MIT/LCS/TR-351

THE COMPLEXITY OF GRAPH LAYOUT AND CHANNEL ROUTING FOR VLSI

Sandeep N. Bhatt

October 1985

This blank page was inserted to preserve pagination.

THE COMPLEXITY OF GRAPH LAYOUT AND CHANNEL ROUTING FOR VLSI

by

SANDEEP NAUTAM BHATT

S.B. Massachusetts Institute of Technology
(1978)

S.M. Massachusetts Institute of Technology
(1980)

Submitted to the Department of Electrical Engineering and Computer Science
in Partial Fulfillment of the Requirements for the Degree of

DOCTOR OF PHILOSOPHY

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

February 1984

© Massachusetts Institute of Technology 1984

Key Words: VLSI graph layout, Bifurcators, Divide-and-Conquer, Fault-tolerance, Wire delay, Restructurable design, Channel routing, density, flux, approximation algorithm

This empty page was substituted for a blank page in the original document.

Acknowledgements

To Charles Leiserson for his constant encouragement and guidance. He taught me how to express myself, and gave me confidence in my abilities. I was fortunate to collaborate with Charles and with Tom Leighton, and to share in their boundless enthusiasm, unrelenting flow of ideas, and friendship. Thanks also to Shafi Goldwasser for graciously agreeing to serve on my thesis committee at the last minute.

To Ron Rivest for his support and gentle guidance through my years in graduate school, and for inspiring me to pursue research in algorithms. Mike Sipser and Christos Papadimitriou stimulated my interest in theory further, Mike with his characteristic knack of making every argument appear beautiful, and Christos with his natural ebullience.

Life would have been deadly dull without my fellow sufferers Keshav, Varghese, and Vinod. Steve Trilling kept up my morale with his endless repertoire of one-liners. This document would never have been prepared without all his help. Benny Chor, Stavros Cosmadakis and John Mitchell made even technical discussions pleasant. Arline, Andy, Flavio and Ray helped maintain my sanity over the years.

To Inna for her warm and steadfast friendhip, and to Susan Landau and Neil Immerman who have helped me in so many ways.

It has been my good fortune to be part of so many families. The Schecters and the Lymans and Mumpuc have been wonderfully supportive and warm. Thank you for being so special.

To Ravin and Swati who will always be concerned.

To Papaji and Mummy who make everything worthwhile.

TABLE OF CONTENTS

Abstract 2
Acknowledgements 3
Table of contents 4

1. PERSPECTIVE 6
   1.1 The complexity of VLSI graph layout 8
   1.2 The complexity of channel routing 11
   1.3 Overview 12

2. ISSUES IN VLSI GRAPH LAYOUT 13
   2.1 The layout model 13
   2.2 Elementary bounds on layout area 14
   2.3 Layouts based on separator theorems 15
   2.4 Eight VLSI graph layout problems 18

3. LAYOUTS FOR TREES 24
   3.1 Layouts for complete binary trees 25
   3.2 Layouts for arbitrary binary trees 28
   3.3 Planar layouts for trees 31
   3.4 The complexity of minimizing edge lengths 35
   3.5 Assembling complete trees 38
   3.6 Collinear layouts and two-color bisectors 40
   3.7 Assembling arbitrary trees 43

4. THE GENERAL FRAMEWORK 46
   4.1 Combinatorial lemmas 47
   4.2 Decomposition trees and bifurcators 51
      4.2.1 Special cases 53
   4.3 Balanced decomposition trees 54
   4.4 Embeddings in the tree of meshes 56
   4.5 Layouts for the tree of meshes 60

5. SOLVING THE LAYOUT PROBLEMS 63

6. THE CHANNEL ROUTING PROBLEM 72
6.1 Manhattan routing within channels 73
6.2 Bounds on channel width 75
6.3 Bounds for other wiring models 77
7. AN APPROXIMATION ALGORITHM FOR MANHATTAN ROUTING 79
7.1 Channel flux 79
7.2 An approximation algorithm for top-to-bottom nets 82
7.3 Running time analysis 89
7.4 The channel routing algorithm 90

8. CONCLUSIONS, EXTENSIONS AND OPEN PROBLEMS 96
8.1 Problems in graph layout 97
8.2 Problems in channel routing 98

BIBLIOGRAPHY 100

CHAPTER 1

Perspective

Advances in integrated circuit technology have had a revolutionary impact on computer system design. A chip today integrates far greater sophistication and computing power than ever before. Fabrication processes have progressed rapidly so that chips with one million components are a reality, and enthusiasts predict chips with upto one hundred million components within a decade. Indeed, it is expected that if ion beam etching techniques become viable for "printing" chips directly, then minimum feature sizes would drop by a factor of ten, thus allowing a hundred-fold increase in the number of components on a chip.

More significantly, the new technology encourages custom design of special purpose integrated systems for solving very large scale sophisticated problems. No longer is it necessary to use a single conventional architecture for solving diverse problems. Instead, the computational structure of a problem may be mapped directly into hardware. This has shifted the emphasis from searching for algorithms, necessarily convoluted to suit a given architecture, to efficient hardware design suited to individual problems.

While this emphasis on greater design flexibility has opened up new directions in computing, a number of difficult problems must be addressed before the emerging technologies can be effectively exploited. Probably the most significant development in easing the awesome task of designing and implementing large systems has been the standardization of design rules and the widespread use of standard building blocks. The design methodology expounded by Mead and Conway [55], and the development of building blocks such as gate-arrays, PLA's, and ROM's has helped shift the emphasis in circuit design from the exclusive domain of electronics to a higher, more functional level, where aspects of circuit layout may be treated in purely

geometrical terms.

This thesis examines various aspects of the circuit layout problem. We address questions such as: why is circuit layout difficult, what properties of a circuit critically determine the quality of its layout, and what kinds of heuristics can help solve layout problems efficiently? These questions are motivated by the need for general techniques for laying out very large circuits. Such basic issues must be addressed before building any automatic or computer-aided design and layout system.

Although the circuit layout problem is not new, progress has been painfully slow. The proliferation of diverse technologies and concerns has only exacerbated the layout problem. On the one hand we desire to minimize layout area, signal delays, and power dissipation, while on the other hand we need to increase reliability by increased redundancy. In addition we require that custom circuits be assembled using standard configurable or restructurable chips as building blocks. It is not at all clear whether these different requirements are compatible or necessarily contradictory.

Part I presents a general theory for VLSI graph layout. Not only does the theory identify structural properties of circuits that critically determine the quality of layouts, but also provides techniques for solving various layout problems. Perhaps the most significant result that emerges is a general framework for solving diverse problems in a simple and uniform manner. In particular, the unified framework provides a layout technique which is suitable for custom layout, and at the same time is efficient with regard to area, delay, and fault-tolerance. Part I consists of Chapters 2 through 5.

Part II examines the channel routing problem. Algorithms for channel routing form the basis of many existing automatic layout systems. Although this problem has received wide attention over the last decade and a number of heuristic algorithms have been proposed, none of these is guaranteed to always determine efficient routings. Approaching this problem from a theoretical viewpoint, we characterize completely the properties that make channel routing difficult. Moreover, we provide a novel, linear-time algorithm that is always guaranteed to find near-optimal solutions. Chapters 6 and 7 constitute Part II of this thesis.

Although the two parts of the thesis investigate different problems, they share a common underlying philosophy. We begin with a theoretical characterization of the properties that make the problems difficult. In the next step, algorithmic techniques are developed for exploiting these properties to solve the problems. Although the results in their present form are primarily theoretical in nature, the techniques provide new insights and approaches for VLSI layout. It is likely that some of the techniques can be adapted for use in practice.

The remainder of this chapter discusses the two parts of the thesis in more detail, and concludes with an outline of the thesis.

1.1. The Complexity of VLSI Graph Layout

In recent years a number of interconnection networks have been proposed for solving diverse problems. For example, one- and two-dimensional arrays of processors are naturally suited to vector and matrix computations [50]. Binary trees are particularly attractive because of their logarithmic depth and have been proposed for a variety of applications including raster graphics [27], databases [75], and direct execution of applicative programming languages [54]. The mesh of trees [19, 44, 57] combines arrays and trees in an elegant manner. By virtue of their sophisticated structure, networks such as the shuffle-exchange network [73], cube-connected cycles network [63], and fast-fourier transform network [76], in which recursive algorithms are programmed conveniently in a natural manner, are computationally more versatile and powerful than the simpler array structures.

Can we exploit the power of sophisticated networks in VLSI? This question becomes increasingly important as problem sizes, and the number of processors increase. It might be relatively simple to fit a thousand processor array on one chip, but can we fit a thousand processor shuffle-exchange network on one chip? Moreover, even if the shuffle-exchange network fits, will its performance, determined by the clockperiod or longest delay, be comparable to the array? To answer such questions, and to compare the relative merits of different networks, it is necessary to develop a general theory for VLSI graph layout.

Research in layout theory was initiated by Thompson [79, 80] who proposed a formal model

for VLSI graph layout and investigated area-time tradeoffs for computing certain functions. Using information-transfer arguments, he obtained strong lower bounds on the layout areas of graphs such as the shuffle-exchange and cube-connected cycles graphs. Subsequently, Leiserson [49, 50] and Valiant [83], focussing on the problem of minimizing layout area, independently developed a divide-and-conquer layout strategy for general classes of graphs. Using elegant combinatorial arguments, Leighton [40, 41] showed that the bounds of Leiserson and Valiant were the best possible in that each class contained graphs for which the bounds were, upto constant factors, optimal. For some graphs however, the bounds were very weak.

Layout area is not the only consideration in choosing one layout over a multitude of possible layouts. In practice, we desire to fabricate *small, inexpensive,* and *easily testable* chips which compute *quickly* and *reliably.* A large number of important engineering issues need to be considered in fulfilling these (possibly conflicting) requirements.

Propagation delays across long wires critically affect the performance of a circuit layout. In pipelined or systolic systems, long delays determine the clockperiod and overall performance of the system. Since propagation delay can be reduced by decreasing wire length, it is important to make the longest wire in the layout as short as possible. Another way to reduce the propagation delay across a long wire is by increasing the size of the transistor that drives the wire; by carefully adjusting transistor sizes to match wire lengths, the clockperiod can be dramatically reduced. Since wire delays determine the efficiency of a chip, it is imperative that techniques to minimize delay be developed within a general theory for VLSI layout.

Fault tolerance is another important design consideration. Fabrication processes are prone to errors so that every wafer invariably contains a small number of defects. Even if a wafer contains a number of defective processors, it may still be possible to use the wafer by configuring wires around the defective processors. This may, for example, be performed by laser restructuring techniques [64]. This ability to wire together processors selectively has considerable impact on sytem design. For example, how should a thousand processor wafer be designed so that a two-dimensional array can be realized using all the good processors, no matter how the defective processors are distributed?

Another major concern is the problem of assembling large systems. Researchers have

proposed networks with as many as one million processing elements [54]. Such systems are clearly too large to fit on a single chip. Whenever any system is larger than a single chip, it is necessary to partition the system among several chips which can be assembled at the printed circuit (or chip carrier) level. What is the most effective way to partition a large system among several chips? This question is pressing because although fabrication technology has been advancing at a rapid pace, the technology for packaging chips has been crawling in comparison: current projections indicate as many as one hundred million components per chip but not more than two hundred off-chip pin connections.

The economics of fabrication technology dictates that it is expensive to make one chip, but cheap to make many copies. For this reason, manufacturers of custom chips have been encouraged to make configurable designs such as gate-arrays, ROM's, and PLA's. The entire chip is manufactured, except for one mask. Given a desired configuration of the chip, a final layer of metallization connects up the circuitry in that way. Most of the design and fabrication costs are thus factored over several chips. Similarly, restructuring techniques allow a chip to be modified after fabrication. For example, "diode-busting" is used to configure PROM's (programmable read only memory) after fabrication. More recent and exciting is the prospect of "laser welding" by which connections between wires can be either made or broken after fabrication by high-intensity laser beams. Such techniques further encourage configurable design of VLSI chips. Thus, we are led to consider how to design efficient layouts which may be configured to realize, for example, arbitrary binary trees or arbitrary rectangular arrays.

Motivated by the engineering issues outlined above, Part I develops a general framework for VLSI graph layout. Within this framework all the diverse concerns mentioned above are dealt with in an efficient and uniform manner. The framework is based on a divide-and-conquer strategy for graph layout which differs significantly from the divide-and-conquer strategy of Leiserson [49, 50] and Valiant [83]. The improved strategy is based on the notion of graph bifurcators introduced by Leighton [42], and provides universally close bounds on important cost functions such as layout area and propagation delay. The results of Part I are based on the papers of Bhatt and Leiserson [8, 9], and Leighton [42]. In addition, the results of Chapters 4 and 5 appear in [7].

1.2. The Complexity of Channel Routing

Although the graph layout problems considered in Part I provide new insights and paradigms for VLSI layout, they are nonetheless abstractions of layout problems encountered in practice. Part II focuses on a specific problem confronting current automatic layout systems.

Channel routing plays a central role in automated layout systems. Most layout systems proceed by first placing modules on a chip, and then wiring together terminals on different modules that should be electrically connected. To solve the latter wiring problem, the chip is heuristically partitioned into a set of rectangular channels, and each channel is assigned a set of wires which are to pass through it. This effectively reduces a difficult "global" wiring problem to a set of disjoint (and presumably easier), "local" channel routing subproblems.

An instance of the channel routing problem is specified by a set of terminals located at fixed positions on two horizontal tracks. Each set of terminals with the same label constitutes a net which must be electrically connected by wires running in horizontal tracks and vertical columns. Figure 1.1 shows a channel with six nets. Horizontal and vertical wire segments are placed on two different layers of interconnect. The objective is to wire up all nets in a way that minimizes the channel width, which is the number of horizontal tracks used for wiring. For example, Figure 1.2 shows a minimum width wiring of the channel in Figure 1.1.

The channel routing problem has been intensively studied for over a decade, and many heuristic algorithms have been proposed for solving the problem [1, 2, 11, 12, 18, 20, 21, 34, 35, 36, 38, 51, 60, 62, 67, 68, 81, 84]. Recently, Szymanski [77] showed that the general problem is NP-complete, and with Yannakakis [78] showed that the problem is NP-complete even when every wire connects exactly two terminals. This might explain why the fast heuristic algorithms developed thus far either produce arbitrarily bad solutions in many cases and/or completely fail on other instances.

Part II of the thesis presents a linear-time algorithm which *always* produces a near-optimal solution. This algorithm is based on the key notion of channel flux which is introduced in Chapter 7. The algorithm originally appears in a paper by Baker, Bhatt, and Leighton [3].

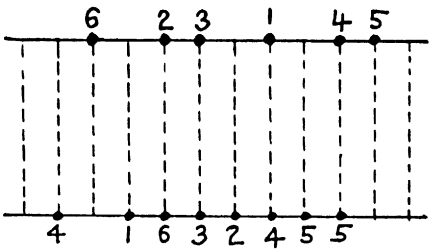


Figure 1.1: *A channel with six nets.*

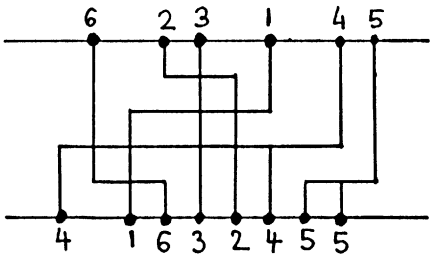


Figure 1.2: *A minimum width routing.*

1.3. Overview

The next four chapters are devoted to VLSI graph layout, and form Part I of the thesis. Chapter 2 outlines Thompson's model for VLSI layout, reviews previous research, and describes important layout problems in a formal setting. Chapter 3 focuses on layouts for the simplest of networks: binary trees. In addition to presenting new layouts with improved bounds on edge lengths, the complexity of producing optimal layouts is examined. The new layout strategy motivates the paradigm for general graph layout presented in Chapter 4. Finally, Chapter 5 shows how the new layout paradigm can be used to efficiently solve the important layout problems of Chapter 2.

Part II of the thesis consists of Chapters 6 and 7. Chapter 6 describes the channel routing problem, its use in automatic layout systems, and briefly reviews previous research. Chapter 7 introduces the concept of channel flux and presents a linear-time approximation algorithm for Manhattan routing.

In conclusion, Chapter 8 summarizes the major results of both parts and outlines a number of important, unresolved problems.

CHAPTER 2

Issues in VLSI Graph Layout

The first three sections of this chapter introduce the layout model developed by Thompson [79, 80] and briefly review previous research in VLSI graph layout. In particular, we discuss the layout strategy of Leiserson [49] and Valiant [83] and note that bounds on layout area based on separator theorems can be very different from the actual minimum layout area. The remainder of this chapter is devoted to formalizing a number of layout questions motivated by engineering considerations.

2.1. The Layout Model

In order to cast VLSI layout problems within a mathematical framework, Thompson [79, 80] developed a formal model for VLSI graph layout. The model is based on, and is consistent with, the VLSI design rules established by Mead and Conway [55]. It is also similar to the widely used Manhattan wiring model. In the *Thomspon grid model*, a layout for a graph is characterized as an embedding within a *two-dimensional grid*. A *two-dimensional grid* is a collection of horizontal and vertical *tracks* spaced apart at unit intervals. A layout for a graph G is specified by an *embedding* which assigns nodes of G to points in the grid where horizontal and vertical tracks intersect, together with an (incidence-preserving) assignment of the edges of G to *paths* in the grid. The paths of the layout are restricted to follow along grid tracks and are not allowed to overlap for any distance (although a vertical path segment may cross a horizontal path segment). In addition, the paths may not cross nodes to which they are not adjacent. For obvious reasons, we restrict our attention to graphs in which no node has degree

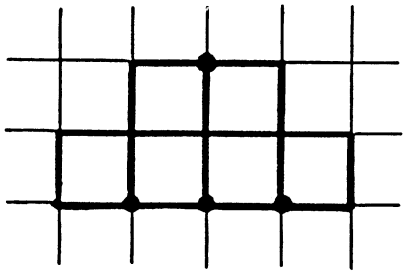


Figure 2.1: *A layout for* K_4.

greater than four. As an example, Figure 2.1 shows a layout for the complete graph on four nodes.

Remark. The results of this thesis extend to variants and generalizations of the Thomspon grid model. For example, graphs with bounded valence greater than four may be laid out by mapping each node to a region of the grid, instead of a single grid point. The results are also applicable to networks with large processors. Techniques for dealing with large processors are described more fully in Chapter 5.

2.2. Elementary Bounds on Layout Area

Although there are a variety of important engineering considerations in choosing one layout for a graph over other possible layouts, the best understood, and perhaps the most desirable cost measure to minimize is *layout area.* The area of a layout is most naturally defined as the area of the "bounding-box" around the layout, and equals the product of the number of vertical tracks and the number of horizontal tracks that contain a node or wire segment of the graph. For example, the layout of Figure 2.1 has area 15. This is not the minimum possible; there is another layout with area 9.

How much area does an N-node graph require? Clearly, the area cannot be less than the number N of nodes. On the other hand, by embedding nodes at equally spaced intervals along a line, and using a distinct horizontal track for each edge (as shown in Figure 2.2), it is clear that the area required for an N-node graph is no greater than $O(N^2)$. These bounds are independent of the structure of the graph and hold for all N-node graphs. In general, however, the minimum area needed to lay out a graph depends on the graph.

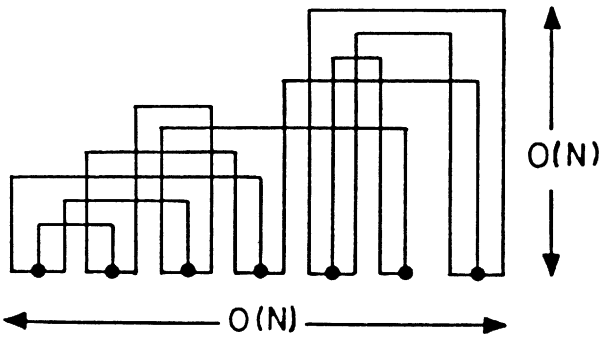


Figure 2.2: *Every N-node graph can be laid out in $O(N^2)$ area.*

Thompson [79, 80] identified *bisection width* as an important property of graphs that affects minimum layout area. The bisection width of an N-node graph is the minimum number of edges which must be removed from the graph in order to disconnect it into two subgraphs each of size at least $\lfloor N/2 \rfloor$. Thompson showed that, up to a constant factor, the layout area can be no less than the square of the bisection width. Therefore, if the bisection width for a graph is known, a lower bound on area can be easily computed. By showing that certain computationally powerful graphs such as the shuffle-exchange graph have large bisection width, Thompson showed that these graphs require large area. In fact, Thompson extended this observation to obtain area-time tradeoffs for computing certain functions.

Leighton [40, 41] identified *crossing number* as another general property that affects layout area. The crossing number of a graph is defined as the minimum number of edge crossings in any drawing of the graph in the plane. It is easy to see that the crossing number of a graph is a lower bound on layout area. Using more sophisticated arguments for special graphs, Leighton also directly obtained lower bounds on *total wire length* (the sum of the lengths of the wires in a layout), which of course is a lower bound on layout area. These techniques are heavily dependent on the recursive structure of the special graphs and are generalized in [7].

2.3. Layouts Based on Separator Theorems

Leiserson [49, 50] and Valiant [83] investigated general properties that provide effective upper bounds on layout area. They independently developed a divide-and-conquer strategy for graph layout and showed, for example, that every N-node tree can be laid out in $O(N)$ area

and that every N-node planar graph can be laid out in $O(N \lg^2 N)$ area. Their technique is based on the notion of separator theorems for graphs.

Definition: A class of graphs which is closed under the subgraph relation is said to have an *$f(x)$-separator theorem* if there exist constants a and b where $0 < a \leq 1/2$ and $b > 0$ such that every N-node graph in the class can be partitioned (by the removal of at most $bf(N)$ edges of the graph) into disjoint subgraphs having $a'N$ and $(1 - a')N$ nodes where $a \leq a' \leq 1 - a$.

Given a class of graphs for which a separator theorem is known (e.g., trees have a 1-separator theorem [52] and planar graphs have a $\sqrt{x}$-separator theorem [53]), it is possible to construct a layout for any N-node graph in the class by using a simple divide-and-conquer approach. For example, Leiserson [49, 50] proved the following upper bounds on layout area.

x^α-separator theorem	*Layout Area*
$\alpha < 1/2$	$O(N)$
$\alpha = 1/2$	$O(N \lg^2 N)$
$\alpha > 1/2$	$O(N^{2\alpha})$

Remark. The layout procedure assumes that a complete recursive decomposition of the graph is given. If a complete decomposition is not given, then there is no known polynomial time algorithm which achieves the upper bounds on area. This severely limits the applicability of separator-based layout strategies to classes of graphs (such as trees or planar graphs) for which decompositions are easily computed.

How good are the preceding area bounds? Thompson [79, 80] and Leighton [40, 41] showed that none of the bounds can be improved. More precisely, they showed that within each class there is a graph for which the bound is optimal. But this does not mean that the bounds are optimal for every graph within a class. In fact, while the bounds are *existentially* optimal, they are not *universally* optimal. For example, an N-node square grid can be laid out in area linear in N, but since the minimum separator theorem for the class of square grids is $\sqrt{x}$, the

best bound obtainable by separator-based layouts is $O(N \lg^2 N)$, which is off by a factor of $O(\lg^2 N)$ from the optimal. Of course, since N-node graphs require area at least N, the bounds for graphs with x^α-separator theorems, $\alpha < 1/2$, are asymptotically universally optimal.

For graphs with larger separator theorems, the discrepancy between the minimum layout area and that given in the table can be much worse. Consider, for example, the N-node graph S_N which consists of $N/\lg N$ disjoint $\lg N$-node expander graphs. An m-node *expander graph* has the property that every subset of k nodes is linked by $\Theta(\min(k, m-k))$ edges to the $m-k$ nodes outside the subset.* The bisection width of such a graph is $\Omega(m)$, and hence the minimum separator theorem is $\Theta(x)$. The existence of trivalent graphs that satisfy this defintion has been known for a long time [28, 31]. In fact, almost all trivalent graphs satisfy this definition. Since each $\lg N$-node expander graph can be trivially laid out in $O(\lg^2 N)$ area, the layout area of S_N is no greater than $O(N \lg N)$. However, Leighton [12] showed that the minimum separator theorem for the class of graphs S_N exceeds $\Omega(x/\lg^2 x)$, so that the area bound from the table above is $O(N^2/\lg^4 N)$, which is much worse than the optimal bound of $O(N \lg N)$.

Remark. Any class of graphs closed under the subgraph relation and containing S_N must also contain expander graphs. Hence, the minimum separator theorem (as defined earlier) for the class is $\Theta(x)$. Instead of defining separator theorems for classes of graphs closed under the subgraph relation, it is more convenient (and general) to define separators for individual graphs in terms of the subgraphs produced by its recursive decomposition. Using the less restrictive (but more useful) definition, it is possible to show that S_N has an $O(N/\lg N)$-separator. The $\lg N$-node expander graphs are split in the upper levels of the decomposition and never appear intact as subgraphs in the lower levels of the decomposition. Leighton [12] proved that even using the most liberal definition, the minimum separator for S_N is at least $\Omega(N/\lg^2 N)$. Any bound on layout area for S_N based on the minimum separator can therefore be no less than $\Omega(N^2/\lg^4 N)$.

