligned} E^4_i &= \langle (u_{i1}, v_{i1}), (v_{i1}, u_{i2}), (u_{i2}, v_{i2}), (v_{i2}, u_{i3}), (u_{i3}, v_{i3}), (v_{i3}, u_{i1}) \rangle \\ u_{ij} &= (y_{ij}, w_i) \in V^1, \qquad j \in \{1,2,3\} \\ v_{ij} &= (\mathbf{1}, (w_i, j)) \in V^2, \qquad j \in \{1,2,3\} \end{aligned}$$

E^4 is the union of disjoint 6-cycles E^4_i which connects vertices associated with 3SAT clause w_i. Each cycle alternates between vertices from V^1 associated with the literals which appear in w_i and vertices from V^2 which are associated with the boolean constant 1.

Finally, let $E = E^1 \cup E^2 \cup E^3 \cup E^4$. Take $k = \frac{|V|}{2}$, take $d(e) = 0$, take $\delta = 1$, and take $T = 1 - \frac{1}{|V|}$.

An example is shown in Figure 2.2. The figure shows the MMCP instance graph constructed from the 3SAT formula $(x_1 \vee \overline{x_2} \vee \overline{x_3}) \wedge (x_1 \vee \overline{x_2} \vee x_3) \wedge (\overline{x_1} \vee x_2 \vee \overline{x_3})$ using the above procedure. To simplify the figure, the edges from $E^1 \cup E^2$ are not drawn; instead these edges are represented by the shaded regions. For each such region, every pair of vertices contained therein is connected in a 2-cycle with edges from $E^1 \cup E^2$. Solid lines indicate edges contained in E^3, while dashed lines indicate edges contained in E^4. As with the previous example, the reader may find it instructional to identify the cycles in E^4 the figure which correspond to the clauses in the 3SAT instance. Examination of the two cycles of E^3 which connects vertices associated with the variable x_1 to those associated with x_2 may also be useful for edification.

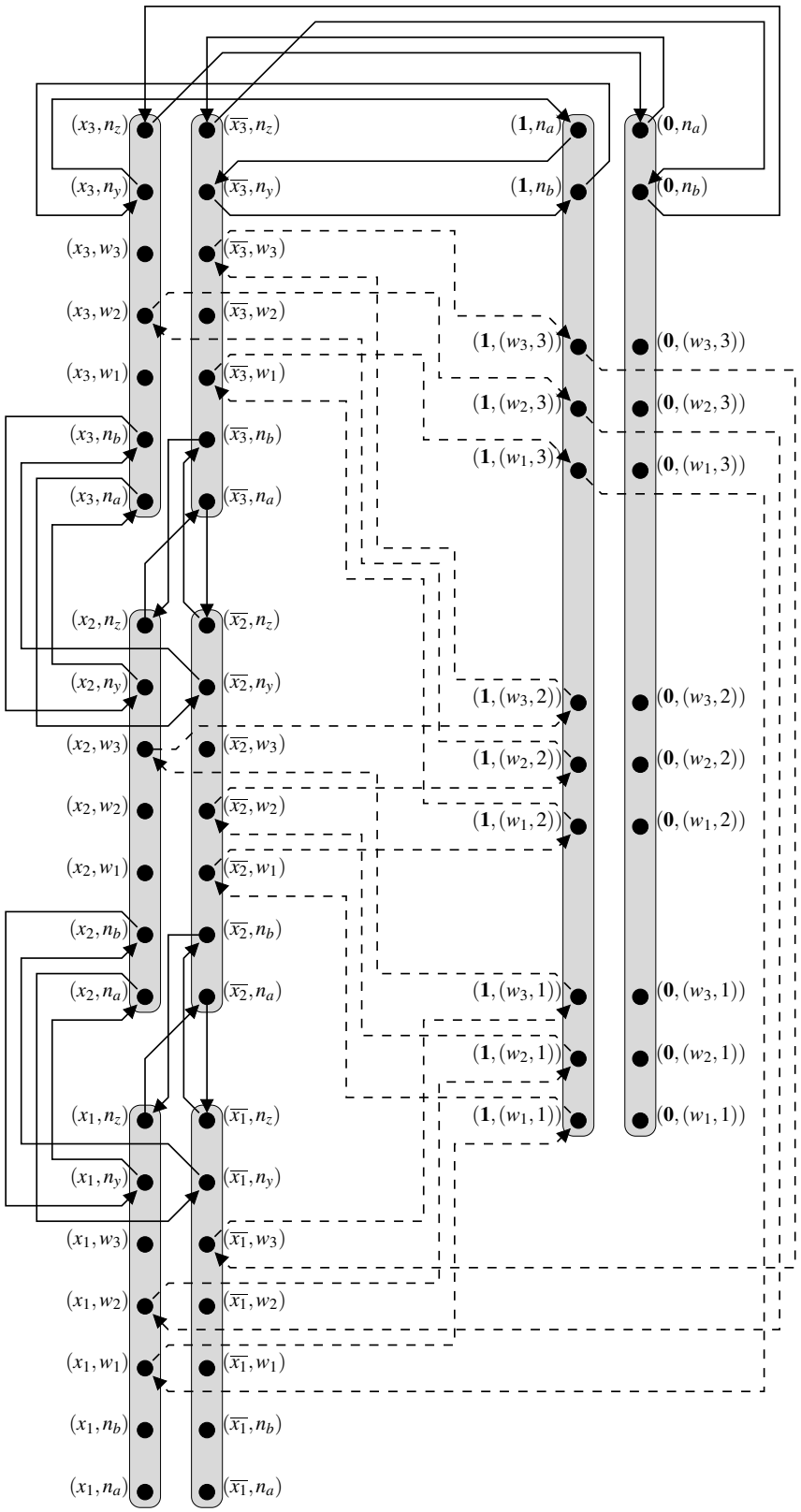


Figure 2.2: Example MMCP (positive cut set delay) instance from 3SAT reduction $(x_1 \vee \overline{x_2} \vee \overline{x_3}) \wedge (x_1 \vee \overline{x_2} \vee x_3) \wedge (\overline{x_1} \vee x_2 \vee \overline{x_3})$. Shaded regions indicate groups of vertices which are pairwise mutually connected with edges (2-cycles) from $E^1 \cup E^2$; solid lines indicate edges from E^3; dashed lines indicate edges from E^4.

It is easy to see that this reduction from 3SAT to MMCP can be done in polynomial time; the size of the generated graph is $|V| = 2|X|(|W|+4)+2(3|W|+2)$ and $|E| = |E^1|+|E^2|+|E^3|+|E^4| = 2|X|(|W|+4)(|W|+3)+2(3|W|+2)(3|W|+1)+4(2|X|)+6|W|$. Now it only remains to be shown that this reduction is valid.

Suppose the 3SAT instance has some satisfying assignment. As before, construct a partition (A,B) of V where A contains those vertices whose corresponding literal has boolean value 1 under the given assignment, and B contains those vertices whose corresponding literal has boolean value 0. Note that $|A| = |B| = \frac{|V|}{2} = k$.

Showing that this partition satisfies the MMCP instance is generally similar to the corresponding portions of the proof of Theorem 2.6, although the details differ.

Lemma 2.14. *Every cycle in G either contains an edge from $E^1 \cup E^2$ or consists solely of edges from a single cycle from $E^3 \cup E^4$.*

Proof of Lemma 2.14. If a cycle in G does not contain any edge from $E^1 \cup E^2$, then it can only contain edges from a single cycle from $E^3 \cup E^4$, as $E^3 \cup E^4$ is the union of disjoint cycles. □

Lemma 2.15. *Suppose (A,B) is a partition derived from a satisfying 3SAT assignment as described above. For every cycle $\ell \in \mathcal{C}(G)$, the partition (A,B) does not cut some edge in ℓ.*

Proof of Lemma 2.15. All edges of $E^1 \cup E^2$ are not cut by (A,B), since all such edges connect vertices associated with a literal to another vertex associated with the same literal.

Now each cycle in E^3 must have some edge which is not cut. Suppose this were not the case; that is, for some cycle ℓ in E^3, all edges of ℓ are cut. Since ℓ is a 4-cycle, there are two vertices in ℓ which lie on the same side of the partition, where one vertex is associated with some literal $y \in Y$ and the other is associated with $\overline{y}$, a contradiction since such vertices must have been assigned to different sides of the partition.

Finally, suppose some cycle $E_i^4 \subseteq E^4$ has no edge which is uncut. Since E_i^4 is a 6-cycle with every second vertex associated with the boolean constant 1, all other vertices of E_i^4 not associated with the boolean constant 1 must lie in B. But this means all literals in the 3SAT clause w_i were assigned the value 0, a contradiction, since this clause would then not be satisfied, and the assignment must satisfy the 3SAT formula.

From the above and Lemma 2.14, every cycle must have some edge which is not cut by (A,B). □

Lemma 2.16. *For every cycle $\ell \in \mathcal{C}(G)$,*

$$\frac{\sum_{(u,v)\in\ell} d'(u,v)}{|\ell|} \leq T$$

Proof of Lemma 2.16. From the choices of d and δ,

$$d'(e) = \begin{cases} 1 & \text{if } e \text{ is cut by } (A,B) \\ 0 & \text{otherwise} \end{cases}$$

Then

$$\frac{\sum_{(u,v)\in\ell} d'(u,v)}{|\ell|} = 1 - \frac{c(\ell)}{|\ell|}$$

where $c(\ell)$ is the number of edges of ℓ which are not cut. The result then falls from Lemma 2.15. □

Therefore the partition derived from the satisfying assignment for 3SAT is a satisfying partition for MMCP.

For the converse, suppose a partition (A,B) of V is given which satisfies the MMCP instance. The following shows that a satisfying 3SAT assignment can be derived from (A,B).

Lemma 2.17. *Suppose (A,B) is a partition which satisfies all the conditions of the MMCP problem. For every cycle $\ell \in \mathcal{C}(G)$, the partition (A,B) does not cut some edge in ℓ.*

Proof of Lemma 2.17. Again

$$d'(e) = \begin{cases} 1 & \text{if } e \text{ is cut by } (A,B) \\ 0 & \text{otherwise} \end{cases}$$

Now since (A,B) satisfies the MMCP instance,

$$\frac{\sum_{(u,v)\in\ell} d'(u,v)}{|\ell|} \leq T < 1$$

so some edge in ℓ must not be cut by (A,B). □

Lemma 2.18. *For all $y \in Y, w_a \in W^1, w_b \in W^1$, the vertex $u = (y,w_a) \in V^1$ lies in A iff the vertex $v = (y,w_b) \in V^1$ lies in A. Also, for all $y \in \{\mathbf{0},\mathbf{1}\}, w_a \in W^2, w_b \in W^2$, the vertex $u = (y,w_a) \in V^2$ lies in A iff the vertex $v = (y,w_b) \in V^2$ lies in A.*

Proof of Lemma 2.18. Directly from Lemma 2.17 and the fact that $E^1 \cup E^2$ contains the 2-cycle $\langle (u,v),(v,u) \rangle$. □

So far, this proof has been generally similar to that of Theorem 2.6. However, unlike the previous proof, we have not yet demonstrated that a consistent assignment of SAT variables can be obtained. The following lemma shows that the vertices associated with any given literal must lie on the opposite side of the partition from the vertices associated with the negation of that literal, due to the particular structure of the cycles in E^3.

Lemma 2.19. *For all $y \in X, w_a \in W^1, w_b \in W^1$, the vertex $u = (y, w_a) \in V^1$ lies in A iff the vertex $v = (\overline{y}, w_b) \in V^1$ lies in B. Also, for all $w_a \in W^2, w_b \in W^2$, the vertex $u = (\mathbf{0}, w_a) \in V^2$ lies in A iff the vertex $v = (\mathbf{1}, w_b) \in V^2$ lies in B.*

Proof of Lemma 2.19. First consider variable $x_1 \in X$, and suppose $u = (x_1, w_a) \in V^1$ lies on the same side of the partition $v = (\overline{x_1}, w_b) \in V^1$ for some $w_a, w_b \in W^1$. WLOG, suppose $u, v \in A$. Then by Lemma 2.18, the vertices

$$u^1 = (x_1, n_y) \in V^1 \qquad \text{and} \qquad v^1 = (\overline{x_1}, n_y) \in V^1$$

must both lie in A as well.

Now consider the 4-cycle E^3_{r1}. This cycle contains the vertices

$$u^{1*} = (x_2, n_a) \in V^1 \qquad \text{and} \qquad v^{1*} = (x_2, n_b) \in V^1$$

In particular,

$$E^3_{r1} = \langle (u^1, u^{1*}), (u^{1*}, v^1), (v^1, v^{1*}), (v^{1*}, u^1) \rangle$$

Since both $u^1 \in A$ and $v^1 \in A$, then by Lemma 2.17, either $u^{1*} \in A$ or $v^{1*} \in A$. But then by Lemma 2.18, $\{(x_2, w_a) : w_a \in W^1\} \subseteq A$. A similar argument using E^3_{s1} yields $\{(\overline{x_2}, w_a) : w_a \in W^1\} \subseteq A$.

Now vertex $u^2 = (x_2, n_y) \in A$ and $v^2 = (\overline{x_2}, n_y) \in A$. Examining E^3_{r2} and E^3_{s2} with the same argument as above yields the conclusion $\{(x_3, w_a) : w_a \in W^1\} \subseteq A$ and $\{(\overline{x_3}, w_a) : w_a \in W^1\} \subseteq A$. Continuing this reasoning, eventually the conclusion is that $A = V$, which is a contradiction, since (A, B) is a satisfying MMCP partition and $|A| \leq k = \frac{|V|}{2}$. Thus the original supposition that u and v lie on the same side of the partition must be false.

Now induction can be used. Given that the lemma is true over all vertices corresponding to variables x_j, $j \in \{1, \ldots, m\}$, the lemma must also be true for vertices corresponding to x_{m+1}, since assuming otherwise yields the contradiction $|A| > k$ using the same argument as above. Similarly, the lemma must hold true for the vertices in V^2. □

Lemmas 2.18 and 2.19 show that a consistent variable assignment for the 3SAT problem can be obtained from the given partition (A,B). WLOG suppose vertex $(\mathbf{1},n_a) \in A$. Then choose the 3SAT variable assignment such that the literal y takes on the value 1 iff $(y,n_a) \in A$.

Lemma 2.20. *The given assignment of variables satisfies the 3SAT instance.*

Proof of Lemma 2.20. Suppose some 3SAT clause w_i is unsatisfied. Then the cycle E_i^4 must alternate between vertices in A and vertices in B, since all literals $y_{ij}, j \in \{1,2,3\}$ for that clause take on the value 0. But then every edge in the cycle E_i^4 is cut, which contradicts Lemma 2.17. Therefore all 3SAT clauses must be satisfied. □

Lemma 2.16 and Lemma 2.20 show that the reduction from 3SAT to MMCP is valid, therefore, under the restriction that $\delta < 0$, MMCP is $\mathcal{NP}$-hard. □

2.3.3 MMCP Is NP-Complete

Theorem 2.21. *MMCP is $\mathcal{NP}$-hard.*

Proof of Theorem 2.21. Directly from Theorem 2.6 and Theorem 2.13. □

Strictly speaking, only one of Theorem 2.6 or Theorem 2.13 is necessary to prove Theorem 2.21. However, since the two cases have different physical interpretations, and the graphs used for each case differ significantly, both are presented here for completeness.

Theorem 2.22. *MMCP is in $\mathcal{NP}$.*

Proof of Theorem 2.22. For the decision problem of MMCP, if the output is 1 the partition (A,B) which satisfies $\mathrm{MMC}(G,d') \leq T$ serves as a certificate which can be checked in polynomial time, since the labeling function d' can be computed in polynomial time given (A,B), and $\mathrm{MMC}(G,d')$ can be computed in polynomial time as well. □

Theorem 2.23. *MMCP is $\mathcal{NP}$-complete.*

Proof of Theorem 2.23. From Theorem 2.21 and Theorem 2.22. □

2.3.4 Generalizations

Recall that the circuit model used in this chapter has registers represented by the vertices of G, and combinational gates abstracted into the delays represented by the labeling d. That is, the MMCP problem considered here was only that of partitioning the registers of the circuit. However, it is possible to extend the above result for the case where the combinational elements are also to be included in the partitioning problem. For this, the *maximum profit-to-time ratio* metric can be substituted for the MMC in the above. Suppose the edges of G are labeled with the function $\tau : E \rightarrow \mathbb{R}$ which represents the number of registers associated with that edge. Purely combinational circuit elements would have a corresponding $\tau(e) = 0$, while individual registers would have $\tau(e) = 1$. The maximum profit-to-time ratio is then

$$\max_{\ell \in C(G)} \frac{\sum_{(u,v)\in\ell} d(u,v)}{\sum_{(u,v)\in\ell} \tau(u,v)}$$

This metric is a generalization of the MMC and it is straightforward to extend the analysis in this chapter to use this more general circuit model instead [DG98, DIG98, CTCG+98].

It is also a straightforward extension to consider the case where the vertices have weights $s(v) : v \in V$ associated with them (i.e. corresponding to the sizes of their respective cells), so that the partition balance conditions become

$$\sum_{v\in A} s(v) \leq k \qquad \text{and} \qquad \sum_{v\in B} s(v) \leq k$$

However, the model using unweighted vertices provides some interesting insight into where the complexity of MMCP originates. The $\mathcal{NP}$-completeness of the traditional integer partitioning problem [GJ79] comes about because of the varying sizes of the elements to be partitioned. With equal element sizes, integer partitioning can be solved trivially. This suggests that the source of complexity of the MMCP problem arises fundamentally from the delay constraints, rather than the partition balance conditions.

It should be noted that the delay model used in the above analysis is very simplistic, in that cut edges are uniformly given a fixed additional delay δ. Certainly more sophisticated delay models such as the geometric embedding model proposed in [Lim00] would be more realistic. Although no straightforward extension of the above analysis to use such delay models is known, it is reasonable to believe that using a more complex delay model will not simplify the partitioning problem.

2.4 Summary

In this chapter, an optimization problem combining partitioning with sequential timing constraints was shown to be be $\mathcal{NP}$-complete. This motivates the need to develop heuristics for physical design under sequential timing constraints.

Chapter 3

Floorplanning With Retiming Constraints

3.1 Introduction

Retiming has the well-known property that, for any retiming, the number of registers in any structural cycle in the circuit must remain constant. The converse of this is not generally true, however. That is, given an original circuit and a target circuit which is both structurally identical and preserves the number of registers in every cycle in the underlying graph, it is not always possible to generate the target circuit from the original in a manner which preserves functionality. However, this converse property does hold for a specific class of circuits, in particular, circuits whose underlying graphs are strongly-connected.

In this chapter, a novel proof of this converse property is presented. A practical application of this theorem is shown in a non-iterative approach to combining retiming and die-level floorplanning for deep-submicron designs. Experimental results show notable improvement in clock cycle times achievable using this technique.

3.1.1 Definitions

For this chapter, we model designs using directed graphs with edge labels. The semantics of the graphs in this chapter differ from that of Chapter 2. Here, the edge labels count the number of registers or sequential elements present on the communication paths between the other circuit elements, which are represented by the vertices.

Let $G = (V, E)$ be a finite strongly-connected directed multigraph.

Definition 3.1 (Path). A *path* P of length $|P|$ in G is a nonempty ordered list of edges

$$P = \langle (u_1, v_1), (u_2, v_2), \ldots, (u_{|P|}, v_{|P|}) \rangle \qquad (u_i, v_i) \in E, i \in \{1, \ldots, |P|\}$$

such that $v_i = u_{i+1}, i \in \{1, \ldots, |P| - 1\}$,

Definition 3.2 (Cycle). A *cycle* in G is a path which begins and ends at the same vertex; that is, $v_{|P|} = u_1$ using the notation above.

The strongly-connected property of G is equivalent to asserting that every edge in E participates in at least one cycle.

Definition 3.3 (Self-Intersecting Cycle). A cycle which contains some edge more than once is said to be *self-intersecting*.

Note that here we use the term cycle to generally include self-intersecting cycles. This is in contrast to Chapter 2, where the term referred to simple (non-self-intersecting) cycles only. We will later see that we can actually ignore self-intersecting cycles for our purposes.

For any subset $V' \subseteq V$, define

$$\alpha(V') = \{(u, v) \in E : u \notin V', v \in V'\}$$
$$\beta(V') = \{(u, v) \in E : u \in V', v \notin V'\}$$

That is, $\alpha(V')$ is the set of edges which enter V' and $\beta(V')$ is the set of edges which leave V'.

Let $S : E \to \mathbb{Z}$ and $S' : E \to \mathbb{Z}$ be labelings of the edges of G with integers.

Definition 3.4 (Retiming Operation). For all $V' \subseteq V, x \in \mathbb{Z}$, we define the *retiming operation* $\Phi(V', x)$ to be the function $\Phi(V', x) : S \to S'$ such that

$$S'(e) = \begin{cases} S(e) + x & \text{if } e \in \alpha(V') \\ S(e) - x & \text{if } e \in \beta(V') \\ S(e) & \text{otherwise} \end{cases}$$

Our notion of retiming here is a somewhat generalized restatement of the original definition of retiming proposed in [LS91]. The original definition is equivalent to the above with the additional restriction $|V'| = 1$.

Definition 3.5 (Compatible Cycle). A cycle ℓ in G is *cycle compatible* with respect to S and S' iff

$$\sum_{e\in\ell} S(e) = \sum_{e\in\ell} S'(e)$$

Definition 3.6 (Compatible Labelings). S and S' are said to be *compatible labelings* iff all cycles in G are compatible with respect to S and S'.

For simplicity of notation, we make use of a particular labeling $T : E \to \mathbb{Z}$ of the edges of G with integers. We call T the *target labeling* for reasons which will become clear later.

Definition 3.7 (Edge Satisfaction). An edge $e \in E$ is *edge satisfied* by S iff $S(e) = T(e)$.

Definition 3.8 (Path Satisfaction). A path P is *path satisfied* by S iff

$$\sum_{e\in P} S(e) = \sum_{e\in P} T(e)$$

Note that the concatenation of two satisfied paths is also a satisfied path.

Definition 3.9 (Cycle Satisfaction). A cycle ℓ is *cycle satisfied* by S iff

$$\sum_{e\in\ell} S(e) = \sum_{e\in\ell} T(e)$$

Note that cycle satisfaction is simply the same as cycle compatibility with $S' = T$.

In the following, we will simply use the terms *compatible* and *satisfied* whenever the referred type is already clear from the context.

3.2 Theoretical Results

Claim 3.10. *If $V' = \{v_1, \ldots, v_{|V'|}\}$, then*

$$\Phi(V', x) = \Phi(\{v_1\}, x) \circ \ldots \circ \Phi(\{v_{|V'|}\}, x)$$

That is, the retiming operation $\Phi(V', x)$ can be formed by applying retiming on the individual vertices in V'.

