MIT/LCS/TR-248

ALGORITHMS FOR INTEGRATED CIRCUIT LAYOUT:
AN ANALYTIC APPROACH

Andrea Suzanne LaPaugh

Algorithms for Integrated Circuit Layout: An Analytic Approach

Andrea Suzanne LaPaugh

November 1980

The research reported here was supported in part by a Xerox Special Opportunity Fellowship and National Science Foundation grant MCS 78-05849.

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

LABORATORY FOR COMPUTER SCIENCE

CAMBRIDGE MASSACHUSETTS 02139

Algorithms for Integrated Circuit Layout: An Analytic Approach

by

Andrea Suzanne LaPaugh

Submitted to the Department of Electrical Engineering and Computer Science
on August 29, 1980 in partial fulfillment of the requirements for
the Degree of Doctor of Philosophy

ABSTRACT

In this thesis, the problem of designing the layout of integrated circuits is examined. The layout of an integrated circuit specifies the position on the chip of functional components and wires interconnecting the components. We use a general model under which components are represented by rectangles, and wires are represented by lines. This model can be applied to circuit components defined at any level of complexity, from a transistor to a programmable logic array (PLA). We focus on the standard decomposition of the layout problem into a placement problem and a routing problem.

We examine problems encountered in layout design from the point of view of complexity theory. The general layout problem under our model is shown to be NP-complete. In addition, two problems encountered in a restricted version of the routing problem -- channel routing -- are shown to be NP-complete. The analysis of heuristic algorithms for NP-complete problems is discussed, and the analysis of one common algorithm is presented.

The major result presented in this dissertation is a polynomial time algorithm for a restricted case of the routing problem. Given one rectangular component with terminals on its boundary, and pairs of terminals to be connected, the algorithm will find a two-layer channel routing which minimizes the area of a rectangle circumscribing the component and the wire paths. Each terminal can appear in only one pair of terminals to be connected, and the rectangle used to determine the area must have its boundaries parallel to those of the component. If any of the conditions of the problem are removed, the algorithm is no longer guaranteed to find the optimal solution.

Thesis Supervisor: Ronald L. Rivest
Title: Associate Professor of Computer Science and Engineering

Key words: VLSI (very large scale integrated) circuit layout, component placement (rectangles), channel routing, NP-completeness, algorithm analysis, heuristic algorithms.

Acknowledgements

I would like to thank my thesis advisor, Ronald Rivest, for his suggestions and support. My discussions with him -- both on general research issues and on technical details -- have been invaluable. I would also like to thank my readers, Jonathan Allen and Gary Miller, for contributing their viewpoints to this thesis. In addition, I would like to thank Alan Baratz, for many interesting discussions on VLSI design and layout; Ron Pinter, for carefully reading a draft of this dissertation; and Errol Lloyd for an enjoyable collaboration studying heuristic algorithms for channel assignment.

I would like to thank my friends at the Laboratory for Computer Science for their fellowship and support. Finally I would like to thank my husband, Michael Lipkowitz, for his encouragement and patience throughout the writing of this thesis.

Table of Contents

Chapter 1: Introduction

The research reported in this thesis is an investigation of algorithms for the layout of integrated circuits. Integrated circuits are formed on silicon chips by creating layers of different substances (e.g. metal, polysilicon) in geometric patterns on the chip through a variety of fabrication techniques. Electronic components are formed by the interaction of regions in the different layers. Wires connecting components are simply regions between two components on a layer. Laying out a circuit consists of determining the patterns for each layer on the chip to create the desired components and interconnections. For example, in nMOS/FET technology, a designer creates a transistor by drawing a region for the polysilicon layer and a region for the diffusion layer which cross when the designs for the two layers are superimposed [Me80]. In general, designing a layout requires knowledge of the interactions between layers for the technology being used and limitations of the fabrication process being used. The goals of the designer are to put as much circuitry as possible in as small an area as possible, and to have it work correctly and as fast as possible. A good example is the layout of a microprocessor, where the amount of information which can be processed, the number of functions which can be performed, and the speed of processing are important.

The layout problem as described above contains a huge number of variables and leaves much room for cleverness by the person designing the layout. It does not lend itself well to an algorithmic approach where a well-defined model and set of operations are employed. However, standard layouts can be used for each component needed in a circuit. These components need not be simple electronic components such as transistors or resistors, but may be logic gates or even higher level circuit subsystems. Given these components and the interconnections necessary to realize the desired circuit function, the layout problem consists of allocating a proper-sized region of the chip for each component and determining the pattern of wires forming the interconnections on each layer. This is the layout problem as we will mean it. We have lost the flexibility of tailoring the layout of each component to the particular application, but have greatly simplified the problem.

In general, we will wish to place a set of components on a possibly multi-layered planar surface. The components will contain *terminals*, which are points to which wires can connect. The desired interconnections will be specified by giving disjoint sets of terminals. Each set of terminals, called a *net*, should be interconnected. The interconnections will be made by wires which define paths between terminals in the layers. There will be constraints which the layout must satisfy, such as a minimum separation between unconnected wires. These constraints are called *design rules*.

The major motivation for developing algorithms to solve the layout problem is the complexity of the integrated circuits being designed. A chip may now contain tens of thousands of transistors. Hand layout of these integrated circuits, even with the aid of computerized graphics, is very costly, time consuming, and error prone. Standardized components, such as logic gate cells, are already employed in the industry to simplify circuit design and layout, and layout by computer has been implemented [Fe76] [Per77]. The tradeoffs are similar to those in computer programming. Programming in assembly language gives a programmer the freedom to devise clever ways to make a program run faster or require less storage. However, when many large, complicated programs need to be written, the savings in time and the conceptual simplification gained by using a high level language and compiler are worth the loss of flexibility. Just as one might write an often used subroutine in assembly language, one might do the layout for a circuit subsystem to be used in many circuits by hand. However, for most projects, the savings in time and the added faith in the correctness of the design due to the simplified structure make the component approach preferable.

Although working automated layout systems do exist, the problem of designing algorithms to do integrated circuit layout is not solved. Many of the existing algorithms place further restrictions on the problem. Typically, they require that all components be the same size in one dimension. Such an algorithm places the components in rows, forming an array. The wiring (called *routing*) is done in the spaces between rows. Also, although several algorithms have been designed and implemented, little analysis of the algorithms has been performed. Empirical evidence is often cited in the literature. However, papers describing algorithms which may not find optimal solutions rarely present mathematical analyses of the quality of the solutions found. For example, one does not find statements of the form "this algorithm always finds a solution within 50% of the optimal." In this dissertation, we discuss layout algorithms from a complexity theory point of view. We focus on the performance of algorithms, both in terms of the quality of the solutions they produce and the running times they require. We work with subproblems of the layout problem since they are easier to approach and are commonly encountered. An example of such a subproblem is the routing problem when components have been placed in rows.

The thesis is organized as follows. In Chapter 2, we review the techniques used by existing automatic layout systems. In Chapter 3, the model we will use for layout problems is presented. We also describe a second model -- the graph theoretic model -- which has proven very useful in characterizing the area required by various interconnection patterns. In Chapter 4, we discuss the complexity of a number of versions and subproblems of the layout problem. A review of previously known NP-completeness results is given. We prove the NP-completeness of a rectangle placement problem and two problems encountered in channel routing. We analyze a previously known heuristic algorithm for one of the channel routing problems. In Chapter 5, a new algorithm is presented for a special case of the routing problem. In this problem, terminals lie on the boundary of one rectangular component. Pairs of terminals must be interconnected. The algorithm finds a minimum area routing for a channel routing model and has running time $O(t^3)$, where t is the number of terminals. By developing this algorithm, we have shown that there are non-trivial routing problems which are not NP-hard. Most routing problems are either known to be NP-hard or are so closely related to NP-hard problems as to be considered intractable without an actual proof of NP-hardness. In Chapter 6, we discuss properties of the algorithm. In Chapter 7, we summarize and present open problems and directions for future research. A review of basic definitions and notation used throughout this thesis is presented in the appendix.

Chapter 2: A Review of Layout Automation Techniques

2.1 Classical Approach to Layout

The problem of placing components on a surface and making the required interconnections in one or more layers is not new. Research on the layout problem was initially done for printed circuit boards in the 1960's. The layout problem for printed circuit boards is closely related to that for integrated circuits -- components are placed on a board and interconnections are made by conducting strips (wires) printed on two or more layers of the board. Wires on different layers are insulated from each other, but wires on the same layer must not cross unless a connection is intended. Depending on the manufacturing technique, a conducting path may be able to change layers only at fixed positions on the board (called fixed vias) or anywhere on the board (called floating vias). Some researchers have used models for the layout problem which they intend to apply to both printed circuit boards and chips [Han76]. However, as we shall discuss below, the objectives and assumptions for printed circuit boards and integrated circuits are different.

Traditionally, layout of circuits has been divided into two phases -- the placement phase and the routing phase. Separate algorithms have been designed for each. In the placement phase, components are assigned positions; in the routing phase, the paths which the wires will use are determined. For printed circuit boards, the goal is usually to minimize the total length of wire used. Generally, the board is divided by a rectilinear grid and components can be placed only at certain locations on the grid. Terminals where wires must attach are at fixed positions on each component; these positions match locations on the grid. Automatic layout systems for integrated circuits have borrowed the algorithms and models from printed circuit board research and expanded on them. Most systems use standard cells of uniform size which are arranged in rows and columns, leaving a grid of horizontal and vertical streets in which connections can be made. Examples of standard cell systems can be found in [Fe76], [Per77].