Thus, while the divide-and-conquer strategy based on separator theorems gives existentially

*The original definition of expander graphs is slightly different from that given here. We adopt this minor variant because it allows nodes of degree no greater than three.

optimal bounds, the bounds can be unacceptably poor in a universal sense. It was the discovery of such large discrepancies that led to the search for an alternative framework for VLSI layout. Within the new framework presented in Chapter 4 we shall see how these large discrepancies are overcome.

2.4. Eight VLSI Graph Layout Problems

As mentioned earlier, there are many important considerations in choosing one layout over a multitude of other possible layouts. The problems in this section are motivated by some engineering concerns fundamental to circuit design and layout. Though not exhaustive, this list covers most of the theoretical issues studied recently. Many of the problems are known to be NP-Complete. The emphasis throughout this thesis is the development of a general unifying framework for dealing with diverse issues in a uniform manner. Within the framework, solutions to some problems are reasonably close to optimal. For other problems, good heuristics are developed or suggested, and general bounds obtained.

Problem 1. *Given a graph G, produce an area-efficient layout for G.*

As mentioned before, minimizing area is a critical concern in VLSI circuit layout. In addition to the work on area-efficient layouts described in the previous section, Dolev, Leighton, and Trickey [22] have shown that determining the minimum layout area of a forest of trees is NP-Complete.

Problem 2. *Given a graph G, produce an area-efficient layout for G with minimax edge length.*

Besides area, speed is another critical factor in chip performance. Signals do not propagate instantaneously across wires, and the longer the wire, the longer the propagation delay. In pipelined or systolic systems, the effect of propagation delays is even more dramatic. The maximum delay determines the clockperiod, and hence the throughput, of the system. To maximize throughput we need to minimize the maximum delay. In short, we must produce layouts so that the longest edge is as short as possible. The minimum, over all layouts, of the

length of the longest edge is called the *minimax* edge length.

Paterson, Ruzzo and Snyder [59] studied the problem of minimizing edge lengths for complete binary trees. They showed that the minimax edge length of an N-node complete binary tree is $\Theta(\sqrt{N}/\lg N)$. Adopting a different strategy based on separator theorems, the next chapter presents a general technique for bounding the maximum edge length of arbitrary trees, while Chapters 4 and 5 extend the techniques to general graphs. The next chapter also shows that minimizing the edge lengths of trees is NP-complete.

Problem 3. *Given a graph, produce an area-efficient layout in which each wire has bounded delay in the capacitive model.*

Although it is certainly true that propagation delay across a wire depends on the length of the wire, there has been little consensus on how fast propagation delay grows as a function of wire length. Thompson [79, 80] assumes propagation delay to be constant, independent of wire length. This might seem unreasonable given the ultimate speed-of-light limitation which indicates that the delay increases linearly with length. The speed-of-light limitation, however, greatly exaggerates the importance of wire delay in determining the speed of circuits. Mead and Conway [55] take into account some of the electrical characteristics of interconnections on MOS integrated circuits, and emphasize the role of wire *capacitance* in determining propagation delay. Recent analysis by Bilardi, Pracchi, and Preparata [10] strongly supports the belief that capacitive effects play the predominant role in determining the speed of MOS circuits.

In a capacitive model, each wire is assumed to present a purely *capacitive load* to the transistor that drives a signal across the wire. This load is proportional to the length of the wire plus the area of the transistor that receives the signal. The delay is proportional to this load divided by the area of the driving transistor. By increasing the size of the driving transistor it is therefore possible to bound the propagation delay, independent of the length of the wire. A second well-known technique for reducing delay across a long wire is to "ramp" the wire with a geometrically increasing series of inverters [55]. The number of intermediate drivers, and hence the delay, is logarithmic in the length of the wire, but an attractive feature is that this process can be carried out without the need to resize the original transistors in the

circuit.

Of course, increasing the size of one transistor or introducing new transistors might force some wires to be stretched to avoid the enlarged transistor area. In other words, decreasing the delay across one wire might force an increase in delay over other wires. Leiserson [47] and Mehlhorn [56] independently posed the question of whether or not the transistors in a layout could be resized so that every wire in the layout has constant propagation delay. Ramachandran [65] investigated the problem of introducing intermediate drivers along long wires to decrease delays, but under the constraint that the topology of the layout remain unchanged. With the restriction that wires can not be rerouted, she showed that logarithmic delay can be achieved, but at the expense of squaring the layout area in the worst case. We allow the layout topology to be changed, and obtain significantly better results.

Problem 4. *Given a graph G, produce a layout for G with few wire crossings.*

An undesirable feature of layouts is the presence of a large number of wire crossings. When two wires cross, they must be on different layers. For faster operation, and less power dissipation, it is advantageous to maximize the total amount of wiring on a layer of low resistance, e.g. the metal layer, while minimizing the wiring on a layer of high resistance, e.g. the polysilicon layer. The net wiring on one layer may be reduced by laying wires on that layer only just before and after two wires cross. If the number of wire crossings is small, the number of contact-cuts which connect wire segments on different layers is small so that the area of the layout is not blown up by the contact cuts which occupy large area. In addition, long wires that are crossed by many other wires are susceptible to cross-talk when all the crossing wires simultaneously carry the same signal.

The *crossing number* of a graph is defined to be the minimum number of wire crossings in any drawing of the graph on the plane. Leighton [40, 41] proved upper and lower bounds on crossing numbers and then used the results to find bounds on layout area. Garey and Johnson [29] showed that determining the crossing number of bipartite graphs is NP-Complete.

Problem 5. *Given a graph, produce an area-efficient regular layout for the graph.*

Some design methodologies, most notably gate-arrays, require that processors be located at fixed positions on a chip. In gate-arrays the processors are placed in a grid pattern with uniform spacing between processors adjacent along every row and column. Such layouts are said to be *regular*. An important advantage of this design restriction is its flexibility: even if the size of every processor is increased, the wiring between processors remains unaffected and the total area remains proportional to the *sum* of the wire area (as computed with unit-size processors) and the processor area. This is because only the $\sqrt{N}$ rows and columns containing the N unit-size processors need to be expanded to accomodate the non-unit-size processors. In non-regular layouts, *every* row and column might have to be expanded since there might be a node in every row and in every column. Increasing the linear dimension of the processors by a factor of s could result in an $\Theta(s^2)$ increase in layout area.

Previous divide-and-conquer layout strategies do not produce regular layouts. Hence, they are not useful in laying out circuits with non-unit-size processors. A good strategy for producing regular layouts would solve the nagging problem of how to cope with variable-size processors.

Problem 6. *Design area-efficient chips that can be configured to realize a large number of graphs.*

Because it is expensive to make one chip but cheap to make many copies, manufacturers of custom chips have been encouraged to make *configurable* designs such as gate-arrays, ROM's and PLA's. In such designs, the entire chip is prefabricated except for one layer. The customer then specifies a configuration for the chip, and the final layer of metalization connects up the circuitry in that particular way. Hence, most of the design and fabrication costs can be factored over many custom chips. Similarly, the fast emerging laser-restructuring technology [64] provides another economical way to customize chips after fabrication is complete. Laser restructuring allows connections between wires to be made or broken after the chip has been fabricated. In either case, it is desirable to design layouts that can be configured from one of a few basic patterns.

Problem 7. *On a wafer which has arbitrarily distributed defective cells, realize a given graph on the good cells.*

In any fabrication process, it is expected that some of the processing cells will be defective. In a two-dimensional array of cells on a wafer in which defective cells are arbitrarily distributed, it may still be possible to use the wafer by configuring wires around the defective cells. This may, for example, be performed by laser restructuring techniques [64]. Given this ability to isolate defective cells, it is important to consider how a graph may be realized on the remaining good cells. This problem has received considerable attention recently [33, 45, 69]. The problem is similar to the general graph layout problem in the Thompson model but with the important restriction that nodes of the circuit can only be mapped to a restricted set of nodes in the grid.

Problem 8. *Given a graph G, assemble G using the minimum number of copies of a single chip having few external pin connections.*

A number of very large networks have been proposed in recent years for implementing priority queues [48], for searching [5], for direct execution of applicative programming languages [54], and for recognizing regular expresions [26]. Some of these networks are too large to fit on a single chip. For example, the tree-structured network of [54] is envisioned to contain as many as one million processing elements. Clearly, such networks must be partitioned over many interconnected chips, so that each chip realizes a small portion of the network.

The technology for packaging chips severely limits the number of external pin connections on a chip. While chips with over a million components are forseeable in the near future, no one predicts a chip with over two hundred external pin connections. This poses a pressing problem in assembling large networks of processors.

Even if a network could be partitioned so that each portion has only a few external connections, it would be economically infeasible to design each chip individually. For instance, it would be prohibitively expensive to design one thousand different chips, each containing a thousand processing elements, to assemble a network of one million processors. For this reason, it is necessary to assemble large systems using copies of a few configurable or restructurable chips. The next chapter presents one solution to the problem of assembling large tree structures

using copies of a single, area-efficient, restructurable chip with few external pin connections.

Within the new framework, efficient solutions are provided for each of these problems. In fact, a single layout simultaneously solves many of these problems efficiently. The framework provides a two-step strategy for solving these problems. First, the graph to be laid out is embedded within a very special network called the *tree of meshes*. For the tree of meshes it is possible to solve all these problems efficiently. In the second step, therefore, a good layout for the tree of meshes also solves these problems for the embedded graph.

CHAPTER 3

Layouts for Trees

A binary tree may not be the best multiprocessor organization, but it has been proposed by many researchers for a variety of reasons. For example, a complete binary tree can be the major component of a priority queue resource [48] and of a smart-memory raster graphics system [27]. A complete binary tree can also serve as a hardware structure for searching [5], for databases [75], or for direct execution of applicative programming languages [54]. Browning [15] proposes a complete binary tree for general-purpose multiprocessing, and two systems based on her ideas are being built at Caltech and Bell Laboratories.

Attention is also directed to binary trees which are not complete. Floyd and Ullman [15] show that strings described by a regular expression can be recognized by processing elements organized as the parse tree of the regular expression. Foster and Kung [25] have a similar scheme based on the simple configurable layout developed by Leiserson [50]. There are other proposals, for example [58, 74], of machine organizations that, while not trees, are nevertheless tree-like.

We shall not debate the merits of the various tree machines here, but shall confine ourselves to understanding their physical organization. In this regard trees are particularly attractive because of their simple interconnection structure. Not only can trees be laid out efficiently, but good layouts for trees also suggest efficient ways to lay out general graphs. Moreover, problems that are intractable for trees are also intractable in general. Thus, by investigating layouts for trees we stand to learn more about general graph layout.

In the following section we examine two well-known layouts for complete binary trees and present a better layout which minimizes (asymptotically) both area as well as maximum edge

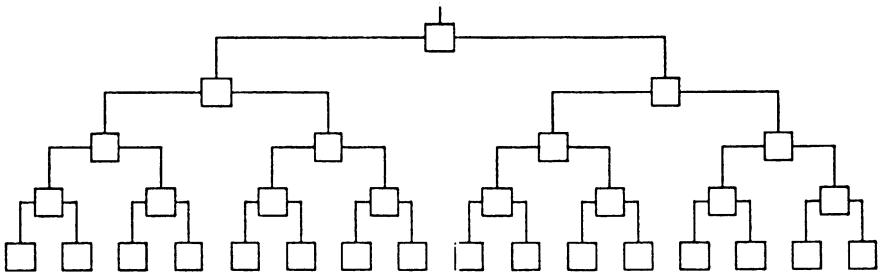


Figure 3.1: *An $O(n \lg n)$ area layout of a complete binary tree.*

length. These bounds are extended to arbitrarily structured trees in Section 3.2, and to planar layouts for trees in Section 3.3. Computing the minimum edge length exactly is shown to be NP-complete in Section 3.4. Section 3.5 describes Leiserson's [50] assembly of large complete trees using multiple copies of a single chip with only four external pin connections. Section 3.6 introduces and examines the *two-color bisection* problem for arbitrary trees. Section 3.7 presents one way to assemble large arbitrarily structured trees using the minimum number of copies of a single restructurable chip with few pins.

3.1. Layouts for Complete Binary Trees

In addition to their usefulness in speeding up computation time by allowing both parallelism and pipelining, complete binary trees are attractive also because they can be laid out efficiently. Figure 3.1 shows the naive layout of a complete binary tree. Since the height of an N-leaf tree is $\lg N$, and the N leaves are spread out over a line of length $2N$, it follows that the area of the layout is $2N \lg N$. Furthermore, the longest edges are at the top level and their length is $\frac{1}{2}N$.

The familiar H-tree layout in Figure 3.2 was originally proposed by Mead and Rem [55]. In contrast to the naive layout which, in a sense is one-dimensional, this layout exploits both dimensions symmetrically. If $S(N)$ is the side of the layout, then we have that $S(1) = 1$ and more generally,

$$S(N) = 2S(N/4) + 1,$$

which yields $S(N) = 2\sqrt{N} - 1$. Consequently, the area of the layout is no greater than $4N$. The longest edges are again at the top level, and their length is no more than $\frac{1}{2}\sqrt{N}$.

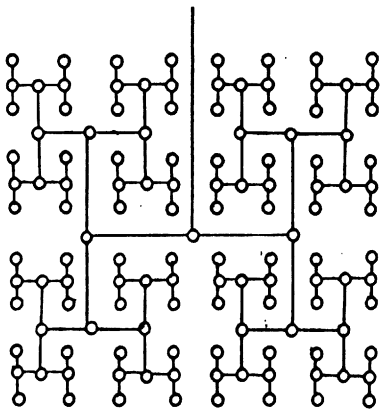


Figure 3.2: *The H-tree layout of a complete binary tree.*

The H-tree layout asymptotically minimizes area but not maximum edge length. Paterson, Ruzzo, and Snyder [59] demonstrated a linear-area layout with maximum edge length $O(\sqrt{N}/\lg N)$. In any layout there are two nodes which are distance $\sqrt{N}$ apart; moreover, these two nodes are connected by a path containing no more than $2\lg N$ tree edges. It follows then that at least one of these edges must have length at least $\sqrt{N}/2\lg N$. Thus, the layout of [59] asymptotically minimizes area as well as maximum edge length. Unfortunately, however, the layout technique of [59] does not extend to more general graphs. The remainder of this section demonstrates another layout with asymptotically optimal area and maximum edge length. The following section generalizes our technique to arbitrary trees and, the next chapter to general graphs.

To illustrate our technique, consider the layout of Figure 3.3 in which the nodes at the second and third levels of the tree have been brought closer to the root so that all edges within the top four levels are of equal length. This "averaging" of edge lengths reduces the maximum edge length from $\frac{1}{2}\sqrt{N}$ to $\frac{5}{12}\sqrt{N}$. Of course, the layout is stretched in the middle in order to accomodate two edges instead of one. This increases the area of the layout, but only slightly, from $4N$ to $4N + 6\sqrt{N}$.

This averaging operation can be carried out further down the tree so that many levels are brought closer towards the root. In order to space top levels of the tree closely together, we embed these levels within an *H-channel* structure shown in Figure 3.4. This structure is

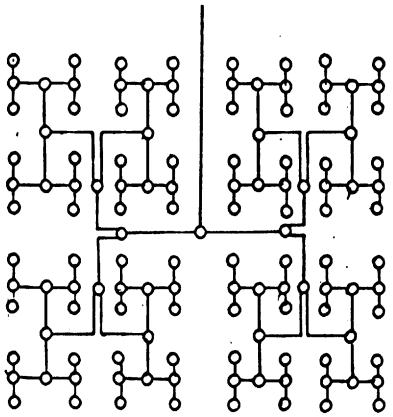


Figure 3.3: *The H-tree layout with shorter edges at the top levels.*

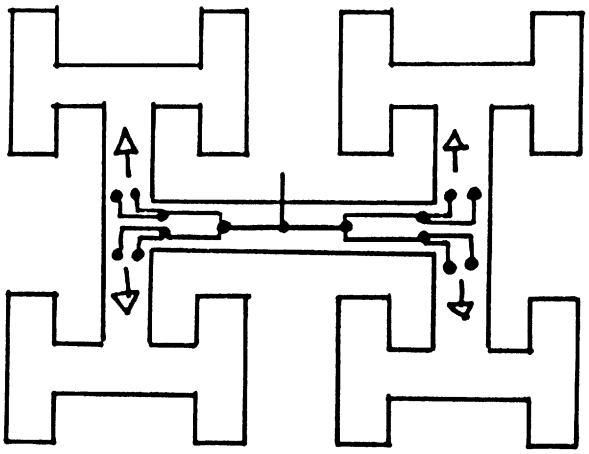


Figure 3.4: *The H-channel structure.*

obtained by taking the H-tree layout of a complete binary tree and blowing up the layout in both dimensions by a suitable factor. The details of the embedding are described next.

Theorem 3.1. *An N-node complete binary tree can be embedded in linear area with maximum edge length $O(\sqrt{N}/\lg N)$.*

Proof. To layout a complete binary tree with N leaves, start with the H-tree layout of a complete binary tree with $\lg^2 N$ leaves which has area $4\lg^2 N$ and maximum edge length $\frac{1}{2}\lg N$. Blow up this layout in either dimension by a factor $\alpha\sqrt{N}/\lg N$, where α is a constant specified later. The area of the layout becomes $4\alpha^2 N$ and the longest channel has length $\frac{1}{2}\alpha\sqrt{N}$.

Next, lay the root at the centre of the H-channel structure and place the second level nodes at distance $\beta\sqrt{N}/\lg N$ from the root on either side. Once again, β is a constant specified later. Place lower levels of the tree as shown in Figure 3.4, with successive levels spaced equally apart. At every corner of the H-channel structure, bisect the tree so that the subtrees embedded within the two substructures are of equal size. Finally, in the lowest level channels lay out the remaining subtrees in the H-tree manner.

We must ensure that every channel is wide enough to accomodate all the nodes in any level embedded within it, and also that the H-tree layouts in the final step fit within the lowest level channels. To satisfy these conditions, let us first calculate the total number of tree levels embedded in all but the lowest level channels. The total length of all channels encountered from the centre of the layout to the end of a terminal channel does not exceed the quantity $2\alpha\sqrt{N}$. Since the distance between successive tree levels is $\beta\sqrt{N}/\lg N$, the number of tree levels embedded is bounded by $(2\alpha/\beta)\lg N$. The total number of tree nodes within any one of these levels is therefore no greater than $N^{2\alpha/\beta}$. If $2\alpha/\beta < 1/2$ then the number of nodes in any level is asymptotically less than the width of a channel which equals $\beta\sqrt{N}/\lg N$. The first condition is therefore satisfied by having $\alpha < \beta/4$.

To ensure that the H-tree layouts at the final step fit within the final channel, it suffices to check that the dimensions of the layout are smaller than the dimensions of the channel. The size of a subtree embedded within a final-level channel cannot be more than $N/\lg^2 N$ because the tree is split into half at each corner. The side of the H-tree layout is no greater than $2\sqrt{N}/\lg N$. By choosing $\alpha > 2$, the side of the channel is guaranteed to be larger than a side of the H-tree layout. Therefore, by choosing $\alpha > 2$ and $\beta > 4\alpha$, we see that the layout can be completed. Finally, the area is linear in N and the maximum edge length is bounded by $O(\sqrt{N}/\lg N)$. ∎

3.2. Layouts for Arbitrary Binary Trees

One property of complete binary trees crucial to the layout of Theorem 3.1 is that a complete binary tree can be bisected into two equal size subtrees simply by removing the root.

At every corner in the H-channel structure, a forest of complete trees is bisected into two equal halves, each "growing" in opposite directions. This controls the size of every subgraph at the final level so that a standard layout fits within a final-level channel.

Arbitrarily structured binary trees are only slightly harder to bisect. Any N-node binary tree can be separated into two components, each with no more than $\lfloor \frac{2}{3}N \rfloor + 1$ nodes, by removing a single edge [52]. (The worst-case occurs for the four-node tree in which one node is adjacent to three others.) Either of the two components might be a forest, but the same result applies to forests, so that the binary tree can be split recursively. By recursively splitting the larger component, a tree can be bisected by cutting at most $O(\lg N)$ edges, or by removing the nodes incident to these edges. The $O(\lg N)$ bound follows because the subgraphs decrease geometrically in size with each cut.

The property that all trees have small bisections was used by Leiserson [49, 50] and Valiant [83] to show that all trees have linear-area layouts. We strengthen this result to show that the maximum edge length of any N-node tree is bounded by $O(\sqrt{N}/\lg N)$. The details of the layout are described in the following Theorem.

Theorem 3.3. *Every N-node tree can be embedded in linear area with maximum edge length $O(\sqrt{N}/\lg N)$.*

Proof. As before, begin with the H-tree layout of a complete binary tree with $\lg^2 N$ leaves, and blow up the layout in either dimension by a factor $\alpha\sqrt{N}/\lg N$, where α is a constant specified later. The area of the layout becomes $4\alpha^2 N$ and the longest channel has length $\frac{1}{2}\alpha\sqrt{N}$.

Find a set of $O(\lg N)$ nodes which bisect the tree and locate them at the center of the layout. Place nodes of the tree in breadth-first levels starting with the bisector set as the roots of the search, so that consecutive levels are distance $\beta\sqrt{N}/\lg N$ apart (β is a constant specified later). At every corner of the H-channel structure, bisect the remaining forest of subtrees so that the subforests embedded within the two substructures are of equal size. Add the new bisector set to the set of nodes from the previous breadth-first level, as shown in Figure 3.5. In the new channel, start with the updated set as the root of a breadth-first search and repeat

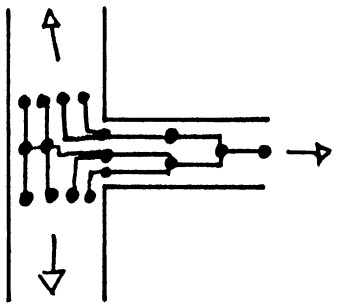


Figure 3.5: *Inserting new bisector sets at every corner.*

the procedure used before. Finally, in the lowest level channels lay out the remaining subtrees using the standard divide-and-conquer layout of Leiserson [49, 50] or Valiant [83].

As before, we need to ensure that every channel is wide enough to accomodate all the nodes embedded within any level, and also that the layouts in the final step fit within the lowest level channels.

Let us first calculate a crude upper bound on the total number of nodes embedded in any one breadth-first level. This quantity is certainly less than the total number of nodes embedded in all but the final-level channels. To bound the latter quantity, suppose that nodes in each bisector set within the H-channel structure are pulled in to the center of the layout, and the remaining nodes placed in breadth-first levels until the final-level channels. Bringing all the bisector sets towards the center can only increase the number of nodes in all but the final-level channels. Since an N-node tree has a bisector of size $O(\lg N)$, the total number of nodes within the union of all bisector sets is bounded by:

$$O\left(\sum_{i=0}^{2\lg\lg N} 2^i \lg \frac{N}{2^i}\right) = O(\lg^3 N).$$

The total length of all channels encountered from the centre of the layout to the end of a final-level channel does not exceed $2\alpha\sqrt{N}$. Since the distance between successive tree levels is $\beta\sqrt{N}/\lg N$, the number of tree levels embedded within the H-channel is bounded by $(2\alpha/\beta)\lg N$. Starting with $O(\lg^3 N)$ nodes as the roots of a breadth-first search, the number of nodes encountered in $(2\alpha/\beta)\lg N$ levels cannot exceed $O(N^{2\alpha/\beta}\lg^3 N)$. Since every node embedded within the H-channel must be in one such breadth-first level, the previous quantity

also bounds the total number of nodes within the H-channel structure. By choosing $2\alpha/\beta < 1/2$, or $\alpha < \beta/4$, we see that the width of a channel asymptotically exceeds the number of nodes in any level within the channel. Therefore, the first condition is satisfied by having $\alpha < \beta/4$.

To ensure that the layouts at the final step fit within a final-level channel, it suffices to check that the dimensions of a layout generated by the Leiserson–Valiant strategy are smaller than the dimensions of the channel. Their layout of an x-node tree is linear in x, i.e., bounded by γx, for all x and some constant γ. In the layout described above, the size of a forest embedded within a final-level channel cannot be more than $N/\lg^2 N$ because the tree is split into half at each corner. The side of a layout at the final level is no greater than $\sqrt{\gamma N}/\lg N$. By choosing $\alpha > \sqrt{\gamma}$, the side of the channel is guaranteed to be larger than a side of the H-tree layout. Therefore, by choosing $\alpha > \sqrt{\gamma}$ and $\beta > 4\alpha$, we see that the layout can be completed. Finally, the area is linear in N and the maximum edge length is bounded by $O(\sqrt{N}/\lg N)$. ∎

3.3. Planar Layouts for Trees

It is sometimes necessary to produce layouts in which distinct edges do not cross one another. Planar layouts have the advantage that only one layer of interconnect is required; by using a low-resistance metal layer, the resulting circuit is not only faster, but also dissipates less power. Many current automatic layout systems reserve a single layer of interconnect for special purposes such as, for example, power and ground connections. In such cases, it is necessary to find good planar layouts. Needless to say, the underlying connection scheme must be planar.