Proof of Claim 3.10. Consider any edge $e = (u, v) \in E$. If $u \notin V', v \in V'$, then $e \in \alpha(V')$ and

$$\begin{aligned}\left\{\Phi(\{v_1\}, x) \circ \ldots \circ \Phi(\{v_{|V'|}\}, x)\right\}(S)(e) &= \Phi(\{v\}, x)(S)(e) \\ &= S(e) + x \\ &= \Phi(V', x)(S)(e)\end{aligned}$$

If $u \in V', v \notin V'$, then $e \in \beta(V')$ and

$$\begin{aligned}\left\{\Phi(\{v_1\},x) \circ \ldots \circ \Phi(\{v_{|V'|}\},x)\right\}(S)(e) &= \Phi(\{u\},x)(S)(e) \\ &= S(e) - x \\ &= \Phi(V',x)(S)(e)\end{aligned}$$

If $u \in V', v \in V'$, then

$$\begin{aligned}\left\{\Phi(\{v_1\},x) \circ \ldots \circ \Phi(\{v_{|V'|}\},x)\right\}(S)(e) &= \{\Phi(\{u\},x) \circ \Phi(\{v\},x)\}(S)(e) \\ &= S(e) - x + x \\ &= S(e) \\ &= \Phi(V',x)(S)(e)\end{aligned}$$

Finally if $u \notin V', v \notin V'$, then

$$\begin{aligned}\left\{\Phi(\{v_1\},x) \circ \ldots \circ \Phi(\{v_{|V'|}\},x)\right\}(S)(e) &= S(e) \\ &= \Phi(V',x)(S)(e)\end{aligned}$$

□

Claim 3.10 shows the equivalence of our notion of retiming with that of [LS91], in that the retiming operation Φ can be decomposed into the retiming transformations proposed in that work.

Let $S : E \to \mathbb{Z}$ and $S' : E \to \mathbb{Z}$ be labelings of the edges of G with integers.

Claim 3.11. *If S is compatible with S', then $\Phi(V',x)(S)$ is a labeling compatible with S' as well.*

Proof of Claim 3.11. Consider any cycle ℓ in G. $|\ell \cap \alpha(V')| = |\ell \cap \beta(V')|$, so

$$\sum_{e \in \ell} S(e) = \sum_{e \in \ell} \Phi(V',x)(S)(e)$$

Thus S is compatible with $\Phi(V',x)(S)$. Compatibility of labelings is an equivalence relation, so $\Phi(V',x)(S)$ must be compatible with S'. □

Claim 3.11 shows that the retiming operation preserves compatibility of all cycles. This result also falls out from the decomposition of retiming operations into single-vertex retiming operations (Claim 3.10) and [LS91].

Theorem 3.12. *Labelings S and S' are compatible iff all non-self-intersecting cycles ℓ in G are compatible with respect to S and S'.*

Proof of Theorem 3.12. One direction ("only if") is trivial. For the other direction ("if"), use induction on the length of a given cycle. For the base case, let ℓ be any cycle of length 2. ℓ cannot be self-intersecting, so ℓ must be compatible with respect to S and S'. Now consider a cycle ℓ of length $|\ell| > 2$, and suppose all cycles of length less than $|\ell|$ are compatible with respect to S and S' (induction hypothesis). If ℓ is non-self-intersecting, ℓ must be compatible with respect to S and S'. If ℓ is self-intersecting, there is an edge e_i which is visited at least twice, so we can write

$$\ell = \langle e_1, \ldots, e_{i-1}, e_i, e_{i+1}, \ldots, e_{j-1}, e_j = e_i, e_{j+1}, \ldots, e_{|\ell|} \rangle$$

for some $j \neq i$. We can then decompose ℓ into two cycles

$$\ell_1 = \langle e_1, \ldots, e_{i-1}, e_i, e_{j+1}, \ldots, e_{|\ell|} \rangle$$
$$\ell_2 = \langle e_{i+1}, \ldots, e_{j-1}, e_j = e_i \rangle$$

so that

$$\sum_{e \in \ell_1} S(e) + \sum_{e \in \ell_2} S(e) = \sum_{e \in \ell} S(e)$$
$$\sum_{e \in \ell_1} S'(e) + \sum_{e \in \ell_2} S'(e) = \sum_{e \in \ell} S'(e)$$

But ℓ_1 and ℓ_2 are cycles of length less than n, so by the induction hypothesis

$$\sum_{e \in \ell_1} S(e) = \sum_{e \in \ell_1} S'(e)$$
$$\sum_{e \in \ell_2} S(e) = \sum_{e \in \ell_2} S'(e)$$

and so ℓ must be compatible with respect to S and S'. □

Theorem 3.12 indicates that we can ignore self-intersecting cycles in our analysis, as compatibility of all simple cycles implies compatibility of all cycles, including self-intersecting cycles.

Theorem 3.13. *Let S and T be compatible integer labelings of the edges of G. Then there exists a finite sequence of successive retiming operations which satisfies all edges (i.e. transforms S to T).*

Proof of Theorem 3.13. For the following discussion, we introduce two auxiliary definitions.

Definition 3.14 (σ-Path). A σ-*path* of length n from v_1 to v_n with respect to S is an ordered list of vertices $\langle v_1, \ldots, v_n \rangle$, $v_i \in V$, such that, for all $1 \leq i \leq n-1$, there exists some edge $e_i \in E$ which is satisfied by S, where either $e_i = (v_i, v_{i+1})$ or $e_i = (v_{i+1}, v_i)$.

Note that if P is a σ-path exists from v_1 to v_n, this does not imply that P is a satisfied path with respect to S. Moreover, the existence of P does not even imply there exists a directed path from v_1 to v_n in the original graph G. However, P does correspond to a path in the underlying undirected graph between v_1 and v_n, where all the corresponding directed edges are satisfied by S.

Definition 3.15 (σ-Closure). For any $v \in V$, define the σ-*closure* of v with respect to S to be the set of vertices

$$K(S,v) = \{v\} \cup \{v' \in V : \text{there exists an } \sigma\text{-path from } v \text{ to } v' \text{ with respect to } S\}$$

Given these definitions we now proceed with the proof.

If S satisfies all edges, then $S = T$ and we are trivially done. Otherwise there exists some edge $e^* = (u^*, v^*) \in E$ for which $S(e^*) > T(e^*)$. Let $V^* = K(S, v^*)$.

Lemma 3.16. $u^* \notin V^*$.

Proof of Lemma 3.16. Suppose $u^* \in V^*$. Then by definition of σ-closure there exists an σ-path from v^* to u^*

$$P = \langle v_1 = v^*, \ldots, v_n = u^* \rangle$$

in G with respect to S. Now consider the construction of an ordinary directed path P' from v^* to u^* by examining every pair of vertices (v_i, v_{i+1}), $1 \leq i \leq n-1$ in the order they appear along P. Since P is an σ-path, there must exist some edge $e_i \in E$, $1 \leq i \leq n-1$, where e_i is satisfied by S and for which one of the following two cases must hold:

- Case 1: $e_i = (v_i, v_{i+1})$. In this case we append the edge e_i to P'.
- Case 2: $e_i = (v_{i+1}, v_i)$. Since every edge participates in at least one cycle, there is a cycle ℓ_i containing e_i. Since S and T are compatible, ℓ_i is cycle-satisfied by S. Since e_i is edge-satisfied as well, then the path $\ell_i \setminus \langle e_i \rangle$ (that is, the path obtained by removing the single edge e_i from ℓ_i) must also be satisfied. Instead of appending e_i to P', we append the path $\ell_i \setminus \langle e_i \rangle$.

Since P' is constructed by adjoining satisfied edges and satisfied paths, P' must be path-satisfied. Now appending e^* to P' gives a cycle ℓ', and ℓ' must be satisfied since S and T are compatible. But then $\ell' \setminus P' = \langle e^* \rangle$ must be satisfied as well, since both ℓ' and P' are satisfied. This is a contradiction, as e^* was chosen to be an unsatisfied edge. Thus $u^* \notin V^*$. $\square$

From Lemma 3.16, $u^* \notin V^*$. But $v^* \in V^*$, so $e^* \in \alpha(V^*)$. Now take $x = T(e^*) - S(e^*)$. Note that

$$\Phi(V^*, x)(S)(e^*) = S(e^*) + x = T(e^*)$$

so applying $\Phi(V^*, x)$ to S yields a labeling where e^* is satisfied.

Now we consider the other edges of E.

Lemma 3.17. *If edge $e' \in E$ is satisfied by S, then e' is also satisfied by $\Phi(V^*, x)(S)$.*

Proof of Lemma 3.17. Suppose $e' = (u', v') \in \alpha(V^*)$. By definition of α, $u' \notin V^*$ and $v' \in V^*$. But since $v' \in V^*$ and e' is satisfied by S, $u' \in V^*$, by definition of σ-closure. This is a contradiction, so $e' \notin \alpha(V^*)$. A similar argument shows $e' \notin \beta(V^*)$. Thus $\Phi(V^*, x)(S)(e') = S(e') = T(e')$. □

By Lemma 3.17, the number of edges satisfied by $\Phi(V^*, x)(S)$ must be strictly greater than the number of edges satisfied by S. Also, Claim 3.11 shows that $\Phi(V^*, x)(S)$ is compatible with T, since S is compatible with T. Therefore, the sequence

$$\begin{aligned} S_0 &= S \\ S_1 &= \Phi(V_0^*, x_0)(S_0) \\ S_2 &= \Phi(V_1^*, x_1)(S_1) \\ &\vdots \end{aligned}$$

converges to T in a finite number of steps, where V_i^* and x_i are the values of V^* and x^* computed according to the analysis above under the labeling S_i. That is, we can successively apply the retiming operation $\Phi(V^*, x)$ to S in order to obtain T. □

Theorem 3.13 indicates that, if a target retiming preserves the number of registers for every cycle in G, then there exists a finite sequence of retiming operations which can generate this target. Moreover, as the proof of this theorem is constructive in nature, we have a procedure to obtain the desired sequence of retiming operations. Thus, the constraint that the number of registers remain constant for all cycles in G is both necessary and sufficient for any valid retiming of G.

3.2.1 Existing Work

This theory was initially developed in the belief that it was a completely novel result in the area of retiming. However, it was subsequently found that the same result was derived in [SSBSV92]. Our proof of the necessary and sufficient conditions for valid retimings differs from

that given in [SSBSV92], most notably in two ways. First, [SSBSV92] only shows that the *temporality* (i.e. sequential behavior) of a circuit is maintained given the number of registers in any cycle is fixed. However, in that work it was not shown that a sequence of valid retiming operations exists which transforms a circuit into a target temporally-equivalent circuit. The proof presented here shows such a sequence does indeed exist. Second, our proof is constructive, so this sequence of retiming operations is explicitly determined. Our proof thus provides a more insightful view of this characterization of valid retimings.

3.2.2 Practical Considerations

One might argue that the condition that G be strongly-connected may be too restrictive in practice. However, note that in a design with no redundancies, every edge must reach some primary output by some path in G; edges which do not connect to a primary output can be removed without affecting the circuit behavior. Additionally, G can be easily modified so that all edges can be reached from some primary input. Thus, with suitable modification to G, edges which do not participate in a cycle must lie on some path from a primary input to a primary output. Hence, adding registers on such edges only contributes to the overall latency of the circuit, and does not change the sequential behavior of the circuit in other ways. By ignoring these edges during retiming (i.e. if we perform retiming on each strongly-connected component of G independently, we thus allow the overall latency to grow unbounded, but do not affect the functionality of the circuit beyond its latency.

An alternate approach to dealing with the condition that G be strongly-connected is to introduce an additional vertex v_{ext} as a "host node" in the graph, representing the environment external to the circuit. Additional edges from *vext* to the primary inputs of the circuit, and from the primary outputs of the circuit to *vext* are also added. Now in this new graph, all edges participate in some cycle with *vext*. This host node approach is commonly used in the literature, e.g. [SSBSV92, CL00].

Recall in the proof of Theorem 3.13 that the construction of the first retiming operation in the sequence which transforms the labeling S to the labeling T begins with the selection of some edge e^* such that $S(e^*) > T(e^*)$. By using this selection criterion, this ensures that the corresponding retiming operation $\Phi(V^*, x)$ which satisfies e^* has $x < 0$. This corresponds to a *forward* retiming operation, where registers are removed from the inputs of V^* and added to the outputs of V^*. Forward retiming operations are generally preferred in practice, as these can be easily realized. On the other

hand, *backward* retiming operations, where registers are removed from the outputs of V^* and added to the inputs of V^*, may not be implementable without modifying reset behavior [TB93, ESS96]. This criterion on the selection of e^* thus avoids the problems associated with backward retiming operations.

3.3 Floorplanning Application

Here we describe an application of our retiming theory to the generation of die-level floorplans under deep-submicron physical conditions. Under current technology trends, wire delays are predicted to grow such that multiple clock cycles will be required for die-level interconnects [Ass97]. Thus, the task of generating a feasible floorplan is tightly coupled with retiming, as the flexibility of placement of modules is directly related to the assignment of registers to the interconnects. That is, interconnects with more registers may be made longer than interconnects with few registers.

3.3.1 Problem Formulation

Designs are modeled as a directed multigraph G, where the nodes of the graph represent large macroblocks, possibly millions of gates each. Here, the blocks represent sequential logic, though we ignore the registers internal to the blocks, as we are only concerned with retiming the registers on the block-to-block interconnects.

Each macroblock n has an associated layout area A_n, though the aspect ratio (width/height) of each block may be flexible. As typical, we assume rectangular blocks only, for simplicity. The edges represent the interconnect between blocks; each edge may be a bus consisting of many wires, though for our purposes we abstract this detail away. We are also given the edge labeling $S(e)$, representing an initial assignment of registers to the interconnects; this represents an initial design/retiming which satisfies the functionality required by the system.

Our problem is thus: find a location (x_n, y_n) and width and height (w_n, h_n) for each macroblock n, and retiming $T(e)$ compatible with $S(e)$ such that

- $w_n h_n \geq A_n$
- No macroblocks overlap

- For edge $e = (m,n)$, $T(e) \geq \lfloor d(m,n)/\phi \rfloor$, where $d(m,n)$ is the interconnect delay from block m to block n, and ϕ is the clock period

We estimate the interconnect delay $d(m,n)$ using a number of assumptions. First, we assume interconnect delay is linearly related to distance, given optimally buffered lines as in [OB98]. Optimal buffering is a reasonable assumption at this level, given the long chip-level interconnect we are dealing with here. Second, we use the Manhattan distance $D(m,n)$ between the centers of the macroblocks as an estimate for the length of the interconnect. Third, we assume the macroblocks are all Moore machines, with registered outputs. This assumption is made to ensure that each interconnect can be treated independently; in the presence of combinational paths which span multiple chip-level interconnects, the inequality constraint for $T(e)$ given above does not necessarily hold. Given the large size of the macroblocks, it is reasonable to assume they will be designed in some regular fashion, so the Moore machine assumption seems to be justified. For simplicity, we assume our registers have no intrinsic propagation delays associated with them, although we can easily model such effects by subtracting such a delay from the clock period ϕ. Finally, we assume there is a combinational delay $t(m,n)$ associated with the inputs to block n coming from block m; this models the fact that, for a general Moore machine, there may be combinational logic between an input and a register. Based on this we have:

$$d(m,n) = \kappa D(m,n) + t(m,n)$$

for some constant κ relating distance to time; κ depends on the physical technology used for the chip itself.

3.3.2 Procedure

Given that *retiming constraints* (the number of registers around each cycle in G) are sufficient to describe all valid retimings of G, one might hope to translate these into placement constraints which are amenable for a floorplanning or placement tool to use. If the placement obeys such placement constraints, the end result will be guaranteed to have a valid retiming. Here we propose a heuristic method to try to satisfy the retiming constraints; as it is a heuristic, the final result may not satisfy all the constraints properly. However, note that we may always reduce the clock frequency in order to satisfy the retiming constraints; increasing the clock period allows violated constraints to become met. Thus, for our methodology, the maximum clock frequency (or minimum clock period) at which the retiming constraints are satisfied becomes a metric for the quality of the floorplan.

Our floorplanning procedure generates a slicing structure derived using recursive mincut partitioning [Bre77, Ott82, WL86]. The implementation for our algorithm uses `HMetis` [KK99] as the core mincut partitioner. We build the slicing tree recursively, using mincut partitioning to determine the structure of the tree at each level. Using mincut partitioning in this manner acts effectively as a heuristic for minimization of wirelength; as a minimum number of edges are cut at each stage, we expect that relatively few wires will have long lengths. This technique effectively implements the same wirelength minimization goals commonly used by other floorplanners.

To heuristically apply our retiming constraints to floorplanning, we adjust the weights on each of the edges in G in order to promote clustering of cycle within a single partition, rather than spreading cycles across partitions. Moreover, intuitively some cycles are more "critical" than others, in the sense that cutting such cycles during partitioning may lead to problems when generating the floorplan. We thus wish to avoid cutting edges which:

- Participate in many cycles
- Participate in cycles which have few registers
- Participate in cycles which have few modules

Empirical observation shows that the following formula gives good results in practice when used for the weighting of edge e:

$$w(e) = \sum_{\ell \in L(e)} M(\ell)/N(\ell)$$

where $L(e)$ is the set of all cycles in which edge e participates, $M(\ell)$ is the number of modules in cycle ℓ, and $N(\ell)$ is the number of registers available in cycle ℓ.

In our experiments, we generate two different floorplans. First we use the wirelength-driven floorplanning technique (i.e. using unweighted edges) to obtain a floorplan. Then we use the edge-weighted method to obtain a second floorplan. Comparing the two results thus give an indication of the improvement possible when using our retiming constraint-driven technique, as a measure of the merit of our method.

3.3.3 Related Work

[CL00, Lim00] proposes a method for simultaneous partitioning, floorplanning and retiming. Their algorithm `GEO` effectively interleaves timing analysis and retiming in a recursive top-down partitioning approach. While our technique shares a common goal, we differ primarily in

that we strive to fully decouple retiming from the task of floorplanning. In [Lim00], it is noted that retiming and timing analysis can be computationally expensive, and so its application is limited. Our work attempts to generate a viable floorplan independent of an explicit retiming, hence doing away with these costly procedures within floorplanning.

3.4 Experiment

Here we describe an experiment to validate our approach for floorplanning under retiming constraints.

3.4.1 Synthetic Benchmarks

Unfortunately, there are no freely available benchmark designs which reflect the large-scale system-on-a-chip designs which we are targeting. To remedy this, we have developed a synthetic benchmark generation technique suitable for the designs which we would like to see. There have been several attempts at generation of realistic synthetic benchmark circuits [DD96, HRGC98, SDC99], but all of these methods use particular features which reflect gate-level designs, rather than the macroblock examples we would like.

We have thus taken the method of [HRGC98] and modified it to reflect our goals. As in the original method, distributions of various quantitative metrics are extracted from a "seed" design, and these parameters are used to randomly generate new designs which have similar characteristics. Our method differs, though, in that the characteristic distributions we considered to be important are the number of output pins per block, degree of fanout per output, block size, and the size of cycles in the design. All but the last characteristic is generated directly from the seed design; in order to generate designs with the correct distribution of cycles, we use a ripup-and-retry technique similar to [HRGC98]. Algorithm 3.1 shows the algorithm we use for this process.

In assigning block sizes, we first obtain the distribution of block sizes from the seed design and fit a binomial distribution curve to these statistics. Block sizes for the generated circuits are generated randomly using this binomial distribution. This curve fitting is done so that the block sizes for our generated circuits have a "smoother" distribution over the full range of the possible sizes.

For each block, we assign it a random fanout count based on the distribution of fanouts from the initial seed design. We do not fit this to a binomial distribution as the fanout counts tend

Algorithm 3.1 Synthetic Benchmark Generation

```
1: Input: initial seed design and target design size n
2: Obtain block size, fanout and cycle count statistics from seed
3: Generate n blocks with random sizes and fanout counts
4: repeat
5:   for i = 2 to max_cycle_size do
6:     if too many cycles of size i then
7:       Choose an edge e of some cycle of size i
8:       Randomly change target of e to some other vertex
9:     else if too few cycles of size i then
10:      Choose a random vertex u
11:      Find a path of length i − 1 from u to some vertex v
12:      Choose a fanout edge of v and change its target to u
13: until no change
```

to be small, discrete values, unlike the area statistics. The target block for the fanouts are initially assigned randomly.

After the initial block generation, the algorithm iterates over all sizes of cycles (i.e. ranging from 2 to the size of the largest cycle in the design). For each cycle size, a target range is obtained by using the number of cycles of that size present in the seed design and allowing for up to 10% variation. If the number of cycles of the given size lies above this range, edges in the graph are modified to remove a cycle of this size. Likewise, if the number of cycles of the given size lies below this range, edges in the graph are modified to add a cycle of this size. In either case, the fanout characteristics of the generated design are not changed.

Although there is no guarantee of convergence with this algorithm, since adding or removing a cycle by modifying an edge can change other cycles in which that edge participates, we found in practice this approach worked well enough for our purposes; runs which failed to terminate in reasonable time could simply be ignored. The 10% variation in the target cycle size distribution can be increased if faster termination is desired, or decreased if closer fidelity to the original seed design is a goal.

Note that the process for generating synthetic designs does not account for $S(e)$, the initial retiming labeling. We generate $S(e)$ for the synthetic circuit by generating a slicing structure floorplan using recursive mincut partitioning, then assigning registers to edges in proportion to their

corresponding edge lengths in the resulting layout. This was done to approximate how such a system might be designed in real life; communication overhead between tightly-coupled blocks would tend to be minimized for performance reasons.

3.4.2 Technology Assumptions

An abstract top-level description of the Alpha 21264 processor was used as the seed design for our experiments in this paper. Although we do not possess an actual Alpha design, high-level architectural descriptions provided information about the basic blocks (functional units, caches, etc.) and their interconnectivity, and a chip micrograph provided information about the relative areas of the blocks [Kes98, Alp00]. A description for this design is shown in Table 3.1. Note the intent here is not to produce an exact replica of the Alpha processor, but rather to provide an abstract but generally realistic model of a modern IC design. We believe that the interconnectivity between blocks for this design will be similar to the high-level designs we will face in the next decade.

Eight benchmarks were synthesized from the Alpha design, ranging from 24 to 32 macroblocks in the generated designs; the original design had 24 blocks. Areas for the blocks were scaled so that the final designs would fit on a square die approximately $24mm$ per side. Note that the relationship between distance and delay is dependent on the physical characteristics of the die; here we need to make some rough estimates for future technology. We assume that the linear propagation constant for long optimally-buffered interconnect is $16\mu m/ps$; note that this yields a delay from corner-to-corner on the die of $3000ps$ (6 clock cycles at $2GHz$). We also assume $t(m,n) = 100ps$ as the input-to-register delay, uniformly for all macroblock inputs.

3.4.3 Results

Table 3.2 shows the results obtained from the eight synthetic benchmark designs generated as described above. The columns labeled *Normal* indicate the layout results using an ordinary floorplanning technique which generates a slicing structure using mincut partitioning. The mincut heuristic used here emulates the wirelength minimization goal typical of current state-of-the-art floorplanning tools. The ϕ column indicates the minimum clock period obtained using the normal technique. The *wlen* column indicates the total length of the macroblock interconnects, estimated using the half-perimeter bounding box model for the nets. The columns labeled *Cycle* indicate the same results using the retiming constraint edge-weighting method described in Section 3.3.2. The

Module Name	Estimated Size	Aspect Ratio	Net Source	Net Sinks	Net Source	Net Sinks
IC1	2.9M	0.73	PC1	IC1 ITB1	IRF2	IU2
DC1	2.8M	0.82	IC1	IM1 FPM1	ALU1	IRF1
FPMUL1	725k	0.61	IM1	IC1	IRF1	ALU1
IQ1	617k	0.50	FPM1	IC1	ALU2	IRF2
LSRU1	612k	0.78	IC1	INT1	IRF2	ALU2
INT1	596k	0.79	INT1	IM1 FPM1	ALU1	MB1
MB1	586k	0.61	IM1	IQ1	ALU2	MB1
FPQ1	515k	0.81	FPM1	FPQ1	IRF1	IRF2
FPM1	515k	0.81	ITB1	PC1	IRF2	IRF1
PC1	488k	0.91	BP1	PC1	MB1	IRF1 IRF2 FPRF1
IM1	432k	0.71	IQ1	IU1	IRF1	MB1
FPADD1	429k	0.97	IQ1	IU2	IRF2	MB1
DTB1	419k	0.74	IQ1	ALU1	FPRF1	MB1
DTB2	419k	0.74	IQ1	ALU2	MB1	DTB1 DTB2 LSRU1
ALU1	404k	0.54	FPQ1	FPADD1	DTB1	MB1
ALU2	404k	0.54	FPQ1	FPMUL1	DTB2	MB1
BP1	337k	0.53	FPQ1	FPDIV1	LSRU1	MB1
FPRF1	296k	0.67	FPADD1	FPRF1	MB1	INT1
IU1	290k	0.75	FPMUL1	FPRF1	INT1	MB1
IU2	290k	0.75	FPDIV1	FPRF1	DC1	MB1
ITB1	284k	0.56	IU1	IRF1	MB1	DC1
FPDIV1	252k	0.57	IRF1	IU1	DC1	INT1
IRF1	217k	0.91	IU2	IRF2	INT1	DC1
IRF2	217k	0.91				

Table 3.1: Abstracted Alpha 21264 design statistics. Module sizes (transistor count) estimated from areas measured from chip micrograph. Netlist derived from functional description.