For placement algorithms, components are generally modeled as points and the board or chip as an array of locations on which the components must be placed. For printed circuits, components are usually standard size packages so that size is not a factor. For integrated circuits, a size parameter may be associated with each point and a capacity with each location. For example, in [Sc76] each location represents a row of standard cells and has a capacity which is the length of the row. In the majority of placement problem formulations, the objective is to find a placement which most facilitates the routing to follow. A function of the placement is chosen as an objective function to be optimized. The function is supposed to be an indication of the difficulty of the routing given the placement. Most often the function is an estimate of the total wire length used for routing. Various estimates of the wire length needed to route one net (set of terminals to be interconnected) are used, e.g. the half-perimeter of the smallest rectangle enclosing all terminals of the net; the length of the shortest spanning tree of the net. The estimate must be easy to compute.

There are two types of placement algorithms -- constructive initial placement and iterative improvement. Most automated layout systems use both, although either can be used alone: iterative improvement can be used with a random initial placement. Constructive initial placement algorithms place components one by one based on their connectivity to components previously placed. In many systems, a designer may choose and place the first components. When the first component is chosen by the algorithm, a special component such as a bonding pad may be chosen, or an arbitrary component may be chosen. Given an initial placement, iterative improvement algorithms move components to improve the objective function. These algorithms are local optimization algorithms. One common technique is to exchange pairs or rearrange larger sets of components and test whether the value of the objective function has been improved. In another method, called Force-Directed Relaxation, a component is moved to a point where the "forces" due to its connections to other components are balanced. A description of various placement algorithms of both types can be found in Chapter 5 of [Des72]. An experimental comparison of several of these algorithms is presented in [Han76].

The routing portion of the layout problem has received much attention in the past ten or fifteen years. Many algorithms have been developed to find "near optimal" solutions to the routing problem when sets of points which must be interconnected are given as input. Most of the algorithms attempt to minimize total wire length. These algorithms almost exclusively use the rectilinear (also known as Manhattan) measure of distance.[1]

Routing can be approached a number of different ways. Nets may be connected using Steiner trees, i.e allowing wires to branch at points between terminals; or may be restricted to spanning trees, where no branching wires are allowed. (See Figure 2.1) Connection paths may be allowed to change layers at arbitrary points, at fixed points, or not at all. The tree of connections for each net may be determined before the actual paths are found -- called wire list determination -- or the trees may be determined as paths are laid out. When each path must lie totally on one layer, the layer assignment, i.e. determining which paths will lie on which layer, is often done first. All paths on a layer must be routed so that they do not cross. However, many of the most recent routers for integrated circuits and printed circuits use the restriction that horizontal wire segments and vertical wire segments are on separate layers. (Horizontal and vertical are the perpendicular directions of the rectilinear metric.) Paths are composed of horizontal and vertical segments. Each change of direction is a change of layer. Typically, two layers are used -- one for each direction. In the projection of the layers on one plane, two paths may intersect within perpendicular segments without being electrically connected. When paths can change layers only at fixed points, called vias, the vias may be assigned to particular connections before routing. For printed circuits, the number and location of vias is often restricted due to the fabrication technology. For integrated

[1] Under the rectilinear metric, points (x_1,y_1) and (x_2,y_2) in a cartesian coordinate system are distance $|x_1-x_2|+|y_1-y_2|$ apart.

circuits, vias are contact cuts [Me80]. They are usually allowed anywhere, although it is desirable to minimize the number used since they require extra area.

Most algorithms to do the actual routing of wire paths fall into one of the following four categories: maze routers, line routers, cell routers, and channel routers. The first algorithms used on printed circuit boards were maze routers. Maze routers find one path at at time. They are based on Lee's algorithm [Lee61] for finding the shortest path between two nodes in a graph. In fact, a path can be found between members of two sets of nodes rather than between two specific nodes. The graph used for routing is a grid graph containing forbidden regions. The algorithm is a breadth first search of the usable grid points. (See Figure 2.2.) Multiple layers may be modeled by using one two-dimensional grid for each layer. Each grid point (for unrestricted vias) or each of a special set of grid points (for fixed vias) on one planar grid is adjacent to corresponding grid points on other planar grids.

Cellular routers [Hit69] also divide up the routing plane into a regular arrangement of cells. Again, each path is found, one at a time, by a breadth first search over all cells. Each cell represents a region of the routing surface, including all layers. This representation is obtained by projecting all layers on one plane, then partitioning the plane into pieces of uniform size and shape. These pieces are the cells. Each cell is large enough to fit more than one wire width on each layer. The routing algorithm must define the entrance and exit points of each routing path in the set of cells traversed by the path. From a given point on a cell boundary, the reachable points along the cell's boundaries are calculated taking into account wires which have been previously routed and have cut off parts of the cell from other parts (Figure 2.3). The determination of the actual paths used within each cell is left for a second algorithm. Cellular routers do not require the large number of grid points which must be represented when using a maze router.

Line routers [Hi69] do not partition the plane. These routers work directly with horizontal and vertical line segments. They build up a path between two points out of these segments. A line router

Figure 2.1: Steiner tree connections versus spanning tree connections.

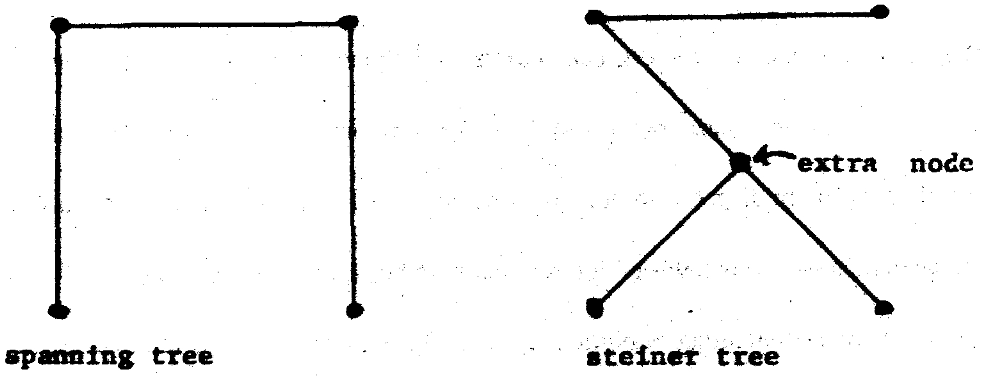
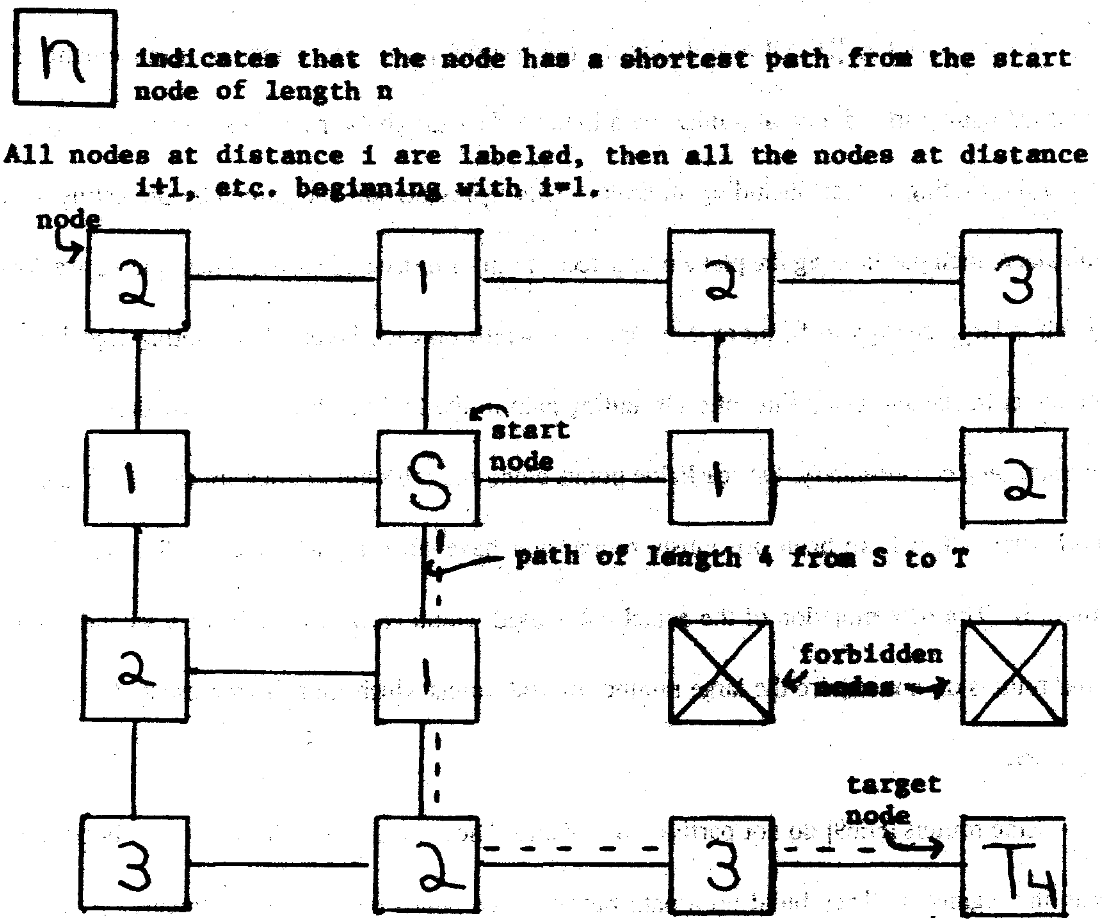


Figure 2.2: Lee's Algorithm.



n indicates that the node has a shortest path from the start node of length n

All nodes at distance i are labeled, then all the nodes at distance i+1, etc. beginning with i=1.

finds paths only in one plane. Paths area determined one at a time. Given two target points to be connected, line segments are extended from the targets until they hit obstacles. Points along these segments from which a line can be extended which will "escape" an obstacle previously hit are found. In this way, "escape lines" are generated until a path between the two target points can be constructed from segments of these escape lines. (See Figure 2.4.)