Planar layouts may require much more area than non-planar layouts. In particular, Valiant [83] demonstrated an N-node planar graph for which every planar layout occupies at least $\Omega(N^2)$ area and has edges of length $\Omega(N)$. On the other hand, Leiserson [49, 50] and Valiant [83] showed that every N-node planar graph can be laid out in $O(N\lg^2 N)$ area with edges of length $O(\sqrt{N}\lg N)$ in Thompson's layout model, which allows distinct wires to cross.

Valiant [83] further showed that every tree has a linear-area planar layout. In other words, the planarity restriction does not affect the asymptotic area requirements of trees. But what

about edge length? Intuitively, the length of a wire can be reduced by taking a short-cut across another wire, instead of going around it. So, an important question is whether the planarity requirement affects the maximum edge length for trees.

Although the layout of Section 3.2 has linear area, and asymptotically optimal edge length in the worst-case, it is not guaranteed to be planar. However, Ruzzo and Snyder [70] showed that this layout could be transformed into a planar layout without increasing edge length asymptotically. The details of their transformation are fairly complicated; in the following Theorem, we present a simpler transformation.

Theorem 3.4. *Every N-node tree has a linear-area planar layout with maximum edge length $O(\sqrt{N}/\lg N)$.*

Proof. The layout proceeds exactly as in the proof of Theorem 3.3, with particular attention paid to the way a bisector set is chosen and to the ordering of nodes within the set. In particular, if a forest of x nodes has to be separated from an N-node tree, $x \leq \lfloor N/2 \rfloor$, then it suffices to remove at most $\lceil \lg x \rceil$ nodes. The key fact is that these nodes can be chosen from a single path in the tree. This path induces a natural linear ordering on the set of nodes removed.

To see this, consider a binary tree rooted at a node of degree either one or two. It is always possible to choose such a root, and if the remainder of the tree is drawn in levels then every internal node has at most two sons. Label each node in the tree by the size of the subtree rooted at that node and below it. Pick any node whose label is no less than x, and both of whose sons have labels less than x. Mark this node. If its label equals x then we have found a node whose removal separates a subtree of the required size. Otherwise, one of its sons must have a label $y \geq \lfloor x/2 \rfloor$, while the other son has label no less than $x - y - 1$. Recursively mark nodes in the subtree rooted at the second son so that the removal of the marked nodes separates a forest of size $x-y-1$. It is easily seen that the marked nodes lie along a path of the original tree. Moreover, the removal of all marked nodes separates a component of size exactly x. Finally, since the first node separates a component of size at least $\lfloor x/2 \rfloor + 1$, it follows that no more than $\lceil \lg x \rceil$ nodes are marked. Figure 3.6 illustrates this procedure.

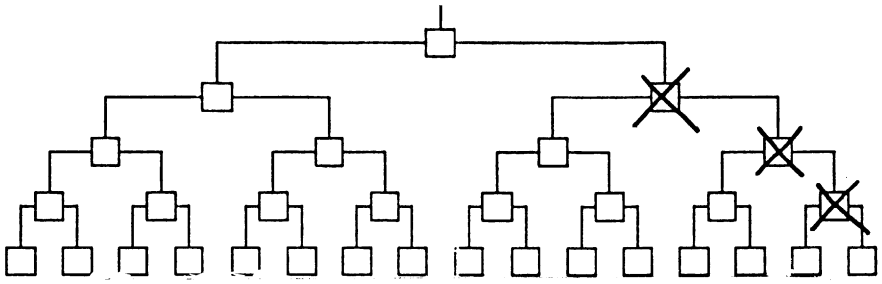


Figure 3.6: *Three cuts separate a subforest of 13 nodes.*

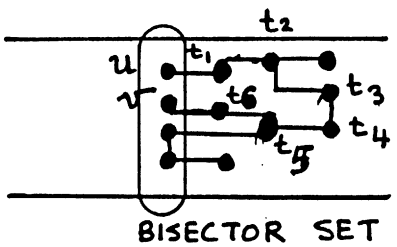


Figure 3.7: *The removed nodes are placed in the order of occurrence along the path.*

Given a tree, use the above procedure to find a set of nodes which bisect the tree, and lie along a path. Place these nodes at the center of the layout in the same order in which they are encountered along the common path. Next, find all nodes adjacent to the bisector set and place them on either side as before. However, the ordering of nodes in these breadth-first levels is chosen as follows: for each pair of nodes u, v that are placed next to each other in the bisector set, if the path connecting them is $u, t_1, t_2, \ldots, t_k, v$, then place nodes t_1 and t_k next to each each other in the second level, as shown in Figure 3.7. The orderings of nodes on either side of the center again satisfy the condition that nodes connected by a path in the forest embedded on that side appear in the order in which they are encountered along the common path.

By placing nodes in every level in the same order in which they lie along a common path within the forest still to be embedded, it is easy to guarantee that the layout is planar inside the channel (see Figure 3.7). All that remains is to guarantee that the layout can be made planar at every corner when new bisector sets are added to a level.

When the end of a channel is reached, the situation is as shown in Figure 3.8. Nodes $u_1, u_2, \ldots, u_n$ are those in the last level of the channel. The subgraph which remains to be embedded is a forest of subtrees. The n nodes can be grouped according to which subtree they

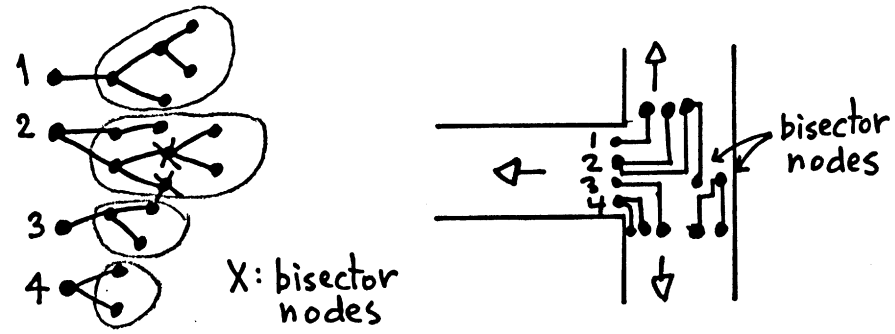


Figure 3.8: *To bisect a forest of trees, only one tree need be separated.*

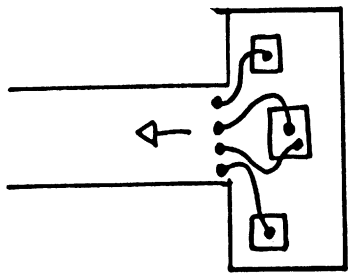


Figure 3.9: *Nodes in the final level may be connected to their subtrees without crossovers.*

belong to, nodes in the same subtree being adjacent within the ordering. To bisect this forest, it suffices to split only one of these subtrees: order the subtrees top-down and pick the lowest one so that the subforest above it contains at most one-half of all nodes in the forest. Split the subtree this node belongs to into two components as required so that the original forest is bisected. By laying out the next breadth-first level and the new bisector nodes as in Figure 3.8, we see that in each of the two lower-level channels the nodes within the same subtree are ordered in the order in which they are encountered along a common path.

Repeating this process further down the H-channel structure, we see that the layout is free of wire crossings. To complete the layout, within the final-level channels we use Valiant's [83] linear-area planar layouts for each remaining subtree. Edges from these subtrees to nodes in the last breadth-first level of the penultimate channel can be inserted without crossovers as shown in Figure 3.9. This completes the planar layout. ∎

3.4. The Complexity of Minimizing Edge Lengths

Thus far we have only showed that every tree can be laid out with maximum edge length bounded by $O(\sqrt{N}/\lg N)$. While this bound is asymptotically optimal for some trees such as the complete binary tree, it is way off for others. For example, a two-ended string with every node connected only to its immediate neighbors can be trivially laid out with every edge of length one, independent of the number of nodes.

This motivates the problem: it Given a tree, produce a layout with minimax edge length. In this section we show that determining the minimax edge length is computationally intractable. The results are quite discouraging – even the problem of deciding if a given tree can be laid out with all edges of unit length is NP-complete.

Theorem 3.5. *Given a tree T, deciding whether or not T has a layout with unit length edges is NP-complete.*

Proof. Observe that the problem is clearly in NP; it is easy to guess a layout and verify that no edge has length greater than one. It remains to show that the problem is NP-hard.

The known NP-complete problem used in the reduction is the NOT-ALL-EQUAL 3CNFSAT problem [29, 72] stated below.

NOT-ALL-EQUAL 3CNFSAT: Given a boolean formula ϕ in 3CNF (conjunctive normal form with three literals per clause), does there exist a truth assignment which satisfies ϕ such that each clause contains at least one false literal?

Given a formula ϕ in 3CNF, we construct a graph G with the property that G can be laid out with all edges of unit length *iff* ϕ is an instance of NOT-ALL-EQUAL 3CNFSAT, i.e., ϕ can be satisfied with at least one false literal per clause. The graph G is constructed from elementary components termed "lines" (Figure 3.10). The crucial property of a line is its rigidity, meaning that in any layout with unit-length edges, nodes $u_1, \ldots, u_n$ must be lined up either horizontally or vertically. Figure 3.10 shows how to connect two lines so that the resulting graph can be laid out in only two ways (ignoring rotations).

Let $x_1, \ldots, x_n$ be the variables, and $C_1, \ldots, C_m$ be the clauses of ϕ. The basic "skeleton"

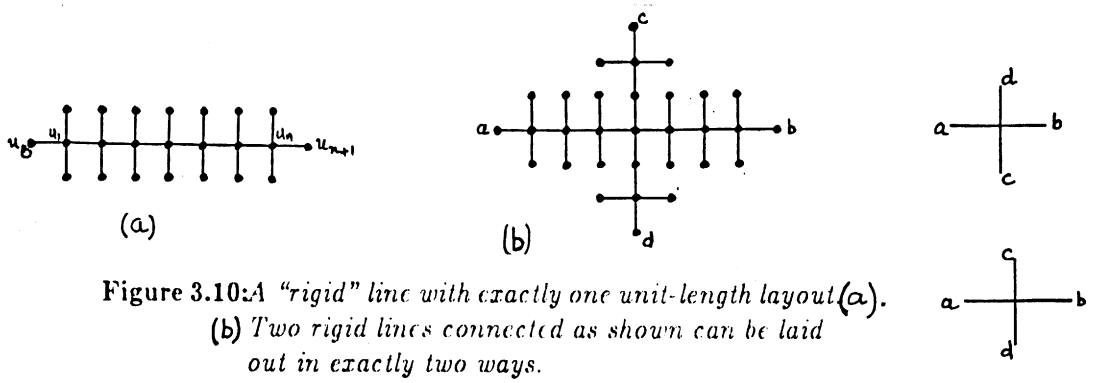


Figure 3.10: *A "rigid" line with exactly one unit-length layout (a). (b) Two rigid lines connected as shown can be laid out in exactly two ways.*

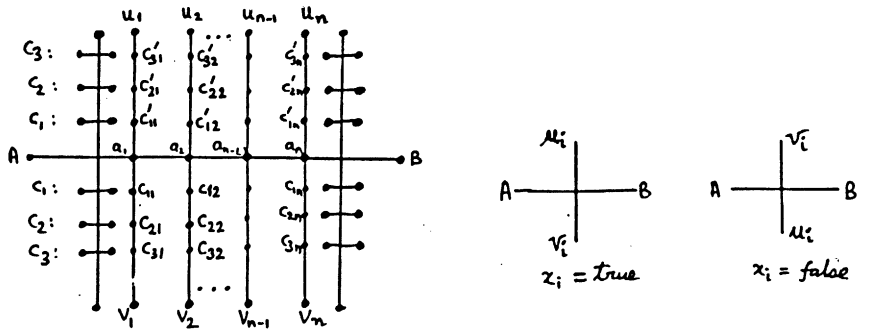


Figure 3.11: *The skeleton of the transformation. Each column represents a variable, while each clause is associated with two rows that are mirror images with respect to the x-axis.*

of G is shown in Figure 3.11. For each j, $1 \leq j \leq m$, the distances (number of intermediate nodes) $\alpha_i - C_{ji}, \alpha_i - C'_{ji}$ are all equal. The line u_i—v_i corresponds to variable x_i, and the two ways of embedding it with respect to the A–B axis correspond to assigning x_i *true* or *false*.

Thus far, there are 2^n possible ways of laying out G with unit length edges, each corresponding to a truth assignment to the variables of ϕ. Next, we encode within G the "structure" of ϕ as described below.

Let clause C_j be denoted $l_{j_1} \vee l_{j_2} \vee l_{j_3}$. If l_{j_i} is positive (x_i) add a "striker" at node C_{j,j_i}. Otherwise, if l_{j_i} is negative ($\overline{x}_i$) add a striker at node C'_{j,j_i}. Finally, for every $k \neq j_1, j_2, j_3$, add strikers *both* at $C_{j,k}$ and at $C'_{j,k}$. For example, if $C_1 = x_1 \vee \overline{x}_2 \vee x_3$, the strikers are added as shown in Figure 3.12.

Think of a node without a striker as a "hole". The rows C_j and C'_j together share three holes, and $2n - 3$ strikers. Because of the boundary constraints at the sides, no more than $n - 1$ of these strikers may lie on any side of the $A - B$ axis. In other words, for each clause

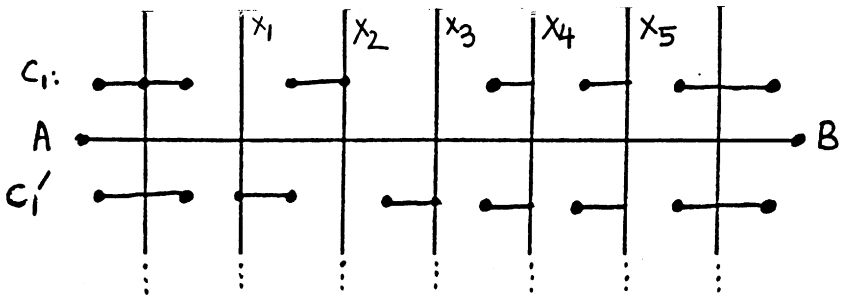


Figure 3.12:*In any unit-length layout each row contains at least one hole; this corresponds to an instance of NAE-3CNFSAT.*

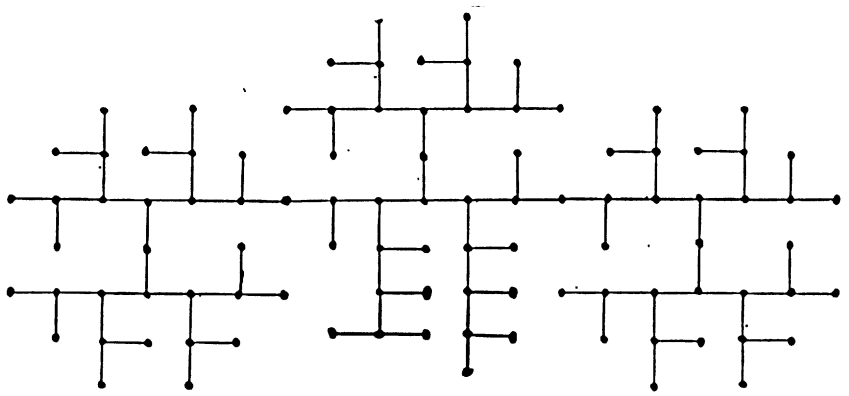


Figure 3.13:*A binary tree which has a unique (upto rotations) unit-length layout.*

there must be at least one hole on either side of the axis in a unit-length layout. For each clause, a hole "above" the axis implies a truth assignment which makes the clause true, while a hole "below" the axis implies at least one false literal within the clause. Therefore, there is a unit-length layout if and only if the formula is satisfiable with at least one false literal per clause. In short, G has a unit-length layout iff ϕ is an instance of NOT-ALL-EQUAL 3CNFSAT. Since the reduction is easily carried out in polynomial time, the theorem follows. ∎

In the above reduction, many nodes had degree four. We may strengthen the result to binary trees with maximum degree 3. A rigid line may be implemented by stringing together binary trees as shown in Figure 3.13. It is not hard to show that the structure is rigid; the key property is that the complete binary tree on 31 nodes has a unique (upto rotations) unit-length layout. This yields the following result.

Corollary 3.6. *Given a binary tree, deciding whether or not it has a layout with unit-length edges is NP-complete.*

3.5. Assembling Complete Trees

Whenever any system is larger than a single chip, it is necessary to partition it among separate chips which can be assembled at the printed circuit (or chip carrier) level. What is the most effective way to partition a large binary tree among several chips?

This question is pressing because although integrated circuit technology has been advancing at a rapid pace, the technology for packaging chips has been crawling in comparison. Packaging technology severely restricts the number of external connections to an integrated circuit. While the number of components per chip is expected to reach one hundred million, no one forsees chips with more than two or three external pin connections.

This section presents Leiserson's scheme [50] for assembling complete binary trees using one kind of chip with only four external pin connections. This chip has been used in tree-machine projects at Caltech and Bell Laboratories [16]. We review this scheme here for its simplicity and because the general scheme developed in Section 3.7 is based on similar ideas.

Figure 3.14 shows how arbitrarily large complete binary trees can be built out of a single chip that has only four off-chip connections. Each chip contains one internal node of the tree, and the remainder of the chip is packed as full as possible with an H-tree layout. The internal node requires three off-chip connections (denoted F, R, and L in the figure) for its father, right son, and left son. The H-tree requires only one off-chip connection (denoted T) to its father.

To interconnect two chips, the unconnected internal node of one of the two chips is selected as the father of the two H-trees. In Figure 3.14 the internal node on the left has been chosen for this purpose. The R pin on this chip is connected to its own T pin, and the L pin is connected to the T pin on the other chip. Considered as a unit, the combined two chips now have the same structure as a single chip — three connections to an internal node and one to the root of a complete binary tree. The pair of chips can be similarly combined with another pair to produce a quadruple of chips, which can in turn be combined, and so forth. Figure 3.15 shows

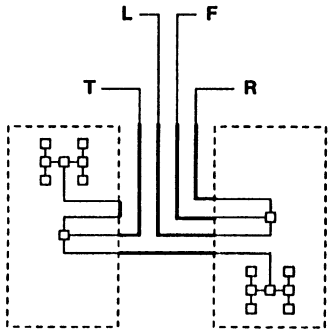


Figure 3.14:*Two chips connected to look like one*

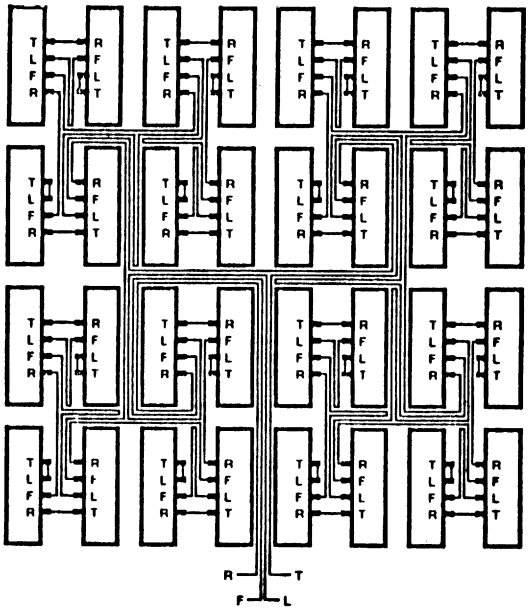


Figure 3.15:*A large complete binary tree assembled using many copies of the same chip.*

a large complete binary tree which has been wired up in this recursive fashion.

Unlike the assembly for complete trees, *configurable* or *restructurable* designs are required for assembling arbitrary binary trees. The reason is simple: a single fixed chip with N processors can realize only one N-node binary tree. In order to realize every N-node binary tree, either a new mask must be designed for each tree, or else connections on the chip must be restructured (for example, by laser) after fabrication. Given the ability to restructure wires on a chip, we ask: *Is there an area-efficient restructurable chip with N processors and m pins ($m << N$) which can be used to assemble every binary tree, independent of its size?*

This question is affirmatively answered in Section 3.7. The solution depends heavily on the results of the next section which considers the problem of partitioning a binary tree into subforests of size N so that every subforest has at most $O(\lg N)$ edges connected to nodes in other subforests. The solution to this problem leads directly to the restructurable chip design of Section 3.7.

3.6. Collinear Layouts and Two-color Bisectors

This section introduces the notion of *two-color bisectors* for trees. Two-color bisectors are a natural extension of graph bisectors, and will be critically used in partitioning graphs for layout. In this section we show how to use two-color bisectors to partition an arbitrary tree into subforests of size N so that every subforest has at most $O(\lg N)$ edges connected to nodes in other subforests. Bounds on the size of two-color bisectors are obtained from collinear layouts developed by Bentley and Leiserson [50].

Definition. Suppose that an N-node graph G has b black nodes and w white nodes. A two-color bisector for G is a set of edges whose removal bisects G into two subgraphs each of size at least $\lfloor N/2 \rfloor$, and such that each contains at least $\lfloor b/2 \rfloor$ black and $\lfloor w/2 \rfloor$ white nodes.

Theorem 3.7. *Every N-node forest of binary trees has a two-color bisector of size no greater than $2 \lg N$.*

Proof. Following Bentley and Leiserson [50], construct a collinear layout for the forest as follows. By removing one edge, separate the forest into two subforests so that neither contains no more than $\lfloor \frac{2}{3} N \rfloor + 1$ nodes [52]. If either component contains more than $\lfloor N/2 \rfloor$ nodes, separate it into two smaller components using the one-separator theorem again. Next, recursively construct collinear layouts for each subforest, and place these layouts side-by-side along the baseline. Finally, as shown in Figure 3.16, connect the two (or three) subforests by

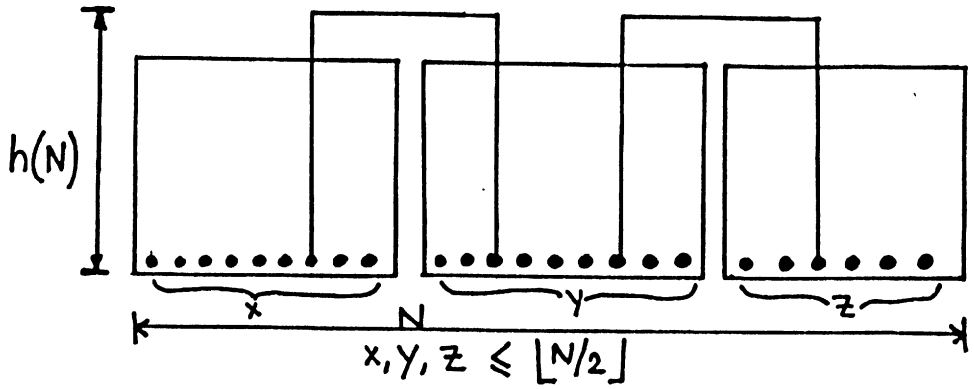


Figure 3.16: *The recursive construction of a collinear layout.*

routing the separator edges on distinct vertical tracks and along a common horizontal track. (For two components this is trivial since only edge is routed; for three components, place the subforest connected to both other subforests in the middle as shown.) For each node there are three vertical tracks to accomodate edges incident to that node.

The height of the layout is determined by a simple recurrence relation. Let $h(N)$ be the height of the layout, so that $h(1) = 0$, and in general,

$$h(N) \leq h(\lfloor N/2 \rfloor) + 1 .$$

A straightforward calculation yields $h(N) \leq \lg N$.

Thus far we have ignored the coloring on the nodes. Suppose there are b black nodes and $N - b$ white nodes. Consider a "window" which overlaps $\lfloor N/2 \rfloor$ consecutive nodes, and place it over the leftmost $\lfloor N/2 \rfloor$ nodes. If more than $\lfloor b/2 \rfloor$ black nodes fall within the window, slide the window one position to the right. Observe that by sliding the window on position, the number of black nodes within the window changes by at most one. Furthermore, by sliding the window all the way to the right, less than $\lfloor b/2 \rfloor$ black nodes would fall within the window. Consequently, there must be an intermediate placement of the window (see Figure 3.17) in which exactly $\lfloor b/2 \rfloor$ black nodes and exactly $\lfloor (N - b)/2 \rfloor$ white nodes are contained within the window. (Such a placement can be obtained in linear time.)

Draw vertical lines through the endpoints of the window in the position obtained above. The edges of the forest intersecting these lines form a two-color bisector of the forest. The size of this two-color bisector is no more than twice the height of the layout; in other words, the size of the two-color bisector is no more than $2 \lg N$. ∎

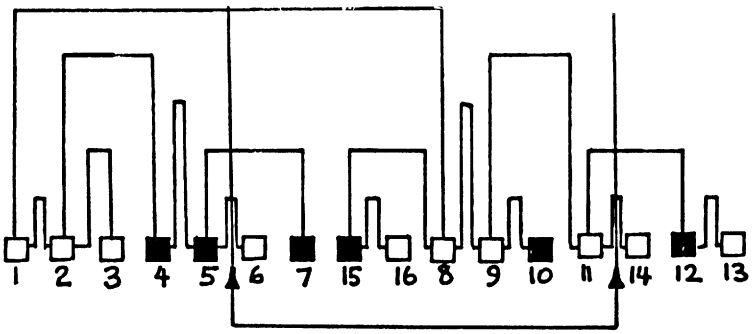


Figure 3.17: *At some point, a window of size $n/2$ slid along the baseline must contain half the black and half the white nodes.*

For our purpose the following variant of two-color bisectors is appropriate. Suppose each node of an N-node forest is assigned a weight from a *bounded* set $\{1, 2, \ldots, k\}$ of weights. We wish to bisect the forest into two equal-size subforests whose total weights differ by at most k. How many edges need be cut? Adapting the argument for two-color bisectors to this variant in a straightforward manner shows again that $2 \lg N$ cuts suffice.