Improvement column gives the percentage improvement (reduction in clock period or wirelength) from the normal layout scheme to the cycle-constrained method.

	Normal		Cycle		Improvement	
	ϕ	wlen	ϕ	wlen	ϕ	wlen
Design	(ps)	(mm)	(ps)	(mm)	(%)	(%)
1	913	388	609	402	33.3	-3.6
2	589	544	530	582	10.0	-7.0
3	584	602	568	633	2.7	-5.1
4	814	543	675	597	17.1	-9.9
5	611	398	562	432	8.0	-8.5
6	657	568	625	623	4.9	-9.7
7	579	608	768	685	-32.6	-12.7
8	766	807	688	935	10.2	-15.9
Average	689	557	628	611	8.9	-9.7

Table 3.2: Results for edge-weighted floorplanning using retiming constraints. *Normal* indicates unweighted floorplanning results, *Cycle* indicates edge-weighted floorplanning results, *Improvement* indicates the improvement seen. ϕ indicates the cycle time of the floorplanned design, *wlen* indicates the wirelength.

On average, our edge-weighting technique decreases the clock cycle for the floorplanned designs by 8.9%, while wirelength increases by 9.7%. The wirelength increase is, of course, expected, as our optimization goal differs from the traditional wirelength metric. These results show that the wirelength penalty is roughly commensurate with the performance improvement.

Note that, for all but Design 7, using the edge-weighting heuristics improves the clock period obtained for the design. For this particular circuit, we found that the critical cycle (i.e. the cycle which constrains the clock period) is broken early in the slicing tree, and hence overly lengthened, using our heuristics. As well, this critical cycle has no extra registers available for retiming, which further constrains the clock period.

One difficulty with the heuristic edge weighting presented here is that it depends on enumerating all simple cycles of the design. The number of cycles is potentially exponential in the number of edges in the graph. For high-level floorplanning tasks, this may not be a problem, as there are relatively few macroblocks and hence relatively few edges. Additionally, typical designs are expected to have fewer cycles than the theoretical worst case. This is demonstrated by the synthetic benchmarks generated here; these designs only have several thousand cycles in their structural graphs apiece. However, the potentially large number of cycles may pose a problem for a standard-cell placement algorithm which must be able to deal with thousands or millions of placeable objects.

One possible way to avoid enumerating all simple cycles is to only consider the *fundamental cycles* of the graph, a technique suggested in [SSBSV92]. Since the number of fundamental cycles is generally much smaller than the number of simple cycles in a graph, this may be an effective technique for avoiding this problem. However, we would have to adjust our weighting heuristic somehow, as we currently use the number of simple cycles in which an edge participates as an implicit part of the weighting function. We address the potential explosion in the number of cycles in Chapter 4. There, heuristic techniques for placement using physical cycle constraints which do not require explicit enumeration of all cycles in the design are given.

3.5 Summary

In this chapter, a new proof for the characterization of valid retimings was given. This proof motivates a heuristic edge weighting technique for floorplanning which captures the flexibility of retiming along global interconnects. Experimental results show these heuristics can be used to yield a tradeoff between design performance and total wirelength.

Chapter 4

Placement Driven By Sequential Timing Analysis

4.1 Introduction

Chapter 3 presented a heuristic technique for incorporating retiming constraints into a floorplanning tool. As previously noted, a potential drawback with the proposed heuristic is its reliance on enumerating all simple cycles in the structural graph representing the design. While this may be acceptable for floorplanning techniques which deal with relatively small numbers of objects, for standard cell placement such a heuristic becomes intractable. This chapter presents two approaches for dealing with placement in the context of sequential optimization which avoids the problem of cycle enumeration. Experimental results on academic and industrial designs indicate these techniques yield significant performance benefits for the final designs after sequential optimization.

4.1.1 Background

Sequential optimization techniques have the potential to significantly improve the performance, area, and power consumption of a circuit implementation to a degree that is not achievable with combinational synthesis methods. The goal is to balance the path delays between registers and thus to maximize the circuit performance without changing its input/output behavior.

Practical sequential optimization methods of interest are retiming [LS83, LS91] and clock skew scheduling [Fis90]. Retiming is a structural transformation that moves the registers in a circuit

without changing the positions of the combinational gates. Retiming, although algorithmically well studied, has gained only limited use because of its impact on the verification flow and the inability to accurately model the load changes caused by register moves. When applied before placement, retiming can perform a coarse balancing of paths delays using wiring estimates. In-place retiming is applied after physical placement and incrementally repositions individual registers based on a precise evaluation of the timing impact including the change of interconnect delays. In-place retiming is limited to local perturbations of the placement and cannot correct for global problems.

Clock skew scheduling preserves the circuit structure by applying non-zero delays to the register clocks — thus virtually moving them in time. In recent years, clock skew scheduling has gained practical acceptance in multiple design flows, typically applied as a post-placement optimization technique. Implementation strategies vary from clock tree topology construction [HKM$^+$03, KF99] to clock tree routing algorithms [XD97]. Recent work on multi-domain clock skew scheduling [RKS03] has demonstrated that even with a very limited number of skew possibilities, almost all of the benefits can be realized; implementing a non-zero clock skew schedule in hardware may be easier than commonly believed. For the purposes of this work, however, we remain implementation independent.

The optimization potential of retiming and clock skew scheduling is bounded by the *critical cycle*, which is among all structural cycles of a circuit the one with the maximum value for $\phi_{cycle} = total\_delay / number\_registers$. If the combinational delays of all paths along this cycle are perfectly balanced by retiming or clock skew scheduling, then the design can be clocked with a period $\phi \geq \phi_{cycle}$.

Traditional timing-driven placement (e.g. [SCK92]) minimizes the overall wire-length with the additional constraint that no combinational path is timing critical, i.e. the sum of the gate and interconnect delays on every path between two registers does not exceed the clock period ϕ. This notion of timing criticality is confined to the paths between single sets of registers and does not adequately capture the timing picture of the circuit when registers positions can be moved. A cyclic set of paths may be non-critical with respect to a flexible register clocking even if some of the individual paths significantly exceeds the cycle period. Likewise, combinational paths that have significant combinational slack may be part of the critical cycle. The combinational delay view of a design does not reveal much information about its true sequential criticality and may easily mislead the placer with respect to the overall optimization problem.

4.1.2 Existing Work

Previous work has addressed the integration of selective sequential optimization techniques and placement, but these attempts remain incomplete. [CL00, Lim00] presents a partitioning and floorplanning approach which considers retiming moves for registers on long interconnects. There, the concept of *sequential slack* is used to express the sequential mobility of a register. These slacks are used as weights in the partitioning algorithm to approximate the sequential criticality of an edge. This work is extended in [CY03] for a multilevel placement algorithm using simulated annealing. However, as discussed in Section 4.3, the sequential slack may dramatically underestimate the edge criticality in the presence of multiple critical regions and thus lead to suboptimal placements; the version of sequential slack computation using the reference point of [CL00] entirely misses the critical cycle in several of our examples. The first version of our algorithm presented in Section 4.4.1 builds on this notion of sequential slack. Unlike the work in [CL00], we guarantee that the most critical cycle will always be identified, since we choose a register which lies on this cycle as the reference point for the sequential slack computation. The second version of our algorithm presented in Section 4.4.2 corrects the problem for all potentially critical cycles by introducing explicit wire-length constraints for circuit loops that are near critical. We use Lagrangian relaxation to handle these constraints in an analytical placement phase similar to the approach presented in [SCK92]. Repeating the timing analysis after each placement iteration additionally improves the odds that such cycles are identified and algorithmic convergence is achieved.

In [YMS03] a budgeting algorithm is presented that computes delay bounds for a traditional placement algorithm under the assumption that retiming can be applied. In contrast to our work, this approach separates the budgeting and placement phases and as a result cannot take the dynamic interaction between placement, wire delays, and sequential optimization into account. A tight integration of sequential timing and placement is needed to capture this complex interaction.

Work on integrating retiming and placement for field programmable gate arrays is described in [SB02]. There, the authors also extend the sequential slack computation technique found in [CL00]. However, they attempt to overcome the shortcomings of [CL00] through a process of random sampling of reference points. While this is certainly better than choosing a single arbitrary reference, this will still be ineffective in identifying the critical cycle if the sampling process does not happen to choose a register which lies on that cycle. [SB02] also demonstrates good results using a net weighting heuristic similar to our technique. However, given their framework of annealing-based placement, it is unclear how to efficiently include explicit cycle constraints, described here in

Section 4.4.2. Our use of a quadratic programming-based formulation allows easy addition of these powerful constraints through Lagrangian relaxation.

4.2 Motivating Example

Timing-driven placement makes possible performance improvements over the pure minimization of wire-length. The portions of interconnect that have been identified as being timing critical can be given increased weight or attention to further reduce their lengths. This objective may come at the expense of other wires, but the slack available on these non-critical nets allows the wire delay increase to not affect the final clock period.

With the availability of in-place retiming or clock skew scheduling, the wires that limit the achievable clock period after sequential optimization are not necessarily the ones that limit the clock period beforehand. Figure 4.1 shows a simple sequential circuit. It is clear that without any changes to the clocking, the path from b to x is period-limiting. However, if the relative arrival of the clock at registers a and b can be moved by 1 delay unit backward and 3 delay units forward, respectively, the paths between x, y, and z will limit the period. The information from static timing analysis is essentially meaningless when register boundaries can be moved.

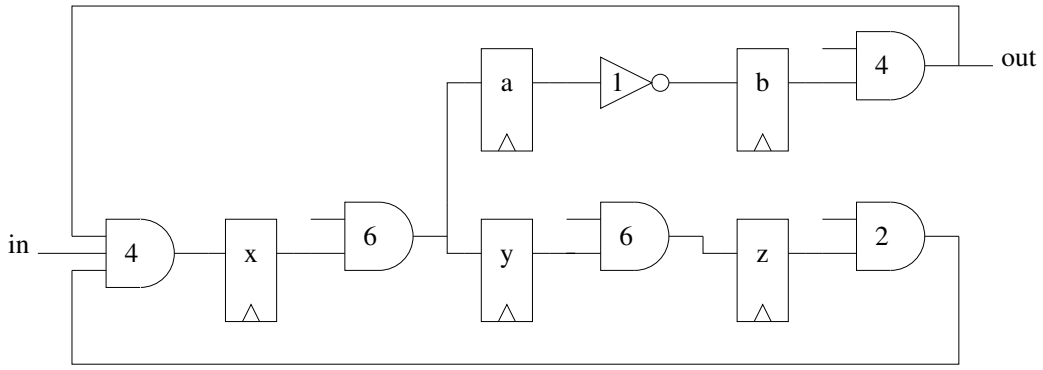


Figure 4.1: An example of two sequential cycles. Combinational gates are labeled with their delays. The lower set of registers $\{x, y, z\}$ forms the critical cycle and will limit the clock period after sequential optimization. However, a static timing-driven tool will incorrectly target the path from b to x, likely at the expense of other paths.

Experimental observation suggests that the failure of static timing analysis to give an accurate picture of the post-sequential optimization timing can be significant in industrial designs. Figure 4.2 plots the distribution of combinational path slack above the distribution of true sequential

path slack in one typical industrial design. It is immediately apparent that they offer very different versions of the timing criticality of the circuit. True sequential slack, described in more detail in Section 4.3.3, measures the amount of delay that can be added to a path before it becomes critical during post-placement retiming or clock skew scheduling. There is clearly more flexibility when sequential optimization techniques are considered; because slack can accumulate across multiple register boundaries, most of the paths in this design see more than one clock period of slack.

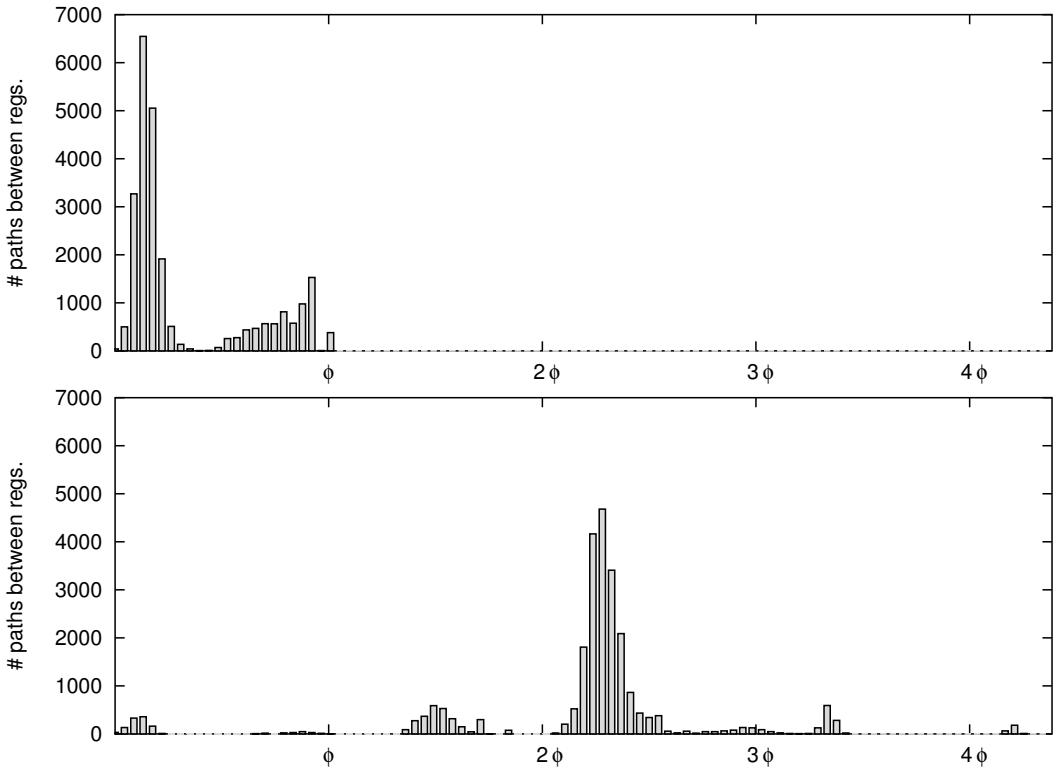


Figure 4.2: Combinational slack and true sequential slack for design Ind08. Combinational slack distribution shown on top, true sequential slack distribution shown on bottom. Horizontal axis indicates slack value after optimal clock skew scheduling, in multiples of clock cycle ϕ. Vertical axis shows number of combinational paths between registers.

The use of inappropriate timing information to guide the placement process can reduce the final design performance. If we again consider the example in Figure 4.1, a decision to shorten the path from b to x at the expense of the path from y to z appears to be beneficial from a static timing standpoint. However, doing so would actually result in an increase in the clock period after sequential optimization. Experimental results, shown in Table 4.1, indicate that this effect is demonstrably present during placement of several industrial designs; timing-driven placement in-

deed yields a design with a smaller clock period before sequential optimization, but such placements have a larger final clock period after sequential optimization, in some cases even worse than what could be achieved without any timing optimization at all. This research addresses this problem. Instead of simply trying to correct the results of combinational timing analysis after placement is finished, however, we attempt to fully exploit the potential of in-place retiming and clock skew scheduling.

Figure 4.3(a) illustrates a placement generated using a traditional QP placement tool based on GORDIAN [KSJ88, KSJA91], with the set of paths that are limiting the post-sequential optimization timing highlighted. Figure 4.3(b) shows the same circuit placed using a timing-driven placer which uses combinational static timing analysis. Figure 4.3(c) shows again the same circuit placed using CAPO, a state-of-the-art placement tool developed at UCLA [CKM00].

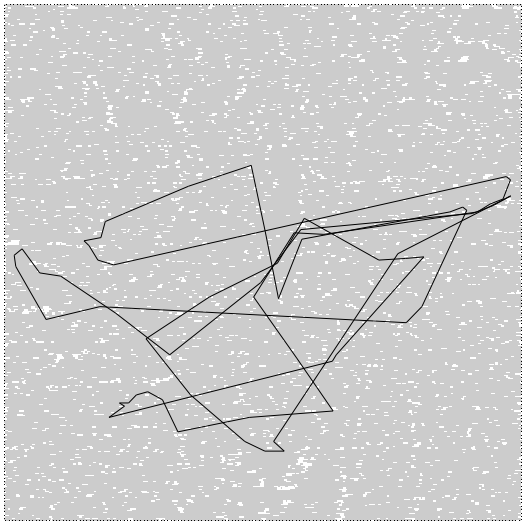
In these three cases, the tools have done a visibly suboptimal job of minimizing the wire segments that are timing-critical; many of them nearly cross the width of the die. Even with register movement through in-place retiming, it is not likely that the severe mislocation of the critical registers in this particular layout could be fully corrected. In contrast, Figure 4.3(d) is the resulting placement from our own sequential placement tool. The critical elements are much more localized, and the contribution of interconnect delay to the clock period is dramatically reduced. In this example, the integration of sequential timing analysis into the physical design process has significantly boosted the utility of clock skew scheduling or in-place retiming.

4.3 Sequential Timing Analysis

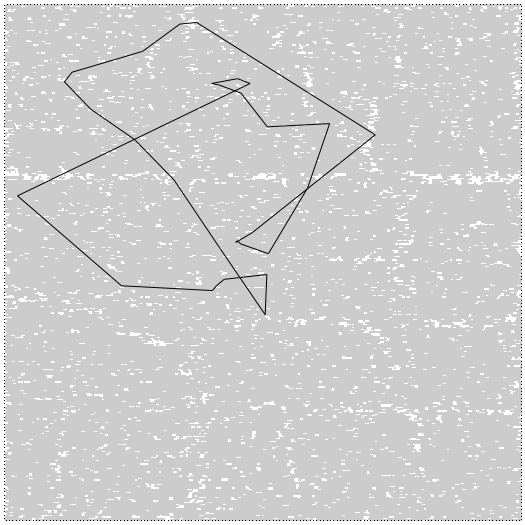
Given a circuit with timing information, we are interested in the minimum clock period achievable under sequential optimization to evaluate its fitness, and to characterize the sequential criticality of each component. There are three phases of this analysis: construction of the sequential timing graph, identification of the minimum feasible clock period, and the determination of sequential slacks.

4.3.1 Constructing The Sequential Timing Graph

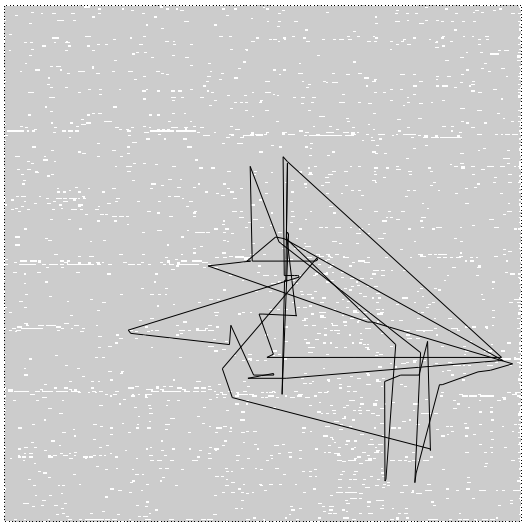
The sequential timing graph $G = (V, E)$ is extracted from a circuit as follows. Take V to be the registers of the design, together with an additional vertex v_{ext} representing the primary inputs and outputs. Add an edge (u, v) to E iff there exists a timing path from u to v in the original circuit.



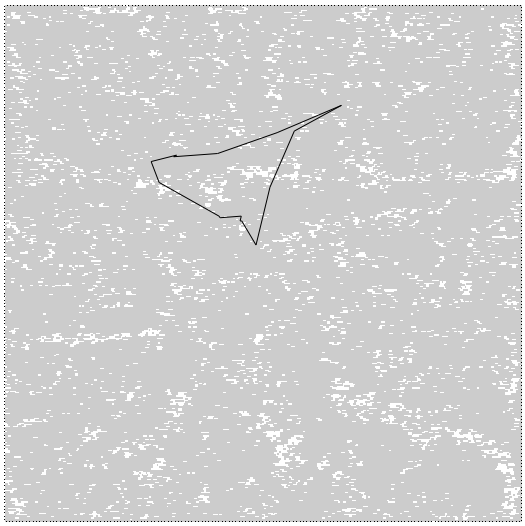
(a) GORDIAN Placement



(b) Combinational Timing-Driven Placement



(c) CAPO Placement



(d) Cycle-Constrained Placement

Figure 4.3: Placements for design Ind08. Critical cycles shown in black; standard cells in gray.

Every edge $e = (u, v)$ is labeled with $d(e)$, the maximum delay between u and v in the circuit. During placement, the estimated wire delays are included in $d(e)$.

4.3.2 Finding The Critical Cycle

Finding the critical cycle is equivalent to computing the *maximum mean cycle* (MMC) for G, which is given by $\max_{\ell \in \mathcal{C}} \sum_{e \in \ell} d(e)/|\ell|$, where $\mathcal{C}$ is the set of all cycles in G. The MMC is equal to the minimum clock period which may be obtained using unconstrained clock skew scheduling. In our implementation, we use Howard's algorithm [CTCG+98] to compute the MMC. This is shown in Algorithm 4.1. [DIG98] suggests that Howard's algorithm is the fastest known algorithm for computing MMC. Our own empirical observations concur with those of [DIG98].

Algorithm 4.1 Howard's Algorithm For Computing MMC

```
1:  for all u ∈ V do {initial guess}
2:     π(u) ← e for some e = (u, v) ∈ E
3:  repeat {main loop}
4:     find all cycles C_π in G_π = (V, {π(u) : u ∈ V} ⊆ E)
5:     for all u ∈ V do {in reverse topological order}
6:        if u = LoopHead(L_u) for some L_u ∈ C_π then
7:           η(u) ← MeanCycleTime(L_u), x(u) ← 0
8:        else {defined recursively}
9:           η(u) ← η(v) where π(u) = (u, v)
10:          x(u) ← x(v) + d(π(u)) − η(v)
11:    for all e = (u, v) ∈ E do {modify π}
12:       if η(u) < η(v) then
13:          π(u) ← e
14:    if π unchanged then
15:       for all e = (u, v) ∈ E do
16:          if η(v) = η(u) and x(u) < x(v) + d(e) − η(v) then
17:             π(u) ← e
18: until no change in π
19: return max_{u∈V} η(u)
```

The general idea behind Howard's algorithm is to maintain a small set of edges π which

starts as an initial guess of the critical cycle in the graph. The set π then has edges added and removed iteratively to monotonically increase the delays seen at the nodes in its induced subgraph (i.e. the subgraph obtained by discarding all edges except those edges in π). When the delays in the induced subgraph are maximized and can no longer be increased by changing π, then we know that π must contain the critical cycle, and thus we obtain the MMC. More details and a proof of correctness of Howard's algorithm can be found in [CTCG+98].