The channel routing technique [Has71] was developed for routings with horizontal and vertical wires on separate layers. The area available for routing is divided into horizontal and vertical *streets*. Each horizontal or vertical street can contain a number of horizontal or vertical path segments, respectively. Paths are first found through the streets without regard to conflicts within each street. After all connections have been globally routed through the streets, the segments within each street are arranged so that no two overlap. The number of parallel lanes or *channels*[1] in each street is minimized. Short segments perpendicular to the street direction must be used from each terminal out into the street. For some algorithms, such short segments are also permitted to allow a path segment to change channels. (See Figure 2.5.) The global routing problem is actually a path-finding problem on a graph where more that one path may use the same edge. The local routing problem (or *channel assignment problem*) resembles a packing problem rather than a path-finding problem. These problems will be described in more detail in Chapter 4.

The algorithms described above are those most often used by layout automation systems. They do not encompass all algorithms. A survey of routing algorithms can be found in Chapter 6 of [Des72] and in [Hi74]. A study of router performance is described in [Kel77]. None of the algorithms are guaranteed to find an optimal solution unless they are allowed to rip up and reroute -- exhaustively searching all possible routings. Furthermore, if the amount of area which can be used is bounded, the

[1] The use of the term channel has become confused in the literature. It is used by some authors to denote a street, by others to denote a lane within a street. We follow the terminology of [Has71].

Figure 2.3: Cellular routing.

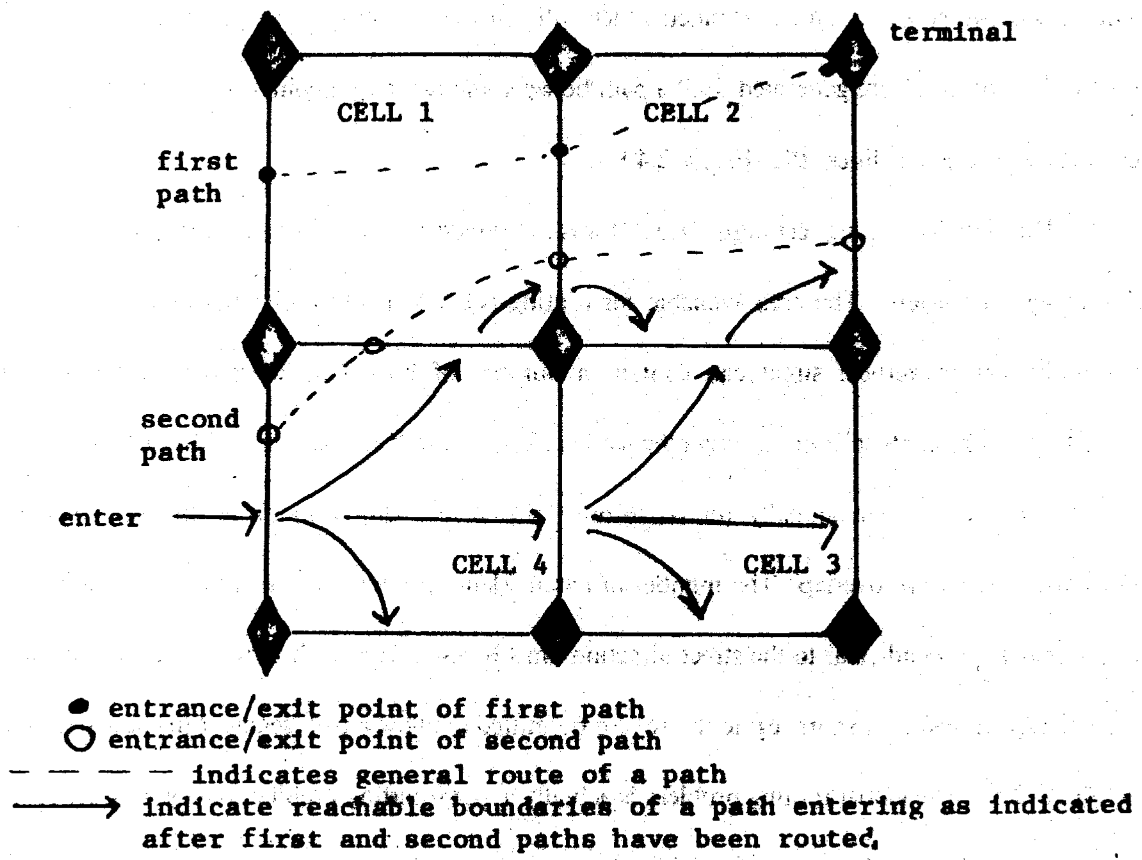
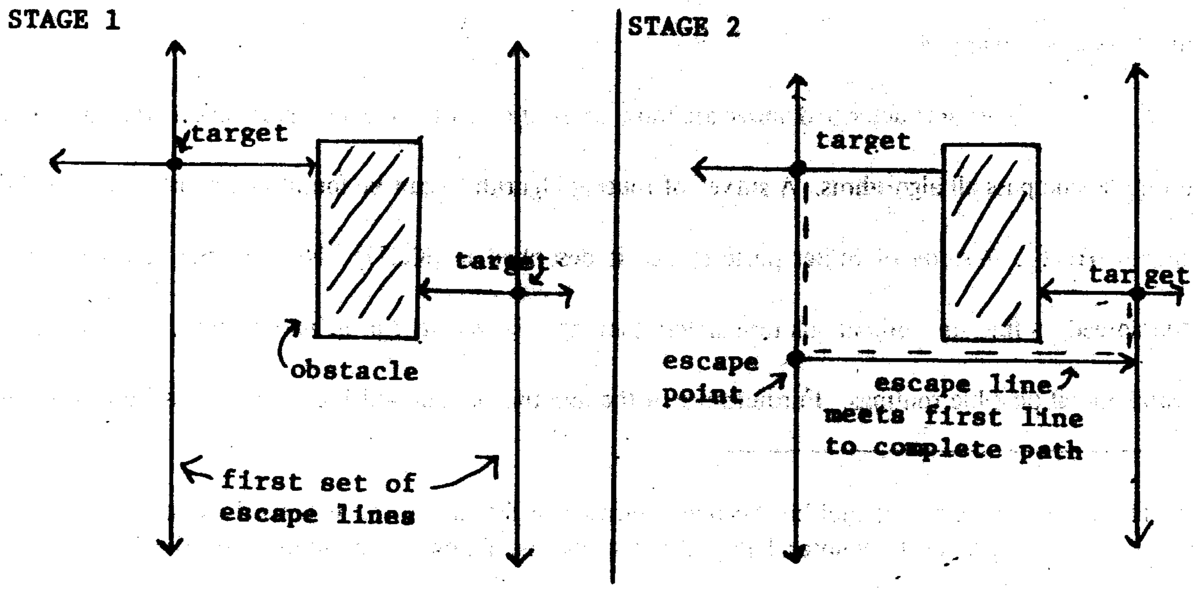

terminal
CELL 1
CELL 2
first path
second path
enter
CELL 4
CELL 3
● entrance/exit point of first path
○ entrance/exit point of second path
– – – – indicates general route of a path
⟶ indicate reachable boundaries of a path entering as indicated after first and second paths have been routed.

Figure 2.4: Line routing.


STAGE 1
target
target
obstacle
first set of escape lines
STAGE 2
target
target
escape point
escape line meets first line to complete path

algorithms are not guaranteed to route all connections, even though such a routing exists. In practice, failed connections are completed manually. Many systems use a combination of algorithms, beginning with faster algorithms to route most connections and using slower but more successful algorithms to route the final connections (e.g. see [Po79]). Some algorithms are more susceptible to the problem of failing to complete all connections than others. When each point-to-point connection is assigned to a single layer and then routed, the paths found first may encircle a terminal not yet connected to anything. It is impossible to find a path to this terminal without changing layers or ripping up a previously routed path. (See Figure 2.6.) For routers of this type, the order in which the paths are assigned for a problem can make a great difference in whether the algorithm is successful in routing all the connections. Given this, it is interesting that Abel [Ab72] concludes from his empirical study that, overall, the performance of such routers is not significantly affected by various ordering criteria. (The router used is a maze router with an added heuristic so that paths do not run next to a row of terminals at minimum spacing from the terminals. The paths being avoided would lead to a large number of blocked terminals.) The channel routing technique is less susceptible to the problem of failure to route all connections since the exact position of each segment is determined after all paths are globally assigned. In fact, if area is not limited, the streets can contain an arbitrarily large number of channels and 100% routing can always be obtained.

Several characteristics of the models used for past research on the layout problem are very restrictive, especially when considering the layout of very large scale integrated (VLSI) circuits, where up to a million transistors are packed on one chip in a very dense arrangement. The use of cells which have one or both dimensions fixed limits the type of components which can be used. One may imagine having in the same circuit a very large component which is an array of registers and a number of small components realizing a special function. Once the components being used are of varying size, it is a waste of space to place them in an array separated by "streets" running the length and width of the chip. The size of each component and the way the components fit together on the chip are important.

Figure 2.5: Channel routing.