Having obtained bounds on the size of two-color bisectors for forests, we wish to use them for partitioning an arbitrary binary tree into subforests of size at most N so that every subforest has $O(\lg N)$ edges connected to nodes in other subforests. This result is established in the following Theorem.

Theorem 3.8. *Every N-node binary tree can be partitioned into $\lceil N/M \rceil$ subforests, each of size at most M, such that no subforest has more than $4 lg M + 8$ edges connected to nodes in other subforests.*

Proof. First bisect the tree into two subforests, each of size at least $\lfloor N/2 \rfloor$, by cutting no more than $\lg N$ edges. Split each subforest recursively as follows: For each node in a recursively split component of size m assign a weight equal to the number of edges incident to that node and which were cut at a previous level. Since the degree of a node is at most three, the weight assigned to a node is at most 2. From the argument following Theorem 3.7, there is a weighted bisector of size no greater than $2 \lg m$ for the component. This weighted bisector divides the number of external connections almost equally (the difference is at most two) between the subcomponents of sizes $\lfloor m/2 \rfloor$ and $\lceil m/2 \rceil$. As seen in Figure 3.18, the number

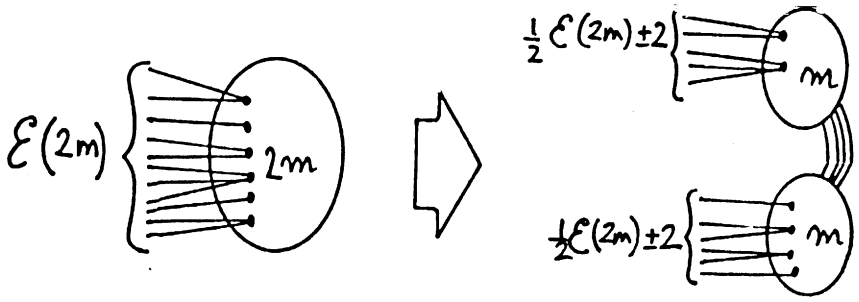


Figure 3.18: *To keep the number of external connections to all subcomponents small when a component is bisected, the external connections must be evenly divided between the subcomponents.*

of external connections into either of the new subcomponents is no more than the size of the weighted bisector plus one-half the number of external connections into the component just split (plus two). This recursive decomposition terminates when each component has size at most M. Letting $\mathcal{E}(m)$ be the number of external connections into any component of size m, we have $\mathcal{E}(N) = 0$, and

$$\mathcal{E}(m) \leq \tfrac{1}{2}\mathcal{E}(2m) + 2\lg(2m) + 2\,.$$

A little calculation shows that $\mathcal{E}(m) \leq 4\lg m + 8$. This means that every subforest of size m in the recursive decomposition has at most $4\lg m + 8$ external edges to other subforests. Substituting M for m, the result follows. ∎

3.7. Assembling Arbitrary Trees

The recursive decomposition of Theorem 3.8 leads directly to the design of an efficient restructurable chip which can assemble all trees. Observe that the layouts developed in earlier sections cannot be used for configurable or restructurable design because the locations at which nodes are embedded are determined by the structure of the tree and are not the same for all trees. The only way to have nodes at fixed locations, independent of the tree structure, is by predetermining the tracks along which edges are routed.

We can predetermine the tracks along which edges are routed by using *restructurable permuters*. A permuter P_k has k terminals on each side of a rectangle and can realize any one-to-one connection between the terminals. The switch shown in Figure 3.19 implements a permuter. It has dimensions $2k \times k$, with the terminals along the longer sides.

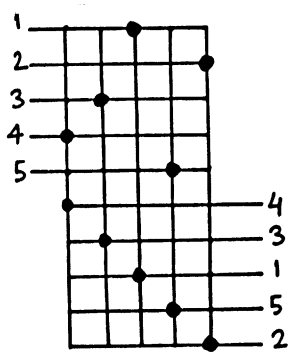


Figure 3.19: *A permuter can realize any set of one-to-one connections between the terminals on its two sides.*

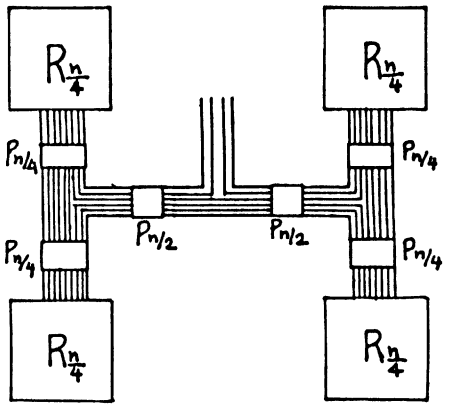


Figure 3.20: *A restructurable chip which can assemble arbitrarily large binary trees.*

The construction of the restructurable chip is recursive and follows the recursive decomposition of Theorem 3.8. We shall use R_m to denote a level of the recursive layout with m nodes, and let R_M denote the restructurable chip of M nodes itself. Figure 3.20 shows how the chip R_M is constructed from four copies of $R_{M/4}$, four copies of $P_{4\lg M}$, and two copies of $P_{4\lg M+4}$. Letting $S(M)$ be the length of the side of the layout, we have $S(1) = 1$ and,

$$S(M) \leq 2S(M/4) + O(\lg M),$$

which yields $S(M) = O(\sqrt{M})$, so that the area is linear in M. The number of pins on R_M is $4\lg M + 8$. We now show that every large tree can be assembled using R_M.

Theorem 3.9. *Suppose each restructurable chip contains M nodes. Then any N-node binary tree can be assembled using $\lceil N/M \rceil$ chips, the minimum possible.*

Proof. Following Theorem 3.8, decompose the tree into $\lceil N/M \rceil$ components, each of size at most M and having no more than $4 \lg M + 8$ external edges to other components. Each of the $\lceil N/M \rceil$ components can be realized on a single chip R_M. To see this, use Theorem 3.8 to recursively decompose each component into single nodes. In this decomposition each subforest of size m has at most $4 \lg m + 8$ external edges. This decomposition may now be mapped directly onto the chip, using the permuters to route edges between different subcomponents. Since the number of external edges at any level is no greater than the size of the permuters at that level, the permuters can realize the desired routing. Nodes of the tree are embedded at fixed positions in the lowest level permuters P_1. Finally, each chip has enough pin connections so that the assembly can be completed off-chip by connecting the chips together as required by the original decomposition. (Permuters are not needed off chip because wires can be routed directly.) ∎

The constant factors on area can be improved if one uses the smaller restructurable permuter P_k with dimensions $(k + O(\sqrt{k})) \times (k + O(\sqrt{k}))$ that follows from the channel routing algorithm of Part II of this thesis. Whereas the simpler permuter from Figure 3.19 requires only two welds to make a connection, the dense layout might require as many as k welds for each connection. Although the total number of welds required by either scheme is $O(M)$, the number per wire is $O(\lg M)$ if the simpler switch is used and $O(\lg^2 M)$ if the channel-routing permuter is used.

In related work, Rosenberg [69] has also considered permuters to obtain a degree of configurability in layouts.

CHAPTER 4

The General Framework

This chapter presents a new framework for general graph layout. Like previous approaches to graph layout, the new framework is based on the divide-and-conquer paradigm. Instead of using a separator theorem to recursively partition a graph, the new framework uses *graph bifurcators*. The notion of a graph bifurcator was introduced by Leighton [12] to overcome the deficiency of separator theorems. Although the differences between bifurcators and separator theorems will be elaborated in this chapter, there are two primary advantages of bifurcators over separator theorems. First, unlike separator theorems, bifurcators may be efficiently computed using either a good graph partitioning heuristic, or from a layout with small area. Second, bifurcators can be used, as in the next chapter, to produce layouts that are efficient in a variety of respects, not layout area alone.

The techniques for general graph layout closely parallel those in Chapter 3 for efficient tree layout. Section 4.1 examines *multi-colored bisectors* for two-ended strings and forests of complete binary trees, and generalizes the results of Section 3.6 to more than two colors. Section 4.2 introduces decomposition trees and bifurcators as generalizations of separator theorems. Section 4.3 considers the problem of balancing decomposition trees, just as Section 3.6 considered the problem of decomposing a tree while balancing the number of external edges among split components. Section 4.4 introduces the *tree of meshes* which is a generalization of the restructurable chip of Section 3.7, and investigates techniques for embedding general graphs within the tree of meshes, given a balanced decomposition tree for the graph. Section 4.5 concludes by developing good layouts for the tree of meshes.

Taken together, an embedding of a graph within the tree of meshes, and a good layout for

the tree of meshes induce a good layout for the embedded graph. The strategy for laying out a general graph, given a decomposition tree is: balance the decomposition tree, embed the graph within the tree of meshes, and lay out the tree of meshes. In Chapter 5 we will see how this strategy can be used to efficiently solve all the layout problems described in Chapter 2.

4.1. Combinatorial Lemmas

This section contains three combinatorial lemmas which provide the foundation for the framework presented in the next section.

Lemma 4.1. *Consider any two-ended string of n colored pearls of k different colors, and let n_i be the number of pearls which are color i for $1 \leq i \leq k$. For any integer $r \geq 2$, the pearls can be partitioned into two sets by cutting the string in no more than $9r^k$ places such that the total number of pearls in each set is $\lfloor n/2 \rfloor$ or $\lceil n/2 \rceil$, the number of pearls of color 1 in each set is $\lfloor n_1/2 \rfloor$ or $\lceil n_1/2 \rceil$, and such that the number of pearls of color $i > 1$ in each set lies between $\lceil (\frac{1}{2} - \frac{1}{2r})n_i \rceil$ and $\lfloor (\frac{1}{2} + \frac{1}{2r})n_i \rfloor$.*

Proof. Let i be a number between 1 and k and let $T(i)$ denote the number of cuts necessary to divide the set of all pearls into two sets that satisfy the constraints of the theorem for colors $1, 2, \ldots, i$. Other than requiring that the total number of pearls be split in half by the cuts, we have made no constraints on the distribution of pearls with colors greater than i. We wish to find a good bound on $T(i)$ in the worst case, i.e., over all choices of $n, k \geq i$, and all possible colorings. In what follows, we will show that $T(1) = 2$ and that

$$T(i) \leq rT(i-1) + 4r + 7$$

for $i > 1$. As a consequence, we can solve the recurrence to conclude that $T(i) \leq 9r^i - 15$ for $r \geq 2$. Thus for $i = k$, at most $9r^k$ cuts are required, as claimed.

For $i = 1$, the argument used in Theorem 3.7 shows that two cuts suffice. Consider a "window" of size $\lfloor n/2 \rfloor$ positioned at the left end of the string. Without loss of generality,

assume that the window covers less than $\lfloor n_1/2 \rfloor$ of the pearls colored 1. Move the window to the right, one pearl at a time until the window covers $\lfloor n_1/2 \rfloor$ pearls of color 1. Since the right half of the string contains more than one-half of all pearls of color 1, there must, by continuity, exist a placement when the window covers exactly one-half of all pearls of color 1. By cutting the string at the endpoints of the window, the portion of the string under the window will contain half of the total number of pearls and half of the pearls colored 1. Hence $T(1) = 2$, as claimed.

For a given $i > 1$, break the string into r segments S_j, $1 \leq j \leq r$, (making $r - 1$ cuts) so that each segment contains at least $\lfloor n_i/r \rfloor$ pearls of color i. Next split each S_j into two subsets S_{j0} and S_{j1} (making a total of $rT(i-1)$ cuts) so that each split satisfies the theorem locally for colors 1, 2,..., $i-1$.

Without loss of generality, assume that S_{j0} contains no fewer pearls of color i than S_{j1}. At this stage, we divide the set C of all pearls into two subsets C_1 and C_2 as follows. Initially, let $C_1 = \bigcup S_{j0}$. If C_1 contains more than $\lfloor (\frac{1}{2} + \frac{1}{2r}) n_i \rfloor$ pearls of color i, remove S_{10} from C_1 and add S_{11}. Repeat this procedure, successively switching S_{20} with S_{21}, S_{30} with S_{31}, and so on until the first time C_1 has at most $\lfloor (\frac{1}{2} + \frac{1}{2r}) n_i \rfloor$ pearls of color i. Such a stage must occur since the number of pearls of color i in C_1 will eventually fall below $\lceil n_i/2 \rceil$ if C_1 and C_2 are completely interchanged. The number of pearls of color i in C_1 after the final switch cannot be less than $\lceil (\frac{1}{2} - \frac{1}{2r}) n_i \rceil - 2$ since every S_j contains no more than $\lceil n_i/r \rceil$ pearls of color i. If the number of pearls of color i in C_1 is $\lceil (\frac{1}{2} - \frac{1}{2r}) n_i \rceil - 1$ or $\lceil (\frac{1}{2} - \frac{1}{2r}) n_i \rceil - 2$, then move either one or two pearls of color i from C_2 to C_1, making no more than four cuts.

We also have to ensure that the total set of pearls and the pearls of the first $i-1$ colors are divided as required. The pearls with colors between 2 and $i-1$ are divided correctly because they were divided correctly at the recursive step. The counts of pearls of color 1 in C_1 and C_2 may differ in size by r, however. To balance the number of pearls with color 1 in each set, we need only remove up to $\lfloor r/2 \rfloor$ pearls colored 1 from the excess set (making at most r cuts) and put them in the deficient set. To balance the difference in the overall sizes of the sets (which now might be as large as $2r + 1$), we need only extract up to $r + 2$ pearls from the larger set (making no more than $2r + 1$ cuts) and put them in the smaller set. Of course, these pearls

must be chosen carefully so that each set retains the required minimum number of pearls of each color. Since pearls are extracted only from the larger set, it is clear that this requirement may be easily satisfied.

The total number of cuts made by the procedure is $rT(i-1)+4r+7$, as claimed. ∎

Using an elegant topological argument, Goldberg and West [32] recently proved that k cuts suffice to divide the pearls of each color exactly in half. This dramatically reduces the number of cuts, and makes our analysis significantly less cumbersome. *All of our layout results may, however, be proved with the weaker Lemma 4.1.* Both results are implementable in polynomial time when the number of colors is fixed, as is the case throughout this thesis.

Lemma 4.2. *Consider any two-ended string of n pearls, n_i of which are colored i, $1 \leq i \leq k$. By cutting the string in k places it is possible to divide the pearls into two sets so that each set has a total of $\lfloor n/2 \rfloor$ or $\lceil n/2 \rceil$ pearls, and $\lfloor n_i/2 \rfloor$ or $\lceil n_i/2 \rceil$ pearls of color i for all i, $1 \leq i \leq k$.*

In the following, we recast Lemma 4.2 in terms of complete binary trees, which will be particularly useful since the recursive decomposition of a graph may be viewed as a tree. The *height* of a tree is the length of the longest path from the root to a leaf, while the *height* of a forest is the maximum height of a tree in the forest. Finally, the *level* of a node in the forest is defined to be the height of the forest minus the length of the longest path from the node to a leaf. (Note that the top level is level zero.)

Lemma 4.3. *Consider a forest of complete binary trees whose n leaves are colored arbitrarily with k colors. Let n_i be the number of leaves colored i for $1 \leq i \leq k$. By removing no more than k nodes (as well as all incident edges) from each internal level of the forest, it is possible to produce a new forest of complete binary trees, some subset of which contains $\lfloor n/2 \rfloor$ or $\lceil n/2 \rceil$ leaves, and $\lfloor n_i/2 \rfloor$ or $\lceil n_i/2 \rceil$ nodes of color i for each i, $1 \leq i \leq k$.*

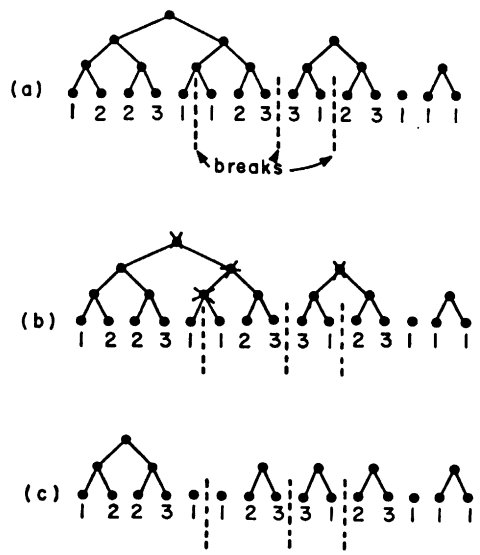


Figure 4.1: *An illustration of the procedure in Lemma 4.3.*

Proof. Draw the trees in the canonical manner and place them side-by-side, in any order, so that the leaves of all trees are placed along a line. By applying Lemma 4.2 to the induced left-to-right ordering on the leaves of the forest, it is possible to break the ordering in no more than k places such that the union of the leaves contained in every other segment contains the desired total number of leaves and the desired number of leaves of each color.

For each break, remove the nodes (and incident edges) which are simultaneously ancestors of the leaf immediately to the left of the break and the leaf immediately to the right of the break. It is easily seen that at most one node is removed from each internal level of the forest for each break. Therefore, no more than k total nodes are removed from each internal level. In addition, the removal of the common ancestors of the leaves neighboring a break divides the associated tree into two or more complete binary trees, at least one on each side of the break. Thus the removal of all such nodes produces a forest of complete binary trees, subsets of which correspond precisely to the sets of leaves between pairs of adjacent break points. Thus the union of the subsets of trees corresponding to every other segment of leaves contains the desired number of leaves of each color. Figure 4.1 illustrates this procedure. ∎

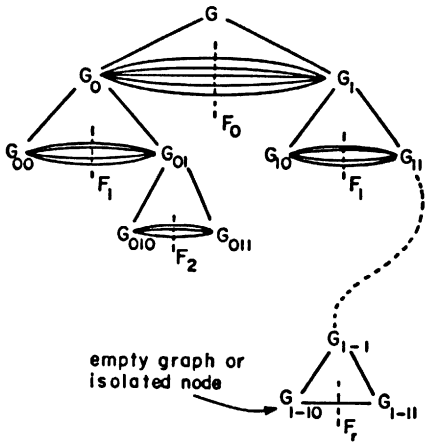


Figure 4.2: *An $(F_0, F_1, \ldots, F_r)$-decomposition tree*

4.2. Decomposition Trees and Bifurcators

The recursive decomposition of a graph into smaller and smaller subgraphs may be viewed as a decomposition tree. In particular, we say that a graph G has an $(F_0, F_1, \ldots, F_r)$-*decomposition tree* if G can be decomposed into two subgraphs G_0 and G_1 by removing no more than F_0 edges from G, and, in turn, both G_0 and G_1 can be decomposed into smaller subgraphs by removing no more than F_1 edges from each, and so on until each subgraph is either empty or an isolated node. Figure 4.2 illustrates this recursive decomposition.

As one might expect, the decomposition of a graph by separator theorems may be viewed as a decomposition tree. It follows by definition that if a class of graphs has an $f(x)$-separator theorem, then there are constants α and β such that each graph in the class has a decomposition tree of the form $(\beta f(N), \beta f(\alpha N), \beta f(\alpha^2 N), \ldots, \beta f(1))$. The converse is not necessarily true. Subgraphs generated at each step of a decomposition by a separator theorem are constrained to be proportional in size, whereas decomposition trees need not satisfy this constraint. Of course, if the decomposition tree has precisely $\lg N$ levels, then subgraphs at each level must be equal in size.

We shall be particularly interested in a special class of decomposition trees, namely *bifurcators,* that is distinct from the class of separators.

Definition. An N-node graph has an α-bifurcator of size F (more simply, an (F, α)-bifurcator) if it has an $(F, F/\alpha, F/\alpha^2, \ldots, 1)$-decomposition tree.

Of particular interest is the class of $\sqrt{2}$-bifurcators. By the definition, we know that an N-node graph has a $\sqrt{2}$-bifurcator of size F if and only if it has an $(F, F/\sqrt{2}, F/2, \ldots, 1)$-decomposition tree. The depth of this tree is no greater than $2 \lg F$. In order to completely decompose an N-node graph into individual nodes, the height of any decomposition tree cannot be less than the $\lg N$. Thus, F must always be at least $\sqrt{N}$. On the other hand, F is always less than $2N$ since every N-node graph with maximum node degree four has at most $2N$ edges.

If a class of graphs has an x^α-separator theorem, where $\alpha \leq 1/2$, and the corresponding decomposition is *balanced* in that every graph is always decomposed into equal-size subgraphs, then it is straightforward to show that every N-node graph in the class has a $\sqrt{2}$-bifurcator of size $O(\sqrt{N})$. Similarly, if a class of graphs has a balanced separator theorem of size x^α with $\alpha > 1/2$, then every N-node graph in the class has a $\sqrt{2}$-bifurcator of size $O(N^\alpha)$.

The converse is not true even if we consider only bifurcators whose corresponding decomposition trees are balanced so that every graph is decomposed into equal-size subgraphs. For example, the N-node graph S_N defined in Section 2.3 has a balanced $\sqrt{2}$-bifurcator of size $O(\sqrt{N \lg N})$ but the smallest separator for this class of graphs is $\Omega(x/\lg^2 x)$.

When translated into bounds on layout area, this seemingly minor difference between bifurcators and separators is greatly magnified. *Graphs with small layout area always have small $\sqrt{2}$-bifurcators, but do not always have small separators.* This is formalized in the following lemma. Later on we will prove the converse: graphs with small $\sqrt{2}$-bifurcators always have small layout area.

Lemma 4.4. *If a graph G can be laid out in area A, then G has a $(\sqrt{A}, \sqrt{2})$-bifurcator.*

Proof. Consider a vertical cut of length $\sqrt{A}$ through the center of the layout. Next, cut each of the sublayouts horizontally through the center. Continuing this sequence of alternating vertical and horizontal cuts, it is easy to see that at the ith step no more than $\sqrt{A}/2^{\lfloor i/2 \rfloor}$ edges are cut from each subgraph. This sequence of cuts yields a $(\sqrt{A}, \sqrt{2})$-bifurcator for G. ∎

4.2.1. Special Cases

Many graphs have decomposition trees in which the number of cuts decreases very slowly as we go lower down the tree. In such cases the number of cuts at higher levels of the tree may be very small. On the other hand, in decomposition trees corresponding to bifurcators, the number of cuts permitted decreases smoothly as we go down the tree. It is conceivable then, that the bifurcator permits far more cuts at higher levels than are necessary. For example, N-node binary trees have decomposition trees of height $O(\lg N)$ in which no more than 1 cut is required at every level. Since the minimum bifurcator is at least $\sqrt{N}$, the decomposition tree corresponding to the bifurcator allows far more cuts at the top levels than needed.

Similarly, some graphs have decomposition trees in which many cuts are required at the top levels, but this number decreases very quickly as we go down the decomposition tree. In such cases, the minimum bifurcator is large so that decomposition trees corresponding to the bifurcator do not underestimate the number of cuts required at the top level. However, they do greatly overestimate the number of cuts at lower levels.

It is useful to separate such extreme cases from a general discussion. Of course, general upper bounds are valid for graphs with extreme decompositions, but they may overestimate the true bound. A particularly important reason for separating these classes is that many computationally useful graphs such as binary trees fall into the first category while cube-connected-cycles and multidimensional meshes fall into the second category.

An N-node graph is defined to have a *type A* $\sqrt{2}$*-bifurcator* if it has an $(O(\sqrt{N}), \sqrt{2})$-bifurcator such that no more than $O((N/2^i)^\alpha)$ cuts, $\alpha < 1/2$, are required for each partition at the ith level of the associated decomposition tree. Observe that at the higher levels of the tree, $i << \lg N$, the number of cuts is far less than the $O(\sqrt{N}/2^{i/2})$ cuts allowed by the usual bifurcator.

Similarly, an N-node graph is defined to have a *type B* $\sqrt{2}$*-bifurcator* if it has an $(O(N^\alpha), \sqrt{2})$-bifurcator, $\alpha > 1/2$, such that only $O((N/2^i)^\alpha)$ edges are cut in any partition at the ith level. Observe that for the lower levels of the tree, $i >> 1$, this quantity is far smaller than the $O(N^\alpha/2^{i/2})$ cuts allowed by the usual bifurcator.

For simplicity, we will prove results only for general $\sqrt{2}$-bifurcators in this thesis. However,

whenever there is a significant difference, results for the special cases are stated separately. The proofs for these special cases are easily worked out, and closely follow the proofs for the general cases.

4.3. Balanced Decomposition Trees

Of particular interest to the layout results reported in this thesis are decomposition trees where at each step of the decomposition, the two subgraphs are nearly equal in size. This section considers such balanced decompositions and gives an effective procedure for transforming an arbitrary decomposition tree into one that is balanced.