Note that not only is the MMC a lower bound on the clock period for the design, but also that it is always possible to find a clock skew schedule for the registers which achieves the MMC, by linear programming duality. We therefore use the MMC as a metric to evaluate the fitness of a design, since we are assured that, given the freedom of clock skew scheduling, the final clock period will be equal to the MMC.

4.3.3 Assigning Sequential Criticality

Once the critical cycle has been identified, the relative sequential criticality of the other vertices can be determined. We use a variant of the sequential timing analysis proposed in [CL00]. There, the concepts of *sequential arrival and required times* and *sequential slack* are presented. Given a target clock period of ϕ, the sequential arrival and required times at all vertices $v \in V$ with respect to a reference vertex v_{ref} can be computed from equations (4.1) through (4.3) using a modified version of the Bellman-Ford algorithm.

$$A_{seq}(v, v_{ref}) = \max_{(u,v)\in E} A_{seq}(u, v_{ref}) + d((u,v)) - \phi \tag{4.1}$$

$$R_{seq}(v, v_{ref}) = \min_{(v,w)\in E} R_{seq}(w, v_{ref}) - d((v,w)) + \phi \tag{4.2}$$

$$A_{seq}(v_{ref}, v_{ref}) = R_{seq}(v_{ref}, v_{ref}) = 0 \tag{4.3}$$

The sequential slack is $S_{seq} = R_{seq} - A_{seq}$.

A_{seq} and R_{seq} represent respectively the earliest and latest relative position in time to which a register can be moved (by retiming or clock skewing) while still meeting timing with respect to the reference point. S_{seq} measures the feasible range of temporal positions for v relative to the reference node. Intuitively, sequential slack represents a metric for quantifying sequential criticality, but we note that this criticality is only with respect to the given v_{ref}. A different choice of v_{ref} will impose different constraints. Figure 4.4 shows an example where the given choice of v_{ref} gives an incorrect value for the criticality of other vertices in the graph.

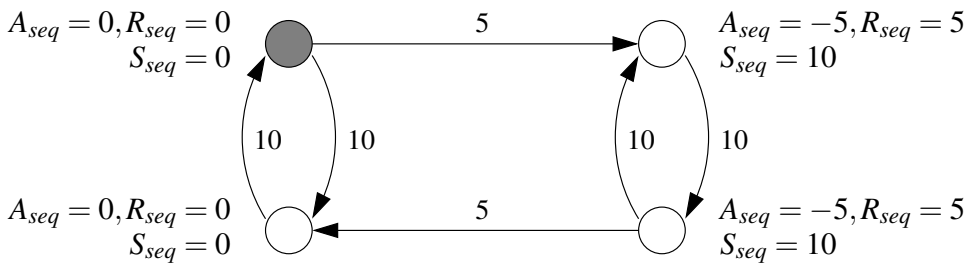


Figure 4.4: An example register timing graph. Clock period $\phi = 10$. Sequential arrival times A_{seq}, required times R_{seq} and slacks S_{seq} are computed with respect to the shaded vertex v_{ref}. The true sequential slack S_{true} for every vertex is zero; ordinary sequential slack does not offer an accurate picture of sequential criticality.

We define the *true sequential slack* as

$$S_{true}(v) = -\max_{(u,v)\in E} A_{seq}(u,v) + d((u,v)) - \phi \tag{4.4}$$

$S_{true}(v)$ gives the true sequential flexibility of v, in that this value is the maximum delay which can be added to the outputs of v without violating the cycle time ϕ. Although the definition is straightforward, in practice computing S_{true} can be prohibitively expensive; using the above equations, this is equivalent to the all-pairs longest path problem on the sequential timing graph. Thus we use the ordinary sequential slack S_{seq} to approximate S_{true} as an estimate of criticality. Note, though, that the potential difference between S_{true} and S_{seq} can be large; we must choose v_{ref} carefully to minimize the potential error. [CL00] takes $v_{ref} = v_{ext}$, but this seems to be a poor choice, as it may completely miss the actual critical cycle. The nodes most likely to impose tight constraints on other nodes are those that are themselves highly constrained, i.e. nodes on the critical cycle itself. We thus use $S_{seq}(v, v_{ref}) \mid v_{ref} \in \textsc{CriticalCycle}(G)$ to measure sequential criticality.

4.4 Placement Driven By Sequential Timing

We introduce a placement algorithm that uses sequential timing information to maximize the potential of post-placement retiming or clock skew scheduling. The general procedure outlined in Algorithm 4.2 involves three phases: sequential timing analysis, the assignment of weights based on sequential criticality, and the introduction of explicit cycle constraints. The algorithm is to some measure independent of the method used to generate placements; the ability to weight nets and to include inequality constraints are the only specific requirements.

Algorithm 4.2 Sequential Slack Weighting

1: sequential timing analysis
2: assign net weights $w(i)$
3: partition $P_0 \leftarrow allcells$
4: **while** $\exists P(|P| > m)$ **do** {GORDIAN main loop}
5: solve global constrained QP
6: bipartition all P where $|P| > m$
7: solve final global constrained QP
8: (optional) do placement with cycle constraints (Algorithm 4.4)
9: legalize placement into rows

We have implemented a modified version of GORDIAN [KSJ88, KSJA91]. In this procedure, phases of global optimization are interleaved with bipartitioning. A quadratic programming (QP) problem is constructed to minimize the total weighted quadratic wirelength

$$\sum \alpha_{ij} \left[(x_i - x_j)^2 + (y_i - y_j)^2 \right]$$

subject to a set of linear constraints. This problem is solved for the entire chip, and the positions of all cells are updated. Based on this information, the cells in every subregion that contains more than m members are bipartitioned to minimize the total number of wires across the cut and maintain reasonably balanced halves.

We utilize two different partitioning techniques. At the topmost levels, where the partitioning is coarse and the information from the QP solution is less useful to guide the partitioning, we use hMetis [KAKS97, KK99] to partition the hypergraph without regard to geometry. For finer divisions, we choose a cut-minimizing spectral partition based on the QP solution, similar to the techniques described in [TKH88, TK91a].

The coordinates of the center of each subregion are computed and a linear center-of-gravity (COG) constraint is imposed on its members. The QP is updated to include these new constraints and the global optimization is repeated. GORDIAN is ideally suited to the requirements described above; nets can be easily weighted in both the global optimization and bipartitioning phases, and additional constraints can be imposed on the solution of the QP.

4.4.1 Sequential Slack Weighting

Each net is assigned a weight proportional to its relative sequential criticality. This is done to give priority to minimizing the lengths of the most critical wires as they are the most likely ones to limit the achievable clock period. After the sequential timing analysis described in Section 4.3, we have a function S_{seq} that gives an approximation of the sequential flexibility at each timing point; this is the inverse of sequential criticality. We use the following equation to compute the net weight $w(i)$:

$$w(i) = 1 + \frac{\beta}{\gamma + S_{seq}(i)/\phi}$$

The constants β and γ are chosen to tune the distribution of weights between the most and least critical nets. This is then applied to every connection α_{ij}, in addition to scaling based on fanout.

This weighting alone is enough to produce layouts with improved sequential timing characteristics, but its limitations should be recognized. Like their combinational counterparts, sequential slacks are inherently incompatible. Also, without computing the true sequential slacks, the problems described in Section 4.3.3 can also arise. Both of these problems can be solved with the introduction of cycle constraints. Our iterative algorithm to handle these constraints helps ensure that we catch all critical cycles.

4.4.2 Explicit Cycle Constraints

Assuming complete flexibility in assigning skew to all registers, for a cycle ℓ in the circuit to satisfy a target clock period ϕ, we must have

$$t_g(\ell) + t_w(\ell) \leq |\ell|\phi$$

where $t_g(\ell)$ is total intrinsic gate delay around ℓ and $t_w(\ell)$ is the total wireload delay around ℓ.

Suppose we have an existing placement P' in which the above constraint is violated. Let $t'_w(\ell)$ be the wireload delay around ℓ for P', and let $d(\ell)$ be the total delay around ℓ for P'. Then we have

$$t_g(\ell) + t'_w(\ell) = d(\ell) \quad \text{and} \quad \frac{t_w(\ell)}{t'_w(\ell)} \leq \frac{|\ell|\phi - t_g(\ell)}{d(\ell) - t_g(\ell)} \triangleq \mu(\ell)$$

which defines the *wireload delay reduction factor* $\mu(\ell)$ necessary for ℓ to have a valid clock skew schedule for the target period.

Let $(\mathbf{x}', \mathbf{y}')$ be the locations of the cells for the given placement P'. We wish to derive a new placement $P = (\mathbf{x}, \mathbf{y})$ which satisfies the above given delay constraints. As an approximation, we

take the wire delay for a cycle as being proportional to the sum of the squared Euclidean distances between cells in that cycle. That is,

$$t_w(\ell) = \eta \sum_{(u,v)\in\ell} (x_u - x_v)^2 + (y_u - y_v)^2$$

where η is a constant. Thus the physical placement constraints are

$$\frac{\sum_{(u,v)\in\ell} (x_u - x_v)^2 + (y_u - y_v)^2}{\sum_{(u,v)\in\ell} (x'_u - x'_v)^2 + (y'_u - y'_v)^2} \leq \mu(\ell) \tag{4.5}$$

The denominator in inequality (4.5) as well as $\mu(\ell)$ are completely determined from the given placement and timing information. Thus inequality (4.5) contains only quadratic terms in $(\mathbf{x}, \mathbf{y})$. Also note that these constraints are convex.

We justify approximating total wire delay with the sum of square Euclidean distances by our use of an iterative algorithm to solve the constrained system. We aim to make only small changes to the layout during each iteration, so that any error in this approximation can be subsequently corrected. Details may be found below.

Lagrangian Relaxation

To realize the placement constraints, we use *Lagrangian relaxation*, a standard technique for converting constrained optimization problems into unconstrained problems. For brevity, we only present a simplified description of this approach here. More information about Lagrangian relaxation can be found in [PR02, GT96, SCK92].

Let $f(\mathbf{x}, \mathbf{y})$ be the sum of square wirelengths over all wires in the design for the placement $(\mathbf{x}, \mathbf{y})$. Recall that the classical analytic placement formulation is simply the unconstrained problem $\min_{\mathbf{x},\mathbf{y}} f(\mathbf{x}, \mathbf{y})$. Our constrained problem is then

$$\min_{\mathbf{x},\mathbf{y}} f(\mathbf{x}, \mathbf{y}) \quad \text{such that} \quad \mathbf{g}(\mathbf{x}, \mathbf{y}) \leq \mathbf{0} \tag{4.6}$$

where the vector $\mathbf{g}$ represents the placement constraints. For each cycle in the design, there is a single element in $\mathbf{g}$ which corresponds to the constraint inequality (4.5) for that cycle. We create the *Lagrangian* $L(\mathbf{x}, \mathbf{y}, \mathbf{k}) = f(\mathbf{x}, \mathbf{y}) - \mathbf{k} \cdot \mathbf{g}(\mathbf{x}, \mathbf{y})$, where $\mathbf{k}$ is a vector of *Lagrangian multipliers*; $\mathbf{k}$ can be thought of as "penalties" which serve to increase the value of the cost function whenever a constraint is violated. The *Lagrangian dual problem* is

$$\max_{\mathbf{k}\geq\mathbf{0}} \min_{\mathbf{x},\mathbf{y}} L(\mathbf{x}, \mathbf{y}, \mathbf{k}) \tag{4.7}$$

Our interest in the dual problem lies in the fact that, for convex problems such as ours, a solution for (4.7) corresponds directly to a solution for the original problem (4.6). We use the standard technique of *subgradient optimization* to solve the dual; see Algorithm 4.3.

Algorithm 4.3 Subgradient Optimization For Lagrangian Dual

1: $\mathbf{k} \leftarrow \mathbf{0}$
2: $\mathbf{x}, \mathbf{y} \leftarrow \arg\min_{\mathbf{x},\mathbf{y}} L(\mathbf{x}, \mathbf{y}, \mathbf{k})$
3: **while** KKT conditions are not satisfied **do**
4: $\quad \mathbf{k} \leftarrow \max(\mathbf{0}, \mathbf{k} + \gamma \cdot \mathbf{g}(\mathbf{x}, \mathbf{y}))$
5: $\quad \mathbf{x}, \mathbf{y} \leftarrow \arg\min_{\mathbf{x},\mathbf{y}} L(\mathbf{x}, \mathbf{y}, \mathbf{k})$

Note that for a fixed $\mathbf{k}$, $\min_{\mathbf{x},\mathbf{y}} L(\mathbf{x}, \mathbf{y}, \mathbf{k})$ can be solved as an ordinary unconstrained quadratic program. Subgradient optimization works by starting with an initial arbitrary $\mathbf{k}$, solving the resulting unconstrained QP, then adjusting $\mathbf{k}$ based on the violated constraints which are found. If a constraint is violated, the corresponding penalty in $\mathbf{k}$ is increased, so that subsequent iterations will move to reduce the violations, since the objective function, including the penalty terms, is to be minimized. Heuristics are available to determine an appropriate step size γ to adjust $\mathbf{k}$; e.g. [SCK92, PR02]. The *Karush-Kuhn-Tucker* (KKT) conditions for stopping the algorithm are described fully in [PR02]. Roughly speaking, the procedure stops once the penalty multipliers grow large enough to force all constraint violations to zero.

Of course, the design may have many cycles, and thus there may be many constraints involved. We propose an iterative technique, given in Algorithm 4.4, which reduces the number of cycles under consideration by ignoring non-critical cycles.

In each iteration, we add the critical cycles found in the current placement to the constraint set S. A clock period T is chosen which we use as a target period for determining the cycle constraints for S. T is decreased slowly from T_c, the feasible clock period for the current placement, down to T_f, the final overall target clock period for the design. A slow adjustment of T helps ensure that we do not overconstrain the current constraint set S while ignoring other cycles. That is, we do not wish to "squeeze too hard" on those cycles which are currently critical, as this may cause some other cycle not under consideration to violate its timing constraint. Also, as noted before, we wish to perturb the placement only by small amounts, so that any error in our quadratic approximation of the wire delays can be corrected.

A significant benefit to using the iterative technique proposed in Algorithm 4.4 is that we are able to correct errors in our estimate of the true sequential slack so that subsequent iterations

Algorithm 4.4 Placement Using Cycle Constraints

1: input: an initial placement
2: $T_c \leftarrow$ current MMC, $S \leftarrow \{$critical cycles$\}$
3: **while** $T_c > T_f$ **do**
4: choose target clock period $T, T_f \leq T < T_c$
5: **for all** cycles $\ell \in S$ **do**
6: add cycle constraint for ℓ with target T
7: remove all cells in S from COG bins
8: solve QP with cycle constraints (ALGORITHM 4.3)
9: reassign all cells in S to nearest COG
10: solve QP with cycle constraints (ALGORITHM 4.3)
11: $T_c \leftarrow$ current MMC, $S \leftarrow S \cup \{$critical cycles$\}$

may have a better estimate of the sequential flexibility of each gate. Recall from Section 4.3.3 that we use $S_{seq}(v, v_{ref})$ to approximate $S_{true}(v)$, the true sequential slack, by choosing v_{ref} to be a vertex on the critical cycle. As the set of critical cycles tends to change with each iteration, this helps to ensure that we compute S_{seq} with respect to several different choices of v_{ref}, so that if we mistakenly identify a critical vertex as non-critical, we will likely correct the mistake in subsequent iterations. In contrast, [CL00] always takes $v_{ref} = v_{ext}$, and so has no opportunity to correct such errors.

COG constraints are commonly used with analytic placement techniques to ensure that the cells are spread out relatively evenly over the entire die area. We also wish our constrained placement to be appropriately spread out over the die area, but we do not wish the COG constraints to overconstrain our solution. We approach this problem using Steps 7–10 in Algorithm 4.4, which allows critical cells to "migrate" to appropriate locations on the die to avoid violation of timing constraints.

As a practical point, we also introduce cycles which are near-critical during each iteration, instead of only the critical cycles, to help reduce the number of iterations performed. Also, the main loop is terminated whenever either of the constrained QPs indicate that the problem may have become overconstrained, as no further improvement becomes possible in such case.

Our approach shares some similarity with that of [SCK92], which also uses Lagrangian relaxation in an analytic placement framework to resolve timing constraints. However, there are several key differences between our work and that of [SCK92]. First, and most important, is that we deal with the cyclic timing constraints which arise during clock skew scheduling, rather than simply

path constraints. Second, we do not use the analytic placement step itself to perform timing analysis. The practical effect of this is twofold: our approach allows us to use general, nonlinear (and nonconvex) wire delay models, and we also do not encounter the degeneracy problems inherent in the constraints which come from timing analysis, as mentioned in [SCK92]. Finally, we enjoy much greater computational efficiency, as our Lagrangian function can be seen as simply augmenting the weights of edges between cells by $\mathbf{k}$. Solving the Lagrangian dual for fixed $\mathbf{k}$ requires no more computation than solving an unconstrained QP for our circuit.

4.4.3 Row Legalization

We use a greedy approach to detailed placement and legalization of the standard cells into rows. Such an approach has the prime benefit of speed. However, instead of direct minimization of wirelength as the search goal, as is typical of most other placement tools, our legalization technique instead seeks to minimize the total perturbation of the final placement with respect to the solution of the last QP obtained during placement (Step 7 of Algorithm 4.2, or Step 10 of Algorithm 4.4). We do this because we wish our placement to be timing-aware. Since the placement obtained by solving the QP respects the timing requirements of the circuit, we wish to deviate from such an ideal solution as little as possible.

Algorithm 4.5 Standard Cell Row Legalization

```
1:  input: a placement solution from QP
2:  Sort cells by their y-axis
3:  Place cells into nearest rows with overflow into adjacent rows
4:  for all rows R do
5:      Solve LP for R to minimize perturbation from QP
6:  while not done do
7:      Sort cells in decreasing order of perturbation
8:      for all cells c do
9:          Move c to minimize perturbation
10:     for all rows R do
11:         Solve LP for R to minimize perturbation
```

Algorithm 4.5 outlines our legalization technique. We first find an initial legal placement solution by putting cells into the nearest rows. Cells are spilled into adjacent rows wherever row capacities are exceeded. Then, for each row, a linear program is formulated and solved to obtain

the placement of each cell in the row. The cost represents the sum of the displacements of each cell from its ideal location (as given by the QP solution), while constraints are added to forbid overlap of adjacent cells within the row.

Once the initial legalized solution is found, we then proceed to make individual cell moves to improve the solution greedily. The cells are sorted in order of descending perturbation from the QP solution; this helps ensure that cells which stand the most to gain are moved first. For each cell, we determine a legal nonoverlapping placement location which minimizes the perturbation from the QP solution, and move the cell to that location. After all cells are moved in this fashion, we compact all rows using the same linear programming technique used to obtain the initial solution. This process is iterative; empirically we find only a small fixed number of iterations is required before most cells find a stable location.

4.5 Experiments

We ran our design flow on a set of thirteen industrial benchmark circuits as well as fourteen synchronous designs freely available for academic use, including the largest designs from the ISCAS89 benchmark suite. The academic circuits were technology mapped using an industrial synthesis tool into a standard cell library chosen arbitrarily from the industrial benchmarks.

The industrial libraries provided with the designs used interpolated lookup-table based models to characterize the cells. Both capacitive load and slew rate dependencies were incorporated in our timing model. The design technology files gave the electrical characterization for the wires; in all cases, we assumed the use of metal layer 3 for routing. We used the half-perimeter bounding box metric as our estimate of the wirelength, noting that our algorithms are actually independent of the wireload estimation technique used, unlike other works, e.g. [SCK92].

Currently, our placement tool can only handle single-row cells, so for the purpose of our experiments, it was necessary to convert larger circuit elements to single-row instances. Double-row cells were given a different aspect ratio, keeping the same area. Large macros were given an arbitrary size so as to fit in a single row. I/O pads were assigned randomly around the die perimeter.

Limitations in our timing analysis tool required some design changes to be made. Transparent latches were treated as ordinary registers, and combinational cycles were broken arbitrarily. Some hard macros did not have timing information associated with them, so for the purpose of timing analysis hard macros were treated as if they were I/Os for the overall circuit. Some designs used multiple clock domains. As we had no additional information regarding the relative phases and

clock frequencies of such, we uniformly regarded the circuits as having only a single clock domain. We note, however, that the techniques described in this paper can be easily extended to multiple clock domains.

Experimental results are shown in Table 4.1. The Size column indicates the number of placed instances for the design. The NW MMC column gives the MMC for the design when no wireload is taken into consideration. This is the minimum feasible clock period and serves as a lower bound on the post-placement timing, though any real placement with non-zero wirelengths will be greater. The REG MMC column shows the MMC achieved for a completely placed design using our placement flow with equal weights attached to the wires; effectively, this is a placement tool similar to GORDIAN. The COM MMC column shows the MMC achieved after placement using a combinational slack-based weighting function for the nets.

The SEQ MMC column indicates the MMC achieved after placement using the sequential slack-based weighting for the nets as described in Section 4.4.1. The percentage figure indicates the *reduction in wire delay* for the SEQ MMC result compared with the COM MMC result. We choose this as a better figure of merit than the absolute reduction in clock period, since no placer can ever hope to reduce the clock period below the no wireload MMC. The Run column indicates the run time for this algorithm, in seconds.

The CYCLE MMC column indicates the MMC achieved after placement using the cycle constraint technique described in Section 4.4.2, again with the percentage indicating reduction in wire delay compared to the combinational-weighted technique, and the Run column indicating the run time in seconds.

We also compare our tool against Capo, a leading-edge placer which focuses on wirelength minimization [CKM00]. As the two placers have very different objectives, we certainly do not expect either one to be competitive in the other's problem domain. However, this comparison quantifies the benefit of using a sequential flexibility-aware placer, rather than choosing an placer which is best-suited for another task. The CAPO MMC column in Table 4.1 shows the MMC obtained after placement using Capo, the Run column indicates the run time for Capo in seconds, and the CYvsCA column indicates the percentage improvement in wire delay of our cycle constraint-based technique compared to the placement from Capo.

We show significant improvement in achievable clock period through application of our algorithm. For the industrial benchmarks, we achieved an overall improvement in wire delay of 23.5% over a combinational slack-weighted placement technique, and 28.3% improvement over the results of Capo.