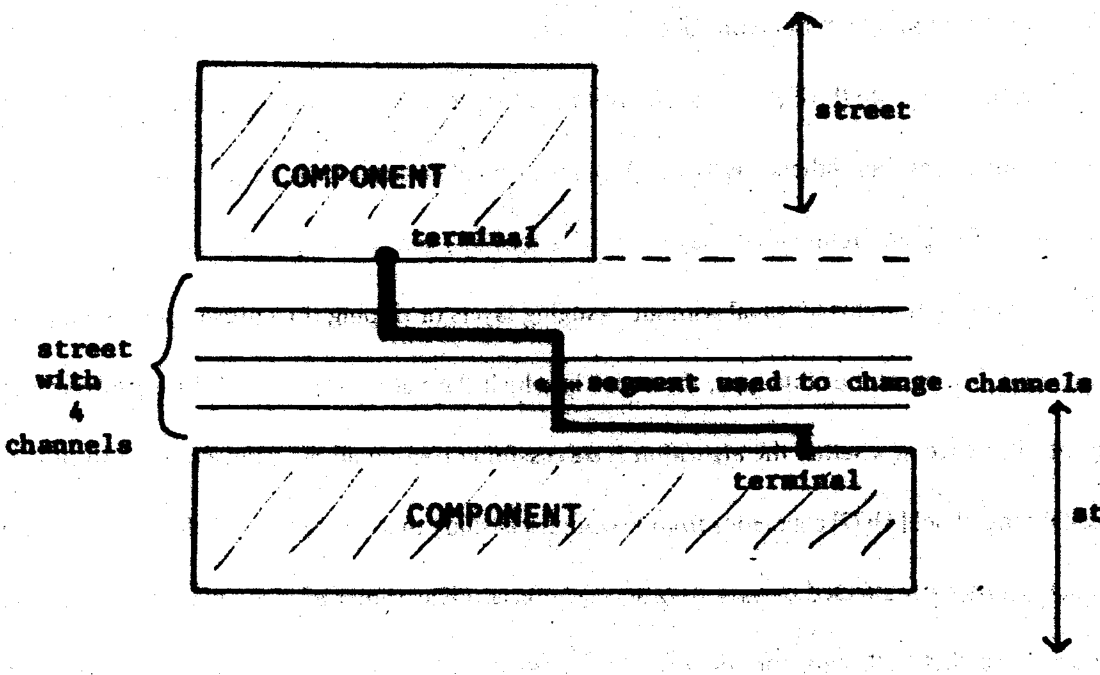
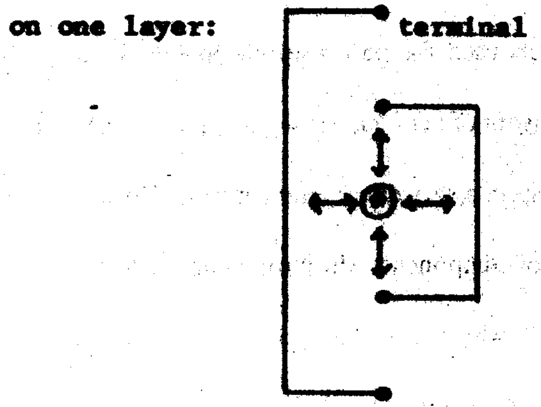


Figure 2.6: Surrounding a terminal.



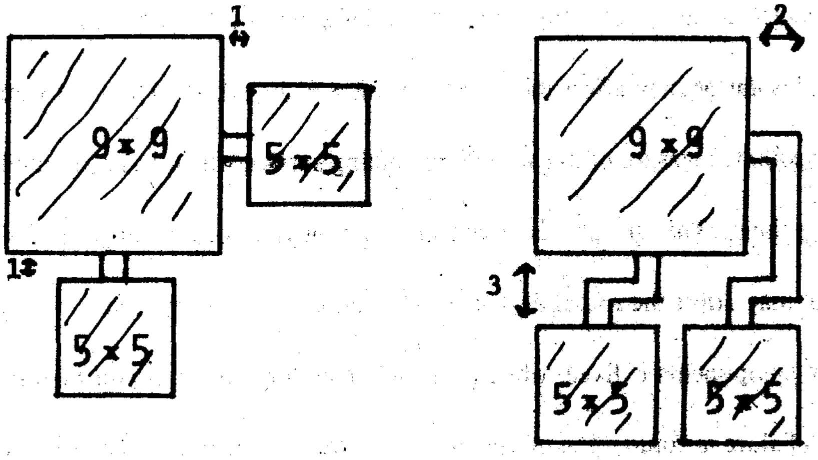
⊙ terminal which is surrounded. No path can reach this terminal.

↔ spacing indicated here is minimum allowed.

When an array of locations is used for cell placement, it is reasonable to use total wire length as a measure of the worth of a particular layout. Different arrangements of components within the array only affect the area of the layout insofar as they affect the interconnection paths taken. The total wire length is an easily computed approximate measure of the amount of routing surface which has been used at any point while routing is being done. This, in turn, is an approximate measure of the congestion of wires in routing areas. Congestion may affect the routability of interconnections still to be made. When the area used for routing between components is fixed, which is usually true for printed circuit boards, too much congestion may result in failure to route some of the interconnections. When the area used for routing can be expanded by moving the components (while keeping their relative positions fixed), as is more likely for integrated circuit design, too much congestion may result in a larger overall layout size.

When components are not of relatively uniform size, their placement has great effect on total layout size. Wire length is no longer a good approximation to layout size. This is illustrated in Figure 2.7. The placement of components and routing of wires interact in a much more complex manner to determine the total size of a layout. The most recently developed layout systems no longer treat component size as a parameter which is of secondary importance. Components are modeled as rectangles -- giving them shape as well as size -- rather than as points with a size parameter attached. When rectangles must be placed on a plane, the way they fit together influences the area used by the placement and the shape of the spaces left for routing. Preas and Gwyn [Pr78] have retained a constructive initial placement based on connectivity but place rectangles in a plane rather than points on a grid. Their iterative improvement phase tries to minimize the area of the circuit by selecting as the candidate for placement modification a component one of whose dimensions contributes to the widest part of the layout in that dimension. This component may be rotated, reflected, shifted, or exchanged with another component. A modification is accepted if the area is reduced. In [La79], the initial placement is produced by dividing a square of area the same as the total area of the components into rectangles of the

Figure 2.7: Tradeoff of wire length versus area.



total area: 15x15 = 225

total wire length: 2

total area: 17x11 = 187

total wire length: 16

where wire width is 1 unit, minimum spacing is 1 unit same area as the individual components. This gives an approximate placement which is modified to fit the actual components. Improvements are made to the placement by using rotations and shifts. Routing is taken into account in the improvement phase. The width of each street is estimated based on global routing and included in the area calculation. Brinkmann [Br76] also uses the technique of dividing a large rectangle into smaller ones to find an approximate placement.

Routing programs for the most recent systems also try to minimize area rather than wire length. Channel routers are used in [Pr78] and [La79]. Channel routers are easy to use when area is the parameter to be optimized since local routing minimizes street widths. Streets can be allowed to shrink or expand as needed to complete the routing. In [Lo79], components are allowed to take positions above the routing area independent of one another so that area is not wasted unnecessarily by aligning the edges of the components.

Wire length remains an important parameter for a layout because it directly affects the quality and speed of signals in the circuit. In a situation such as that shown in Figure 2.7 where both size and total wire length cannot be minimized at the same time, a tradeoff must be made. In this context, it may be decided that wire length is most important regardless of the resulting circuit size. At other times, only a maximum wire length might be imposed on certain interconnections in the circuit. To make things more complex, we might imagine a situation in which the requirement was that two interconnections have approximately the same length, say when two outputs of one component are the inputs to another component. In short, the desired measure for the worth of a layout can be made very complicated if enough factors are considered. Physical quantities may depend on other layout properties such as the density of wires in an area. In [Agu77] and [Ru77], a system is described in which total power is minimized and timing constraints are observed. In [No76], connections whose delay must be minimized can be designated "critical" and treated special.

2.2 Alternate Approaches to Layout Automation

The approaches described in Section 2.1 separate the placement and routing phases of layout. The input is a set of components -- either specified as points or rectangles -- and a set of interconnections to be made either among the components as points or among terminal points on the components. There are alternate approaches in the literature where the topological aspects of layout are modeled using graph theory. When finding the layout of a circuit, placement and routing are considered together by finding a planar embedding of a graph modeling the circuit. The models do not necessarily associate one node with each component. Each component may be modeled as a set of nodes and edges, e.g. a cycle. Interconnections among a set of terminals may be modeled as a set of edges or a set of nodes and edges. For example, the branch point of a wire may be represented as a point under a Steiner tree representation of connections. Several models have been suggested. A summary can be found in [van76]. Graph embedding techniques have the advantage that placement and routing interact completely at the topological level. However, geometry -- the size and shape of the components -- is completely ignored and must be accounted for separately.

In the standard approach and in the graph embedding approach, the only information about the layout provided by the input is the set of nets. In an alternate approach based on stick diagrams [Me80], topological information about the desired layout is also provided. In stick diagrams, regions in various layers of the integrated circuit are represented as lines. For example, in nMOS technology, a polysilicon line and a diffusion line crossing represent a transistor. The relative positions of components and the general path of each wire are indicated. The layout automation system must expand the "sticks" into rectangles of the proper dimensions based on design rules, and modify the layout so that components and wires fit together with the proper spacing. Examples of such systems are STICKS [Wi77] and CABBAGE [Hs79]. These programs attempt to pack the layout as much as possible while satisfying the design rules and maintaining the original relative positions of components and wires. This technique exploits the human designer's ability to do overall layouts, i.e. rough sketches. The program FLOSS [Ch77] also packs a rough manual layout; it works from a hand drawn sketch. Other systems for which the designer does the general layout use a symbolic representation of the layout [Gi76], [Per73]. The symbolic layout is produced by the designer and expanded into a fully specified layout automatically. Other techniques for design automation use special structures such as Programmable Logic Arrays (PLAs), with known layouts. Functional specifications can be automatically converted into PLA implementations [Ay79]. Special interconnection patterns can also be exploited to assist in design [Jo79].