Formally, a decomposition tree for a graph G is *balanced* if each subgraph G_w in the tree is the father of two subgraphs G_{w0} and G_{w1} such that the number of nodes in the subgraphs differ by at most 1. In addition, we say that a decomposition tree is *fully balanced* if it is balanced, and if for every subgraph G_w in the tree, the set of edges connecting $G - G_w$ to G_w is divided into two subsets of nearly equal size by the partition of G_w into G_{w0} and G_{w1}. (Here we allow the number of edge connections in the two subgraphs to differ by a small constant, say 5. For the purposes of simplicity, however, we shall often ignore such small differences and assume that the nodes and connections are split evenly between the two subgraphs.)

Somewhat surprisingly, any decomposition tree may be transformed into a fully balanced one at little or no cost. We prove this in the following theorem which generalizes earlier results in [9, 40, 41, 42].

Theorem 4.5. *Let G be any N-node graph with an $(F_0, F_1, \ldots, F_r)$-decomposition tree T. Then G has a fully balanced $(F'_0, F'_1, \ldots, F'_{\lg N})$-decomposition tree, such that for $0 \leq i \leq \lg N$,*

$$F'_i = 6 \sum_{s=i}^{r} F_s$$

Proof. Let Γ be a forest of complete binary trees consisting initially of the decomposition tree T. Color the leaves of T with two colors according to whether or not the subgraph of G

associated with the leaf is empty. Apply Lemma 4.3 ($k = 2$) to Γ, removing the indicated nodes and edges of T. Each node of T corresponds naturally to a set of edges of G, namely the edges whose removal splits the associated subgraph in two. Removing a node of T corresponds to removing this cutset of edges from G. Since no more than 2 nodes are removed from each level of Γ, the number of edges removed from G in applying Lemma 4.3 does not exceed $2\sum_{s=0}^{r} F_s$, which is less than F'_0.

Further note that G is divided into two disjoint subgraphs of nearly-equal-size by the removal of these edges. Each subgraph, in turn, corresponds in a natural way to a subforest of complete binary trees in Γ. Consider one such subgraph G_0 and color the leaves of the associated forest of complete binary trees Γ_0 using six colors as follows:

> If the leaf corresponds to an empty subgraph, color the leaf with color 1. Otherwise, if the single node corresponding to the leaf is incident to exactly j edges of G removed earlier, $0 \leq j \leq 4$, then color the leaf with color $j + 2$.

By applying Lemma 4.3 ($k = 6$) to Γ_0, it is clear that G_0 can be decomposed into two disjoint subgraphs G_{00} and G_{01} of nearly-equal-size such that the number of edges from $G - G_0$ to G_{00} is nearly-equal to the number of edges from $G - G_0$ to G_{01}. Since at most 6 nodes were removed from each level of Γ_0 and since Γ_0 does not contain the root of T, we can conclude that no more than $6\sum_{s=1}^{r} F_s = F'_1$ edges were removed from G_0.

By applying the above argument recursively, the desired fully-balanced decomposition tree is obtained. With each application of Lemma 4.3, the total number of leaves in each forest is cut in half at each step so that the biggest tree in any forest corresponding to a subgraph decreases in height by at least one. Also, $\lg N + 1$ levels suffice since the size of each subgraph is also halved at each step. ∎

Theorem 4.6. *Every graph with a $\sqrt{2}$-bifurcator of size F has a fully balanced $\sqrt{2}$-bifurcator of size $6(2 + \sqrt{2})F$.*

Proof. Immediate from Theorem 4.5, since $\sum_{i \geq 0} 2^{-i/2} \leq 2 + \sqrt{2}$. ∎

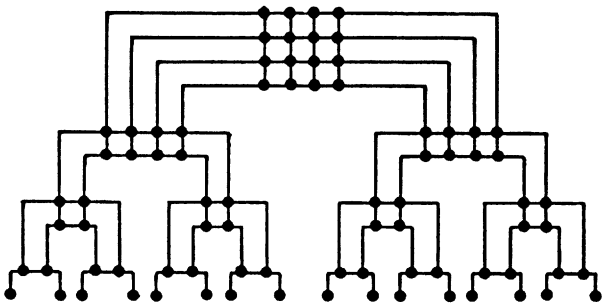


Figure 4.3: *The* 4×4 *tree of meshes* T_4.

Remark. The procedure described in Theorems 4.5 and 4.6 can be implemented in polynomial time.

4.4. Embeddings in the Tree of Meshes

Leighton [40, 41] introduced the tree of meshes as an example of a planar graph that cannot be laid out in linear area. He also showed that every N-node planar graph can be embedded in an $O(N \lg N)$-node tree of meshes. In this section, we define the tree of meshes and describe a general strategy for embedding a graph in the tree of meshes.

The *tree of meshes* is formed by replacing each node of a complete binary tree with a mesh and each edge by several edges which connect meshes at consecutive levels. More precisely, the root of the complete binary tree is replaced by an $n \times n$ mesh (it is assumed that n is a power of 2), the nodes at the second level are replaced by $n \times n/2$ meshes, those at the third level by $n/2 \times n/2$ meshes, and so on until the leaves of the tree are replaced by 1×1 meshes. As shown in Figure 4.3, each edge of the tree is replaced with edges connecting nodes on one side of the higher-level mesh to the top row of the mesh at the lower level. The resulting graph is called the $n \times n$ tree of meshes T_n. It is not difficult to see that T_n, has $N = 2n^2 \lg n + n^2$ nodes.

In many cases, we use only the top levels of the tree of meshes. The subgraph consisting of levels $0, 1, \ldots, p$ ($p \leq 2 \lg N$) of T_n is called a *truncated tree of meshes* $T_{n,p}$.

Theorem 4.7. *There is a constant c such that every N-node graph G with an $(F, \sqrt{2})$-bifurcator can be embedded in $T_{cF,2\lg\frac{N}{F}}$. Moreover, the embedding is regular in the sense that F^2/N nodes of G are embedded in a regular fashion each of the N^2/F^2 bottom-level meshes of $T_{cF,2\lg\frac{N}{F}}$.*

Proof. We first use Theorem 4.6 to construct a fully-balanced $\sqrt{2}$-bifurcator of size $6(2+\sqrt{2})F$ for G. We then use the internal meshes of $T_{cF,2\lg\frac{N}{F}}$ to route the edges that were removed in the upper $2\lg\frac{N}{F}$ levels of the fully balanced decomposition tree for G. The subgraphs in the $(2\lg\frac{N}{F})$th level of the decomposition tree (each of which has $\lfloor F^2/N \rfloor$ or $\lceil F^2/N \rceil$ nodes) are then embedded in the meshes on the bottom level of the truncated tree of meshes.

The internal meshes are used as restructurable permuters. As we saw in Section 3.7, terminals on opposite sides of a mesh can be connected in any order through the mesh. In general, if the number of wires routed through a mesh does not exceed any side-length of the mesh, a routing may always be found. Similarly, a graph with M nodes can always be embedded in a $4M \times 4M$ mesh with nodes placed in a regular fashion.

Consider only the top $2\lg\frac{N}{F}+1$ levels of a fully balanced decomposition tree for G. Each of the subgraphs at level $2\lg\frac{N}{F}$ of the decomposition tree has $N(1/2)^{2\lg\frac{N}{F}} = F^2/N$ nodes. (For simplicity we shall assume that F^2/N is an integer.) Furthermore, if E_i is the maximum number of edges between $G - G_i$ and G_i, where G_i is a subgraph in the decomposition tree at level i, then it is easy to see that $E_0 = 0$ and by Theorem 4.6, that

$$E_i \leq \frac{1}{2}E_{i-1} + 6(2+\sqrt{2})\frac{F}{2^{(i-1)/2}}$$

for $1 \leq i \leq 2\lg\frac{N}{F}$. Solving the above recurrence, we obtain:

$$E_i \leq 6(2+\sqrt{2})\frac{F}{2^{(i-1)/2}}\sum_{s\geq 0}(\sqrt{2}/2)^s,$$

and thus

$$E_i \leq 6(2+\sqrt{2})^2\frac{F}{2^{(i-1)/2}}.$$

We now embed G in $T_{cF,2\lg\frac{N}{F}}$. First, embed each of the $(2\lg\frac{N}{F})$-level subgraphs of the decomposition tree in the bottom level meshes. This can be done if the side of each mesh at level $2\lg\frac{N}{F}$ exceeds $4F^2/N$. This is true provided

$$cF/\sqrt{2}^{2\lg\frac{N}{F}} \geq 4F^2/N.$$

For $c \geq 4$, this inequality is easily satisfied.

Next embed the additional edges through the upper-level meshes in the natural way. No more than $2E_{i+1}$ edges pass through any ith level mesh. Thus the routing can be performed if the smaller side of the ith level meshes exceeds $2E_{i+1}$. In other words, we must have:

$$cF/2^{\lceil i/2\rceil} \geq 12(2+\sqrt{2})^2 F/2^{i/2}.$$

A simple calculation shows that the inequality is satisfied for sufficiently large c. ∎

Remark. Throughout the thesis, we express bounds using the term $\lg\frac{N}{F}$. For all practical purposes, F is much smaller than N and this term is greater than one. Should the value of F be larger, however, we shall still define $\lg\frac{N}{F}$ to be at least one. Similar interpretations are assumed for $\lg\lg\frac{N}{F}$ and for $\lg\lg\lg\frac{N}{F}$. The conventions avoid the annoying (and trivial) cases when F is very large without complicating the analysis further.

In the preceding embedding, all the nodes of G were mapped to meshes at the bottom level of the truncated tree of meshes. Thus, edges between nodes in different meshes might have to be routed through as many as $4\lg\frac{N}{F}$ meshes. Such long edges are undesirable for a variety of reasons. It is natural to ask whether an embedding can be found in which each edge can be routed through fewer intermediate meshes. This is answered in the following theorem.

Theorem 4.8. *There exist constants c and k such that every N-node graph G with an $(F, \sqrt{2})$-bifurcator can be embedded in $T_{cF, 2\lg \frac{N}{F}}$ and such that no edge is routed through more than k intermediate meshes.*

Proof. We adopt a slight variant of the strategy used in the previous theorems. The balancing and embedding are done simultaneously and in the same manner as before, except at levels $0, k, 2k, 3k, \ldots$ (where k is a constant specified later). At these levels, we embed the nodes that are incident to edges previously cut, and we cut the previously uncut edges incident to these nodes. Of course, this could triple the number of cut edges every k levels but if k is sufficiently large, this happens infrequently and is not harmful. At all other levels the procedure is the same as before, using 6 colors and Lemma 3 to partition the decomposition tree. The process terminates after $2 \lg \frac{N}{F}$ levels.

As before, the embedding is accomplished by using meshes as switching boxes for routing edges. We must ensure that the number of edges routed through any mesh does not exceed the side lengths of the mesh. The calculation is the same as before except that the number of cut edges is tripled at every kth level. Thus the recurrence for E_i is

$$E_i \leq \frac{1}{2}(3^{1/k})E_{i-1} + 6(2 + \sqrt{2})\frac{F}{2^{(i-1)/2}}.$$

Here, we have (without loss of generality) increased number of cut edges by a factor of 3 initially and by a factor of $3^{1/k}$ at each level instead of increasing the number of cuts by a factor of 3 at every kth level. Solving the recurrence, we find

$$E_i \leq 18(2 + \sqrt{2})\frac{F}{2^{(i-1)/2}} \sum_{s \geq 0} \left(\frac{\sqrt{2}}{2} 3^{1/k}\right)^s.$$

For $k \geq 4$, the sum converges to a constant. The remaining analysis is the same as in the previous theorems except that the constants are larger. ∎

Remark. It is worthwhile to point out here that Theorems 7 and 8 could also have been proved using Lemma 4.1 instead of Lemma 4.2. The nodes of G would still be balanced in the decomposition tree but the cut edges could only be split 1/3 - 2/3 at each decomposition.

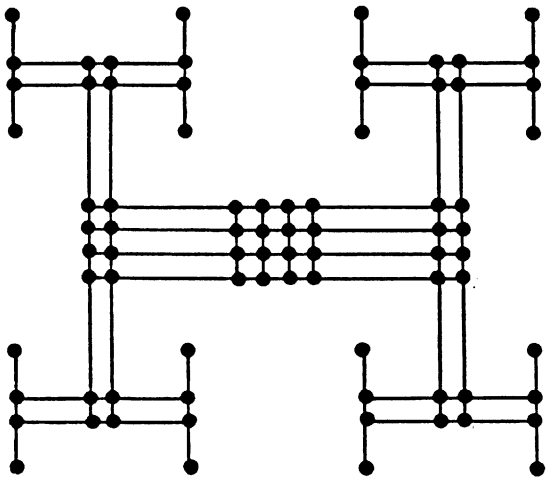


Figure 4.4: *The H-layout of the tree of meshes*

While this increases the value of the sum, it still converges to a constant. (This is because, for sufficiently large k, $\frac{2\sqrt{2}}{3}3^{1/k} < 1$.) Hence, k and c would be larger but the statements of the theorems remain the same.

4.5. Layouts for the Tree of Meshes

Thus far we have considered only the problem of embedding graphs in the tree of meshes. How do we lay out the tree of meshes efficiently? Clearly, any layout for the tree of meshes also gives a layout for every graph that can be embedded within the tree of meshes. In this section we develop two different layouts for the tree of meshes.

The first layout is a straightforward modification of the "H-tree" layout for complete binary trees [55]. The modified layout is obtained by expanding each node of the complete binary tree into a mesh of the appropriate size. Figure 4.4 shows this layout. It is easy to see that if $S(F)$ denotes the side of the layout for T_F, then $S(1) = 1$, and

$$S(F) \leq 2S(F/2) + O(F),$$

which gives $S(F) = O(F \lg F)$. This means that the area of the layout for T_F is bounded by $O(F^2 \lg^2 F)$. As shown in [40, 41], this bound is optimal.

For truncated trees of meshes, such as considered in Theorems 4.7 and 4.8, a similar result holds.

Theorem 4.9. *The truncated tree of meshes $T_{F,2\lg\frac{N}{F}}$ has a layout of area $O(F^2\lg^2\frac{N}{F})$.*

Proof. The obvious restriction of the H-layout to the top levels suffices. ∎

Although the mesh edges in the layout shown in Figure 4.4 have length 1, the edges between meshes can be quite long (nearly half the side of the layout). By pulling in meshes closer towards the top level, we can reduce the length of the longest edge considerably. This technique was introduced in Chapter 3 to produce minimax edge length layouts for trees, and generalizes to graphs with known bifurcators. This layout will later be used to find layouts with short edges for graphs embedded within the truncated tree of meshes.

Theorem 4.10. *The truncated tree of meshes $T_{F,2\lg\frac{N}{F}}$ can be laid out in area $O(F^2\lg^2\frac{N}{F})$ so that mesh edges have length 1 and edges between meshes have length at most $O(F\lg\frac{N}{F}/\lg\lg\frac{N}{F})$.*

Proof. Consider the H-tree layout of a complete binary tree of height $2\lg\lg\lg\frac{N}{F}$, and having $(\lg\lg\frac{N}{F})^2$ leaves. Expand each linear dimension by a factor $\beta = \Theta(F\lg\frac{N}{F}/\lg\lg\frac{N}{F})$, so that each edge of the H-tree layout becomes a channel of width β and each node becomes a $\beta\times\beta$ square. The resulting area is $(\beta\lg\lg\frac{N}{F})^2 = \Theta(F^2\lg^2\frac{N}{F})$.

Since the channels are much wider than the side of any mesh, we can stack many meshes within one channel. In particular, as seen in Figure 4.5, we embed the top level mesh at the center of the layout with the second-level meshes on either side. In the first stage of the layout, the meshes in the top levels are placed together in a breadth-first manner. Meshes at successive levels are equally spaced at distance $\Theta(F\lg\frac{N}{F}/\lg\lg\frac{N}{F})$ apart.

We need to ensure that every channel is wide enough to accomodate the meshes stacked within it. To this end, let us suppose that all meshes embedded in the first stage are stacked together in the same channel. Of course, this is a gross overestimate, but suffices for our argument. Since the path from the root to a leaf in the original $(\lg\lg\frac{N}{F})^2$-leaf H-layout has length $\Theta(\lg\lg\frac{N}{F})$, a total of $c\lg\lg\frac{N}{F}$ levels of $T_{F,2\lg\frac{N}{F}}$ are embedded in the first stage. The value of the constant c depends on the values of the other constants in the Θ-terms and can be made as small as necessary.

The total number of meshes embedded in the first stage is no more than $2^{1+c\lg\lg\frac{N}{F}}$. Each

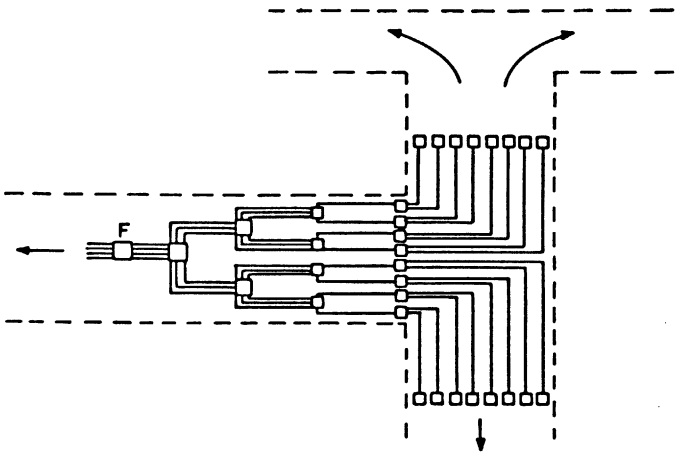


Figure 4.5: *An improved layout for the tree of meshes.*

mesh has side length no greater than F, so to stack all these meshes within one channel of side β, it suffices to have:

$$F2^{1+c\lg\lg\frac{N}{F}} \leq O\left(\frac{F\lg\frac{N}{F}}{\lg\lg\frac{N}{F}}\right),$$

which is easily satisfied when $c \leq 1/2$. Hence every channel has sufficient width to stack all the ith level meshes across the channel for any $i \leq c\lg\lg\frac{N}{F}$.

In the second stage, we embed the remaining meshes in the $\beta \times \beta$ squares. A total of $(\lg\frac{N}{F})^c/(\lg\lg\frac{N}{F})^2$ copies of an $O(\lg\frac{N}{F})$ level $\frac{F}{(\lg\frac{N}{F})^{c/2}} \times \frac{F}{(\lg\frac{N}{F})^{c/2}}$ truncated tree of meshes must be embedded in each of the $(\lg\lg\frac{N}{F})^2$ $\beta \times \beta$ regions to accomplish this. Using the layout described in Theorem 4.9 for each copy, the total area required in each region is

$$\Theta\left(\frac{(\lg\frac{N}{F})^c}{(\lg\lg\frac{N}{F})^2}\frac{F^2}{(\lg\frac{N}{F})^c}\lg^2\left(\frac{N}{F}\right)\right) = \Theta\left(\frac{F^2\lg^2\frac{N}{F}}{(\lg\lg\frac{N}{F})^2}\right).$$

This is precisely the amount of area available in each $\beta \times \beta$ region. Hence the embedding is possible.

It remains to verify that the edges between meshes have length $O(F\lg\frac{N}{F}/\lg\lg\frac{N}{F})$. This is easily done since meshes in adjacent levels were spaced distance $\Theta(F\lg\frac{N}{F}/\lg\lg\frac{N}{F})$ apart in the first stage, and since meshes in adjacent levels were located in the same $\beta \times \beta$ region in the second stage. ∎

CHAPTER 5

Solving the Layout Problems

Using the framework described in the previous section, we are now ready to present general solutions to the eight problems posed in Chapter 2. The layout framework of Chapter 4 applies directly to most of these problems, supporting our belief that the divide-and-conquer strategy based on bifurcators is an efficient paradigm for VLSI graph layout. In particular, the tree of meshes emerges as an extremely versatile network for graph layout. While specific instances of some problems might be better solved using different techniques, the framework provides a novel and uniform approach for VLSI layout which effectively addresses various unrelated issues. The solutions presented in this section are evaluated by comparing them with known lower bounds.

Problem 1. *Given a graph G, produce an area-efficient layout for G.*

By Theorem 4.7, every N-node graph with an $(F, \sqrt{2})$-bifurcator can be embedded in the truncated tree of meshes $T_{O(F),2\lg\frac{N}{F}}$. Next, by Theorem 4.9, the truncated tree of meshes can be laid out in $O(F^2 \lg^2 \frac{N}{F})$ area. Therefore, every N-node graph with an $(F, \sqrt{2})$-bifurcator can be laid out in $O(F^2 \lg^2 \frac{N}{F})$ area.

As a consequence of Lemma 4.4, every N-node graph whose smallest $\sqrt{2}$-bifurcator is F, *must* occupy at least F^2 area. For otherwise the graph would have a $\sqrt{2}$-bifurcator strictly smaller than F. Therefore, for *every* graph the upper bound is at most a factor of $O(\lg^2 \frac{N}{F})$ worse than optimal, i.e., the area bound is *universally* close to optimal.

The bounds are also *existentially optimal.* Leighton [7, 42] has shown the existence of N-node graphs with minimum $\sqrt{2}$-bifurcator F which require area at least $\Omega(N \lg^2 \frac{N}{F})$. In

other words, no strategy based on bifurcators alone can asymptotically improve upon the divide-and-conquer framework.

Special Cases. Graphs with $(F, \sqrt{2})$-bifurcators with either of the special forms described in Section 4.2.1 have $O(F^2)$-area layouts. Thus, for example, N-node trees have $O(N)$-area layouts.

Problem 2. *Given a graph G, produce an area-efficient layout for G with minimax edge length.*

From Theorem 4.8 we know that every N-node graph with an $(F, \sqrt{2})$-bifurcator can be embedded in the truncated tree of meshes $T_{O(F),2 \lg \frac{N}{F}}$ so that no edge passes through more than a constant number of intermediate meshes. Furthermore, the layout for the truncated tree of meshes given in Theorem 4.10 guarantees that every edge between meshes has length bounded by $O(F \lg \frac{N}{F} / \lg \lg \frac{N}{F})$, and that every edge within a mesh has length one. Combining these two theorems, we see that every N-node graph with an $(F, \sqrt{2})$-bifurcator has an $O(F^2 \lg^2 \frac{N}{F})$-area layout with maximum edge length bounded by $O(F \lg \frac{N}{F} / \lg \lg \frac{N}{F})$.

This bound, too, is existentially optimal [7]. In other words, there exist N-node graphs with minimum $\sqrt{2}$-bifurcator F whose minimax edge length is $\Omega(F \lg \frac{N}{F} / \lg \lg \frac{N}{F})$.

Unfortunately, the bounds are not universally close to optimal. The only general lower bound on minimax edge length for N-node graphs whose minimum $\sqrt{2}$-bifurcator is F, is $\Omega(F^2/N)$. This general lower bound is also existentially optimal.

The problem of minimizing maximum edge length appears to quite difficult. Although the preceding bounds are disappointingly weak, they are the best known. Recall that in Chapter 3 we showed that even determining if a tree can be laid out with minimax edge length one, is NP-complete.

Special Cases. The minimax edge length bounds for graphs with special $(F, \sqrt{2})$-bifurcators are $O(\sqrt{N} / \lg N)$ for type A $\sqrt{2}$-bifurcators and $O(F)$ for type B $\sqrt{2}$-bifurcators.

Problem 3. *Given a graph, produce an area-efficient layout in which each wire has bounded delay in the capacitive model.*

First we formalize some details of the model. As usual, a graph describes a connection of processors, with an edge corresponding to a bidirectional link between two processors. Each node is a processing element which contains one driver and one receiver for each incident edge. Every transistor in a processing element has the same size. Thus, in our layouts, a node may be represented by a long and skinny box of constant thickness, with length equal to the area of an internal transistor. Since each node has bounded degree, a box will be just big enough to contain all the transistors in the corresponding processor. Note that different nodes in the layout will have different lengths, but the same thickness. We assume that the grid spacing is adjusted so that nodes and edges have unit thickness and may be laid along grid lines. Although wires are allowed to cross, we will not allow nodes to cross; this corresponds to transistors not overlapping. Similarly, wires and nodes may not cross. The propagation delay over a wire of length l driven by a transistor of area D with capacitive load A is proportional to $(l + A)/D$. The capacitive load presented to a transistor equals the sum of incident wire lengths and areas of adjacent transistors.

Theorem 5.1. *Every N-node graph G with an $(F, \sqrt{2})$-bifurcator has a bounded-delay layout of area $O(F^2 \lg^2 \frac{N}{F})$.*

Proof. As in Theorem 4.8, embed G in a tree of meshes so that adjacent nodes are mapped to meshes no more than a constant number of levels apart. Since the dimensions of meshes at successive levels, as well as the lengths of edges connecting adjacent meshes in the layout of Theorem 4.9, decrease at the same geometric rate, we know that the length of an edge of G is proportional to the side lengths of the meshes that contain the corresponding nodes. Assign to each node an area that is proportional to the side lengths of the mesh in which it is embedded. Thus, the capacitive load on any node, which equals the sum of the areas of all the incident edges and adjacent nodes, is proportional to the area of the node. In other words, every wire in the layout has bounded delay.