		NW	REG	COM	SEQ			CYCLE			CAPO		CYvs
Design	Size	MMC	MMC	MMC	MMC	%	Run	MMC	%	Run	MMC	Run	CA%
simon	1112	1.17	1.43	1.31	1.30	7.1	3	1.30	7.1	29	1.41	6	45.8
s13207	2179	0.87	1.63	1.28	1.29	-2.4	7	1.29	-2.4	34	1.73	8	51.2
s15850	2496	1.62	2.52	2.32	2.25	10.0	8	2.17	21.4	106	2.36	9	25.7
dsip	3653	0.74	1.09	1.17	1.06	25.6	9	1.06	25.6	80	1.12	25	15.8
diffeq2	4463	1.55	2.04	1.89	1.79	29.4	15	1.79	29.4	28	2.01	27	47.8
tseng	4579	1.55	1.81	1.82	1.75	25.9	15	1.75	25.9	29	1.75	27	0.0
bigkey2	6117	0.67	1.25	1.29	1.29	0.0	24	1.28	1.6	287	1.29	36	1.6
s38584	6903	0.74	2.38	2.35	2.18	10.6	34	2.17	11.2	85	2.33	35	10.1
s38417	7544	0.74	1.06	1.03	1.13	-34.5	43	1.09	-20.7	265	1.05	35	-12.9
s35932	8871	0.62	2.08	1.86	1.87	-0.8	47	1.87	-0.8	113	2.00	32	9.4
80386	9144	2.89	4.26	3.96	3.81	14.0	73	3.81	14.0	152	4.19	74	29.2
viper	10212	2.51	3.44	3.51	3.30	21.0	66	3.27	24.0	135	3.24	69	-4.1
elliptic3	10591	1.34	1.70	1.69	1.65	11.4	58	1.62	20.0	127	1.69	84	20.0
clma	25157	4.16	6.59	6.88	6.50	14.0	241	6.50	14.0	309	6.19	132	-15.3
Average % Improvement, Academic Designs						9.4			12.2				16.0
Ind01	6038	3.55	4.30	3.97	3.80	40.5	39	3.80	40.5	45	4.12	24	56.1
Ind02	12800	3.36	11.19	7.04	7.72	-18.5	119	7.72	-18.5	212	8.75	56	19.1
Ind03	13633	13.55	19.19	18.55	15.55	60.0	129	15.33	64.4	145	15.94	83	25.5
Ind04	26503	1.06	1.56	1.51	1.42	20.0	469	1.42	20.0	549	1.45	192	7.7
Ind05	27518	5.32	11.04	11.23	10.49	12.5	332	10.51	12.2	786	9.55	140	-22.7
Ind06	41077	3.75	5.17	4.75	4.55	20.0	964	4.65	10.0	3014	4.95	295	25.0
Ind07	52751	5.67	6.04	5.79	5.80	-8.3	1523	5.74	41.7	1689	5.88	402	66.7
Ind08	56861	1.37	1.79	1.64	1.57	25.9	1438	1.56	29.6	2671	1.74	365	48.6
Ind09	101884	7.65	12.95	11.45	10.82	16.6	7032	10.82	16.6	9670	11.47	999	17.0
Ind10	116921	4.20	4.79	4.45	4.39	24.0	8489	4.39	24.0	9237	4.65	896	57.8
Ind11	137536	4.27	11.54	8.57	8.53	0.9	8119	8.53	0.9	10875	8.73	1087	4.5
Ind12	140729	7.88	10.53	9.66	9.53	7.3	8826	9.53	7.3	9272	10.09	1133	25.3
Ind13	152066	5.08	5.43	5.31	5.18	56.5	8422	5.18	56.5	10153	5.24	1695	37.5
Average % Improvement, Industrial Designs						19.8			23.5				28.3

Table 4.1: Results for sequentially-aware placement. Size indicates number of placed instances. NW MMC indicates MMC when no wireloads are taken into consideration. REG indicates ordinary GORDIAN-style placement, COM indicates combinational slack-weighted placement. SEQ indicates sequential slack-weighted placement, CYCLE indicates cycle-constrained placement, CAPO indicates CAPO placement. % indicates wire delay improvement compared to COM, Run indicates run times in seconds. CYvsCA% indicates wire delay improvement of CYCLE compared to CAPO.

Note that overall the benefits of using the cycle constraints, compared to simply using the sequential slack-weighting heuristic, was lower for the academic benchmarks than for the industrial circuits. Our observation is that, for smaller circuits, adding cycle constraints tends to draw the cells close together, causing significant cell overlap in the QP solution. Such heavy overlap makes legalization more difficult, and all of the performance gains made from drawing the critical cycles close together are lost when legalization spreads the overlapping critical cells apart. We have added some simple heuristics to halt the addition of cycle constraints whenever sufficient overlap is seen. However, more work needs to be done in this area.

We note that such excessive cell overlap tends to happen more often with small designs rather than large ones. We conjecture that large designs tend to have more I/O pins on their periphery, which gives rise to more spreading of cells in the QP solution. One may therefore wish to avoid the use of cycle constraints, and only utilize the sequential slack weighting heuristic for small designs.

The wirelength-optimizing version of our tool was 10.7% worse in total wirelength compared to Capo. With the addition of cycle constraints, an 18.8% increase was measured over the wirelength-optimizing implementation. One certainly expects that a final layout which meets the timing constraints of the design will have longer wirelength than a layout which is done purely with minimization of wirelength as a goal. The key feature is that even with this wirelength penalty, the timing still improves. We also note that our tool currently does very little to control the total wirelength of the final placed design. As there are many nets in the design which are not critical, there is much opportunity for us to further reduce wirelength, especially during row legalization. Recall our legalization procedure seeks only to minimize the perturbation of the QP solution. For non-critical sections of the design, it makes sense to ignore the QP solution and focus on the traditional approach of wirelength minimization instead. Additionally, more modern partitioning techniques, such as those found in Capo, can replace our existing partitioning algorithm we use, especially at the coarsest levels of partitioning where the information provided by the QP solution is limited.

Run times are measured in CPU seconds on a 2GHz Pentium 4 processor. Although it may seem that our run times are significantly worse than the excellent Capo tool, one must note that our design flow includes timing analysis and additional physical constraints for performance optimization, which are completely lacking in Capo.

4.6 Discussion And Future Work

Retiming and clock skew scheduling offer significant opportunities for improving design performance, but current physical design tools do not yet exploit these techniques to their fullest potential. By optimizing the placement of the most sequentially critical components, we have demonstrated that it is possible to significantly improve the post-sequential optimization timing.

Another benefit from sequential-driven placement is that the aggressive reduction of interconnect length in the most sequentially critical cycles will result in their spatial localization. This greatly simplifies the complexity of the distribution problem for clocks with multiple skews. Table 4.2 suggests that most registers will not even require any skewing. In a traditional placement, the relative locations of the most critical combinational paths are unconstrained and can be uniformly distributed across the chip, thereby necessitating the need for a clock network with tightly controlled skew across the die. Since the registers with zero or near-zero slack will be grouped, the effort to accurately distribute multiple clock domains can be concentrated on this region.

Clock Offset	Fraction of Registers
20% or more	0.04
10% to 20%	0.01
5% to 10%	0.01
2% to 5%	0.01
-2% to 2%	0.90
-2% to -5%	0.00
-5% to -10%	0.00
-10% to -20%	0.01
-20% or less	0.03

Table 4.2: Clock skews necessary to implement MMC for cycle-constrained placement across academic designs. Clock offset is as a percentage of the final clock period.

Much can be done to improve our current tool. As noted before, addressing total wirelength is an important consideration and we have already formulated several ways to approach this challenge. The weighting of nets by their criticality is admittedly somewhat ad hoc, so we may use the ideas of [TK91b] to provide a stronger mathematical basis for our weighting function. Implementation improvements will allow us to run on larger designs and to present more datapoints by which to judge our work; runtime can also be improved. Future work includes addressing the actual implementation details of retiming and clock skew scheduling, adding in-place resynthesis optimizations during the placement procedure, and extending the sequentially-aware timing model

to other aspects of the synthesis flow.

4.7 Summary

In this chapter, two techniques for incorporating sequential timing analysis in a placement framework were shown. One uses a heuristic net weighting technique, while the other uses explicit cycle constraints in a Lagrangian relaxation framework. These techniques yield considerable improvement in final design performance compared to conventional placement techniques.

Chapter 5

Floorplanning Of Asynchronously-Communicating Systems

5.1 Motivation

The problem of timing closure in deep submicron design has motivated recent research in the area of what we call *asynchronously-communicating* systems. Purely synchronous designs generally cannot tolerate arbitrary insertion of registers, as the delaying of signals for even a single clock cycle may change the functionality of the circuit. However, it is possible to design systems where the correct behavior of the circuit is independent of the delays for at least some of the communication between circuit elements. This has an advantage in making the design process simpler, as the designer can then ignore the delay for such interconnects and still maintain correct behavior of the final circuit. This is especially true for today's design flows, where the logical functionality is often designed far in advance of the physical implementation. The early-stage logic designer's task is greatly simplified by being able to ignore the additional delays which will be incurred in the physical implementation.

In the literature, asynchronously-communicating systems are known by various names. For instance, they are called *latency-insensitive* systems in [C$^+$99, CSV00] and *wire pipelined* systems in [CM05]. To avoid varying terminology, we will use the moniker asynchronously-communicating uniformly throughout this work.

We note that asynchronously-communicating systems need not be implemented in a strictly asynchronous design methodology. The communication between circuit elements can be implemented with synchronous logic using handshaking techniques [C+99]. However, the fundamental nature of the communication remains asynchronous even with such a synchronous implementation; correct operation of the circuit is guaranteed regardless of the actual delay of the communication channel.

The use of asynchronously-communicating design techniques poses a different challenge for retiming and physical design. In contrast to the purely synchronous case, we have extra design flexibility, since the length of the wires for asynchronously-communicating interconnects is no longer bounded by the clock cycle time of the design. Such wires can generally be made longer than they could be made in a purely synchronous system. However, the extra delay incurred by extending these wires does impact the performance of the design. Essentially, use of asynchronously-communicating design techniques relax the *timing constraints* into a *timing cost* instead.

5.2 System Performance Estimation

Because of the effects of wire delay on performance, we must utilize techniques to estimate the performance of our system in order to predict the impact of our physical design. Most straightforward is *cycle-true discrete event simulation*, which simply takes the system and a set of typical inputs and simulates the system running on the given input. The drawback of this technique is that it is relatively slow.

Queuing theory has been used to a great extent in performance prediction of computer systems [All80, BCMP75]. Computation elements are modeled as service centers with queues; data flow in the system is represented as "customers" moving between the queues. The appeal of using queuing networks for performance estimation is that analytic solutions for system performance can be obtained under certain assumptions. However, one primary assumption is that queue lengths are allowed to grow unbounded. This may be a reasonable approximation for simulation of larger computer systems, where the service centers represent objects such as disk caches and computer systems with large amounts of memory, but for low-level integrated circuit design where buffers tend to be small and of fixed size, the assumption of unbounded queue length is not justified.

Petri nets have often been used in the asynchronous design community for modeling and performance estimation. An excellent overview of this area can be found in [Mur89]. For simple Petri nets analytic solutions for performance estimation can be obtained, however the systems found

in practice give rise to structure not amenable to a closed-form solution.

5.3 Existing Work

Recently a number of approaches to integrating floorplanning with performance estimation for asynchronously-communicating systems have been proposed. Both [EMW+04] and [CM05] propose using a cycle-true discrete event simulation to profile the relative utilizations of each interconnect, then use net weighting in floorplanning to keep those nets which are used most often shorter. This approach does not account for the relative criticality of the nets, however. A highly-utilized net may not have any impact on the overall performance of the system, if other nets are the real limiters of performance. [CM05] mentions this shortcoming in their assumption that the net utilizations are independent.

[LSLH04] proposes a *trajectory-based* approach, which uses a single cycle-true discrete event simulation to obtain a piecewise-linear model of the system performance as a function of the interconnect delays. This model is used to guide the floorplanning process. While [LSLH04] cites reasonably good fidelity of the model in their experiments, it is not clear how well this approach will scale. With the increasing impact of interconnect delay, floorplanning will have a greater effect on the delays, and thus there is an increasing need to evaluate the system performance model at points further away from the initial characterization.

5.4 Our Approach

There are two key ingredients to our approach to floorplanning of asynchronously-communicating systems. The first ingredient is the use of a simplified simulation model for performance estimation. Use of a simplified model avoids the costly computation incurred with cycle-true simulation. For this purpose, we choose to use a timed Petri net to model the operation of the macroblocks to be floorplanned. We use simulation on the Petri net to obtain the performance estimation rather than seek a likely non-existent closed-form solution. Non-determinism is introduced using weights on the Petri net transitions which give the relative probabilities of the transitions firing.

To model the flow of data through the system, a *token source* (i.e. a transition with no incoming edges) is used to simulate an input to the system, and a *token sink* (i.e. a transition with no outgoing edges) is used to simulate an output. The rate of arrival of tokens at the token sink becomes the throughput of the system.

The second ingredient in our approach is to use net weighting based on the *sensitivity* of the system performance on the interconnect delays. That is, we consider a linearization of the performance as a function of the interconnect lengths about some given intermediate floorplanning solution.

Using the Petri net model of our system, we can obtain the sensitivities by labeling the tokens of the Petri net with two pieces of information, the time of arrival at the place where the token currently resides, and a list of positive integers, one element per transition, representing the number of times the token was involved in the firing of each transition. During the simulation of the Petri net, when a transition fires the latest arriving token among those removed by the firing is used to provide the labeling of the tokens added to the system by the firing. This reflects the fact that it is the latest arriving token which lies on the critical path and is the bottleneck for the system; all other tokens have freedom to be delayed without affecting system performance. The new tokens are labeled with the list of firing counts taken from the critical token with the count for the just-fired transition incremented.

Tokens arriving at the token sink are thus labeled with the number of times the token passes through the transitions which restrict the throughput. If we sum all these figures for the tokens arriving at the token sink, we then obtain the relative sensitivity of the system to the delays for each transition.

To see this, consider the following formalization: Let S be the set of tokens in the system and T be the set of the transitions. Let $g : S \rightarrow \mathbb{R}$ be the labeling of arrival times on the tokens, and let $f : S \times T \rightarrow \mathbb{Z}^+$ be the function which defines the labeling of the firing counts on the tokens; that is, $f(s,t)$ is the the number of times token s was associated with transition t. Suppose the delays of the transitions are given as $h : T \rightarrow \mathbb{R}$. Then we have the relationship

$$g(s) = \sum_{t \in T} f(s,t) \cdot h(t)$$

since the arrival time of a token is simply the sum of all the transition times associated with that token. The sensitivity of the arrival time with respect to the transition delay of t is then

$$\frac{\partial g(s)}{\partial h(t)} = f(s,t)$$

While the set of labelings on the tokens may seem like a lot of data to manipulate, consider that the number of tokens is relatively small. For the Petri nets representing a macroblock, usually there will only be a single token representing the internal state of the macroblock in a one-hot encoding scheme. For interconnects, there will be one token representing the presence of data on that

interconnect. Additionally, we only need to label the tokens with only the subset of transitions associated with the interconnects; transitions internal to the macroblocks can be ignored for the purpose of sensitivity. Under these assumptions, for a system containing m macroblocks and n interconnects between macroblocks, we expect there to be at most $m+n$ tokens, each labeled with $n+1$ integers. As typical high-level floorplanning problems operate on perhaps dozens of macroblocks and hundreds of interconnects, the size of this data is relatively small.

Algorithm 5.1 shows our proposed floorplanning flow for asynchronously-communicating systems. We use the same slicing tree-based floorplanning framework described in Chapter 3. The key difference is that instead of weighting nets based on the criteria described in Chapter 3, we instead use the sensitivities derived from the simulation step.

Algorithm 5.1 Asynchronously-Communicating Floorplanning Flow

1: Simulate unfloorplanned system and obtain net sensitivities
2: Use recursive bipartitioning to obtain slicing floorplan
3: Low-temperature annealing on slicing tree
4: Re-simulate floorplanned system to obtain new performance and net sensitivities
5: **repeat**
6: Low-temperature annealing on slicing tree
7: Re-simulate floorplanned system to obtain new performance and net sensitivities
8: **until** no improvement

We use an iterative solution so as to incorporate the changes in the floorplan solution to the performance estimation. In other words, we re-linearize our performance estimate about the new operating point. Termination of the iterations occurs when there is no improvement in the solution.

5.5 Experiment

Due to the proprietary nature of such information, we encountered difficulty in obtaining real floorplanning design examples with detailed architectural descriptions. To work around this, we constructed a system based on the description of the Intel Pentium 4 processor with hyperthreading technology as described in [MBH$^+$02, MPS02].

5.5.1 Model Description

While not complete, the description from [MBH+02] was sufficient to determine a set of macroblocks, their rough functionality as a Petri net implementation, and the interconnects. The resulting design had 32 macroblocks and 46 interconnects. An illustration of the processor pipeline is shown in Figure 5.1.

We chose the relative probabilities of the transitions in the instruction dispatch unit to reflect a workload of 10% floating point operations, 80% integer operations and 10% load/store operations. The figures which should be used here are actually dependent on the workload to be simulated, but we believe this distribution gives a reasonable workload. The token source was set to generate one token (i.e. one instruction) per clock cycle, so that the throughput of the system would be one token per cycle if there were no pipeline stalls.

Figure 5.2 illustrates some examples of the Petri net modules composing the processor design. Figure 5.2(a) shows the Instruction Fetch module. The place marked with a token acts as the token source for the overall design. The place labeled OUT TC represents the output interconnect going to the Trace Cache (the level 1 instruction cache); this is a signal indicating the instruction fetch was found in the trace cache and can be read immediately without decoding. The place labeled OUT TLB represents the output interconnect going to the Translation Lookaside buffer; this is a signal indicating a trace cache miss so that the instructions must be loaded from the L2 Instruction Cache. The numbers on the transitions before and after the slash indicate the weighting given to the transition and the delay of the transition, respectively. For this instruction fetch unit, we assume 95% (19/20) of instructions are hits from the cache and can be fetched in one clock cycle, while 5% (1/20) are cache misses. These assumptions are, of course, dependent on the workload being simulated; if one wishes to use different assumptions, one can change the model appropriately. In either case, the determination of whether the instruction resides in the TC can be done in a single cycle.

Edges marked with bubbles indicate *inhibiting edges* [Mur89]. These edges have the semantics that if a token is at the place connected to by the edge, the corresponding transition may not fire. We use this mechanism to prevent multiple tokens from appearing at any given place. This effectively emulates the notion that a register may only contain a single value, and the inhibiting edge emulates the handshaking signal from the asynchronous communication channel.

The L2 Instruction Cache itself (including the cache controller) is modeled as shown in Figure 5.2(b). This is also an example of how conflicts for resources are generally modeled in our

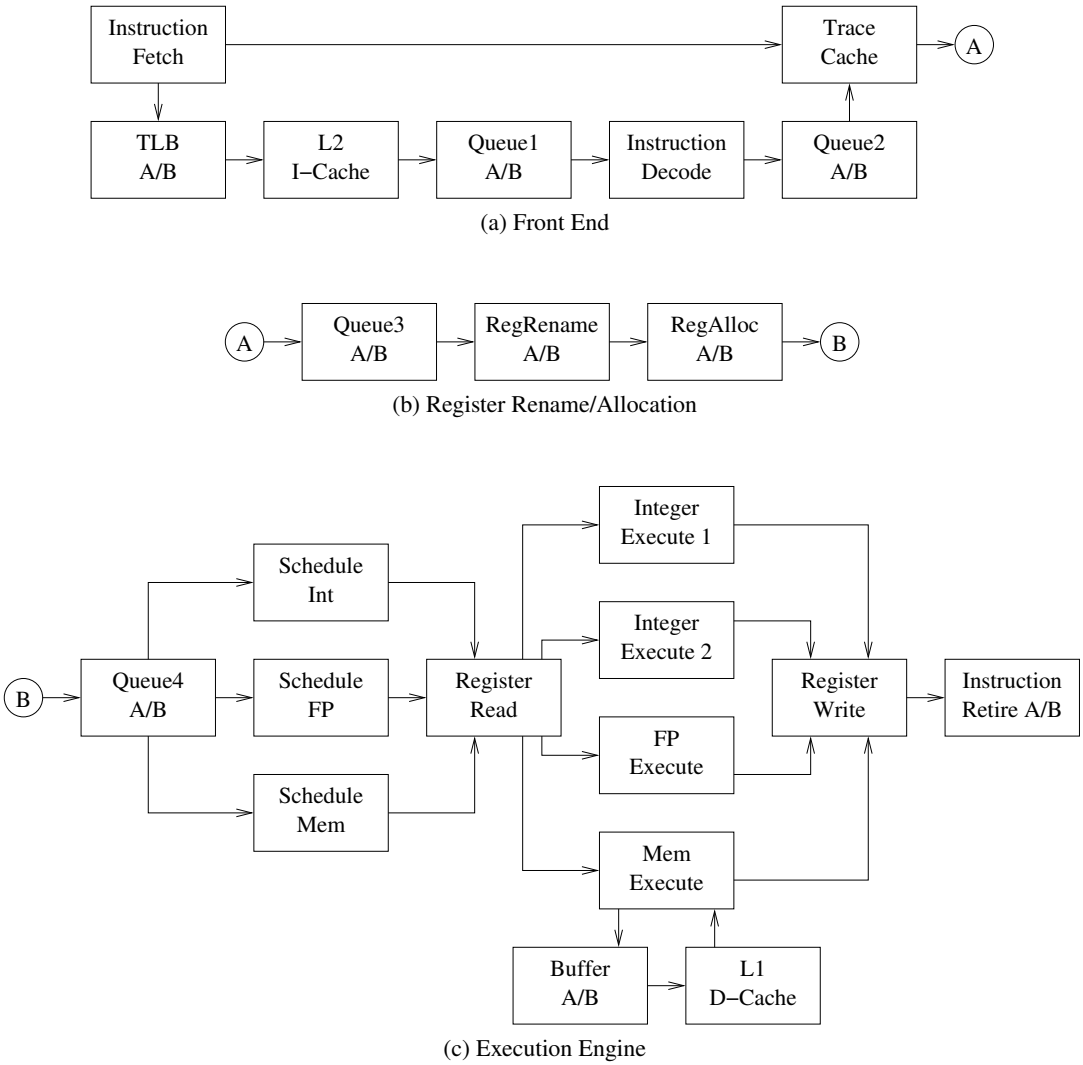
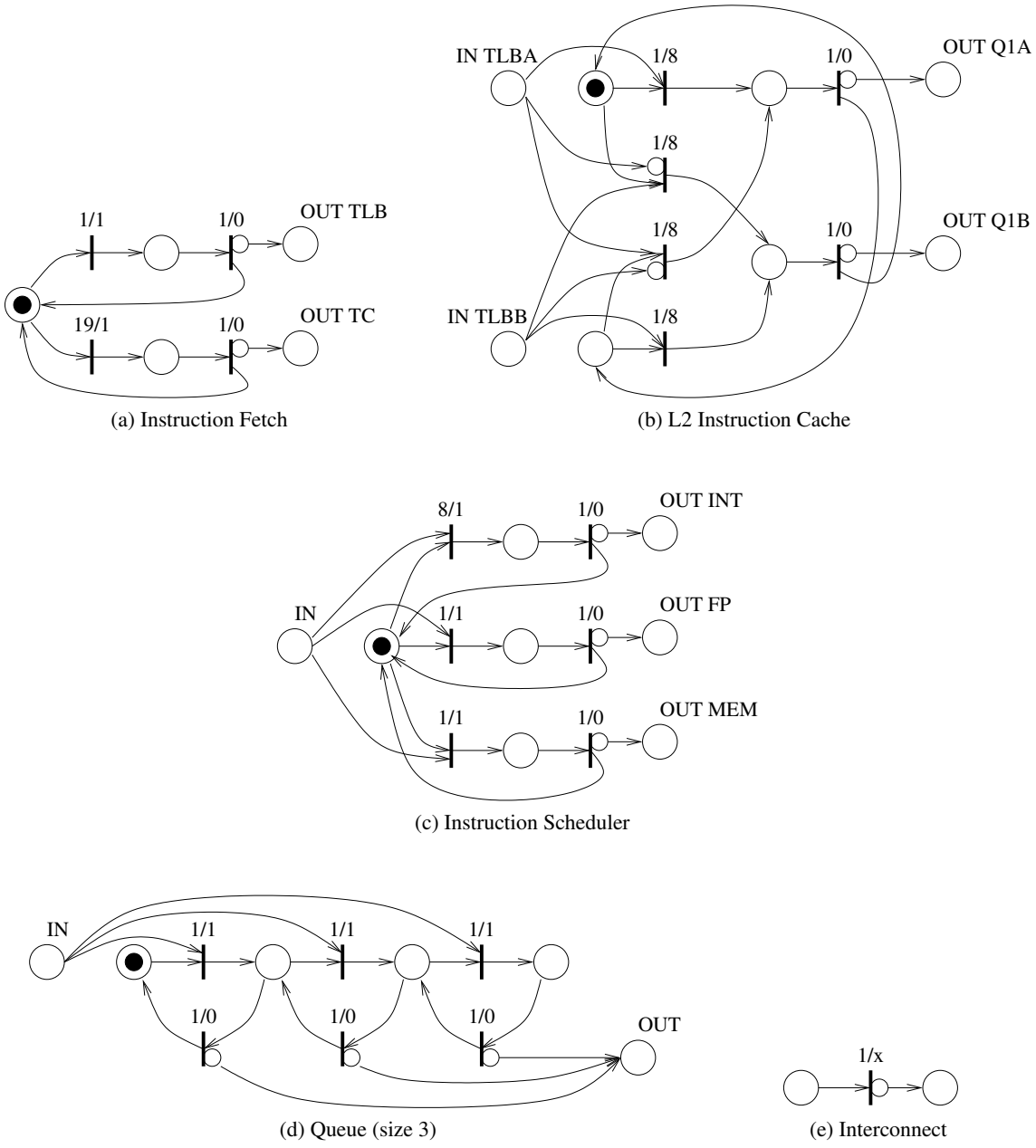


Figure 5.1: Example processor pipeline. Modules replicated for hyperthreading support are labeled with A/B.