2.3 Summary

All of the above techniques have been developed to automate, at least in part, the design of circuit layouts. The research reported in this dissertation is restricted to placement and routing as described in the first section. Although many algorithms have been designed and tested, little mathematical analysis of the algorithms has been performed. The techniques of complexity theory are not regularly applied to layout problems. One notable exception is the work of So, Ting, Kuh and others ([Ku79], [So74], [Ti76], [Ti78], [Ti79], [Ts79]) for printed circuit board routing. In this work, a routing problem for printed circuit boards with fixed vias in columns is broken up into several problems, ending with a number of instances of a routing problem for a row of terminals on a single layer. The problems are analyzed. Necessary and sufficient conditions for a single-row, single-layer routing to be optimal are developed. However, the model is not well suited to integrated circuits. The research reported in this dissertation also focuses on particular subproblems of the layout problem. The problems are motivated by the channel routing model of interconnections.

Chapter 3: Models for Integrated Circuit Layout

As is evident in the discussion in the previous chapter, most models of circuits for layout use points or rectangles to model components. In the research presented in Chapters 4 through 6, we use rectangles. The model is described precisely below. In the second section of this chapter, we describe a graph model in which components are points. We discuss its use in proving bounds on the area required by circuits with certain interconnection patterns.

3.1 The Geometric Model

We have chosen to use a rectangle model of components because it captures the geometric and topological aspects of the layout problem. Components are rectangular in shape and variable in size. Wires lie on any of several layers and are of uniform width. The model was guided by the design rules presented in [Me80] for nMOS technology, but it is intended to be applicable to many technologies. Formally, the model is as follows.

The input for a layout problem will consist of a set of components and a set of nets. Each component, C_i, will be a rectangle with given dimensions x_i and y_i. At given locations on the boundary of a component C_i are terminals t_{ij}, $j = 1,...,n_i$, where n_i is the number of terminals on component C_i. Each net is a set of terminals, and the nets are pairwise disjoint. Each net represents a collection of terminals which must be electrically connected. The layout problem is to place the components on a plane and form the interconnections specified by the nets in the minimum possible area. The interconnections can be made in any of N layers, where N is specified a priori. The layers are numbered 1 through N; layer i is considered to be adjacent to layers i+1 and i-1, for $2 \leq i \leq N-1$. Typically, N is two. The wires used to interconnect terminals have a uniform width w. There is a minimum spacing between wires on a layer, between components, and between a wire and a component of s.

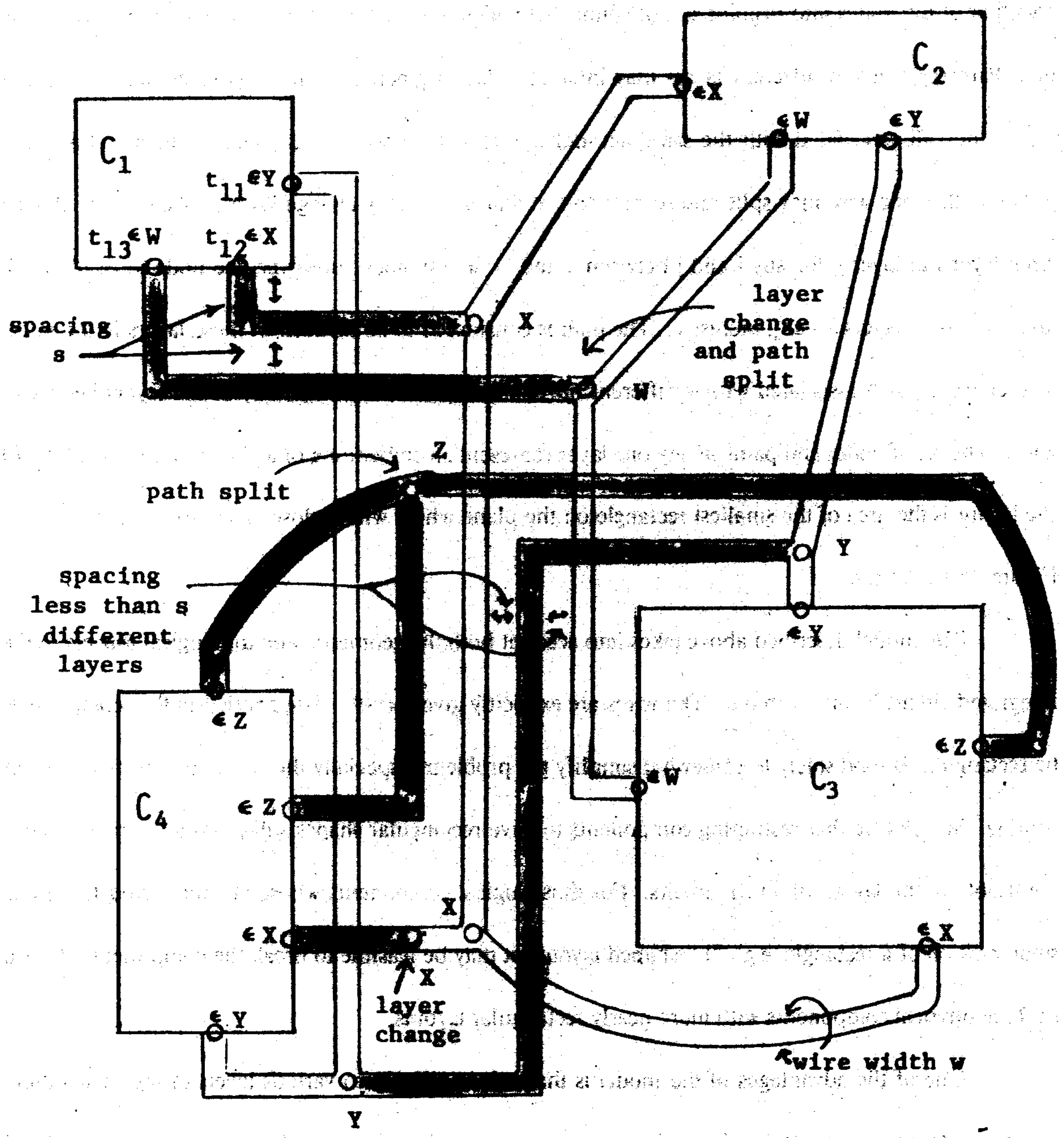
The interconnections form paths which lie on the surface of the plane not covered by the components. These paths go between *nodes*. A node is a terminal or an additional point on one or more layers of the plane. Each path segment between two nodes has a designated layer. The paths induce a partition of the nodes into disjoint sets of connected nodes; each set contains terminals from exactly one net. Path segments in different layers may intersect. Path segments in the same layer cannot intersect unless they are associated with the same net and intersect at a node. Additional points are allowed as nodes so that one wire may split into several and so that a path may change layers. A path may change from layer i to layer j, for any i and j between 1 and N, at any node; however, the node must lie on all layers between layers i and j, inclusive. The path is considered to lie on each of these layers at the node. Therefore, no path associated with a different net can intersect the node on any of the layers between i and j. The set of nodes and paths in any one layer represent an embedding of a planar graph. The area of the layout is the area of the smallest rectangle on the plane which will enclose all components and wires. Figure 3.1 illustrates.

The model described above takes into account both the geometric and topological aspects of the integrated circuit layout problem. The wires are explicitly given width. The restriction that components be rectangular is used solely to somewhat simplify the problem--especially the calculation of free area for routing. We believe that restricting components to have rectangular shapes still provides a model that is applicable to the layout of VLSI circuits. For those logical components whose layouts would fill only a small portion of a rectangle, e.g. "L"-shaped layouts, it may be feasible to break the component into two or three physical components with more nearly rectangular layouts.

One of the advantages of the model is that it is applicable to various levels of modularization. Components may represent transistors, logic gates, or even arithmetic units, allowing any conceptual level of design. A hierarchical approach to layout such as that used by Preas and Gwyn [Pr78] can easily be taken using this model.

Additional parameters can be allowed as inputs to the layout problem in this model. Upper

Figure 3.1: Illustration of the geometric model for layout design.



W,X,Y,Z are nets
O indicates a node. Nodes are labeled by the net being connected.

▬ indicates layer 1; ═ indicates layer 2
There are only 2 layers.

bounds on the length and width or the total area used by the layout may be given. Alternatively, a bound on the aspect ratio, i.e. the ratio of the length to the width, of the layout area may be given. This would restrict the shape of the layout. Any of these limitations may be necessary to insure that the layout produced is usable. For example, suppose a minimum area solution resulted in an extremely long and thin chip. The shape could prohibit the fabrication or the packaging of the chip. Another possibility is that the circuit being laid out is a component at a higher level in a design hierarchy. The bounds might be derived from requirements at the higher level. Bounds such as these provide a range of solutions within which the algorithm should work. It is also possible to introduce wire length requirements to the layout problem. As mentioned in Chapter 2, wire length is an important factor in signal quality. Individual nets may be given upper bounds on the total wire length of their interconnections. (Wire length can be measured using the center line through a wire.) Relationships between the wire lengths for various nets may also be imposed.