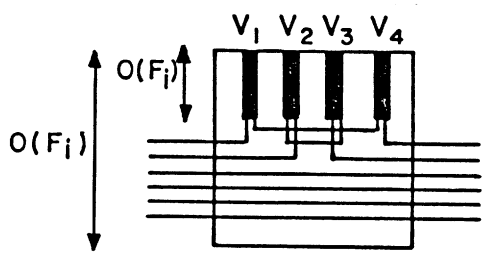


Figure 5.1: *Laying out expanded nodes in a mesh.*

We need to ensure that each enlarged node can be accomodated in its assigned mesh without blowing up the area of the layout by more than a constant factor. This can be done by increasing the dimensions of each mesh by a constant factor, and laying out the nodes and incident edges as shown in Figure 5.1. Notice that the nodes do not overlap other nodes or wires. The area of each node remains proportional to the side lengths of the mesh containing it, and thus the delay across every wire is bounded. ∎

Special Cases. Similarly, graphs with special $(F, \sqrt{2})$-bifurcators have $O(F^2)$-area bounded-delay layouts. Thus, for example, every N-node tree has an $O(N)$-area bounded-delay layout.

Theorem 5.1 implies that the area bounds for bounded-delay layouts are no worse than the best known *general* area bounds for Problem 1. However, it is not known whether or not there exists a graph for which any bounded-delay layout requires asymptotically greater area than the minimum area layout. In the following corollary, we show that any increase in area need not be large.

Corollary 5.2. *Any layout of area A for an N-node graph can be transformed into a bounded-delay layout of area $O(A \lg^2 \frac{\sqrt{A}}{N})$.*

Proof. By Lemma 4.4, an area A layout yields a $(\sqrt{A}, \sqrt{2})$-bifurcator which can be quickly found. Next, by Theorem 5.1, a bounded-delay layout of area $O(A \lg^2 \frac{\sqrt{A}}{N})$ can be easily constructed. Observe that this transformation is effective. ∎

Problem 4. *Given a graph G, produce a layout for G with few wire crossings.*

The layouts for the truncated tree of meshes in Theorems 4.9 and 4.10 do not have any edge crossings. Since every N-node graph G with an $(F, \sqrt{2})$-bifurcator can be embedded within the truncated tree of meshes $T_{O(F),2\lg\frac{N}{F}}$, this means that the number of crossings in the layout for G cannot exceed the number of nodes in $T_{O(F),2\lg\frac{N}{F}}$. In other words, the number of crossings in the layout for G is bounded by $O(F^2 \lg \frac{N}{F})$.

Once again, this bound too is existentially optimal [7]. Moreover, if the minimum $\sqrt{2}$-bifurcator F of an N-node graph is asymptotically greater than $\sqrt{N}$, the number of crossings in the layout for G is no more than a factor $O(\lg \frac{N}{F})$ times optimal.

Special Cases. Graphs with special $(F, \sqrt{2})$-bifurcators can be laid out with $O(F^2)$ crossings.

Problem 5. *Given a graph, produce an area-efficient regular layout for the graph.*

In Theorem 4.7, we showed how to embed any N-node graph G with an $(F, \sqrt{2})$-bifurcator in $T_{cF,2\lg\frac{N}{F}}$ for some constant c. Moreover, the nodes of G were divided evenly among the N^2/F^2 bottom-level meshes of $T_{cF,2\lg\frac{N}{F}}$ and in each bottom-level mesh, the nodes of G were embedded in a regular fashion. Thus to produce an $O(F^2 \lg^2 \frac{N}{F})$-area layout for G that is regular, we need only produce a layout for $T_{cf,2\lg\frac{N}{F}}$ for which the nodes at the $(2\lg\frac{N}{F})$th level are located in a regular fashion. In fact, we can do much better, as we show in the following theorem.

Theorem 5.3. *The truncated tree of meshes $T_{O(F),2\lg\frac{N}{F}}$ can be laid out in $O(F^2 \lg^2 \frac{N}{F})$ area so that, for every level i, all nodes within ith level meshes are placed in a regular fashion.*

Proof. The first step is to construct a $\Theta(\lg \frac{N}{F})$-layer three-dimensional layout [46] of the truncated tree of meshes. Fold the connections between the root of the tree of meshes and each of its two sons so that the sons fit naturally on a second layer over the root mesh. Fold the connections to each of the meshes at the next lower level so they fit, on the third layer, directly over the meshes on the second layer, and so forth. This generates a $\lg \frac{N}{F}$-layer three-

dimensional layout, with each layer occupying linear area. By projecting the three-dimensional layout onto the plane in the manner of Thompson [80, pp. 36-38], the result follows. (The same layout can be constructed by interleaving the meshes at each level.) ∎

Special Cases. The $O(F^2)$-area layouts for graphs with special $\sqrt{2}$-bifurcators are also regular.

Problem 6. *Design area-efficient chips that can be configured to realize a large number of graphs.*

In Theorem 4.7 we showed that every N-node graph with an $(F, \sqrt{2})$-bifurcator can be embedded in a truncated tree of meshes such that the nodes of the graph are embedded in a regular fashion in the bottom-level meshes of $T_{cF,2\lg\frac{N}{F}}$. In fact, the nodes can be mapped to fixed positions within the meshes. Therefore, if we lay out the truncated tree of meshes on a chip with processors at these fixed positions, we have a configurable chip for all graphs with the corresponding bifurcator. This yields the following result. Observe that the area bounds for configurable layouts are the same as for unrestricted layouts.

Theorem 5.4. *Every N-node graph with an $(F, \sqrt{2})$-bifurcator has a configurable layout of area $O(F^2 \lg^2 \frac{N}{F})$.*

Proof. Simply make the connections in the meshes after the rest of the chip has been fabricated. Recall that we used the meshes as crossbar switches in Theorem 4.7. ∎

Special Cases. Similarly, graphs with special bifurcators have $O(F^2)$-area configurable layouts. The $O(N)$-area restructurable tree layout of Chapter 3 is such an example.

Problem 7. *On a wafer which has arbitrarily distributed defective cells, realize a given graph on the good cells.*

Theorem 4.7showed how to embed any N-node graph G with an $(F, \sqrt{2})$-bifurcator in the truncated tree of meshes $T_{O(F),2\lg\frac{N}{F}}$. The embedding had the property that nodes of the graph could be mapped to fixed positions within the meshes at the bottom level. Accordingly, we fixed processors at each of these positions.

Faulty processors on a wafer therefore correspond to faulty processors in the truncated tree of meshes, the correspondence being induced via the layout for the tree of meshes. It is clearly no longer possible to realize G in the faulty tree of meshes. However, it is possible to realize a smaller graph with a similar structure using only the functioning processors.

More formally, consider a class of graphs for which any N-node graph in the class has a $\sqrt{2}$-bifurcator of size $O(f(N))$ where the function f is such that $f(x)/\sqrt{x}$ is nondecreasing for increasing x. For example, $f(x) = \sqrt{x}$ for the class of square meshes (as well as for the class of trees or the class of planar graphs). In what follows, we will show how to embed any M-node graph from the class in any $T_{cf(N),2\lg\frac{N}{f(N)}}$ that has M functioning processors where $N \geq M$ and c is a sufficiently large constant.

In particular, we will show how to embed $T_{f(M),2\lg\frac{M}{f(M)}}$ in the faulty tree of meshes. By applying Theorem 4.7 to the smaller tree of meshes embedded within the faulty one, this will prove our claim. Thus the layout strategy developed in Chapter 4 is impervious to the existence of faulty processors. This result substantially generalizes and simplifies a similar result proved by Leighton and Leiserson for embedding meshes around faults in [45].

Theorem 5.5. *Given the preceding constraints on N, M, c and f, a completely functioning truncated tree of meshes $T_{f(M),2\lg\frac{M}{f(M)}}$ with M processors can be embedded in any partially functioning truncated tree of meshes $T_{cf(N),2\lg\frac{N}{f(N)}}$ with N processors (M of which are functioning) so that the processors of the former are mapped onto the functioning processors of the latter.*

Proof. Label the functioning processors in each tree of meshes from 1 to M by counting from left to right across the bottom level of each graph. (Recall that the processors are evenly distributed on the bottom level.) Map the kth processor of $T_{f(M),2\lg\frac{M}{f(M)}}$ onto the kth functioning processor of $T_{cf(N),2\lg\frac{N}{f(N)}}$. Route the edges of the former graph through the meshes of the latter in the usual way, at the same time embedding meshes of the former in blocks within the meshes of the latter.

It remains to show that the capacity of each mesh in $T_{cf(N),2\lg\frac{N}{f(N)}}$ is sufficient for the

embedding. Consider a mesh X on the ith level of $T_{cf(N),2\lg\frac{N}{f(N)}}$. This mesh has side lengths $cf(N)/2^{i/2}$ and at most $N/2^i$ functioning processors below it in the bottom level of the graph. The only meshes and edges of $T_{f(M),2\lg\frac{M}{f(M)}}$ that are embedded in X are those that correspond to roots of the forest of complete binary trees formed by removing the corresponding interval of (at most $N/2^i$) processors in $T_{f(M),2\lg\frac{M}{f(M)}}$. These roots are identified by splitting $T_{f(M),2\lg\frac{M}{f(M)}}$ (as in Lemma 4.3) at the two endpoints of the interval. There are at most two roots at each level in the resulting forest and the sum of their side lengths (a geometrically decreasing sum) is proportional to $f(M)/2^{j/2}$ where j is such that $M/2^j \leq N/2^i$. (Remember that there are at most $N/2^i$ processors in the leaves of the forest so that the height of the largest complete binary tree in the forest is j where $M/2^j \leq N/2^i$.) Thus the sum of the side lengths of the meshes embedded in X is $O\left(\frac{f(M)}{2^{i/2}}\sqrt{\frac{N}{M}}\right)$ which, for sufficiently large c, is less than $cf(N)/2^{i/2}$ (this is the side length of X), since $N \geq M$ and $f(x)/\sqrt{x}$ is a nondecreasing function. Hence X is large enough and the embedding is possible. ∎

Special Cases. A similar argument works for graphs with special bifurcators.

Problem 8. *Given a graph G, assemble G using the minimum number of copies of a single chip having few external pin connections.*

Suppose that we wish to assemble N-node graphs with $(F, \sqrt{2})$-bifurcators but that each chip contains only m nodes, where $m < N$. Consider a chip consisting of a truncated tree of meshes $T_{O(\frac{\sqrt{mF}}{\sqrt{N}}),O(\lg\frac{\sqrt{mN}}{F})}$, with the m processors divided equally among the bottom-level meshes, and external pin connections to the top of the top level mesh. Two copies of this chip may be wired together to form a truncated tree of meshes with $2m$ processors. Thus, graphs with twice as many processors can be assembled with two chips than can be assembled on a single chip. More generally, we have the following result.

Theorem 5.6. *There is a universal restructurable chip with m processors and $O(\frac{\sqrt{m}F}{\sqrt{N}})$ external pins, occupying area $O(\frac{F^2m}{N}\lg^2\frac{\sqrt{mN}}{F})$, such that every N-node graph with an $(F,\sqrt{2})$-bifurcator can be assembled using multiple copies of the universal chip. Furthermore, the number of chips used in the assembly is the minimum possible.*

Proof. Consider the top $\lg N - \lg m$ levels of a fully balanced decomposition tree of G. Each of the subgraphs at level $\lg N - \lg m$ has $N/2^{\lg N - \lg m} = m$ nodes, and has a $\sqrt{2}$-bifurcator of size $O(\frac{\sqrt{m}F}{\sqrt{N}})$. By Theorem 4.7, each of these subgraphs can be realized with a single universal chip consisting of a truncated tree of meshes $T_{O(\frac{\sqrt{m}F}{\sqrt{N}}),O(\lg\frac{\sqrt{mN}}{F})}$ whose area is bounded by $O(\frac{F^2m}{N}\lg^2\frac{\sqrt{mN}}{F})$, and which has $O(\frac{\sqrt{m}F}{\sqrt{N}})$ external pin connections. To complete the assembly, the chips are wired up by making connections between pins on different chips as given by the decomposition tree. ∎

A noteworthy consequence of this result is that when $F = O(\sqrt{N})$, the restructurable chip has $O(\sqrt{m})$ pins, which is independent of the size of the network to be assembled. This is the best possible. To realize networks with larger bifurcators, the parameters of the restructurable chip depend on the size of the network assembled.

Special Cases. For graphs with special bifurcators, the same is true except that only $O(F^2)$ area is used on each chip. For type A $\sqrt{2}$-bifurcators, the number of pins needed is much lower. For example, N-node trees require only $O(\lg m)$ pins per chip (Theorem 3.9). As is the case for all planar graphs, the number of pins does not depend on the number of nodes. This is because N-node planar graphs have $\sqrt{2}$-bifurcators of size $O(\sqrt{N})$.)

CHAPTER 6

The Channel Routing Problem

While the layout problems considered in Part I provide new insights and paradigms for VLSI graph layout, they are nevertheless abstractions of problems encountered by current automatic layout systems. In this second part (Chapters 6 and 7) we shall study the widely encountered channel routing problem which forms the basis of a popular paradigm for automatic layout.

The typical routing problem is characterized by a set of rectangular modules with terminals at fixed positions along module boundaries. Labels on the terminals specify the required connections – all terminals with the same label must be electrically connected. The problem is to wire together all terminals that have the same label.

Most layout systems proceed in two phases: *placement* and *routing*. In the placement phase the modules are located at fixed positions, and the required connections are later made in the routing phase by running wires around and in between the modules. Of course, the two phases go hand-in-hand; a placement for which a complete routing is impossible is of little use. The intractability of obtaining optimal solutions in either phase demands that efficient heuristics be developed for practical use.

Introduced by Hashimoto and Stevens in 1971 [34], *channel routing* has become a very popular and successful heuristic for routing integrated circuits. As illustrated in Figure 6.1, after the modules have been placed, the chip is heuristically partitioned into a set of rectangular channels, and each channel is assigned a set of wires which are to pass through it. This effectively reduces a difficult "global" wiring problem to a set of disjoint (and presumably easier), "local" channel routing subproblems.

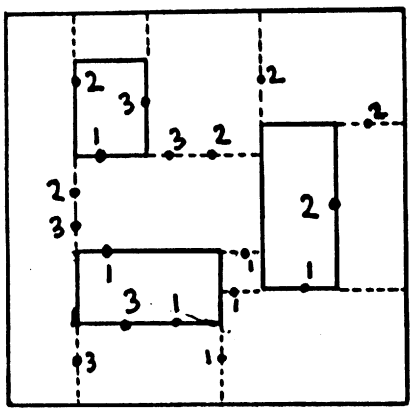


Figure 6.1: *Reducing the global wiring problem into a set of channel routing subproblems.*

The performance of the overall strategy is largely determined by the algorithm used to solve the individual channel routing subproblems. For this reason, the channel routing problem has been intensively studied for over a decade, and many heuristic algorithms have been proposed for solving the problem [1, 2, 11, 12, 18, 20, 21, 34, 35, 36, 38, 51, 60, 62, 67, 68, 81, 84]. Although many of these heuristics have proved reasonably successful in practice, there are instances (albeit theoretical) when the heuristics either produce arbitrarily bad solutions or fail to produce any solution. Chapter 7 presents a fast approximation algorithm which is guaranteed to produce a solution close to optimal. The remainder of this chapter, however, poses the problem in a formal framework and briefly reviews some of the previous work on channel routing.

6.1. Manhattan Routing Within Channels

The channel routing problem may be described as follows. A *channel* consists of a *two-layer* rectangular grid of *columns* and *tracks* (rows). *Terminals* are located on the *top* and *bottom* tracks at grid points. The number of tracks between the top and bottom tracks is the *width* of the channel. Each set of terminals to be electrically connected constitutes a *net*, and distinct nets are disjoint. A net with r terminals is called an r–point net. The width may be varied by moving the tracks vertically; however, the tracks are not allowed to slide horizontally. In other words, the columns are fixed. We also assume that there are no *trivial nets* (two-point

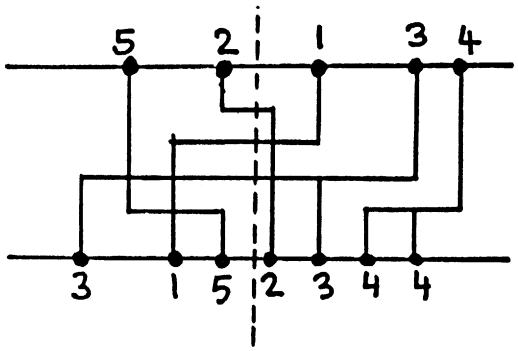


Figure 6.2: *Manhattan routing within a channel. Vertical cuts measure channel density.*

nets with both terminals in the same column).

The objective of the channel routing problem is to wire together all terminals in each net in a way which minimizes channel width. Wires may be routed on either layer, along any track between the top and bottom tracks, and along any column. There is no restriction on the number of columns at either end. Electrically disjoint wires may *cross* at grid points on different layers, but may not overlap for any distance even on different layers. A wire may change layers at a grid point, in which case no other electrically disjoint wire may pass through that grid point on either layer.

In the *Manhattan wiring* model, these constraints are satisfied by restricting all horizontal wire segments to lie on one layer, and all vertical segments to lie on the other layer. For a wire to turn a corner it has to change layers, which requires a contact cut. Clearly, distinct wires cannot share a corner since that would violate the constraint that only one wire may change layers at any point. For obvious reasons, Manhattan routing is also referred to as *layer per direction* or *reserved layer* routing. Figure 6.2 illustrates an example of Manhattan routing in a channel.

Remark. The channel routing problem described above is a simpler version of *switchbox* routing in which terminals are located on all sides of a rectangular channel. In many instances, such as when two large modules are placed next to each other, terminals lie only along two opposite sides of a channel. For this reason, and because switchbox routing problem is much more difficult, engineers have focussed attention primarily on the simpler channel routing problem.

6.2. Bounds on Channel Width

Consider a vertical cut which slices the channel in two (see Figure 6.2). Every net which has a terminal on both sides of the cut is said to be *split* by the cut. Since at least one wire must cross the vertical cut for each split net, it follows that at every point the channel must be at least as wide as the number of nets split by a vertical cut through that point. In short, channel width can be no less than *channel density*, which is defined as the maximum number of nets split by a vertical cut. For example, the channel of Figure 6.2 has density three.

Can every channel with density d be routed in $O(d)$ tracks? In practice, most channels can be routed in d plus two or three tracks. In general however, this is far from the truth. Brown and Rivest [14] gave examples of two-point net channels, with n terminals, whose density is one, but for which channel width can be no less than $\sqrt{2n}$. Since we shall employ an identical argument later, their result is rederived below.

Theorem 6.1 (Brown-Rivest). *Consider the two-point, n-net (shift-one) channel in which terminal i is located in column i on the top track, and in column $i+1$ on the bottom track. Any Manhattan routing for this channel must have width at least $\sqrt{2n} - 1$.*

Proof. Suppose that a routing of width w is given. Since the top and bottom terminals of any net lie in different columns, each wire in the routing must use a horizontal track to change columns at least once. Now, if a wire changes from column i to column j along track y $(1 < y < n)$ then either the vertical segment $(j, y-1) - (j, y)$ or the segment $(j, y) - (j, y+1)$ can not have a wire laid on it. Otherwise, as seen in Figure 6.3, two different nets will overlap at point (j, y).

In other words, whenever a wire changes columns within the channel, it must change to a *blank column*, one which has no wire in one incident vertical segment. A wire may also change columns by exiting across a side of the channel along a horizontal track.

How many wires can change columns along the first horizontal track? Since all grid points on the top track are occupied, a wire can change columns only by exiting the channel. But, since segment overlaps are prohibited, at most two wires can change columns in this way.

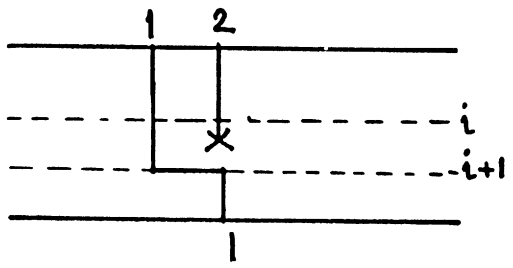


Figure 6.3: *A wire can only turn into a blank spot.*

Observe that whenever a wire exits the channel, one blank segment is created along a column.

The number of wires that can change columns on any horizontal track is bounded by the number of blank vertical segments incident to that track, plus two (for wires that exit the channel). If 2 wires change columns on the first horizontal track, this creates two empty vertical segments incident to the second track, so that 4 wires can change columns on the second track, and so on. In general, it is easy to see that the number of wires that can change columns on track y is at most $2y$ when $y \leq \lfloor w/2 \rfloor$ and at most $2(w+1-y)$ otherwise.

Summing over all horizontal tracks, the total number of wires that can change columns is consequently no greater than

$$\sum_{0 \leq y \leq \lfloor w/2 \rfloor} 2y + \sum_{\lfloor w/2 \rfloor + 1} 2(w - y + 1),$$

which is always less than $\frac{1}{2}(w+1)^2$. Finally, since every wire connecting a net has to change columns, we have

$$\tfrac{1}{2}(w+1)^2 \geq n,$$

or, $w \geq \sqrt{2n} - 1$, thus proving the result. ∎

An obvious question that arises is: Can every channel be quickly routed in minimum width? Unfortunately, the general problem is NP-complete [77], and remains NP-complete even for two-point nets [77, 78]. This might help explain why none of the current heuristics is even guaranteed to find solutions that are close to optimal.

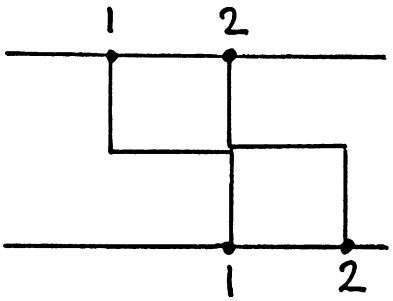


Figure 6.4: *In the knock-knee wiring model, two wires may share a corner as long as they remain on different layers.*

6.3. Bounds for Other Wiring Models

While Manhattan wiring rules ease the task of mask fabrication, less restrictive wiring models are also occasionally used. For example, some manufacturers may permit wires to change direction within a layer, or may allow non-rectilinear wiring. Similarly, other manufacturers may provide more than two layers of interconnect. It is important to consider how variations in the wiring rules affect the routability of channels.

In the *knock-knee* wiring model, wires are allowed to change direction within a layer, and wires on different layers may share a grid point as long as neither one changes layers at that point. The routing illustrated in Figure 6.4 is permissible in the knock-knee model, but not in the Manhattan model. Channel density of course remains a lower bound on channel width. Rivest, Baratz, and Miller [67] investigated the channel routing problem under the knock-knee wiring model. They showed that every two-point net channel with density d can be routed in width $2d - 1$, independent of the number of nets. In view of Theorem 6.1, this implies that the knock-knee wiring model is more powerful than the Manhattan wiring model. Leighton [43] gave a construction for channels with density d which cannot be routed in less than $2d - 1$ tracks, so that the Rivest, Baratz, and Miller algorithm is optimal in the worst case. For multi-point net channels, their algorithm guarantees a routing of width at most $4d - 1$.

Preparata and Lipski [62] consider the channel routing problem under the knock-knee model, but with three layers of interconnect instead of only two. With this extra layer, they guarantee that every two-point net channel with density d can be optimally routed using exactly d tracks. Moreover, this routing can be accomplished quickly. For multi-point net channels,

their algorithm guarantees a routing of width no greater than $2d$.

The problem of "river routing," which is single-layer channel routing, has also received considerable attention [21, 23, 51, 81]. Under the single-layer restriction, there exist fast algorithms for channel routing. In particular, Leiserson and Pinter [51] also examine the problem of placing movable modules along the top and bottom tracks so as to minimize the horizontal "spread" and width of a channel. Pinter [61] also studies the problem of river routing within polygonal regions with terminals along the perimeter of the polygon. Finally, LaPaugh [39] studies the problem of wiring terminals placed along the perimeter of a rectangular module where the wires are on two layers, but are restricted to lie outside the module.

CHAPTER 7

An Approximation Algorithm for Manhattan Routing

Brown and Rivest's lower bound for the one-shift example indicates that channel density is not the only fundamental limitation on channel width. Motivated by their argument, Section 7.1 introduces the concept of *channel flux,* which provides another fundamental limitation on channel width. Unlike density, flux is a local phenomenon and captures the amount of "congestion" within a channel.

Flux and density together completely characterize the difficulty of Manhattan routing. Section 7.2 presents a linear-time algorithm which routes every two-point net channel in width proportional to its flux and density. This settles a conjecture of Brown and Rivest that their lower bounds are tight to within a constant factor. Moreover, in practice, flux is extremely small so that the algorithm for two-point nets uses no more than a constant number of tracks more than density. Section 7.3 analyzes the running time of the algorithm, while Section 7.4 extends the algorithm to multi-point net channels.