(a) Instruction Fetch

(b) L2 Instruction Cache

(c) Instruction Scheduler

(d) Queue (size 3)

(e) Interconnect

Figure 5.2: Example Petri net models for processor modules. Numbers indicate transition weights/delays.

system. The controller accepts cache read requests from the two separate TLBs. The cache controller must respond to requests from the TLBs as they arrive, however if both TLBs have pending requests for data, the cache controller alternates between the two for *fairness*. Two places (one containing an initial token, the other directly below that) represent the internal state of the cache controller; that is, which TLB is to be next serviced in case of a conflict. The topmost and bottommost transitions labeled 1/8 fire when a token arrives at the input which is to be next served; the middle two transitions labeled 1/8 correspond to the case where there is a request from the TLB which was last served, but there is no request from the other TLB. In this latter case the controller must accept the request even though it breaks the alternation between TLBs.

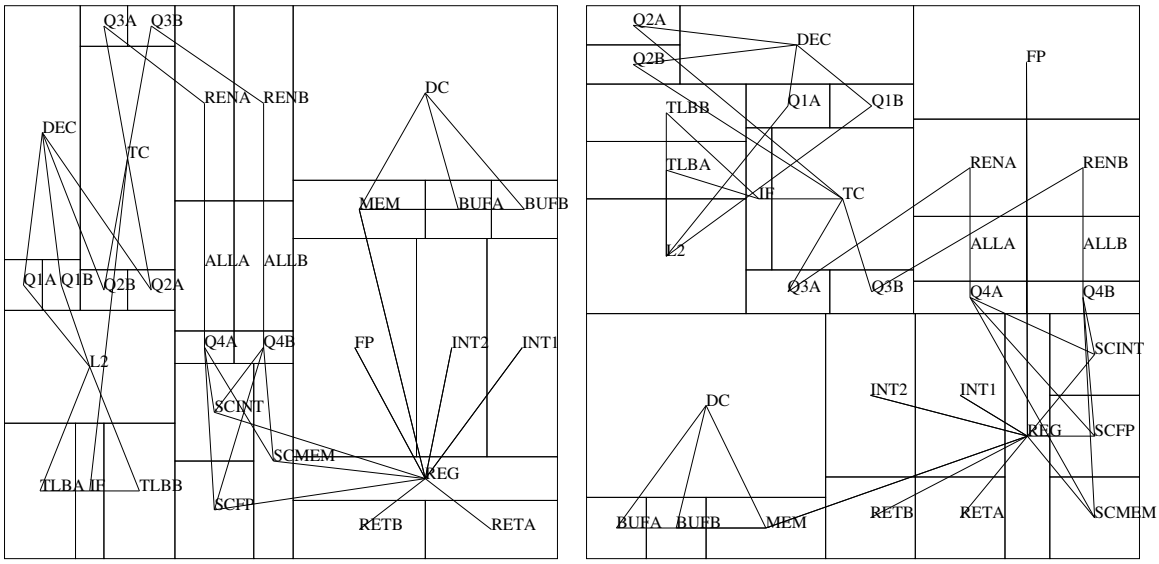
The delay of 8 on the transitions in the L2 Cache are used to model the delay of fetching from the cache itself. Again, the value one chooses here depends on the actual cache behavior one wishes to model.

Figure 5.2(c) shows part of the Instruction Scheduler module, which separates the types of instructions (integer, floating point and memory access) into separate queues. Input to the scheduler is represented by the presence of a token at the place marked IN. The relative weights given to the transitions yield the distribution of instruction types.

Queues make up a significant portion of the Pentium 4 processor design. Figure 5.2(d) illustrates the Petri net representation for a queue which can hold up to three elements (the queues used for our example processor were all 64-element queues). The token represents the state of the queue (the number of elements in the queue). Any transition which removes a token from the input place increments the state, while adding a token at the output place decrements the state. The delays of 1 on the increment transitions ensure that the queue can only consume one token from the input place per clock cycle. The delays of 0 on the decrement transitions allow tokens to be both added to the queue and removed from the queue in a single clock cycle; since the queues all feed modules which consume at most one token per clock cycle, the inhibiting edges on the decrement transitions ensure that only one token can be removed from the queue per clock cycle, as should be. The exception to this is the integer unit instruction scheduler, which can issue two integer instructions at once. Here this is modeled using two possible token consumers in parallel.

The model we use for interconnect is shown in Figure 5.2(e). Here there is a single transition with an inhibiting edge, representing an asynchronous communications channel of delay x with end-to-end handshaking. If one wishes to model a pipelined interconnect strategy, several such stages can be placed in series, and each given a delay of 1.

We chose the factor relating wire length to delay to yield an 8 cycle delay for a wire con-



(a) Uniform Weighted

(b) Sensitivity Weighted

Figure 5.3: Floorplans for example processor.

	Wirelength	Throughput	Time
Sensitivity Weighting	37254	0.71	3494
Uniform Weighting	18456	0.63	383
Mixed Weighting	23766	0.67	3546

Table 5.1: Results for asynchronously-communicating floorplan experiment. Throughput in instructions per cycle. Time is execution time in seconds on a 450 MHz Pentium 2 machine.

necting opposite corners of a square die of sufficient area to contain the entire Pentium 4 design. For the simulation runs, 10 million simulated machine cycles were used to derive the net sensitivities.

5.5.2 Experimental Results

We compare the results from our asynchronously-communicating system floorplanner with the same tool with uniform net weighting (i.e. without any simulation or sensitivity analysis) in Table 5.1. In the latter case there is no need for iteration in the algorithm. Figure 5.3 shows the floorplan results.

We can see that, as expected, using the net weights based on sensitivities degrades the wirelength, albeit more substantially than we would like. We also show results for a "mixed" weight-

ing scheme, where we take the largest sensitivity among all interconnects and add that figure to all net weightings. This essentially provides an even weighting between the performance optimization metric and the wirelength optimization metric. Such a mixed weighting scheme allows the designer to balance between the wirelength and the system performance as desired.

5.6 Summary

We have demonstrated a system for the floorplanning asynchronously-communicating system which takes advantage of simplified modeling to allow performance estimation within an iterative improvement scheme. We use sensitivities of performance with respect to interconnect delays as a net weighting approach to optimize for performance.

The primary difficulty with this approach lies in obtaining adequate architectural models for the system. However, in the real world, an architectural level design is typically developed before designers commence work on a more detailed level. We believe an abstract model can be developed readily starting from this. Alternatively, we may seek techniques for developing an abstract model from a cycle-true simulation, possibly similar in nature to the techniques proposed in [LSLH04]. It remains to be seen how accurate such an approach would be, however.

Chapter 6

Clock Skew Scheduling Under Variability

6.1 Introduction

The manufacturing process inevitably introduces differences between the idealized design generated by the circuit designer and the final manufactured product. A goal of circuit design is to create a final design which is robust to such *process variation*. A natural approach to dealing with this problem is to design conservatively; given estimates as to how large the process variations are, the designer can consider the worst-case variations, and account for these during the initial design process.

Process variation affects the problem of clock skew scheduling. Variability of the delays present in the clock tree distribution network of a synchronous digital circuit can change the timing of registers, and hence can affect the performance of the design. Conservative design in this case means that the clock skew scheduling, or assignment of latency to the registers, should account for the worst-case clock tree variability.

In this chapter we present the problem of clock skew scheduling given the presence of uncertainty in the delays associated with the clock distribution tree in a digital synchronous circuit. A model for this problem and a sufficient condition for its optimal solution is presented. This condition can be used as a stopping criterion for an iterative algorithm for this problem.

6.2 Notation

Let $G = (V_L \cup V_C, E_L \cup E_C)$ be the directed graph representing our circuit, where V_L is the set of leaf nodes in the clock tree (registers), V_C is the set of internal nodes in the clock tree, $E_L \subseteq V_L \times V_L$ is the set of edges representing combinational delay paths between the leaf nodes, and $E_C \subseteq V_C \times (V_L \cup V_C)$ is the set of edges representing the clock distribution from the internal nodes of the clock tree. All edges are directed in the direction of the signal or clock flow.

Let $\delta_{ij} \in \mathbb{R}$ be the delay associated with each combinational delay edge $(i, j) \in E_L$. Let $\mu_{ij} \in \mathbb{R}$ be the minimum delay associated with each internal clock tree edge $(i, j) \in E_C$.

Let $T \in \mathbb{R}$ be the clock period.

Let $\varepsilon \in [0, 1]$ be the factor of uncertainty in delay in the clock tree. A nominal delay of d corresponds to an actual delay in the range $[(1-\varepsilon)d, (1+\varepsilon)d]$.

We require the clock tree subgraph $G_C = (V_L \cup V_C, E_C)$ be a tree. That is, there is a distinguished vertex $v_0 \in V_C$ called the *root* which has no incoming edges, and, for every other vertex $v \in (V_L \cup V_C) \setminus \{v_0\}$, there is a unique path in G_C from v_0 to v. Equivalently, every vertex except the root must have exactly one incoming edge in E_C.

For each edge $(i, j) \in E_L$, there is an associated vertex in V_C representing the *lowest common parent* between the endpoints of that edge. Take the vertices which appear both in the path from v_0 to i and in the path from v_0 to j. Among these vertices, the one which is furthest away from v_0 (has a path from v_0 with the greatest number of edges) is the lowest common parent of (i, j). Let $P_{ij} \in V_C$ denote the lowest common parent of $(i, j) \in E_L$. It is easy to show that P_{ij} is unique given (i, j).

6.3 Problem Formulation

We wish to find latency assignments for all vertices which minimizes the clock period while satisfying the timing requirements for the graph. Ignoring the clock distribution tree and the uncertainty in delay, this is equivalent to finding $(T, \mathbf{x})$ with $x_i \in \mathbb{R}, i \in V_L$ which satisfies

$$\min T$$

$$x_i + \delta_{ij} - T \leq x_j, \qquad (i, j) \in E_L$$

This is recognizable as the LP dual of the ordinary maximum mean cycle problem.

When the clock distribution network is added with uncertainty in the associated delays,

we introduce additional variables $x_i \in \mathbb{R}, i \in V_C$, and the problem becomes

$$\begin{aligned} &\min T \\ x_i + \varepsilon(x_i - x_{P_{ij}}) + \delta_{ij} - T &\leq x_j - \varepsilon(x_j - x_{P_{ij}}), \qquad (i,j) \in E_L \\ x_i + \mu_{ij} &\leq x_j, \qquad (i,j) \in E_C \end{aligned}$$

which can be rewritten as

$$\begin{aligned} &\min T \\ -(1+\varepsilon)x_i + (1-\varepsilon)x_j + 2\varepsilon x_{P_{ij}} + T &\geq \delta_{ij}, \qquad (i,j) \in E_L \\ -x_i + x_j &\geq \mu_{ij}, \qquad (i,j) \in E_C \end{aligned} \tag{6.D}$$

We denote this problem (6.D) to indicate that this is the *dual* of the maximum mean cycle problem with clock tree latencies.

Note that (6.D) has one constraint for every edge in G. Given an assignment for $(T, \mathbf{x})$ which is feasible in (6.D), we say an edge is *critical* iff the corresponding constraint is satisfied strictly at equality.

Claim 6.1. (6.D) *has a feasible solution.*

Proof of Claim 6.1. One can easily construct a feasible solution. Assign latencies x_i in topological order starting from v_0, according to the inequalities corresponding to the edges in E_C. After all latencies have been assigned, compute T to satisfy all inequalities corresponding to the edges in E_L. □

Claim 6.2. (6.D) *has an optimal solution where all latencies x_i are non-negative.*

Proof of Claim 6.2. If $(T, \mathbf{x})$ is feasible in (6.D), then $(T, \mathbf{x} + \theta \cdot \mathbf{1})$ is also feasible in (6.D), where $\theta \in \mathbb{R}$ and $\mathbf{1}$ is the vector with all elements equal to 1. Let $(T, \mathbf{x})$ be optimal in (6.D), and choose $\theta = -\min x_i$. □

6.4 Fundamental Theorem Of Markov Chains

Here we state the Fundamental Theorem Of Markov Chains, which we make use of later. First, however, we present some definitions.

Definition 6.3 (Stochastic Vector). A vector $\mathbf{p} \in \mathbb{R}^m$ is said to be *stochastic* iff each entry $p_i \in [0,1]$ and $\sum_{i=1}^{m} p_i = 1$.

Definition 6.4 (Column Stochastic Matrix). A matrix B is *column stochastic* iff every column of B is stochastic.

Theorem 6.5. *(Fundamental Theorem Of Markov chains) If B is a square column stochastic matrix, then there exists a stochastic vector $\mathbf{p}$ such that $B\mathbf{p} = \mathbf{p}$.*

Proof of Theorem 6.5. Given in [Mon]. As this source is no longer generally available, we briefly reproduce the salient points of the proof here. Let K be the set of all stochastic vectors in $\mathbb{R}^m$. Let B be a square column stochastic matrix. We make the following observations:

Observation 6.6. *K is bounded. That is, there exists $M \in \mathbb{R}, M > 0$ such that the Euclidean norm $\|\mathbf{x}\| \leq M$ for all $\mathbf{x} \in K$.*

Observation 6.7. *K is closed.*

Observation 6.8. *K is convex.*

Observation 6.9. *For all $\mathbf{x} \in K$, $B\mathbf{x} \in K$.*

Lemma 6.10. *(Markov-Kakutani Theorem) Let T be an affine transformation and K a non-empty compact convex subset of $\mathbb{R}^m$. If T maps K into K them T has a fixed point in K.*

Proof of Lemma 6.10. Choose any $\mathbf{z} \in K$ and define $\mathbf{x}_n = \frac{1}{n}(\mathbf{z} + T\mathbf{z} + T^2\mathbf{z} + \ldots + T^{n-1}\mathbf{z})$ where $T^k = T \circ T^{k-1}$ and $T^1 = T$. By convexity of K, $\mathbf{x}_n \in K$. By compactness of K, there exists a convergent subsequence $(\mathbf{x}_{n_j})_{j=1}^{\infty}$ with limit $\mathbf{x} \in K$. Since $(\mathbf{x}_{n_j}) \to \mathbf{x}$ and T is continuous (since T is affine), then $(T\mathbf{x}_{n_j}) \to T\mathbf{x}$. By the Heine-Borel theorem, K is bounded so there is an $M > 0$ such that $\|\mathbf{a}\| \leq M$ for all $\mathbf{a} \in K$. Therefore

$$\begin{aligned}
\|\mathbf{x}_n - T\mathbf{x}_n\| &= \|\frac{1}{n}(\mathbf{z} + T\mathbf{z} + T^2\mathbf{z} + \ldots + T^{n-1}\mathbf{z}) - \frac{1}{n}(T\mathbf{z} + T^2\mathbf{z} + T^3\mathbf{z} + \ldots + T^n\mathbf{z})\| \\
&= \|\frac{1}{n}(\mathbf{z} - T^n\mathbf{z})\| \\
&\leq \frac{1}{n}(\|\mathbf{z}\| + \|T^n\mathbf{z}\|) \\
&\leq \frac{1}{n}(M + M)
\end{aligned}$$

Replacing n by n_j we get $0 \leq \|\mathbf{x}_{n_j} - T\mathbf{x}_{n_j}\| \leq \frac{2M}{n_j}$ and letting $j \to \infty$ we conclude $0 \leq \|\mathbf{x} - T\mathbf{x}\| \leq 0$. This means $\mathbf{x} = T\mathbf{x}$. Thus $\mathbf{x}$ is a fixed point of T. □

Now let T be defined as the affine transformation $T(\mathbf{x}) = B\mathbf{x}$. By Observation 6.9, T maps K into K. By Observation 6.6 and Observation 6.7, and using the Heine-Borel Theorem, K is compact. By Lemma 6.10, T has a fixed point $\mathbf{p} \in K$. □

6.5 Theoretical Results

In this section, we present a sufficient condition for optimality in the latency assignment problem with uncertainty.

Theorem 6.11. *Let* $(T,\mathbf{x})$ *be a latency assignment feasible in* (6.D)*. If every vertex in* $(V_L \cup V_C)$ *has at least one outgoing edge which is critical, then* $(T,\mathbf{x})$ *is an optimal solution of* (6.D)*.*

Proof of Theorem 6.11. Let E_Z be the set of edges which are critical for (6.D) under the assignment $(T,\mathbf{x})$. Let $E'_L = E_L \cap E_Z$ and let $E'_C = E_C \cap E_Z$. Consider the following linear program:

$$\begin{aligned} &\min T' \\ -(1+\varepsilon)x'_i + (1-\varepsilon)x'_j + 2\varepsilon x'_{P_{ij}} + T' \geq 0, \qquad & (i,j) \in E'_L \\ -x'_i + x'_j \geq 0, \qquad & (i,j) \in E'_C \end{aligned} \tag{6.RD}$$

We denote this problem (6.RD) to indicate that this is the *restricted dual* problem for (6.D) under the assignment $(T,\mathbf{x})$. The restricted dual has the interpretation that a feasible solution to (6.RD) indicates a direction in which we can move from $(T,\mathbf{x})$ so as to remain feasible in (6.D). That is, given a feasible solution $(T',\mathbf{x}')$ to (6.RD), we can find some $\theta \in \mathbb{R}, \theta > 0$ such that $(T + \theta T', \mathbf{x} + \theta\mathbf{x}')$ is feasible in (6.D).

Note that $\mathbf{0}$ is feasible in (6.RD), so at optimality we must have $T' \leq 0$.

Lemma 6.12. $(T,\mathbf{x})$ *is not optimal for* (6.D) *iff the optimal solution for* (6.RD) *has* $T' < 0$*.*

Proof of Lemma 6.12. Suppose $(T,\mathbf{x})$ is not optimal for (6.D). Let the optimal solution of (6.D) be $(T^*,\mathbf{x}^*)$. Now take $(T',\mathbf{x}') = (T^* - T, \mathbf{x}^* - \mathbf{x})$. $(T',\mathbf{x}')$ must be feasible in (6.RD), and $T' < 0$, so the optimal solution for (6.RD) has $T' < 0$.

Now suppose the optimal solution for (6.RD) is $(T',\mathbf{x}')$, where $T' < 0$. Then we can find $\theta \in \mathbb{R}, \theta > 0$, such that $(T + \theta T', \mathbf{x} + \theta\mathbf{x}')$ is feasible in (6.D). But $T + \theta T' < T$, so $(T,\mathbf{x})$ is not optimal for (6.D). □

Consider the following linear program derived from (6.RD):

$$\begin{aligned} &\min -1 \\ -(1+\varepsilon)x'_i + (1-\varepsilon)x'_j + 2\varepsilon x'_{P_{ij}} \geq 1, \qquad & (i,j) \in E'_L \\ -x'_i + x'_j \geq 0, \qquad & (i,j) \in E'_C \end{aligned} \tag{6.RD$'$}$$

Lemma 6.13. (6.RD) *has* $T' < 0$ *at optimality iff* (6.RD$'$) *has a feasible solution.*

Proof of Lemma 6.13. Suppose (6.RD) has an optimal solution $(T^*, \mathbf{x}^*)$, where $T^* < 0$. Then $\mathbf{x}' = -\mathbf{x}^*/T^*$ is feasible in (6.RD′).

Now suppose (6.RD′) has a feasible solution $\mathbf{x}'$. Then $(T' = -1, \mathbf{x}')$ is feasible in (6.RD), so (6.RD) has $T' < 0$ at optimality. □

The dual of (6.RD′) is

$$\begin{aligned} -1 + \max \sum_{(i,j)\in E'_L} y_{ij} & \\ (1-\varepsilon)\sum_{(i,v)\in E'_L} y_{iv} - (1+\varepsilon)\sum_{(v,j)\in E'_L} y_{vj} + \sum_{(i,v)\in E'_C} z_{iv} = 0, & \qquad v \in V_L \\ \sum_{(i,v)\in E'_C} z_{iv} - \sum_{(v,j)\in E'_C} z_{vj} + 2\varepsilon \sum_{P_{ij}=v,(i,j)\in E'_L} y_{ij} = 0, & \qquad v \in V_C \\ y_{ij} \geq 0, & \qquad (i,j) \in E'_L \\ z_{ij} \geq 0, & \qquad (i,j) \in E'_C \end{aligned} \tag{6.RP′}$$

Lemma 6.14. *(Farkas' Lemma)* (6.RD′) *is infeasible iff* (6.RP′) *has some feasible solution for which* $\sum y_{ij} > 0$.

Proof of Lemma 6.14. Consider the case where (6.RP′) has no feasible solution for which $\sum y_{ij} > 0$. Note that $\mathbf{0}$ is a feasible point in (6.RP′), so the optimal value of (6.RP′) is -1. By duality, the optimal value of (6.RD′) is -1 as well, and hence (6.RD′) is feasible.