In the following chapters, we will put restrictions on the types of layouts allowed within the above model. We will restrict ourselves to rectilinear paths and orthogonal orientations for components. Directions "horizontal" and "vertical" perpendicular to each other will be chosen, and all components will be placed so that their boundaries are parallel to one of these directions. Also, all paths will be composed of horizontal and vertical segments. This restriction is imposed for two reasons. First, the problems examined are motivated by currently used layout algorithms. As discussed in the previous chapter, the rectilinear metric is almost always used. Second, the restriction limits the number of possible solutions and makes the problems easier to analyze. Most of the time, we also restrict horizontal and vertical segments to be on different layers. Given that only horizontal and vertical wire segments are being used, this assumption not only greatly simplifies the description of allowable paths (those that may intersect at a point but do not overlap) but is reasonable when only two layers are available. In this case, two segments running in parallel on different layers, one on top of the other, prohibit any path from crossing from one side of these segments to the other. Any pair of terminals, one on each side of the segments, which must be connected will require a path which goes around the parallel segments. Therefore, one would rarely want such segments. Also, if the segments are extremely long, they may cause electrical problems due to capacitance. Restricting adjacent layers to contain segments in perpendicular directions eliminates these potential problems.

There are several technical aspects of the layout problem which the model does not take into account. We discuss these below and indicate how the model can be extended or modified to include them. In actual component design, a component may have several terminals to which a particular connection can be made. These terminals may be either physically equivalent, i.e. they are connected inside the component, or logically equivalent, i.e. you can't tell them apart functionally. An example of logically equivalent terminals is the set of input terminals of an "and" gate. The problem of deciding which terminal to use in connecting a particular net is called the *pin assignment problem* for printed circuit boards, and is usually solved before a routing algorithm is used [Hi74]. Both physically equivalent and logically equivalent terminals can be modeled as sets of terminals. A set of equivalent terminals, rather than an individual terminal, would be a member of the set defining a net. A set of physically equivalent terminals would appear in only one net, while a set of logically equivalent terminals would appear in several nets, but no more nets than the number of terminals in the set. When layout is complete, each set of interconnected terminals must contain at most one terminal from each set of logically equivalent terminals. This model of equivalent terminals is used by van Cleemput [van76].

In integrated circuits, layers are made of different materials and different design rules may apply. Allowing the width of wires and separation between objects to vary between layers is a minor modification to our model, although the resulting layout problem is more difficult. However, the design rules for various layers can be more complicated than the model will allow. For example, in nMOS technology [Me80], wires in diffusion and polysilicon layers cannot cross. Therefore, we model them as one layer. However, the minimum distance between two wires on either polysilicon or diffusion is different from the minimum distance between wires of different layers. This is not provided for in our model. We must use the largest of the actual required separations. Luckily, the metal layer in nMOS can cross both diffusion and polysilicon wires. If it could only cross one, then the actual layout problem would not be captured by our model: two layers could be used for interconnection, but a concept of "coloring" wires in each layer according to the design rules would have to be added. Another design rule which we have not accounted for concerns changing layers. When a conducting path changes layers, a contact cut must be made to connect the layers through an insulating layer. The contact cut requires a square area larger that the width of the wire. This is not taken into account in our area calculation. Even worse, some wires on a particular layer may be significantly wider than other wires on the same layer, depending on the electrical load on the wire. For example, the wires supplying power to a large circuit usually look like the human arterial system: there is a large main wire and a network of wires of decreasing size reaching every part of the circuit. We recognize that our model overlooks many details of actual layout design, but we feel that it is a reasonable approximation of the major issues in the layout of integrated circuits.

3.2 A Graph Embedding Model

We now discuss the uses of a model in which a circuit is represented as a graph. We call this model the *graph embedding model*. Each component is represented as a node in the circuit graph, and each connection to be made is represented as an edge between two nodes. The layout problem is defined as the problem of embedding the circuit graph in a two-dimensional grid graph. An embedding maps each node representing a component to a node of the grid. This mapping is one-to-one. Each edge representing a connection is mapped to a path in the grid graph. This path can only contain two grid nodes which correspond to component nodes -- those that correspond to the endpoints of the edge being embedded. Paths corresponding to distinct edges are allowed to intersect at grid nodes but are not allowed to use the same grid edge. To make such an embedding possible, each node in the circuit graph is restricted to having at most four edges adjacent to it, i.e. it is restricted to be of degree at most four. Such an embedding is an edge-disjoint homeomorphic embedding. Given an embedding, there are two measures of area which we will use. The first, which we will call the *node area* and denote A_N, is a count of the number of grid nodes used as images of components and on paths which are images of edges. The second, called *rectangle area* and denoted A_R, is the number of nodes contained in a rectangle whose boundaries lie on grid edges and which circumscribes all nodes used in the embedding, i.e. all nodes counted by A_N. For a circuit graph, C, let $A_N(C)$ and $A_R(C)$ denote the minimum node area and rectangle area, respectively, over all embeddings of C. Obviously, $A_N(C) \leq A_R(C)$. An embedding is shown in Figure 3.2.

The model presented above can be viewed as describing a layout which uses only horizontal and vertical wire segments (segments in the two grid directions), each direction on a separate layer. Alternatively, the nodes of the grid may be viewed as representing unit area squares on a plane. Each grid edge adjacent to a node represents a boundary of the corresponding square. Two paths which use a node need not actually intersect on the plane -- they may run diagonally, cutting across opposite corners of the corresponding square. (See Figure 3.3.) Thompson [Th80] uses a model of circuit layout which divides the plane into such unit squares. He views connections as running primarily in a single layer of metal. Crossovers are achieved by using short runs in a second layer such as polysilicon.

The graph embedding model lumps all terminals of a component into one point. The point-to-point connections made by wires are pre-determined and represented in the graph for the circuit. If Steiner tree interconnections of the terminals are desired (i.e. branching wires), these must be explicitly modeled in the circuit graph by adding a node for each branch point and edges from each branch point to components or other branch points. The model does not represent the area required by components or

Figure 3.2: Embedding a graph.

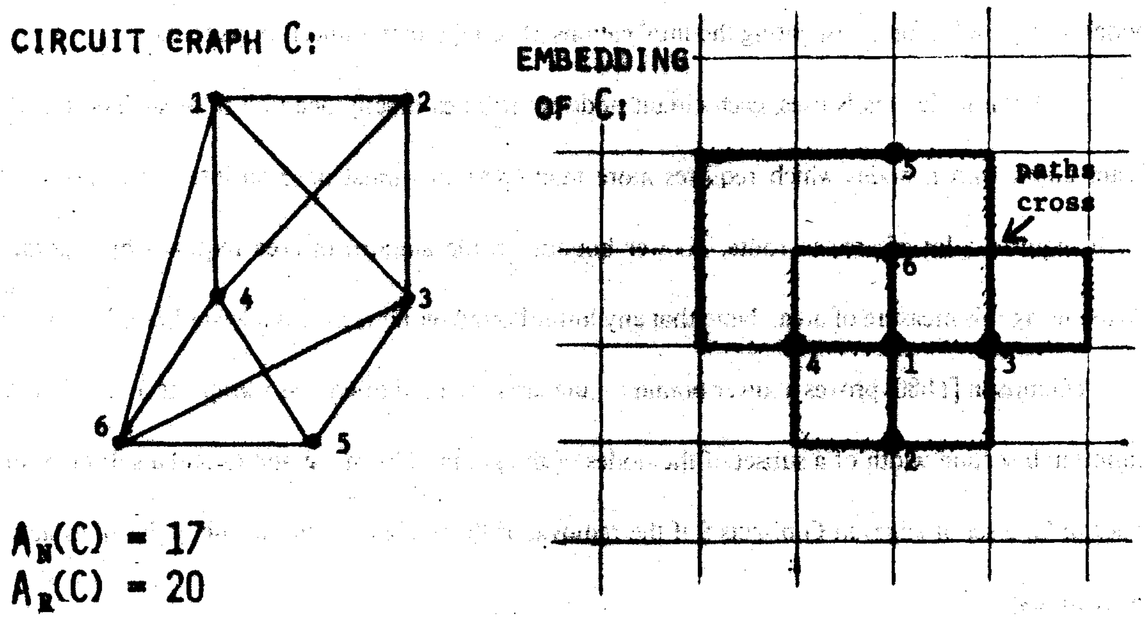


Figure 3.3: A node representing a square.

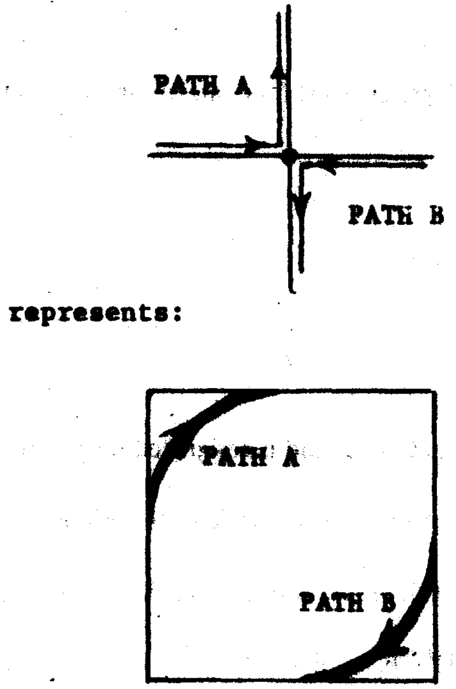
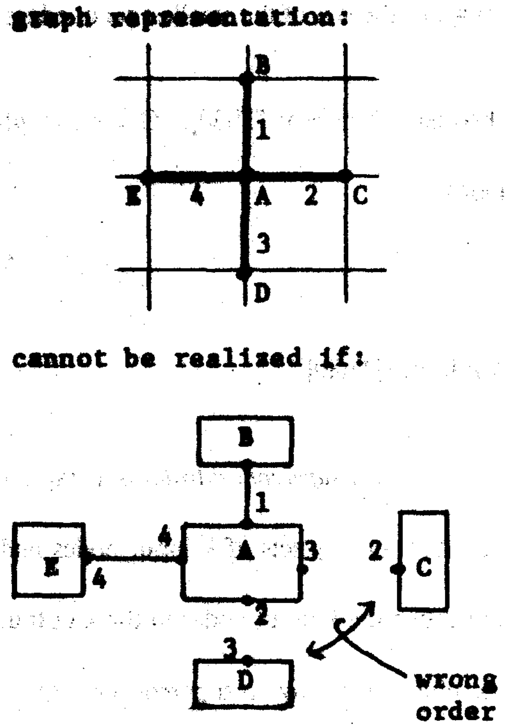


Figure 3.4: Effects of the order of terminals around a component.



the fact that terminals have a specific order around the component. This order causes some routings for the graph embedding model to be impossible in the geometric model (see Figure 3.4.). However, the model is very useful for investigating the implications of certain interconnection patterns.