7.1. Channel Flux

While channel density provides a fundamental limitation on channel width, it fails to capture the local congestion inside a channel. For example, while the one-shift channel has low density, the channel width must nevertheless be large to overcome congestion within the channel. This congestion arises from the fact that every column in the top track contains a terminal whose mate lies in a different column along the bottom track. Since wires in adjacent columns may not both "turn right" along a common track without colliding, many horizontal

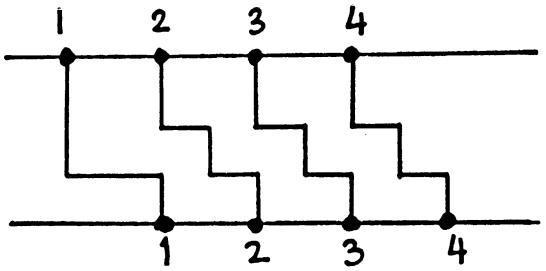


Figure 7.1: *The modified one-shift channel can be routed in width two.*

tracks are needed to complete the wiring.

In striking contrast, consider modifying the one-shift channel by making every alternate column blank. While this channel is globally similar to the one-shift, it can be routed using only two horizontal tracks as shown in Figure 7.1. This channel is not locally congested because the empty columns enable many wires to simultaneously turn along the same horizontal track.

We now introduce the concept of channel flux to measure congestion. Although there are a variety of ways to measure congestion, we choose here a simple definition which permits a clean analysis. In Section 7.4 we vary the definition slightly to obtain better bounds.

Suppose that instead of making vertical cuts in the channel, we instead make a horizontal cut which isolates a set of contiguous columns from one track. Observe that we can vary the *size* of a cut (measured by the number of columns within the cut) as well as its position. As before, we say that a net is split by a horizontal cut if it contains terminals both within the cut and outside. For any given position of a cut we can measure the number of distinct nets split by the cut.

Intuition suggests that the greater the number of distinct nets split by a cut, the greater the congestion is within the cut. Moreover, the larger the size of a congested cut, the larger the channel width, because if the region of local congestion is very large, then so is the overall global congestion of the channel. This intuition is formalized below. As mentioned earlier, we restrict attention only to channels which do not contain any trivial nets.

Definition. The flux of a channel is the largest integer f for which there exists a horizontal cut of size $2f^2$ which splits at least $2f^2 - f$ nontrivial nets.

For example, the one-shift channel has flux $\Omega(\sqrt{n})$ because a horizontal cut of size n which isolates the top track splits n nets. Similarly, the modified one-shift of Figure 7.1 has flux one. For the flux to equal two there must be a cut of size 8 which splits at least 6 nets, but since every alternate column in either track is blank no such cut exists.

Using Brown and Rivest's argument for the one-shift channel, we next show that flux is indeed a lower bound on channel width.

Theorem 7.1. *Every channel with density d and flux f requires channel width at least $max(d, f)$.*

Proof. Find a horizontal cut of the channel which spans $2f^2$ columns and splits at least $2f^2 - f$ nontrivial nets. For each nontrivial net split by the cut, choose any two terminals from different columns that lie on opposite sides of the cut.

Consider the channel formed by the set of chosen terminals, i.e., assume that all columns which do not contain a chosen terminal are blank. This new channel consists of at least $2f^2 - f$ nontrivial two-point nets. Moreover, at most f of the $2f^2$ columns spanned by the original cut may be empty. By the same argument used to prove Theorem 6.1, no more than $f + 2$ of the nontrivial nets can be routed into the correct column on the first track: f into empty columns and one out each side of the cut. After the first track, there are at most $f + 2$ empty columns, the extra two having possibly been created by wires exiting across the side of the cut in the first track. Thus, at most $f + 4$ nontrivial nets can be routed into the correct column on the second track. In general, at most $f + 2i$ nontrivial nets can be routed into the correct column on the ith track.

Let w be the minimum width for which a wiring exists. By the preceding argument, the total number of nets that can change columns anywhere in the channel is no greater than $\sum_{i=1}^{w}(f + 2i) = wf + w(w + 1)$. But since at least $2f^2 - f$ nontrivial nets must eventually be routed, it follows that $wf + w(w + 1) \geq 2f^2 - f$, or $w \geq f$. Thus the original problem requires a channel of width at least f. Finally, since the density d also is a lower bound on channel width, the Theorem follows. ∎

Flux is negligibly small in practice, and for all purposes never exceeds three or four. One explanation for this is that terminals are movable; it is good engineering practice to leave enough empty space so that if the channel is congested, then the terminals can be moved slightly to allow a better wiring. Moreover, many columns contain less than two terminals, and a large fraction of nets contain terminals that are close together on the same side of the channel. These are precisely the conditions that make flux small. Finally, unlike density, flux is a local phenomenon and is less likely to grow with the size of a channel or the total number of nets. As an example, Deutsch's "difficult problem" [20] has 72 nets, 174 columns and density 19, but the flux is just 3.

7.2. An Approximation Algorithm for Top-to-bottom Nets

In this section we present a linear-time approximation algorithm for routing channels with two-point nets. It is assumed that each net is nontrivial and has exactly two terminals, one each on the top and bottom tracks. The next section extends this algorithm to general multi-point net channels.

The input to the algorithm may be presented in one of two ways. It might consist of a list of columns, each entry describing the terminals in the top and bottom tracks in that column (possibly none). A more compact representation is a list of nets, each net itself being a list describing the positions of terminals in that net. The algorithm outputs a detailed wiring of the channel. The length of the output is proportional to the total wire area used to route the channel.

The running time of the algorithm will be measured as a function of the shortest possible output. This is more reasonable than measuring time as a function of the length of the input because the length of the output is always at least as large as the length of the input. In fact, the output is generally much longer than the length of the input.

With this convention for measuring the running time, it is straightforward to see that either input representation described above may be converted to the other in linear time. Moreover, if the total number of columns in the channel is c, and if the channel has flux f and density d,

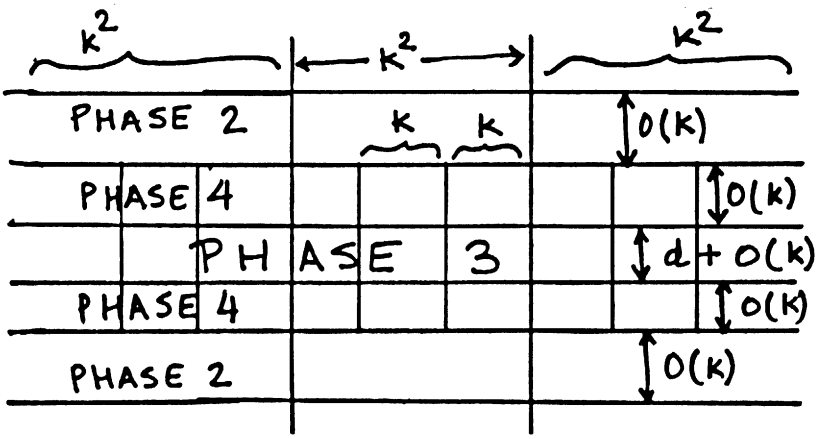


Figure 7.2: *The regions routed in each phase.*

the minimum area required to route the channel is at least $\Omega(c(d+f))$. The running time of our algorithm is bounded above by $O(c(d+f))$, so that it is a linear-time algorithm.

The algorithm proceeds in four phases. Figure 7.2 sketches the regions routed within the different phases. The first two phases distribute empty columns uniformly across the channel, thereby dividing the channel into blocks each containing a small number of empty columns. This creates a new channel routing problem with possibly higher density, but with reduced flux. The third phase, the heart of the algorithm, routes the correct number of wires between blocks, without worrying about which columns within a block these wires lie in. Finally, the fourth phase routes the wires within each block into the correct column. The empty columns within each block allow a block to be wired independently of other blocks, so that every block is wired simultaneously on the same horizontal tracks.

The Top-to-bottom Channel Routing Algorithm

Phase 1: *Partition the channel into groups.*

Find the least integer k such that the channel can be partitioned into groups of k^2 consecutive columns, each group containing at least $3k$ *empty* grid points in both the top and bottom tracks. (An empty grid point is one at which no terminal is placed.) This can be accomplished by trying successive values for k (starting with $1, 2, 3, \ldots$) until the constraint is satisfied.

The definition of flux guarantees that k does not exceed $6(f+1)$. For, suppose that $k = 6(f+1)$ does not satisfy the constraint. Then some group of $36(f+1)^2$ columns contains less than $18(f+1)$ empty grid points on one track. If we partition this group into 18 blocks, each of size $2(f+1)^2$, then one of them must have less than $(f+1)$ empty grid points on one track. But this means that the flux is at least $f+1$—a contradiction.

Phase 2: *Distribute empty points uniformly.*

Divide each group of k^2 columns into k blocks of k columns each. Route wires from the first 3 points (if non-empty) on the top track of each block into columns that are empty on the top track. Since each group has at least $3k$ empty points on the top track, this routing can be easily accomplished using no more than $3k$ horizontal tracks. Repeat the same for the bottom track, so that the original channel is reduced to one which can be partitioned into blocks of size k such that the leftmost 3 columns of each block are empty. The significance of having 3 empty points in each block will be made clear in the detailed interblock routing of Phase 3. Observe that although the density of the resulting channel may be greater than the density d of the original channel, it can be no greater than $d+6k$.

Phase 3: *Route wires between blocks.*

This phase routes the correct number of wires between different blocks: if x nets have one terminal in the top track of block A and the second terminal in the bottom track of block B, then route x wires from the top track of block A to the bottom track of block B. It is not necessary that the wires be routed into the correct columns, but only that the correct number are routed between blocks. This phase is relatively complicated and forms the core of the overall strategy. At most $d+3k$ horizontal tracks are used. Details are descibed later in this section.

Phase 4: *Route wires within each block.*

At the end of Phase 3, all that remains is the problem of routing within each block. Each block has at most k nets and at least three empty columns. The location of each net is determined in Phases 2 and 3. Each net may be routed entirely within its block using,

for example, the algorithm of Kawamoto and Kajitani [36], which uses no more than $\frac{3}{2}k$ horizontal tracks. Moreover, every block can be simultaneously routed on the same horizontal tracks, so that this phase uses at most $\frac{3}{2}k$ tracks.

Specifically, the nets are routed one per track: the order of routing is determined by constraints caused by a top terminal for one net lying above a bottom terminal of another net. When a cycle of constraints occurs, one net of the involved cycle is temporarily routed into an empty column to eliminate one constraint, and routed to its other terminal after the other nets in the cycle have been routed. Two tracks are used to route the last net in each such cycle of constraints. ∎

Next, we present the detailed routing of Phase 3. Each net is first classified into one of three categories. If both terminals of a net lie in the same block then the net is said to be a *vertical net.* Otherwise, if the terminals are in different blocks and if the top terminal is to the left of the bottom terminal, then the net is called a *falling net.* Finally, if the terminals are in different blocks and if the top terminal is to the right of the bottom terminal, then the net is called a *rising net.*

The interblock routing procedure performs a left to right scan across the channel, routing each block completely before proceeding to the next block. Between any two consecutive blocks, the rising nets run along the upper horizontal tracks, the falling nets run along the lower tracks, and every empty horizontal track lies between the tracks containing the rising and falling nets.

In some cases a wire must be routed through previously routed blocks on the left before it can proceed to the right. This requires that space be maintained for wires to *backtrack* (pun intended) when necessary. By keeping the empty tracks between the rising and falling nets within each block, we can coalesce the empty tracks in consecutive blocks to form the *pyramid* shown in Figure 7.3. Pyramids are crucial to backtracking; as an example, Figure 7.3 illustrates how a "blocked" wire can backtrack through the pyramid on its way right. After a wire backtracks through the pyramid, the pyramid is updated as shown.

The following outline describes the interblock routing procedure in detail. Each of the steps is illustrated in Figure 7.4. Figure 7.4a shows the initial situation just before a new

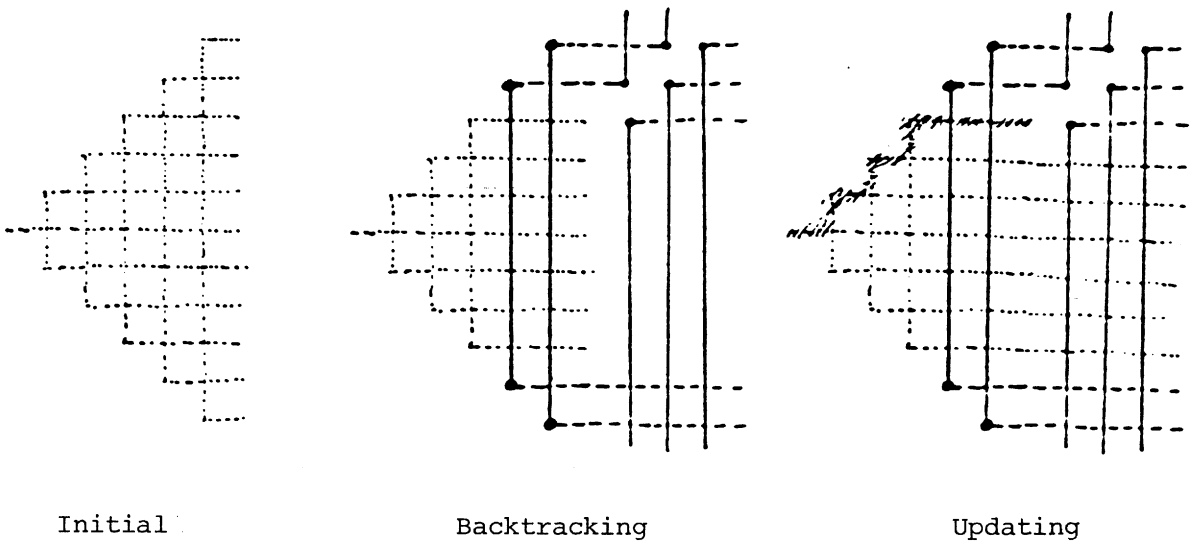


Figure 7.3: *Maintaining a pyramid for backtracking.*

block is entered. The arrows on the tracks indicate whether the net is a rising/falling net that terminates within the block, or whether the net terminates in a different block on the right. The empty tracks are contained within the pyramid shown. In the case when the block to be routed is the leftmost block, the pyramid contains all horizontal tracks and extends to the left of the channel.

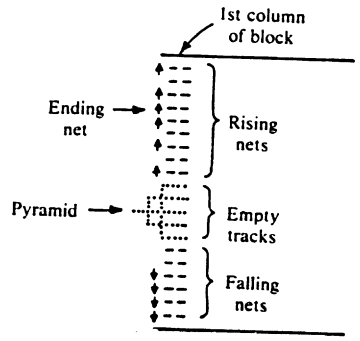
The Interblock Routing Procedure

Step 1: Ending nets.

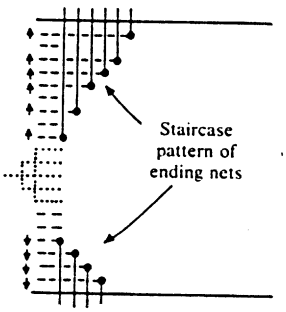
Nets with one terminal in a block on the left and the other in the current block are called *ending nets.* By moving the lowest ending rising net upward and the highest ending falling net downward wherever possible, the ending nets can be routed in a *staircase pattern* as shown in Figure 7.4b.

Step 2: Continuing nets.

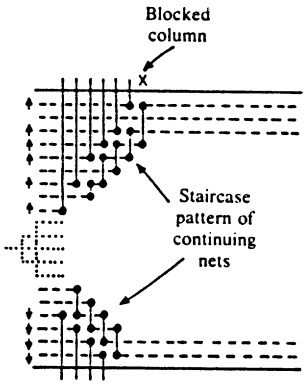
Nets with one terminal in a block on the left and the other terminal in a block to the right of the current block are called *continuing nets.* Route the rising (falling) continuing nets



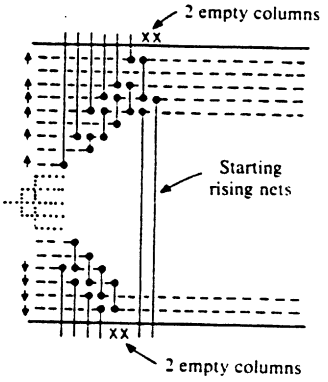
7.4a *Before routing starts (ending nets are marked with arrows).*



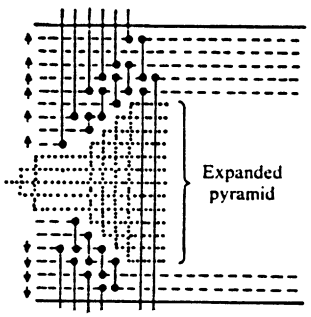
7.4b *Routing ending nets in Step 1.*



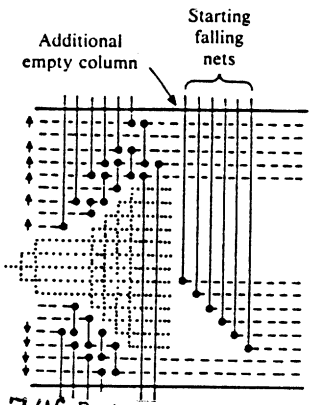
7.4c *Routing continuing nets in Step 2.*



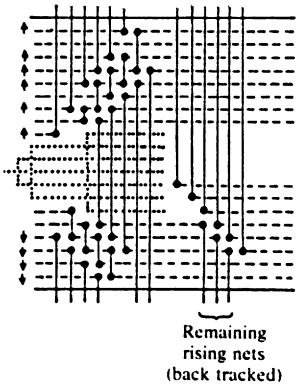
7.4d *Balancing columns in Step 3.*



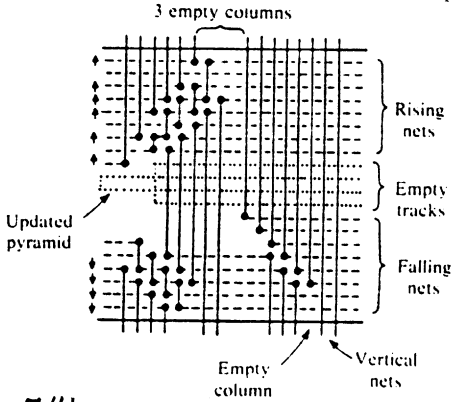
7.4e *Updating the pyramid structure.*



7.4f *Routing starting falling nets in Step 4.*



7.4g *Routing remaining starting rising nets in Step 4.*



7.4h *Completed routing of block and pyramid structure.*

through the block by shifting them up to higher (lower) tracks in a staircase pattern that fits the staircase pattern of the ending nets.

As shown in Figure 7.4c, the staircase pattern of the continuing nets blocks one grid point in the top track as well as in the bottom track (unless the block has no ending nets). In other words, no net can begin at the grid points shown. However, remember that Phase 2 provides at least 3 empty grid points on either track in each block. Since we are free to place these empty grid points in any position, we still have at least two empty points remaining on either track.

Step 3: Balancing.

Suppose the number of ending rising nets is greater than the number of ending falling nets. Balance the difference by routing some *starting* rising nets (those which originate in the block) as shown in Figure 7.4d. In case there are more ending falling nets than ending rising nets, follow a symmetrically opposite procedure.

In order to ensure that every empty column remains between the rising and falling nets it may be necessary to force one more empty grid point on the bottom track. Similarly, one grid point in the top track is forced to be empty because it is blocked by the rightmost starting rising net. At the end of this step, observe that the pyramid may be updated as shown in Figure 7.4e.

Step 4: Starting nets.

Suppose again that the number of ending rising nets is greater than the number of ending falling nets. After balancing the columns in Step 3, route all the starting falling nets as shown in Figure 7.4f. Observe that one more grid point on the bottom track is blocked, and therefore must be empty. Follow a symmetric procedure in the opposite case.

Step 5: Remaining nets.

At this stage either starting rising nets or starting falling nets remain to be wired. Suppose that some starting rising nets remain. Route these nets as shown in Figure 7.4g, making

use of the pyramid to backtrack whenever necessary. In case the number of remaining starting nets equals the number of starting falling nets routed in Step 4, then route the last starting rising net using the empty column from Step 3.

Step 6: Vertical nets.

Route the vertical nets in the natural way as shown in Figure 7.4h. Note that no extra empty points are required. ∎

Figure 7.4h shows the complete routing for the block, as well as the updated pyramid structure. Observe that the initial conditions are satisfied for routing the next block on the right. Furthermore, note that no more than 3 points on any track are required to be empty, so that Phase 2 of the main algorithm distributes sufficiently many empty grid points throughout the channel.

Since every ending net is routed before every starting net, the total number of horizontal tracks used is no greater than $d + 6k$, the density of the resulting channel at the end of Phase 2. Consequently, the number of horizontal tracks used by the main algorithm is at most $d + 15k = d + O(f)$.

7.3. Running Time Analysis

To analyze the running time of the algorithm we shall calculate the running time of each phase separately. Suppose that a channel has c columns, density d, and flux f. Then, as shown earlier, $\Omega(c(d + f))$ is a lower bound on the minimum area needed to wire the channel. As shown below, this is also an upper bound on the running time of the algorithm.

The first phase computes the smallest integer k for which the channel can be divided into groups of k^2 columns each such that every group has at least $3k$ empty grid points in both the top and bottom tracks. The value of k is computed by successively trying every integer (starting with $1, 2, \ldots$) until the condition is satisfied. For any possible value i, the size of each group is i^2 and there are c/i^2 groups in all. The required condition can easily be checked for each group in time $O(i^2)$ so that the total time is $O(c)$. The total time for Phase 1 is therefore

no more than $O(ck)$. But, since $k \leq 6(f+1)$, this is no greater than $O(cf)$.

In the second phase, empty columns are evenly distributed among the different blocks within each group. Each wire runs along one horizontal track so that the time is no more than the total length of wire laid out. Since no more than $3k$ tracks are used, the total wire length does not exceed $O(ck) = O(cf)$.

Phase 3 is slightly more complicated to analyze. As long as wires do not change direction, the time to lay them out is never more than the length of wire laid. However, whenever a wire must turn a corner or backtrack, the time requirements can potentially increase. A priori, it seems that maintaining the pyramid structure is time consuming; furthermore, the time to update the pyramid each time can be significantly large.

Fortunately, however, the pyramid is only an aid in understanding why the algorithm works correctly; there is no need to explicitly maintain the pyramid at all. Any time a wire must backtrack, all we really have to do is to simultaneously backtrack along the uppermost and lowermost empty tracks until a column, which is empty between the two tracks, is encountered. In fact, following this procedure gives the same routing as with the pyramid. It is relatively straightforward to argue that, with the modified strategy, the total time spent in Phase 3 is no more than $O(c(d+k)) = O(c(d+f))$.

Finally, Phase 4 requires no more than $O(cf)$ time. Each channel routing subproblem of size k can be routed in time $O(k)$ using $O(k)$ tracks. The total time over all subproblems is therefore $O(ck) = O(cf)$.

Summing up, we conclude that the running time of the algorithm is dominated by Phase 3, and does not exceed $O(c(d+f))$, which is linear in the area of the minimum area routing.

7.4. The Channel Routing Algorithm

The algorithm of Section 7.3 routed two-point nets which had one terminal in the top track and the other in the bottom track. This section extends the algorithm to multi-point nets. As before, the algorithm is divided into four phases. Once again, we assume that the channel has no trivial two-point nets, and has density d and flux f.

The General Channel Routing Algorithm

Phase 1: *Partition the channel into groups.*

Find the least integer k for which the channel can be partitioned into groups of k^2 consecutive columns, such that a horizontal cut of size k^2 which isolates either the top or bottom track of any group splits at most $k^2 - 3k$ nets. The value of k may be found by trying successive values (starting with $1, 2, \ldots$) until the required condition is satisfied.

As before, it may be verified that the value of k is bounded by $O(f)$, where f is the flux of the channel.

Phase 2: *Distribute empty points uniformly.*

For each track within a group count the number p of empty points. If $p \geq 3k$, then distribute the empty points as before. If $p < 3k$ then there are at least $3k - p$ duplicate terminals within the group and on the same track. Choose any $3k - p$ duplicated terminals and connect these to other terminals from the same net using one horizontal track for each such net.

Next, pick one representative terminal for each duplicated net connected above. The duplicate terminals, being already connected, may be ignored so that each group now has at least $3k$ empty points on either track. Distribute these empty points uniformly as before so that each block of size k has at least 3 empty points. Observe that the total number of horizontal tracks used is $O(k) = O(f)$.

Phase 3: *Route wires between blocks.*

Although the basic strategy is the same as before, the major difference is that a net may have representative terminals in many different blocks. (Within a block choose any one representative terminal, if it exists, on each track.) The modified interblock routing procedure is described later in this section, and uses no more than $2d + O(f)$ tracks.