Now consider the case where (6.RP′) has some feasible solution for which $\sum y_{ij} > 0$. Observe that if $(\mathbf{y}, \mathbf{z})$ is feasible in (6.RP′), then for all $\theta \in \mathbb{R}, \theta > 0$, $(\theta \cdot \mathbf{y}, \theta \cdot \mathbf{z})$ is also feasible in (6.RP′). Thus the value of (6.RP′) is unbounded, so by duality (6.RD′) is infeasible. □

Suppose every vertex $v \in V_L \cup V_C$ has at least one outgoing edge which is critical. Take an arbitrary subset of the critical edges $E'_Z \subseteq E_Z$ so that the graph $G_Z = (V_L \cup V_C, E'_Z)$ has exactly one outgoing edge for every vertex. Consider the following problem generated by adding constraints to

(6.RP′):

$$\begin{aligned}
&\max \sum_{(i,j)\in E'_L} y_{ij} \\
(1-\varepsilon)\sum_{(i,v)\in E'_L} y_{iv} - (1+\varepsilon)\sum_{(v,j)\in E'_L} y_{vj} + \sum_{(i,v)\in E'_C} z_{iv} &= 0, \qquad v \in V_L \\
\sum_{(i,v)\in E'_C} z_{iv} - \sum_{(v,j)\in E'_C} z_{vj} + 2\varepsilon \sum_{P_{ij}=v,(i,j)\in E'_L} y_{ij} &= 0, \qquad v \in V_C \\
y_{ij} &\geq 0, \qquad (i,j) \in E'_L \\
z_{ij} &\geq 0, \qquad (i,j) \in E'_C \\
y_{ij} &= 0, \qquad (i,j) \notin E'_Z \\
z_{ij} &= 0, \qquad (i,j) \notin E'_Z
\end{aligned}$$

Since the above problem is a restriction of (6.RP′), any feasible solution for this problem is also feasible for (6.RP′). Let $E''_L = E'_L \cap E'_Z$ and $E''_C = E'_C \cap E'_Z$, and let $e : (V_L \cup V_C) \to E'_Z$ be the bijective mapping from the vertices of G to their corresponding outgoing edges in E'_Z. Also, we can perform the substitution $\mathbf{y}' = (1+\varepsilon)\mathbf{y}$. Simplifying the above problem yields the following equivalent problem:

$$\begin{aligned}
&\max \left(\frac{1}{1+\varepsilon}\right) \sum_{(i,j)\in E''_L} y'_{ij} \\
\left(\frac{1-\varepsilon}{1+\varepsilon}\right) \sum_{(i,v)\in E''_L} y'_{iv} - y'_{e(v)} + \sum_{(i,v)\in E''_C} z_{iv} &= 0, \qquad v \in V_L \\
\sum_{(i,v)\in E''_C} z_{iv} - z_{e(v)} + \left(\frac{2\varepsilon}{1+\varepsilon}\right) \sum_{P_{ij}=v,(i,j)\in E''_L} y'_{ij} &= 0, \qquad v \in V_C \\
y'_{ij} &\geq 0, \qquad (i,j) \in E''_L \\
z_{ij} &\geq 0, \qquad (i,j) \in E''_C
\end{aligned} \tag{6.RP″}$$

Note that there is exactly one equation in (6.RP″) corresponding to each vertex in $V_L \cup V_C$.

For every $v \in V_C$, define

$$R_v = \left\{u \in V_C : (u = v) \vee ((u,w) \in E''_C, w \in R_v)\right\}$$

That is, $u \in R_v$ iff there exists a path from u to v whose edges are all contained in E''_C.

Lemma 6.15. *Given* (6.RP″)*, for all* $v \in V_C$*,*

$$z_{e(v)} = \left(\frac{2\varepsilon}{1+\varepsilon}\right) \sum_{u\in R_v} \left(\sum_{P_{ij}=u,(i,j)\in E''_L} y'_{ij} \right)$$

Proof of Lemma 6.15. We use induction on $|R_v|$. If $|R_v| = 1$, then $R_v = \{v\}$, and v must have no incoming edge in E''_C. Rearranging the equality in $(6.\text{RP}'')$ corresponding to v gives

$$z_{e(v)} = \left(\frac{2\varepsilon}{1+\varepsilon}\right) \sum_{P_{ij}=v,(i,j)\in E''_L} y'_{ij}$$

as required.

If $|R_v| > 1$, then v must have exactly one incoming edge $(w,v) \in E''_C$. Also, by definition of R_v, $R_w = R_v \setminus \{v\}$. By the induction hypothesis,

$$z_{e(w)} = \left(\frac{2\varepsilon}{1+\varepsilon}\right) \sum_{u\in R_w} \left(\sum_{P_{ij}=u,(i,j)\in E''_L} y'_{ij} \right)$$

Rearranging the equality in $(6.\text{RP}'')$ corresponding to v yields

$$\begin{aligned} z_{e(v)} &= z_{e(w)} + \left(\frac{2\varepsilon}{1+\varepsilon}\right) \sum_{P_{ij}=v,(i,j)\in E''_L} y'_{ij} \\ &= \left(\frac{2\varepsilon}{1+\varepsilon}\right) \sum_{u\in R_w} \left(\sum_{P_{ij}=u,(i,j)\in E''_L} y'_{ij} \right) + \left(\frac{2\varepsilon}{1+\varepsilon}\right) \sum_{P_{ij}=v,(i,j)\in E''_L} y'_{ij} \\ &= \left(\frac{2\varepsilon}{1+\varepsilon}\right) \sum_{u\in R_v} \left(\sum_{P_{ij}=u,(i,j)\in E''_L} y'_{ij} \right) \end{aligned}$$

as required. □

Now consider only the system of linear equalities in $(6.\text{RP}'')$, ignoring the inequality constraints. We can rewrite this system in matrix form by letting

$$\mathbf{p} = \begin{bmatrix} \mathbf{y}' \\ \mathbf{z} \end{bmatrix}$$

where $y'_{e(u)}$ and $z_{e(v)}$ lie in the same row in $\mathbf{p}$ as the constraint corresponding to vertices u and v, respectively, so that $A\mathbf{p} = \mathbf{0}$, where A is the matrix representing the linear equalities in $(6.\text{RP}'')$.

Lemma 6.16. *If a stochastic vector* $\mathbf{p}$ *is feasible in* $(6.\text{RP}'')$*, then* $\sum y'_{ij} > 0$.

Proof of Lemma 6.16. Suppose $\sum y'_{ij} = 0$. By Lemma 6.15, every z_{ij} can be expressed as the sum of a subset of the y'_{ij}, times a constant. Hence $z_{ij} = 0$ as well, so $\mathbf{p} = \mathbf{0}$. But $\mathbf{p}$ is stochastic, a contradiction. Thus some $y'_{ij} > 0$, so $\sum y'_{ij} > 0$. □

Let $B = A + I$, where I is the identity matrix. We now have $B\mathbf{p} - I\mathbf{p} = 0$, or $B\mathbf{p} = \mathbf{p}$.

Lemma 6.17. *B is a column stochastic matrix.*

Proof of Lemma 6.17. The variable $y_{ij}, (i,j) \in E_L''$ appears exactly three times in the linear system of equalities: once with coefficient -1 in the equality corresponding to the vertex i, once with coefficient $\frac{1-\varepsilon}{1+\varepsilon}$ in the equality corresponding to the vertex j, and once with coefficient $\frac{2\varepsilon}{1+\varepsilon}$ in the equality corresponding to the vertex P_{ij}. Note that these coefficients sum to zero. The variable $z_{ij}, (i,j) \in E_C''$ appears exactly twice: once with coefficient -1 in the equality corresponding to the vertex i, and once with coefficient $+1$ in the equality corresponding to the vertex j. Again, these coefficients sum to zero. Thus the columns of A must each sum to zero, so by construction the columns of B must each sum to 1. Since $\varepsilon \in [0,1]$, all the elements of B lie in $[0,1]$ as well. □

Now we complete the proof of Theorem 6.11. For the latency assignment $(T, \mathbf{x})$, we can construct the linear program (6.RP″) given the critical edges. Now from Lemmas 6.17 and 6.5, we know there exists a feasible solution to (6.RP″). In particular, the fixed point $\mathbf{p} = [\mathbf{y}'; \mathbf{z}]$ of $B\mathbf{p} = \mathbf{p}$ satisfies the equality constraints, and since $\mathbf{p}$ is stochastic, it must also satisfy the inequality constraints $\mathbf{p} \geq \mathbf{0}$ of (6.RP″) as well.

By Lemma 6.16, this feasible solution of (6.RP″) must have $\sum y'_{ij} > 0$. But by construction of (6.RP″), the point $(\mathbf{y} = \frac{1}{1+\varepsilon}\mathbf{y}', \mathbf{z})$ must be feasible in (6.RP′), and $\sum y_{ij} > 0$.

By Lemma 6.14, (6.RD′) is infeasible, so by Lemma 6.13, the optimal solution of (6.RD) has $T' = 0$. By Lemma 6.12, $(T, \mathbf{x})$ is optimal for (6.D). □

6.6 Summary

While we do not propose a particular iterative solution for the latency assignment problem here, we hope to modify one of the many such algorithms for variation-free optimal latency assignment to a procedure which accounts for variations. In this respect, Theorem 6.11 becomes valuable in that it gives a stopping criterion for an iterative solution to the latency assignment problem. If a given assignment is such that every vertex has at least one outgoing edge which is critical, then we know an optimal solution has been reached. This can also be used to guide the choice of how to modify the clock network during each step of the iterative latency assignment algorithm.

The analysis presented in this chapter assumes a fixed clock tree topology. Of course, a designer may have flexibility in the clock tree design, in order to minimize the impact of variability on the final performance. However, treatment of such design techniques is outside the scope of the work presented here.

Chapter 7

An Experimental Nonlinear Programming Technique For Floorplanning

7.1 Introduction

In the past, formulation of EDA problems in terms of general nonlinear programs had limited use. Software packages capable of tackling such problems were quite limited in the size of the problems which they could handle. However, modern nonlinear program solvers have reached a state of maturity where they have the potential to be useful in many applications. For example, LANCELOT [CGT92], a large-scale general-purpose optimization package, has been used successfully to solve nonlinear circuit optimization problems for designs over 1500 gates [VC99]. With the ability to tackle problems with over 9000 variables and 10000 constraints, it is worth examining the application of LANCELOT to other EDA problems of similar size.

In this chapter, we describe an experimental technique to use LANCELOT in a floorplanning context. The floorplanning problem is formulated and an algorithm which uses nonlinear programming at its core is proposed.

7.2 Problem Formulation

We consider the problem of floorplanning rectangular soft modules (i.e. modules with fixed areas but whose aspect ratios may be modified) to minimize a linear combination of total die area and wirelength.

Formally, we are given a graph $G = (V,E)$, with V representing the modules to be floorplanned, and E representing the interconnects. We are also given areas $A_v \in \mathbb{R}$ for all modules v, and parameters $\gamma \in \mathbb{R}$, $\alpha_A \in \mathbb{R}$ and $\alpha_L \in \mathbb{R}$, representing the maximum allowed aspect ratio for any module, the cost weighting factor for area and the cost weighting factor for wirelength, respectively. The goal is then to find $W \in \mathbb{R}$, $H \in \mathbb{R}$, the overall width and height of the die, respectively, and, for each module v, module locations $x_v \in \mathbb{R}$, $y_v \in \mathbb{R}$, widths $w_v \in \mathbb{R}^+$ and heights $h_v \in \mathbb{R}^+$, such that all of the following conditions hold:

- All modules fit on the die: For all $v \in V$

$$x_v - \frac{1}{2}w_v \geq 0 \qquad \text{and} \qquad x_v + \frac{1}{2}w_v \leq W$$
$$y_v - \frac{1}{2}h_v \geq 0 \qquad \text{and} \qquad y_v + \frac{1}{2}h_v \leq H$$

- Modules fit in their allocated areas: For all $v \in V$

$$w_v h_v = A_v \tag{7.1}$$

- No modules overlap: For all $u \in V, v \in V, u \neq v$

$$|x_u - x_v| > \frac{1}{2}(w_u + w_v) \qquad \text{or} \qquad |y_u - y_v| > \frac{1}{2}(h_u + h_v) \tag{7.2}$$

- Modules respect the aspect ratio limit: For all $v \in V$

$$\frac{1}{\gamma} \leq w_v/h_v \leq \gamma$$

- A weighted sum of total area and wirelength is minimized. Here we use the sum-of-squares metric for wirelength, measured from the module centers. For the cost function we take

$$\alpha_A WH + \alpha_L \sum_{(u,v)\in E} (x_u - x_v)^2 + (y_u - y_v)^2$$

Note that some simplification is possible; we may eliminate the area constraints (7.1) and substitute $h_v = A_v/w_v$ in the above. Also note that W and H are completely determined given the above constraints.

The floorplanning problem is nonlinear. Moreover, the problem is not smooth: the non-overlap constraints (7.2) are not continuously differentiable in the region of interest. We take a two-phase approach to dealing with the non-smoothness of the problem. We first use a smooth approximation to the floorplanning problem to derive a layout which may contain some violations of the above constraints, but is close to a legal floorplan. Given this approximate solution, we apply a legalization procedure which resolves the violations to produce a valid floorplan.

7.3 Existing Work

In nearly all existing works in high-level floorplanning, the resolution of the non-overlap constraints is performed through the generation of topological constraints for the modules. A common representation of these topological constraints is an *HV-constraint graph pair*, which is a pair of acyclic directed graphs, where each graph contains a vertex for each module. In the H-constraint graph, a directed edge (u,v) represents the constraint $x_v > x_u + \frac{1}{2}(w_u + w_v)$ (an *H-constraint*), while in the V-constraint graph, a directed edge (u,v) represents the constraint $y_v > y_u + \frac{1}{2}(h_u + h_v)$ (a *V-constraint*). Additionally, for every pair of modules $\{u,v\}$, there is exactly one edge between those modules (either (u,v) or (v,u)) in the HV-constraint graph pair.

Given an HV-constraint graph pair, the non-overlap constraints can be simplified to linear constraints. Moreover, it is easy to see that any solution to the floorplanning problem corresponds to at least one (that is, not necessarily unique) HV-constraint graph pair. Based on this observation, most of the existing floorplanning works approach floorplanning as a discrete search problem through the space of possible HV-constraint graph pairs, typically using simulated annealing as the core algorithm. Examples of this approach include the sequence-pair technique [MFNK96], the bounded sliceline grid [NFMK96], the O-Tree representation [GCY99], and the corner block list [H$^+$00]. Discrete search techniques are problematic to implement in a general nonlinear programming framework, however.

A number of works address the nonlinearities arising from the area constraints (7.1). [CK00] uses a linear programming approach to approximate these constraints. [MK98] uses a transformation technique to construct a convex optimization problem with convex constraints. Although these techniques are useful in simplifying the area constraints (7.1), they require an HV-constraint

graph pair be provided as input, and so are not complete solutions to the floorplanning problem in themselves. That is, they must be augmented with a strategy to search the space of feasible HV-constraint graph pairs.

7.4 Technique

Here we present our proposed nonlinear floorplanning technique.

7.4.1 Overall Flow

Algorithm 7.1 gives the overall flow for the nonlinear floorplanning technique. The procedure is iterative, and alternates between two subroutines, PACK and SMOOTH. PACK finds a legal packing for the modules given their current locations and aspect ratios as a guess as to their relative placements. SMOOTH formulates a nonlinear program for a smoothed version of the floorplanning problem and solves it using the current locations as the starting point for LANCELOT. PACK and SMOOTH are described in Sections 7.4.2 and 7.4.3, respectively.

Algorithm 7.1 Overall Nonlinear Floorplanning Algorithm

```
1: Input: initial module locations and aspect ratios
2: PACK (Algorithm 7.2)
3: Measure wirelength
4: repeat
5:    SMOOTH (Algorithm 7.3)
6:    PACK (Algorithm 7.2)
7:    Measure wirelength
8: until no improvement
```

The starting point for the algorithm is arbitrary and need not be legal. We initially choose module locations randomly and set aspect ratios to 1. A call to PACK is made prior to the first call to SMOOTH in order to resolve any initial overlaps among the modules. This is in order to provide good initial positions to LANCELOT for the smoothed floorplanning procedure.

7.4.2 Packing Algorithm

The packing algorithm is shown in Algorithm 7.2. The goal here is to take the current module locations and aspect ratios, which may violate some of the floorplanning constraints, and derive a legal floorplan within the die boundary with minimally perturbation. First, topological constraints are extracted from the current module locations. Given these additional constraints, a nonlinear program is formulated for the new constrained floorplanning problem. LANCELOT is then invoked to solve this problem.

Algorithm 7.2 Packing Algorithm

1: Input: current module locations
2: Extract topological constraints from module locations
3: Generate nonlinear program for packing using topological constraints
4: Invoke LANCELOT to solve nonlinear program

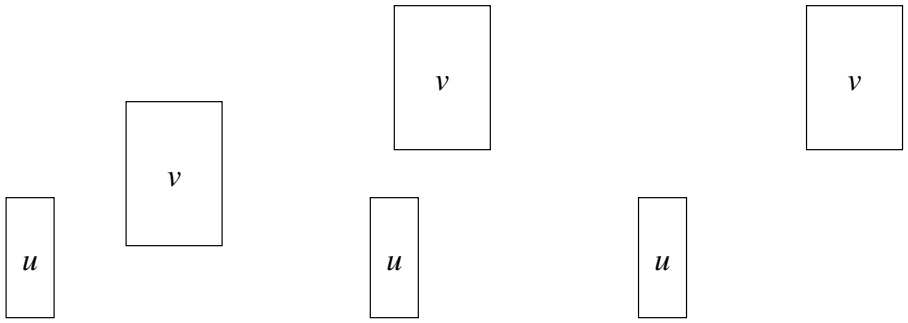
We use the following technique to derive an HV-constraint graph pair from the current module locations. For every pair of modules $\{u, v\}$, we must choose whether to add an H-constraint edge or a V-constraint edge between u and v, and we wish this choice to fall naturally from the current locations and aspect ratios of the modules. Assume for the moment $x_u < x_v$ and $y_u < y_v$ and the modules do not overlap in their current placement. We are particularly interested in the following properties:

- If there exists a horizontal line which passes through both modules u and v, we must add a constraint in the H-constraint graph between the modules. Figure 7.1(a) illustrates this situation. In this case, adding a V-constraint between the modules does not make sense, as the current placement would already violate such a constraint.

- If there exists a vertical line which passes through both modules u and v, we must add a constraint in the V-constraint graph between the modules. Figure 7.1(b) illustrates this situation. This is analogous to the previous property.

- If neither a vertical nor horizontal line exists which intersects both modules, we have flexibility to choose add either an H-constraint or V-constraint between the modules. See Figure 7.1(c) for an example.

We propose the following technique for extracting suitable topological constraints between the modules. Suppose we could *scale* the modules (that is, increase their sizes) uniformly



(a) H-constraint (b) V-constraint (c) Either H or V

Figure 7.1: Examples for topological constraint determination. Desired constraint types indicated.

and continuously until u and v abut along some edge. The abutting edge indicates a natural constraint between the modules. If the abutting edge of the scaled modules were horizontal we add a vertical constraint between the modules. Likewise, if the abutting edge were vertical, a horizontal constraint is natural. This procedure captures the properties for the constraints as described above and can also be easily computed. We formally determine the relative constraint between modules u and v by computing the following *sizing factors*:

$$k^x_{u,v} = \frac{2|x_u - x_v|}{w_u + w_v} \qquad \text{and} \qquad k^y_{u,v} = \frac{2|y_u - y_v|}{h_u + h_v}$$

We observe the following properties of these sizing factors:

Observation 7.1. *If the modules u and v are kept in their current locations but resized by the sizing factor $k^x_{u,v}$ simultaneously, the resulting floorplan will have the right edge of u collinear with the left edge of v; that is, $x_u + k^x_{u,v}(\frac{1}{2}w_u) = x_v - k^x_{u,v}(\frac{1}{2}w_v)$. Likewise, if the modules are scaled by $k^y_{u,v}$, then the top edge of u becomes collinear with the bottom edge of v.*

Observation 7.2. *If the modules u and v are resized by some factor s which is less than $k^x_{u,v}$, then the right edge of u will lie to the left of the left edge of v; that is, $x_u + s(\frac{1}{2}w_u) < x_v - s(\frac{1}{2}w_v)$. An analogous situation arises in the y-direction.*

Observation 7.3. *The modules u and v can be scaled up to a factor of* $\max(k^x_{u,v}, k^y_{u,v})$ *before overlap results between the modules.*

Based on these observations, we add an H-constraint between u and v if $k^y_{u,v} > k^x_{u,v}$, otherwise we add a V-constraint.

Note that the calculation of $k^x_{u,v}$ and $k^y_{u,v}$ do not depend on the previous assumptions that $x_u < x_v$ and $y_u < y_v$; that is, regardless of the relative locations of the modules, the comparison of $k^x_{u,v}$ and $k^y_{u,v}$ still generates the proper constraints between u and v. Moreover, we can also remove the assumption that the current module locations do not overlap. We thus determine the direction of the constraint edge between u and v by the relative locations of the modules. For instance, if we must add an H-constraint edge, this edge is chosen to be (u,v) if $x_u < x_v$, else we choose edge (v,u).

In theory, degenerate situations may arise where $x_u = x_v$ and $y_u = y_v$, i.e. the modules are exactly coincident, or when $k^x_{u,v} = k^y_{u,v}$. In such cases, the physical information provides no useful guide to the constraint addition process, and we can only choose a constraint arbitrarily.

Once we have generated an HV-constraint graph pair, the non-overlap constraints are simplified to become linear inequalities. The resulting system is passed to LANCELOT to be solved.

7.4.3 Smoothed Floorplan Algorithm

The packing algorithm of Section 7.4.2 is limited in that it is useful for resolving violations of the floorplanning constraints described in Section 7.2, but does not explore larger movements of modules. We propose using the smoothed floorplanning algorithm here as a method for generating global movements of modules. This algorithm is shown in Algorithm 7.3.

Algorithm 7.3 Smoothed Floorplan Algorithm

1: Generate smoothed version of non-overlap constraints

2: Invoke LANCELOT to solve nonlinear program

Recall the non-overlap constraints are (7.2):

$$|x_u - x_v| > \frac{1}{2}(w_u + w_v) \qquad \text{or} \qquad |y_u - y_v| > \frac{1}{2}(h_u + h_v)$$

These constraints are non-smooth due to the disjunctive *or*. Our goal is to relax these constraints so that the problem given to LANCELOT is smooth and more readily solvable. Since we can use the packing algorithm (Algorithm 7.2) to resolve overlaps in the final solution, we do not need to consider module overlaps at this stage. Note that for a fixed (x_u, y_u), the constraints define a "keepout" rectangle which constrains the placement of module v.

The non-overlap constraints can be written in terms of the ℓ^∞-norm as:

$$\left|\mathbf{q}_{uv}(\mathbf{p}_u - \mathbf{p}_v)^T\right|_\infty > 1$$

where

$$\mathbf{p}_u = \begin{bmatrix} x_u \\ y_u \end{bmatrix}, \qquad \mathbf{p}_v = \begin{bmatrix} x_v \\ y_v \end{bmatrix}, \qquad \mathbf{q}_{uv} = \begin{bmatrix} 2/(w_u + w_v) \\ 2/(h_u + h_v) \end{bmatrix}$$

The idea we use here is to replace the non-smooth ℓ^∞-norm with the smooth ℓ^2-norm. This smoothing yields

$$\left| \mathbf{q}_{uv} (\mathbf{p}_u - \mathbf{p}_v)^T \right|_2 > 1$$

or

$$\frac{4}{(w_u + w_v)^2}(x_u - x_v)^2 + \frac{4}{(h_u + h_v)^2}(y_u - y_v)^2 > 1 \tag{7.3}$$

The keepout region becomes an ellipse under this relaxation as shown in Figure 7.2.