When node area is uses, each circuit node contributes exactly one unit of area. Therefore, any circuit graph with n nodes which requires more than $O(n)$ area must have an interconnection pattern which requires a lot of area to route. Lower bounds on the amount of area required by a graph are proven using this measure of area. Note that any lower bound on node area is a lower bound on rectangle area. Thompson [Th80] proves a lower bound on the area required to embed a graph as a function of the minimum bisection width of a subset of the nodes of the graph. Given a graph G and a subset, S, of the nodes of G, a set of edges in G bisects S if the removal of these edges partitions the nodes of G into two sets such that

i) $\lfloor |S|/2 \rfloor$ of the nodes of S are in one set and $\lceil |S|/2 \rceil$ are in the other set;

ii) After removing the edges, there are no paths between nodes in different sets.

Let w_S be the size of the smallest set of edges bisecting S in G. Then:

Theorem (Thompson [Th80]): Given a graph G with nodes of degree at most four, and a subset, S, of the nodes,

$$A_N(G) \geq (w_S^2)/4.$$

Proof: See [Th80].

An *n-superconcentrator* is a graph with n designated input nodes and n designated output nodes such that for any sets of k input nodes and k output nodes, $1 \leq k \leq n$, there are k node disjoint paths connecting the k input nodes to the k output nodes [Val75]. For the set of input nodes, I, $w_I \geq \lfloor n/2 \rfloor$. Therefore, for any n-superconcentrator, G, $A_N(G) \geq (n-1)^2/16$. Since for any n, there are n-superconcentrators with $O(n)$ nodes, all of bounded degree [Pi77], there are graphs with n nodes which require node area $\Omega(n^2)$. Note that although the superconcentrators in [Pi77] do not have degree at most four, it is straightforward to reduce the degree of each node by adding a node for each edge. The number of nodes added is then bounded by the original number of edges. Figure 3.5 illustrates.

Thompson also defines average and worst case information complexities of a function. He derives a lower bound on the average or worst case time required by a graph to compute a function:

average (or worst case) time $\geq$

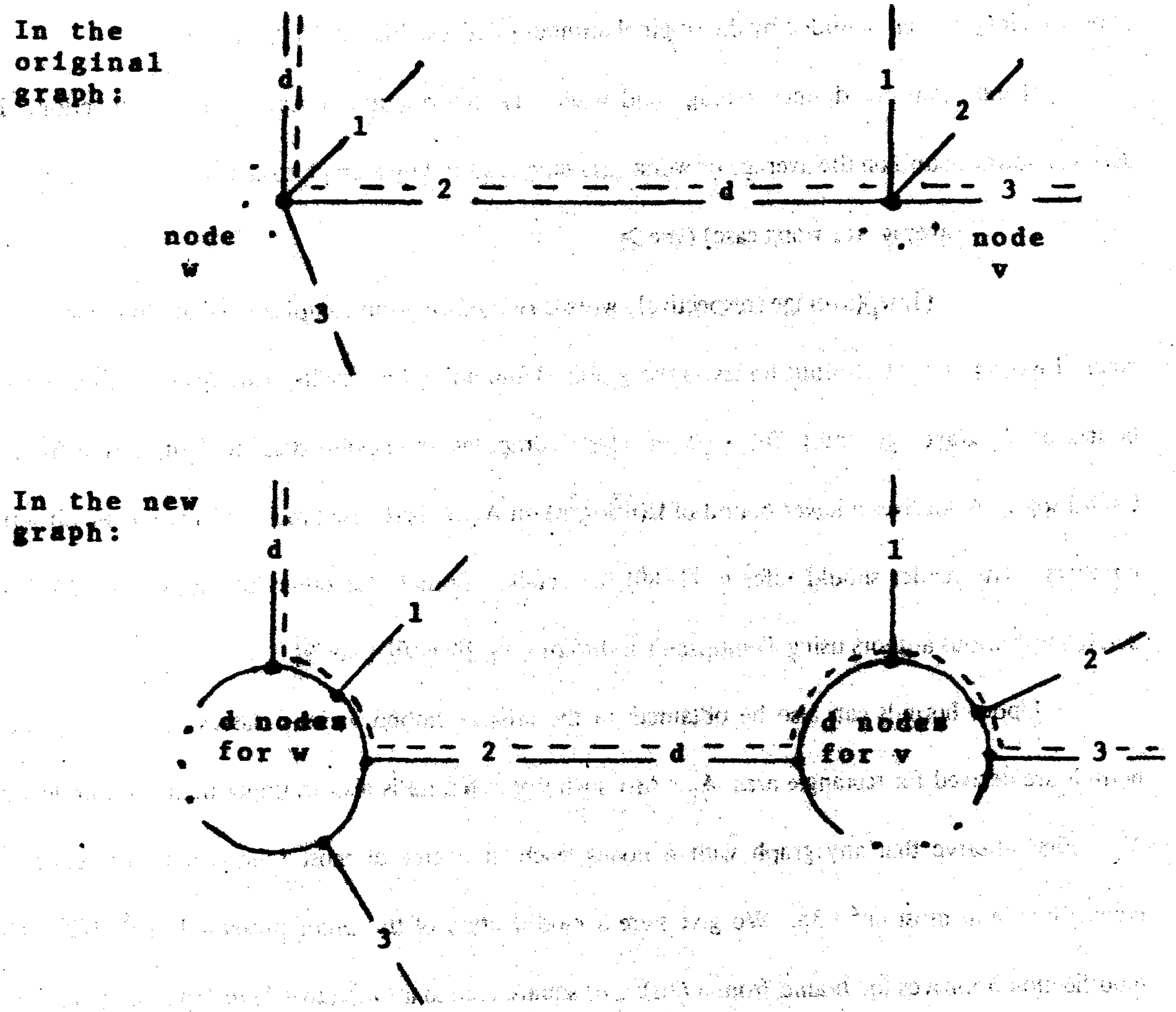
$(1/w_I)$(average (respectively worst case) information complexity of the function)

where I is a special set of input nodes in the graph. Combining the results, Thompson obtains a lower bound on A_Nx(average time)^2 for a graph which computes an n-point discrete fourier transform of $\lfloor n/8 \rfloor^2 \log^2 n$; he derives a lower bound of $\Omega(n^2 \log^2 n)$ on A_Nx(worst case time)^2 for a graph which sort n numbers. The reader should refer to [Th80] for details. Bounds for other functions have also been derived by various authors using Thompson's technique, e.g. [Abs80], [Sav79].

Upper bounds can also be obtained on the area to embed various classes of graphs. Upper bounds are derived for rectangle area, A_R. Any such upper bound is also an upper bound for node area, A_N. First observe that any graph with n nodes, each of degree at most four, can be embedded in rectangle area at most $6n^2+3n$. We give here a modification of the proof presented in [Val79]. This modification improves the bound from a (3n)x(3n) square area to a (3n)x(2n+1) rectangular area. Since n-superconcentrators require $\Omega(n^2)$ area, proportional to n^2 area is both necessary and sufficient for embedding an n-superconcentrator.

The embedding achieving rectangle area of at most $6n^2+3n$ is shown in Figure 3.6. The directions used below refer to the figure. The nodes are embedded in one vertical column of the grid -- one node every three grid points. There is a column for each edge, either to the left or the right of the column in which the nodes are embedded. The path representing an edge must reach the column for the edge from each node representing an endpoint of the edge. Therefore, each path includes two horizontal

Figure 3.5: Reducing the degree of nodes in a superconcentrator.