Phase 4: *Route wires within each block.*

This phase remains essentially unchanged. The only difference is that within each block the representative terminal of any net should be connected to all its duplicates. Although the choice of representatives determines the number of horizontal tracks used, this never exceeds $O(f)$. ∎

Next, we present the detailed interblock routing of Phase 3. Each net is first classified into one of four categories. A net whose leftmost terminal on the top track lies in the same block as its leftmost terminal on the bottom track is called a *vertical net.* If the leftmost top terminal (i.e., on the top track) of a net falls in a block to the left of the block containing the leftmost bottom terminal (i.e., on the bottom track) of the net then the net is said to be a *falling net.* Conversely, if the block containing the leftmost top terminal of a net is to the right of the block containing the leftmost bottom terminal of the net then the net is called a *rising net.* Finally, if all terminals of a net lie on the same track (either top or bottom) then the net is called a *same-side net.*

In addition, each net is divided into a rising portion and a falling portion. The *rising portion* of a net links the block containing the leftmost terminal to the blocks containing terminals in the top track of the channel. The *falling portion* of a net links the block containing the leftmost terminal to the blocks containing terminals in the bottom track of the channel. The interblock routing procedure connects the top terminals with the bottom terminals using a single connection emerging from the block containing the leftmost terminal. Figure 7.5 illustrates the rising and falling portions of a net and where the connection is made. Observe that not every net is required to have both a rising as well as a falling portion.

As before, the procedure ensures that between consecutive blocks tracks containing rising portions of nets are above every empty track and that every empty track is above the tracks containing falling portions of nets. This allows us to once again maintain a pyramid structure for backtracking.

The routing proceeds block-by-block from left to right in the middle $2d + O(f)$ tracks of the channel. Each block is routed in seven steps described below. The steps are numbered to

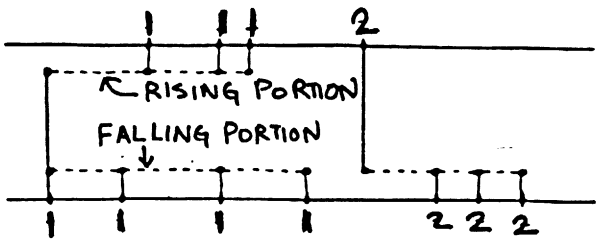


Figure 7.5: *Dividing nets into rising and falling portions. Some nets may have only a falling/rising portion.*

coincide with the algorithm of Section 7.3. Figure 7.6 shows a complete routing of a block.

The Interblock Routing Procedure

Step 1: Ending nets.

Route the ending nets (those which do not have a terminal to the right of the current block) in staircase patterns at the left end of the block.

Step 2: Continuing nets.

Route the continuing nets (those with a terminal in a block to the right of the current block) in staircase patterns nestled against those generated in Step 1. If a continuing net also has a representative terminal in the current block, then place the terminal to the right of the staircase and make a connection as shown in Figure 7.6.

Step 2.5: Starting same-side nets.

Route every same-side net whose leftmost terminal lies in the current block in a staircase fashion, bringing wires from the bottom (top) track to the lowest (highest) available empty track.

Step 3: Balancing.

If more columns have been used at the top of the channel than at the bottom, make up the difference by routing the rising portions of some starting rising nets. If the opposite case holds, follow the symmetric procedure.

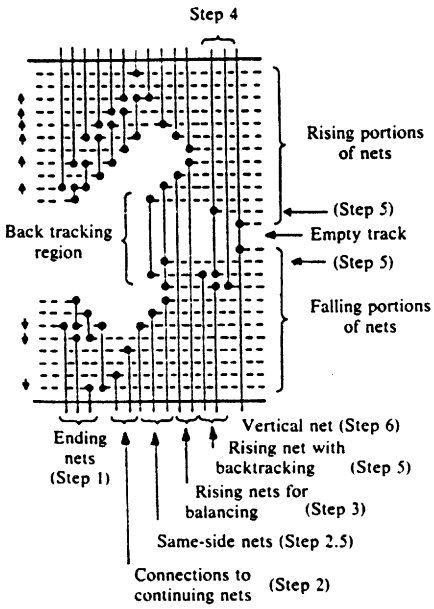


Figure 7.6: *Complete Phase 3 routing within a block.*

Step 4: Starting nets.

Route the falling portions of starting falling nets (or the rising portions of starting rising nets depending on which was in excess in Step 3).

Step 5: Remaining nets.

Route the remaining rising portions of starting rising nets (or the falling portions of remaining starting falling nets), using the pyramid for backtracking if necessary. Furthermore, route the falling portions of starting rising nets and the rising portions of starting falling nets in the straightforward way using empty tracks.

Step 6: Vertical nets.

Route the vertical nets in empty columns as before. ∎

Since the rising and falling portions of each net are effectively separated, the interblock routing procedure requires no more than $2d + O(f)$ horizontal tracks. As before, it can be argued that the overall algorithm runs in linear time, and routes a channel of density d and flux f in width $2d + O(f)$. To summarize, we have shown the following.

Theorem 7.2. *Every multi-point net channel with density d and flux f can be routed in width no greater than $2d + O(f)$ in linear time.*

Furthermore, if every net is a same-side net or only has a rising portion or a falling portion (but not both) then the number of tracks used is $d + O(f)$. In particular, for two-point net channels we have the following result.

Theorem 7.3. *Every two-point net channel with density d and flux f can be routed in width $d + O(f)$ in linear time.*

CHAPTER 8

Conclusions, Extensions and Open Problems

This thesis was motivated by the need for a clearer understanding of various issues in circuit layout. The techniques developed provide new insights and approaches for VLSI layout. Although the results in their present form are theoretical in nature, it is likely that some of the techniques can be adapted for use in practice.

The two parts of the thesis share a common underlying methodology. First, the critical properties that determine the quality of a layout are identified. In the next step, these properties are effectively exploited to obtain good layouts. Thus, for example, the minimum bifurcator of a graph gives a lower bound on layout area, and good layouts can be found quickly if a decomposition is available. Similarly, flux and density give lower bounds on channel width; they also provide the basis for a fast, provably good channel routing algorithm.

The strategy for VLSI graph layout in Part I provides a simple and uniform technique for solving a variety of layout problems efficiently. The unified framework is suitable for custom layout, and at the same time is efficient with regard to area, delay, and fault-tolerance. The tree of meshes, in particular, emerges as a surprisingly versatile and powerful network for circuit layout. A priori, there is no reason to believe that such diverse concerns can be handled simultaneously in a compatible manner, let alone within a common framework.

Approaching the channel routing problem from a theoretical viewpoint, Part II characterizes the properties that make Manhattan routing difficult. These properties then form the basis of a new, linear-time approximation algorithm that is guaranteed to always find a near-optimal routing. In contrast, although the problem had been studied intensively for over a decade from an engineering viewpoint, all previous heuristics could be made to perform ar-

bitrarily poorly on certain inputs.

These results notwithstanding, a number of problems are left unresolved in this thesis. The following sections mention some of the more important open problems, and also sketch extensions to the results reported. More details on specific problems may be found in [7].

8.1. Problems in Graph Layout

The divide-and-conquer strategy based on graph bifurcators has also been successfully applied by Leighton and Rosenberg [46] to the study of three-dimensional VLSI circuit layout. In addition, the techniques and results are also applicable to graph and data-structure embeddings, and also provide bounds on one- and two- dimensional bandwidth minimization.

Question 1. How much area is required to lay out an N-node planar graph? The best universal upper bound is $O(N \lg^2 N)$ [49, 83] while the best existential lower bound (for the tree of meshes) is $\Omega(N \lg N)$ [40, 41].

Question 2. Is there a polynomial time algorithm for laying out trees with edges not much longer than the minimax edge length? The best tree layout algorithm (Chapter 3) produces layouts with edges of length $\Theta(\sqrt{N}/\lg N)$. Although this is optimal for some trees, it is way off for others.

Question 3. Is there a better way to realize a network in an environment that contains defective processors? The results of Chapter 5 guarantee that any graph can be realized using the good processors provided the "channels" have width $\Omega(\frac{F}{\sqrt{N}} \lg \frac{N}{F})$ in a regular layout. Although this bound is optimal for some networks [7], it is not known to be optimal for simpler networks such as two-dimensional arrays.

Question 4. Is there a provably good heuristic for graph bisection? Any such heuristic could be used to find efficient decomposition trees and bifurcators, which, in turn, could be used to produce good layouts [7, 42]. There are many heuristics which do very well in practice [13, 17, 24, 37, 66, 71]. Analyzing these or developing new heuristics along similar lines is likely to have an impact on VLSI layout.

Question 5. Can the framework be extended to deal with processors of variable size and shape? While it is relatively easy to deal with equal-size processors, any progress toward the general problem would be very interesting.

8.2. Problems in Channel Routing

While the algorithms of Chapter 7 are fast and are guaranteed to produce near-optimal routings, the analysis of the constant factors leaves much room for improvement. In particular, the actual number of tracks used by the algorithm may be much less than the upper bounds indicate.

For example, if the empty grid points are already uniformly distributed to begin with, then Phase 2 needs to perform only a minor redistribution of empty points. Consequently, the upper bound of $6k \leq 36(f+1)$ tracks to redistribute empty points, is a gross overestimate. On the other hand, if the empty points are not uniformly distributed, but are bunched together in groups, then the actual lower bound is underestimated by flux. To see this, observe that along a horizontal track at most two wires can turn into a blank column inside a bunch of empty columns. However, the lower bound argument for flux does not take the density/frequency of blank points into consideration. Since flux underestimates the true bound in this case, once again, we see that the performance of the algorithm is much better in relation to the actual value than what the bounds indicate.

In addition, it is possible to obtain tighter bounds more directly, by redefining the notion of flux. Rather than making horizontal cuts in the channel, it is better to employ the argument to "windows," i.e., groups of contiguous columns. This is the idea adopted by Brown and Rivest in their lower bound arguments. The advantage of this lower bound strategy is that if many wires are forced to change columns within the window, then the lower bound is very high. On the other hand, if many wires exit across the sides of the window, then the width must again be large since at most two wires can exit the window along a horizontal track. Is it possible to redefine the notion of flux to capture some of these bounds? What is the best definition for flux? Finally, do multi-point nets really require $2d + O(f)$ tracks, or will $d + O(f)$ suffice?

At a more general level, it would be interesting to investigate the applicability of flux to other wiring problems, such as, for example, the switchbox problem. In conclusion, we mention that Baker, Bhatt, and Leighton [3] extend the results of the Manhattan wiring model to the case where contact cuts are larger than wires. In this case it turns out that flux is never more than a constant, so that density is the sole limiting factor on channel width.

Bibliography

[1] S. Alford, *DYCHAR: A Channel Router which uses dynamic channel assignment,* S.B. thesis, Dept. of Electrical Engineering and Computer Science, M. I. T., (1980).

[2] T. Asano, T. Kitahashi, and K. Tanaka, "On a method of realizing minimum width wiring," *Electronics and Communications in Japan* Vol. **J59-A**, 2 (1976).

[3] B. S. Baker, S. N. Bhatt, and F. T. Leighton, "An approximation algorithm for Manhattan routing," *Fifteenth Annual Symposium on Theory of Computing* (1983).

[4] B. S. Baker and R. Y. Pinter, "An algorithm for the optimal placement and routing of a circuit within a ring of pads," *Twenty fourth Annual IEEE Symposium on Foundations of Computer Science* (1983).

[5] J. Bentley and H. T. Kung, "A tree machine for searching problems," *Proceedings of the 1979 International Conference on Parallel Processing, IEEE* (1979).

[6] S. N. Bhatt and S. Cosmadakis, "The complexity of minimizing wire lengths in VLSI Layouts," unpublished manuscript, M.I.T., (1982).

[7] S. N. Bhatt and F. T. Leighton, "A framework for solving VLSI graph layout problems," *JCSS* (to appear).

[8] S. N. Bhatt and C. E. Leiserson, "Minimizing the longest edge in a VLSI layout," M.I.T. VLSI Memo 82-86, (1982).

[9] S. N. Bhatt and C. E. Leiserson, "How to assemble tree machines," *Fourteenth Annual ACM Symposium on Theory of Computing* (1982).

[10] G. Bilardi, M. Pracchi, and F. Preparata, "A critique and appraisal of VLSI models of computation," *Proceedings CMU Conference on VLSI Systems and Computations* (1981).

[11] T. Bolognesi, *A Channel Routing Algorithm Bounding Channel Width and Maximum Wire Length,* M.S. thesis, (1982).

[12] T. Bolognesi and D. Brown, "A channel routing algorithm with bounded wire length," unpublished manuscript, (1982).

[13] M. A. Breuer, "Min-cut placement," *Journal of Design Automation and Fault Tolerant Computing* Vol. **1**, No. 4, (October 1977), 343-362.

[14] D. J. Brown and R. L. Rivest, "New lower bounds on channel width," *Proceedings CMU Conference on VLSI Systems and Computations* (1981).

[15] S. Browning, *The Tree Machine: A Highly Concurrent Computing Environment,* Ph.D. thesis, Dept. of Computer Science, California Institute of Technology, (1980).

[16] S. Browning, "private communication," (April, 1981).

[17] T. Bui, *On Bisecting Random Graphs,* S.M. thesis, Dept. of Electrical Engineering and Computer Science, (1983). Also appears as M.I.T. LCS Technical Report 287.

[18] M. Burtstein and R. Pelavin, "Hierarchical channel router," *IBM Research Report* Vol. **RC 9715 (42907)**, (1982).

[19] P. R. Cappello and K. Steiglitz, "Area-efficient VLSI structures for multiplying at clock rate," Technical Report 289, Department of EECS, Princeton University, (1981).

[20] D. N. Deutsch, "A 'Dogleg' channel router," *Proceedings 13th. IEEE Design Automation Conference* (1976).

[21] D. Dolev, K. Karplus, A. Siegel, A. Strong, and J. Ullman, "Optimal wiring between rectangles," *Proceedings 13th ACM Symposium on Theory of Computing* (1981).

[22] D. Dolev, F. T. Leighton, and H. Trickey, "Planar embeddings of planar graphs," M.I.T. LCS Technical Memo 237, (1983).

[23] D. Dolev and A. Siegel, "The separation for general single-layer wiring barriers," *Proceedings CMU Conference on VLSI Systems and Computations* (1981).

[24] C. M. Fiduccia and R. M. Mattheyses, "An almost linear algorithm for partitioning networks," unpublished manuscript, (1982).

[25] M. Foster and H. T. Kung, "Recognize regular languages with programmable building-blocks," *VLSI 81,* J. Gray, ed., Academic Press, New York, (1981).

[26] R. Floyd and J. Ullman, "The compilation of regular expressions into integrated circuits," *Twenty-First Annual IEEE Symposium on Foundations of Computer Science* (1980).

[27] H. Fuchs, J. Poulton, A. Paeth, and A. Bell, "Developing pixel-planes, a smart memory-based raster graphics system," *Proceedings, MIT Conference on Advanced Research in VLSI* P. Penfield, ed.,, (1982).

[28] O. Gabber and Z. Galil, "Explicit constructions of linear size superconcentrators," *Proceedings 20th Annual IEEE Symposium on Foundations of Computer Science* (1979), 364-370.

[29] M. R. Garey and D. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness,* Freeman and Company, San Francisco, (1979).

[30] M. R. Garey and D. S. Johnson, "Crossing number is NP-complete," unpublished manuscript, (1982).

[31] J. R. Gilbert, *Graph Separator Theorems and Sparse Gaussian Elimination,* Ph.D. thesis, Dept. of Computer Science, Stanford University, (1980).

[32] C. Goldberg and D. West, "Bisection of of circle colorings," unpublished manuscript, (1982).

[33] J. Greene and A. El Gamal, "Area and delay penalties in restructurable wafer-scale arrays," *Third Caltech Conference on VLSI* R. Bryant, ed., Computer Science Press, (1983).

[34] A. Hashimoto and J. Stevens, "Wire routing by optimizing channel assignment within large apertures," *Proceedings 8th. IEEE Design Automation Workshop* (1971).

[35] D. Hightower, "The interconnection problem - a tutorial," *Computer* Vol. 7, 4 (1974).

[36] T. Kawamoto and Y. Kajitani, "The minimum width routing of a 2-row 2-layer polycell-layout," *Proceedings 16th. IEEE Design Automation Conference* (1979).

[37] B. W. Kernighan and S. Lin, "An efficient heuristic procedure for partitioning graphs," *The Bell System Technical Journal* (1970), 291-307.

[38] B. Kernighan, D. Schweikert, and G. Persky, "An optimal channel-routing algorithm for polycell layouts of integrated circuits," *Proceedings 10th. IEEE Design Automation Workshop* (1973).

[39] A. S. LaPaugh, *Algorithms for Integrated Circuit Layout: an Analytic Approach,* Ph.D. thesis, Dept. of Electrical Engineering and Computer Science, M. I. T., (1980).

[40] F. T. Leighton, *Layouts for the Shuffle-Exchange Graph and Lower Bound Techniques for VLSI,* Ph.D. thesis, Dept. of Mathematics, Massachusetts Institute of Technology, (1981). A revised version appears as *Complexity Issues in VLSI,* Foundations of Computing Series, M.I.T. Press (1983).

[41] F. T. Leighton, "New lower bound techniques for VLSI," *Twenty-Second Annual Symposium on Foundations of Computer Science, IEEE* (1981).

[42] F. T. Leighton, "A layout strategy for VLSI which is provably good," *Fourteenth Annual ACM Symposium on Theory of Computing* (1982).

[43] F. T. Leighton, "New lower bounds for channel routing," M. I. T. VLSI Memo 82-71, (1981).

[44] F. T. Leighton, "Parallel computation using meshes of trees," *Proceedings 1983 Osnabruck Workshop on Graph theoretic Concepts in Computer Science* (1983).

[45] F. T. Leighton and C. E. Leiserson, "Wafer-scale integration of systolic arrays," *Twenty-Third Annual IEEE Symposium on Foundations of Computer Science* (1982).

[46] F. T. Leighton and A. L. Rosenberg, "Three dimensional circuit layouts," M.I.T. VLSI Memo 102, (1982).

[47] C. E. Leiserson, "A model for VLSI computation," Thesis proposal, CMU, (1979).

[48] C. E. Leiserson, "Systolic priority queues," *Proceedings of the Caltech Conference on Very Large Scale Integration,* C. Seitz, ed., California Institute of Technology, (1979).

[49] C. E. Leiserson, "Area-efficient layouts (for VLSI)," *Twenty-First Annual Symposium on Foundations of Computer Science, IEEE* (1980).

[50] C. E. Leiserson, *Area-Efficient VLSI Computation,* Ph.D. thesis, Dept. of Computer Science, Carnegie-Mellon University, (1981). Also published by M.I.T. Press 1983.

[51] C. E. Leiserson and R. Y. Pinter, "Optimal placement for river routing," *Proceedings CMU Conference on VLSI Systems and Computations* (1981).

[52] P. M. Lewis, R. E. Stearns, and J. Hartmanis, "Memory bounds for recognition of context-free and context-sensitive languages," *IEEE Symposium on Switching Circuit Theory and Logical Design* (1965).

[53] R. J. Lipton and R. E. Tarjan, "A separator theorem for planar graphs," *A Conference on Theoretical Computer Science,* University of Waterloo, (1977).

[54] G. A. Magó, "A network of microprocessors to execute reduction languages, Parts I and II," *International Journal of Computer and Information Sciences* (December, 1979).

[55] C. Mead and L. Conway, *Introduction to VLSI Systems,* Addison-Wesley, (1980).

[56] K. Mehlhorn, "personal communication," (1982).

[57] D. Nath, S. N. Maheshwari, and P. C. P. Bhatt, "Efficient VLSI networks for parallel processing based on orthogonal trees," *IEEE Transactions on Computers* (July 1983).

[58] T. A. Ottmann, A. L. Rosenberg, and L. J. Stockmeyer, "A dictionary machine (for VLSI)," *IEEE Transactions on Computers* Vol. **C-31**, (1982).

[59] M. Paterson, W. Ruzzo, and L. Snyder, "Bounds on minimax edge length for complete binary trees," *Thirteenth Annual ACM Symposium on Theory of Computing* (1981).

[60] G. Persky, D. Deutsch, and D. Schweikert, "A minicomputer-based system for automated LSI layout," *Journal of Design Automation and Fault-Tolerant Computing* Vol. **1**, 3 (1977).

[61] R. Y. Pinter, "On routing 2-point nets across a channel," *Proceedings of the 19th IEEE Design Automation Conference* (1982).

[62] F. Preparata and W. Lipski, "Three layers are enough," *Proceedings Twenty third Annual IEEE Symposium on Foundations of Computer Science* (1982).

[63] F. Preparata and J. Vuillemin, "The cube-connected cycles: a versatile network for parallel computation," *Twentieth Annual IEEE Symposium on Foundations of Computer Science* (1979).

[64] J. Raffel, "On the use of nonvolatile programmable links for restructurable VLSI," *Proceedings of the Caltech Conference on Very Large Scale Integration* (1979).

[65] V. Ramachandran, "On driving many long lines in a VLSI layout," *Proceedings Twenty third Annual IEEE Symposium on Foundations of Computer Science* (1982).

[66] R. L. Rivest, "The "PI" (Placement and Interconnect) System," *Proceedings 19th. IEEE Design Automation Conference* (1982).

[67] R. L. Rivest, A. Baratz, and G. L. Miller, "Provably good channel routing algorithms," *Proceedings CMU Conference on VLSI Systems and Computations* (1981).

[68] R. L. Rivest and C. M. Fiduccia, "A greedy channel router," *Proceedings 19th. IEEE Design Automation Conference* (1982).

[69] A. Rosenberg, "Routing with permuters: toward reconfigurable and fault-tolerant networks," Technical Report CS-1981-13, Duke University, (1981).

[70] W. Ruzzo and L. Snyder, "Minimum edge length planar embeddings of trees," *Proceedings CMU Conference on VLSI Systems and Computation* (1981).

[71] A. Sangiovanni-Vincentelli, L. Chen, and L. Chua, "An efficient heuristic cluster algorithm for tearing large-scale networks," *IEEE Transactions on Circuits and Systems* Vol. **CAS-24**, No. 12, (1977), 709-717.

[72] T. J. Schaefer, "The complexity of satisfiability problems," *Proceedings 10th Annual ACM Symposium on Theory of Computing* (1978).

[73] J. T. Schwartz, "Ultracomputers," *ACM Transactions on Programming Languages and Systems* Vol. **2**, (1980).

[74] C. Séquin, A. Despain, and D. Patterson, "Communication in X-tree, a modular multiprocessor system," *ACM 78 Proceedings* (1978).

[75] S. Song, "A highly concurrent tree machine for database applications," *1980 International Conference on Parallel Processing* (1980).

[76] H. Stone, "Parallel processing with the perfect shuffle," *IEEE Transactions on Computers* Vol. C-20, (1971).

[77] T. Szymanski, "Dogleg channel routing is NP-Complete," unpublished manuscript, (1981).

[78] T. Szymanski and M. Yannakakis, personal communication, (1982).

[79] C. D. Thompson, "Area-time complexity for VLSI," *Eleventh Annual ACM Symposium on Theory of Computing* (1979).

[80] C. D. Thompson, *A Complexity Theory for VLSI,* Ph.D. thesis, Dept. of Computer Science, Carnegie-Mellon University, (1980).

[81] M. Tompa, "An optimal solution to a wire-routing problem," *Proceedings 12th. ACM Symposium on Theory of Computing* (1980).

[82] J. D. Ullman, *Computational Aspects of VLSI,* Computer Science Press, (1983).

[83] L. G. Valiant, "Universality considerations in VLSI circuits," *IEEE Transactions on Computers* (February, 1981).

[84] T. Yoshimura and E. Kuh, "Efficient algorithms for channel routing," U. C. Berkeley Electronics Research Laboratory Memo. M80/43, (1980).

CS-TR Scanning Project
Document Control Form

Date : 3/30/95

Report # LCS-TR-351

Each of the following should be identified by a checkmark:
Originating Department:

☐ Artificial Intellegence Laboratory (AI)
☒ Laboratory for Computer Science (LCS)

Document Type:

☒ Technical Report (TR) ☐ Technical Memo (TM)
☐ Other:_____

Document Information

Number of pages: 104 (114-IMAGES)
Not to include DOD forms, printer intstructions, etc... original pages only.

Originals are:
☒ Single-sided or
☐ Double-sided

Intended to be printed as :
☐ Single-sided or
☒ Double-sided

Print type:
☐ Typewriter ☐ Offset Press ☐ Laser Print
☐ InkJet Printer ☒ Unknown ☐ Other:_____

Check each if included with document:

☐ DOD Form ☐ Funding Agent Form ☒ Cover Page
☒ Spine ☒ Printers Notes ☐ Photo negatives
☐ Other: _____

Page Data:

Blank Pages(by page number):_____

Photographs/Tonal Material (by page number):_____

Other (note description/page number):

Description : Page Number:

IMAGE MAP (1-3) UNNUMBERED TITLE, ABSTRACT, ACKNOWLEDGEMENT PAGES (4-104) PAGES #'ED 4-104 (105-108) SCANCONTROL, COVER, SPINE, PRINTERS NOTES (109-114) TARGETS

Scanning Agent Signoff:

Date Received: 3/30/95 Date Scanned: 4/6/95 Date Returned: 4/6/95

Scanning Agent Signature: Michael W. Cook

Scanning Agent Identification Target

Scanning of this document was supported in part by the **Corporation for National Research Initiatives,** using funds from the **Advanced Research Projects Agency** of the **United states Government** under Grant: **MDA972-92-J1029.**

The scanning agent for this project was the **Document Services** department of the **M.I.T Libraries.** Technical support for this project was also provided by the **M.I.T. Laboratory for Computer Sciences.**

Scanned

Date: 4/06/1995

M.I.T. Libraries
Document Services