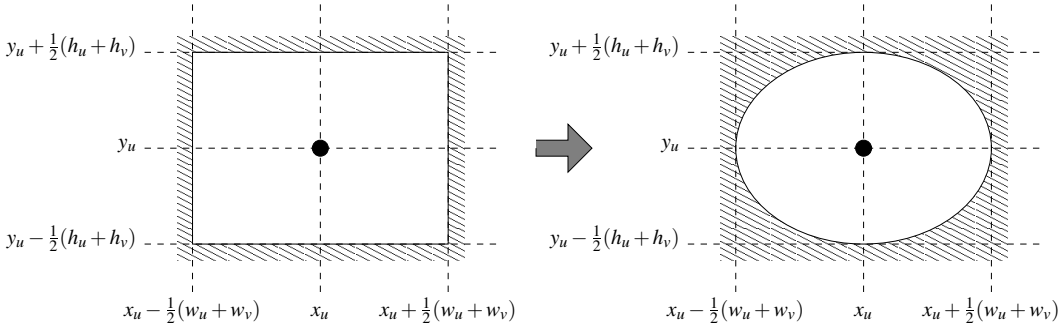
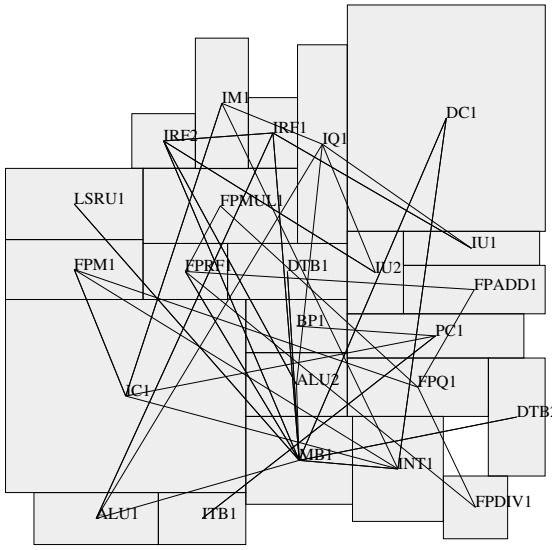


Figure 7.2: Smoothing of non-overlap constraints. Shading indicates feasible region for module v relative to module u.

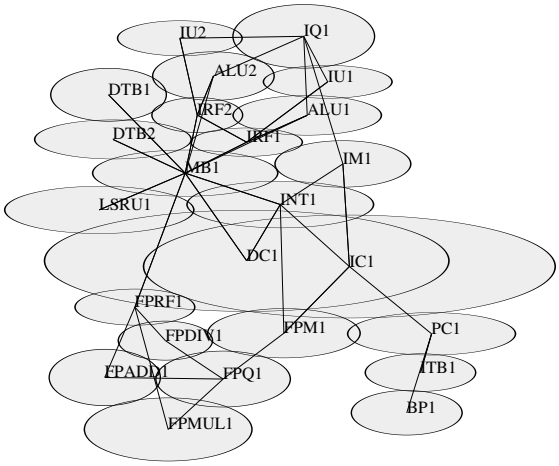
In theory, higher-order norms (ℓ^4, ℓ^6, etc.) yield smooth approximations which are closer to the original non-overlap constraints. In practice, we found the ℓ^2-norm gave just as good results using LANCELOT with much faster run times.

7.5 Experiments

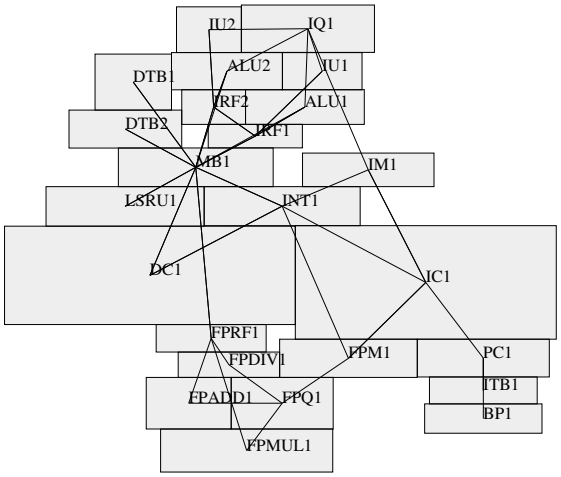
To illustrate the algorithm, Figure 7.3 shows the results of our nonlinear programming floorplanner when run on the abstracted Alpha design described in Section 3.4. Figure 7.3(a) shows a floorplan generated by starting with a random arrangement and applying the packing algorithm. Figure 7.3(b) shows the result after the first application of the smoothed floorplanning procedure; modules are shown as ellipses to illustrate the effects of the smoothing on the overlaps of the modules. Figure 7.3(c) shows the result after packing this placement.



(a) Random (wlen = 250654)



(b) Smoothed (wlen = 17201)



(c) Packed (wlen = 25221)

Figure 7.3: Nonlinear programming floorplan for abstract Alpha 21264 design. Nets are shown in black lines, and sum-of-square wirelength metrics indicated. Visualizations are independently scaled so that absolute sizes may not be comparable between figures.

Figure 7.3(b) illustrates the prime shortcoming of our floorplanning algorithm. There tends to be much wasted area as the smoothed constraints fail to spread the modules evenly over the die area. The poor spreading in the solution to the smooth floorplan carries forward to the packing. Figure 7.3(c) shows much wasted space in the top right corner of the die. This poor utilization remains regardless of the weighting factors α_A and α_L chosen for the nonlinear cost function.

We show results for our nonlinear floorplanning algorithm applied to problems derived from the ISCAS85 circuit benchmarks [Yan91]. The unmapped designs were considered as floorplanning instances, with the nodes from the original netlist representing the modules to be floorplanned. The connectivity between modules was taken as the nets between the nodes, and the sizes of the modules was taken as proportional to the number of minterms in the original BLIF representation. We took the maximum aspect ratio parameter $\gamma = 4$. For this experiment, we arbitrarily took $\alpha_A = \alpha_L = 1$, noting that for real floorplanning problems these factors would have to be considered more carefully, based on the needs of the designer.

The results of our algorithm are shown in the columns labeled *Nonlinear* in Table 7.1. We compare these results to a solution based on simulated annealing using the sequence-pair technique [MFNK96]. We chose an annealing schedule which resulted in the same run time as was taken for our nonlinear floorplanning algorithm. This makes the comparison "fair", in the sense that both techniques would be given the same amount of computing resources. The results from the annealing method are shown in the columns labeled *Annealing* in Table 7.1. We note that the nonlinear algorithm yields consistently better results in both wirelength and area than the annealing technique when executed for the same amount of time.

		Nonlinear			Annealing		Long	
Design	Size	Wlen	Area	Time	Wlen	Area	Wlen	Area
cm150a	37	1020	472	872	1198	584	994	362
cm151a	21	325	320	426	465	380	298	214
cm152a	12	107	96	128	120	122	98	84
cm42a	17	769	236	376	1125	301	741	212
cm82a	11	103	134	173	153	311	112	118

Table 7.1: Results for nonlinear floorplanning. Size indicates number of modules. Wlen indicates total wire length using sum-of-squares metric. Area indicates total die area. Time indicates run time in seconds. Nonlinear indicates the method described in this chapter, Annealing indicates quick annealing, Long indicates long annealing.

Of course, setting the annealing schedule for the same run time as the nonlinear floor-

planner is an artificial restriction on the annealer. As a further comparison, we used an annealing schedule which ran for 3 hours on each of the designs. The results of this are shown in the *Long* columns of Table 7.1. This yielded consistently better results than our nonlinear floorplanning. Although our experiments were not exhaustive, we could find no technique to readily improve the nonlinear floorplanning results.

7.6 Summary

We have developed a new, nonlinear programming-based technique for for floorplanning. For the same amount of run time, our technique compares very favorably to the annealing-based, sequence-pair technique proposed by [MFNK96]. However, our method compares poorly against the annealing-based method, given additional run time for the annealer.

Visual inspection of the floorplans resulting from our nonlinear programming-based technique suggest that there are problems with the spreading of modules over the die. This could perhaps be alleviated by borrowing techniques from placement, e.g. the dissection technique from GORDIAN [KSJ88]. However, it is not clear that further exploration along this avenue will be fruitful, given the relatively good results obtained from using the annealing technique of [MFNK96].

Here we have not explored the interaction of this floorplanning technique with retiming or clock skew scheduling, having limited our study to wirelength and area optimization for now. However, net weighting approaches such as those presented in Chapter 3 and Chapter 4 can be applied in a straightforward manner. Whether other useful techniques for introducing timing or sequential flexibility in our nonlinear floorplanning framework exist remains an open issue.

Chapter 8

Conclusions

The progress of technology has highlighted the need to work towards the goal of integrating previously disparate design and optimization techniques. To this end, we have shown theoretical results to justify our approach to the problem of integration of physical design and sequential optimization, and successfully applied a number of sequentially-aware floorplanning and placement techniques to various benchmark designs.

There are still a number of open questions and avenues of research left to explore in this area. While our placement techniques have been validated on industrial design examples, we do note that more experimental results would be beneficial in demonstrating the benefits of our floorplanning approaches. Also, there are other design goals, such as routability and congestion, that can be explicitly addressed to further extend this work. However, our work demonstrates that the integration of physical design and sequential optimization can be successful for physically-aware timing optimization of digital circuits in today's wire-dominant technologies.

Bibliography

[A+99] Charles J. Alpert et al. Analytic engines are unnecessary in top-down partitioning-based placement. *VLSI Design*, 10(1):99–116, 1999.

[All80] Arnold O. Allen. Queueing models of computer systems. *IEEE Computer*, pages 13–24, April 1980.

[Alp00] Exploring Alpha power for technical computing. Technical report, High Performance Technical Computing Group, Compaq Computer Corporation, April 2000.

[Ass97] Semiconductor Industry Association. National technology roadmap for semiconductors, 1997.

[BCMP75] Forest Baskett, K. Mani Chandy, Richard R. Muntz, and Fernando Palacios. Open, closed and mixed networks of queues with different classes of customers. *Journal of the ACM*, 22(2):248–260, April 1975.

[Bre77] Melvin A. Breuer. Min-cut placement. *Journal of Design Automation and Fault-Tolerant Computing*, 1(4):343–362, October 1977.

[BV04] Ulrich Brenner and Jens Vygen. Legalizing a placement with minimum total movement. *IEEE Transactions on Computer-Aided Design*, 23(12):1597–1613, December 2004.

[C+99] Luca. P. Carloni et al. A methodology for "correct-by-construction" latency insensitive design. In *IEEE/ACM International Conference on Computer-Aided Design*, pages 309–315, 1999.

[CGT92] Andrew R. Conn, N.I.M. Gould, and Ph.L. Toint. *LANCELOT: A Fortran Package for Large-Scale Nonlinear Optimization (Release A)*. Springer-Verlag, 1992.

[CK00] Pinhong Chen and Ernest S. Kuh. Floorplan sizing by linear programming approximation. In *Design Automation Conference*, pages 468–471, 2000.

[CKM+99a] Andrew E. Caldwell, Andrew B. Kahng, S. Mantik, I.L. Markov, and A. Zelikovsky. On wirelength estimations for row-based placement. *IEEE Transactions on Computer-Aided Design*, 18(9):1265–1278, September 1999.

[CKM99b] Andrew E. Caldwell, Andrew B. Kahng, and Igor L. Markov. Design and implementation of the Fiduccia-Mattheyses heuristic for VLSI netlist partitioning. In *Workshop on Algorithm Engineering and Experimentation*, pages 177–193, 1999.

[CKM00] Andrew E. Caldwell, Andrew B. Kahng, and Igor L. Markov. Can recursive bisection alone produce routable placements? In *Design Automation Conference*, pages 477–482, 2000.

[CL00] Jason Cong and Sung Kyu Lim. Physical planning with retiming. In *IEEE/ACM International Conference on Computer-Aided Design*, pages 1–7, 2000.

[CLR90] Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Introduction to Algorithms*. The MIT Press, 1990.

[CM05] Mario R. Casu and Luca Macchiarulo. Floorplan assisted data rate enhancement through wire pipelining: A real assessment. In *International Symposium on Physical Design*, pages 121–128, 2005.

[CSV00] L.P. Carloni and A.L. Sangiovanni-Vincentelli. Performance analysis and optimization of latency insensitive protocols. In *Design Automation Conference*, pages 361–367, 2000.

[CTCG+98] J. Cochet-Terrasson, G. Cohen, S. Gaubert, M. McGettrick, and J.-P. Quadrat. Numerical computation of spectral elements in max-plus algebra. In *Proceedings of the IFAC Conference on System Structure and Control*, July 1998.

[CY03] Jason Cong and Xin Yuan. Multilevel global placement with retiming. In *Design Automation Conference*, pages 208–213, 2003.

[DD96] J. Darnauer and W.W.-M. Dai. A method for generating random circuits and its application to routability measurement. In *ACM International Symposium on Field-Programmable Gate Arrays*, pages 66–72, 1996.

[DG98] Ali Dasdan and Rajesh K. Gupta. Faster maximum and minimum mean cycle algorithms for system performance analysis. *IEEE Transactions on Computer-Aided Design Of Integrated Circuits and Systems*, 17(10):889–899, 1998.

[DIG98] Ali Dasdan, Sandra S. Irani, and Rajesh K. Gupta. An experimental study of minimum mean cycle algorithms. Technical Report UCI-ICS 98-32, University of California, Irvine, 1998.

[DJS91] K. Doll, F.M. Johannes, and G. Sigl. DOMINO: Deterministic placement improvement with hill-climbing capabilities. In *IFIP International Conference on VLSI*, pages 91–100, 1991.

[DS95] R.B. Deokar and S.S. Sapatnekar. A graph-theoretic approach to clock skew optimization. In *IEEE International Symposium on Circuits and Systems*, pages 407–410, 1995.

[EJ98] H. Eisenmann and F.M. Johannes. Generic global placement and floorplanning. In *Design Automation Conference*, pages 269–274, 1998.

[EMW+04] Mongkol Ekpanyapong, Jacob R. Minz, Thaisiri Watewai, Hsien-Hsin S. Lee, and Sung Kyu Lim. Profile-guided microarchitectural floorplanning for deep submicron processor design. In *Design Automation Conference*, pages 634–639, 2004.

[ESS96] G. Even, I. Spillinger, and L. Stok. Retiming revisited and reversed. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 15:348–357, 1996.

[Fis90] J.P. Fishburn. Clock skew optimization. *IEEE Transactions on Computers*, C-39:945–951, 1990.

[FM82] C.M. Fiduccia and R.M. Mattheyses. A linear-time heuristic for improving network partitions. In *IEEE Design Automation Conference*, pages 175–181, 1982.

[GCY99] Pei-Ning Guo, Chung-Kuan Cheng, and Takeshi Yoshimura. An O-tree representation of non-slicing floorplan and its applications. In *ACM/IEEE Design Automation Conference*, pages 268–273, 1999.

[GJ79] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, New York, 1979.

[GT96] E.G. Golshtein and N.V. Treityakov. *Modified Lagrangians and Monotone Maps in Optimization*. John Wiley and Sons, 1996.

[H+00] Xianlong Hong et al. Corner block list: An effective and efficient topological representation of non-slicing floorplan. In *IEEE/ACM International Conference on Computer-Aided Design*, pages 8–12, 2000.

[HKM+03] Stephan Held, Bernhard Korte, Jens Maßberg, Matthias Ringe, and Jens Vygen. Clock scheduling and clocktree construction for high-performance ASICs. In *IEEE/ACM International Conference on Computer-Aided Design*, pages 232–239, 2003.

[HMS02] Bo Hu and Malgorzata Marek-Sadowska. FAR: Fixed-points addition & relaxation based placement. In *International Symposium on Physical Design*, pages 161–166, 2002.

[HRGC98] M.D. Hutton, J. Rose, J.P. Grossman, and D.G. Corneil. Characterization and parameterized generation of synthetic combinational benchmark circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 17(10):985–996, October 1998.

[ITR03] The international technology roadmap for semiconductors. Technical report, Semiconductor Industry Association, 2003.

[KAKS97] George Karypis, Rajat Aggarwal, Vipin Kumar, and Shashi Shekhar. Multilevel hypergraph partitioning: Application in VLSI domain. In *Design Automation Conference*, pages 526–529, 1997.

[Kar78] Richard Karp. A characterization of the minimum cycle mean in a digraph. *Discrete Mathematics*, 23:309–311, 1978.

[Kes98] The Alpha 21264 microprocessor architecture. Technical report, Compaq Computer Corporation, 1998.

[KF99] Ivan S. Kourtev and Eby G. Friedman. Synthesis of clock tree topologies to implement nonzero clock skew schedule. In *IEE Proceedings on Circuits, Devices, Systems*, pages 321–326, 1999.

[KK99] George Karypis and Vipin Kumar. Multilevel k-way hypergraph partitioning. In *Design Automation Conference*, pages 343–348, 1999.

[KMR04] A.B. Kahng, I.L. Markov, and S. Reda. On legalization of row-based placements. In *ACM Great Lakes Symposium on VLSI*, pages 214–219, 2004.

[KSJ88] Jürgen M. Kleinhans, Georg Sigl, and Frank M. Johannes. GORDIAN: A global optimization/rectangle dissection method for cell placement. In *IEEE/ACM International Conference on Computer-Aided Design*, pages 506–509, 1988.

[KSJA91] Jürgen M. Kleinhans, Georg Sigl, Frank M. Johannes, and Kurt J. Antreich. GORDIAN: VLSI placement by quadratic programming and slicing optimization. *IEEE Transactions on Computer-Aided Design*, 10(3):356–365, March 1991.

[Lim00] Sung Kyu Lim. *Performance Driven Circuit Partitioning*. PhD thesis, University of California, Los Angeles, 2000.

[LS83] C.E. Leiserson and J.B. Saxe. Optimizing synchronous systems. *Journal of VLSI and Computer Systems*, 1(1):41–67, January 1983.

[LS91] C.E. Leiserson and J.B. Saxe. Retiming synchronous circuitry. *Algorithmica*, 6(1):5–35, 1991.

[LSLH04] Changbo Long, Lucanus J. Simonson, Weiping Liao, and Lei He. Floorplanning optimization with trajectory piecewise-linear model for pipelined interconnects. In *Design Automation Conference*, pages 640–645, 2004.

[MBH+02] Deborah T. Marr, Frank Binns, David L. Hill, Glenn Hinton, David A. Koufaty, J. Alan Miller, and Michael Upton. Hyper-threading technology architecture and microarchitecture. *Intel Technology Journal*, 6(1):4–15, February 2002.

[MFNK96] H. Murata, K. Fujiyoshi, S. Nakatake, and Y. Kajitani. VLSI module placement based on rectangle-packing by the sequence-pair. *IEEE Transactions on CAD/ICAS*, 15(12):1518–1524, 1996.

[MK98] Hiroshi Murata and Ernest S. Kuh. Sequence-pair based placement method for hard/soft/preplaced modules. In *International Symposium on Physical Design*, pages 167–172, 1998.

[Mon] `http://www.maths.monash.edu.au/mth3111/Markov.pdf`.

[MPS02] William Magro, Paul Petersen, and Sanjiv Shah. Hyper-threading technology: Impact on compute-intensive workloads. *Intel Technology Journal*, 6(1):58–66, February 2002.

[Mur89] Tadao Murata. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, 77(4):541–579, April 1989.

[NFMK96] S. Nakatake, K. Fujiyoshi, H. Murata, and Y. Kajitani. Module placement on BSG-structure and IC layout applications. In *IEEE/ACM International Conference on Computer-Aided Design*, pages 484–491, 1996.

[OB98] R.H. Otten and R.K. Brayton. Planning for performance. In *ACM/IEEE Design Automation Conference*, pages 122–127, 1998.

[Ott82] R.H.J.M. Otten. Layout structures. In *IEEE International Large Scale Systems Symposium*, pages 349–353, 1982.

[PKL98] P. Pan, A.K. Karandikar, and C.L. Liu. Optimal clock period clustering for sequential circuits with retiming. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 17(6):489–498, June 1998.

[PR02] Panos M. Pardalos and Mauricio G.C. Resende, editors. *Handbook of Applied Optimization*. Oxford University Press, 2002.

[PS98] Christos H. Papadimitriou and Kenneth Steiglitz. *Combinatorial Optimization: Algorithms and Complexity, Dover Edition*. Dover Publications, Inc., 1998.

[RKS03] K. Ravindran, A. Kuehlmann, and E. Sentovich. Multi-domain clock skew scheduling. In *IEEE/ACM International Conference on Computer-Aided Design*, 2003.

[SB02] Deshanand P. Singh and Stephen D. Brown. Incremental placement for layout-driven optimizations on FPGAs. In *IEEE/ACM International Conference on Computer-Aided Design*, pages 752–759, 2002.

[SBSV92] N. Shenoy, R.K. Brayton, and A.L. Sangiovanni-Vincentelli. Graph algorithms for clock schedule optimization. In *IEEE/ACM International Conference on Computer-Aided Design*, pages 132–136, 1992.

[SCK92] A. Srinivasan, K. Chaudhary, and E.S. Kuh. RITUAL: A performance driven placement algorithm. *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, 39(11):825–840, November 1992.

[SDC99] Dirk Stroobandt, Jo Depreitere, and Jan Van Campenhout. Generating new benchmark designs using a multi-terminal net model. *Integration, The VLSI Journal*, 27(2):113–129, July 1999.

[SK99] Dennis Sylvester and Kurt Keutzer. Rethinking deep-submicron circuit design. *IEEE Computer*, 32(11):25–33, November 1999.

[SSBSV92] N.V. Shenoy, K.J. Singh, R.K. Brayton, and A.L. Sangiovanni-Vincentelli. On the temporal equivalence of sequential circuits. In *ACM/IEEE Design Automation Conference*, pages 405–409, 1992.

[Szy92] T.G. Szymanski. Computing optimal clock schedules. In *ACM/IEEE Design Automation Conference*, pages 399–404, 1992.

[TB93] H.J. Touati and R.K. Brayton. Computing the initial states of retimed circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 12(1):157–162, January 1993.

[TK91a] Ren-Song Tsay and Ernest Kuh. A unified approach to partitioning and placement. *IEEE Transactions on Circuits And Systems*, 38(5):521–533, May 1991.

[TK91b] R.S. Tsay and J. Koehl. An analytic net weighting approach for performance optimization in circuit placement. In *Design Automation Conference*, pages 620–625, 1991.

[TKH88] Ren-Song Tsay, Ernest S. Kuh, and Chi-Ping Hsu. PROUD: A sea-of-gates placement algorithm. *IEEE Design & Test Of Computers*, 5(6):44–56, December 1988.

[VC99] Chandu Visweswariah and Andrew R. Conn. Formulation of static circuit optimization with reduced size, degeneracy and redundancy by timing graph manipulation. In *IEEE/ACM International Conference on Computer-Aided Design*, pages 244–251, 1999.

[VC04] N. Viswanathan and C. Chu. FastPlace: An efficient analytical placement technique using cell spreading and iterative local refinement. In *International Symposium on Physical Design*, pages 26–33, 2004.

[WL86] D. Wong and C. Liu. A new algorithm for floorplan design. In *Design Automation Conference*, pages 101–107, 1986.

[WS00] Maogang Wang and Majid Sarrafzadeh. Modeling and minimization of routing congestion. In *ASP-ACM/IEEE Design Automation Conference*, pages 185–190, 2000.

[XD97] Joe Gufeng Xi and Wayne Wei-Ming Dai. Useful-skew clock routing with gate sizing for low power design. *J. VLSI Signal Process. Syst.*, 16(2-3):163–179, 1997.

[Yan91] S. Yang. Logic synthesis and optimization benchmarks user guide version 3.0. Technical report, Microelectronic Centre of North Carolina, 1991.

[YMS03] Chao-Yang Yeh and Malgorzata Marek-Sadowska. Delay budgeting in sequential circuit with applications on FPGA placement. In *ACM/IEEE Design Automation Conference*, pages 202–207, 2003.