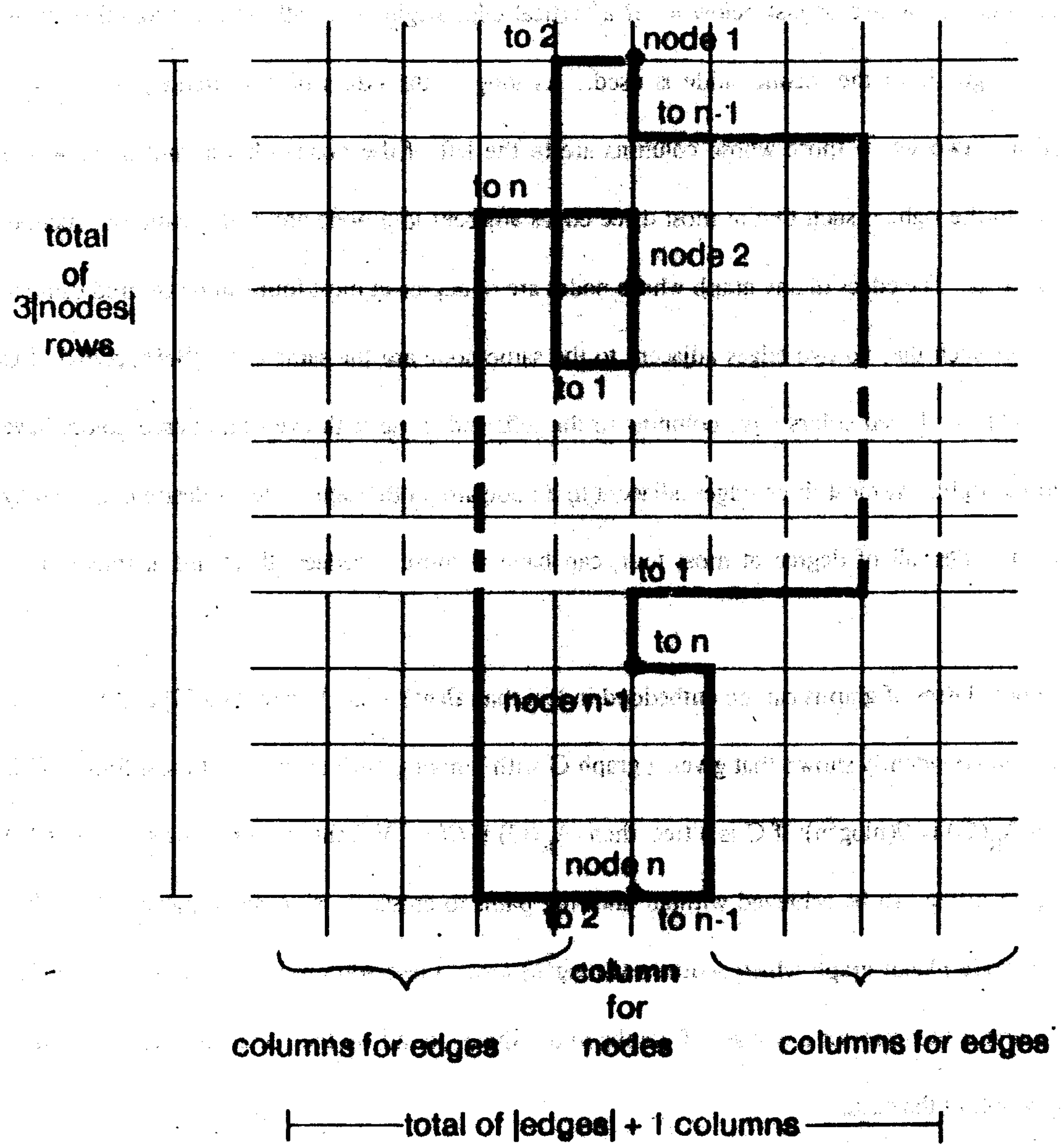


d is the degree of the nodes.

— — — indicates a path.

All sets of node disjoint paths are preserved.

Figure 3.6: Embedding an arbitrary graph in $O(n^2)$ area.



segments between the column containing the embedded nodes and the column for the edge. The first (last) segment of the path is either one of these segments or a vertical edge from the endpoint of the path to the grid node just above or just below it. If a vertical edge begins the path, the horizontal segment going left or right from the second node is used. As long as the edges of the circuit graph can be partitioned into two sets -- those whose columns are to the left of the node column, and those whose columns are to the right -- such that at most three edges adjacent to a node are in the same set, such an embedding exists. The edges of any graph whose nodes are of degree at most four can be colored using at most six colors such that no two edges adjacent to the same node are the same color [Be66, p.262]. Let edges with odd numbered colors have columns to the left, and those with even numbered colors have columns to the right. At most three edges adjacent to a node are on the same side, as desired. Since any graph with n nodes, all of degree at most four, can have at most 2n edges, there are at most 2n+1 columns.

Other classes of graphs can be embedded in less than $\Omega(n^2)$ area. Valiant [Val79] and Leiserson [Lei80] have independently shown that given a graph G with n nodes, each of degree at most four, if G is planar, then $A_R(G)$ is $O(n\log^2 n)$; if G is a tree, then $A_R(G)$ is $O(n)$. Valiant actually shows that an $O(n)$ embedding for a tree can be achieved without allowing paths to cross. It is an open question whether there is an n node planar graph which requires $\Omega(n\log^2 n)$ area. Leiserson proves a general result which relates a separator theorem for any class of graphs to an upper bound on the rectangle area required to embed any graph of the class.

The results which we have reviewed above illustrate that the graph embedding model is useful for proving bounds on layouts for particular classes of graphs and for proving time/space bounds for function implementation. Using it, we can identify easy and hard interconnection patterns to route. In the rest of this dissertation we will be interested in algorithms to actually do the layout. Therefore, we will use the rectangle model described in Section 3.1.

Chapter 4: Complexity of Layout Problems

Regardless of the exact formulation of the layout problem, we are interested in finding an efficient algorithm which computes an optimal layout. If this is not possible, we would like an efficient algorithm which computes an optimal layout much of the time and a good layout the rest of the time. This algorithm may actually be a collection of algorithms to solve subproblems which together give a layout. Again, we would like the algorithms for the subproblems to find solutions efficiently. However, most problems associated with circuit layout are NP-complete. The definition of an NP-complete problem is given below. From a practical point of view, the NP-completeness of a problem indicates that it is probably impossible to find an efficient algorithm which solves the problem.

4.1 NP-Completeness and Heuristic Algorithms

Two major classes of problems in complexity theory are the classes P and NP. A problem is in P (NP) if there is a deterministic (nondeterministic) Turing machine and a polynomial P such that the Turning machine solves any instance of the problem with an input of length n in a number of steps no greater than $P(n)$. The *length* of an input is the length of its representation as a character string in a predetermined character set. We will not formally define Turing machines here. The interested reader should see [Ah74]. Very often, a nondeterministic Turing machine solves a problem by "guessing" a solution and testing to see if its guess is actually a solution. When the problem of interest is a minimization (or maximization) problem, it is reformulated so that candidate solutions can be tested independent of each other. In the new problem, a parameter k is part of the input. A solution to the new problem is a feasible solution to the old problem for which the quantity to be minimized (maximized) is less than (respectively greater than) k. A feasible solution is one which satisfies all requirements of the old problem except optimality. For example, if the original problem is to find a minimum area layout of a circuit, the new problem, given the circuit and parameter k, is to find a layout of the circuit of area less than k. Under this formulation, a feasible solution can be tested by criteria depending only on the feasible solution to see if it is a solution. When the minimization formulation of the problem is used, feasible solutions must be compared against each other to find the actual solutions.

The number of steps taken by a deterministic Turing machine is polynomially related to the number of steps taken under a model of computation that corresponds to the instruction set of a computer if the lengths of numbers operated on is taken into account [Ah74]. Therefore, any problem which is in P can be solved in a polynomial number of steps by an algorithm in a high level programming language. The number of steps executed by an algorithm is referred to as the time taken by the algorithm. We would like all the problems we need to solve to be in P.

The question of whether P = NP is one of the major open questions in complexity theory. It is believed that P ≠ NP. Problems in NP have been found whose membership in P implies NP = P. These problems are called *NP-complete* problems. A problem is *NP-hard* if for each problem in NP, there is a polynomial *P* and a transformation computable by a deterministic Turing machine in a polynomial number of steps which transforms an instance of the problem in NP with an input of length n to an instance of the NP-hard problem with an input of length *P*(n). An NP-complete problem is one which is NP-hard and is in NP. There are no known deterministic algorithms for NP-complete problems which take a number of steps polynomial in the length of the input. The fact that such well studied problems as integer programming and the travelling salesman problem are NP-complete [Gar79] strongly suggests that NP ≠ P. Therefore, proving a problem NP-complete is very strong evidence that any algorithm which solves the problem will be time consuming.

The most common way to prove that a problem is NP-complete is to find a reduction from a known NP-complete problem to the new problem which can be executed in deterministic polynomial time. Then, since the composition of polynomials is still a polynomial, all problems in NP can be reduced through the known NP-complete problem to the new problem in deterministic polynomial time.

When a problem has been proven NP-complete, the usual course is to try to find an algorithm which runs in polynomial time and finds a solution most of the time. When the problem proven NP-complete is the parameterized version of an optimization problem, an algorithm which can find solutions close to optimal, if not optimal is desired. Heuristics are developed which can direct the algorithm towards a good solution. When the algorithm is searching through a large (exponential) number of possible solutions, heuristics remove large numbers of possible solutions based on the likelihood that none will be optimal solutions.

Very often a heuristic algorithm is tested and validated by empirical evidence. All of the algorithms discussed in Chapter 2 are heuristic algorithms for placement and routing. All have been judged by comparing the solutions they produce to manually produced solutions for the same problems. These algorithms are also compared to each other to judge their worth. In contrast, in this thesis, heuristic algorithms for optimization problems are judged by the relationship of the solutions they produce to optimal solutions. Consider a layout problem in which minimum area is desired. Let $\text{area}_{alg}(C)$ be the area of the layout for circuit C found by a particular algorithm. Let $\text{area}_{opt}(C)$ be the minimum area layout of C. Then define the *worst case performance* of the algorithm, denoted $\text{wc}_{alg}(n)$, as the maximum over all circuits of size n of $\text{area}_{alg}(C)/\text{area}_{opt}(C)$. Define the *average case performance of the algorithm*, $\text{avg}_{alg}(n)$, as the average over all circuits of size n of $\text{area}_{alg}(C)/\text{area}_{opt}(C)$, where the average is taken with respect to a predetermined distribution of circuits. The size of a circuit can be defined in various ways depending on the circuit model, e.g. the number of terminals, or the sum of the number of components and the number of nets. The size should be defined so that the length of the input specifying the circuit to the algorithm is polynomial in the size.

Average case performance is more likely to correspond to the observed performance of an algorithm, especially if the average is taken over "realistic" circuits. However, it is often very difficult to analyze. In this dissertation, the analysis is limited to worst case performance. If a lower bound on

$area_{opt}$ and an upper bound on $area_{alg}$ can be derived, an upper bound on wc_{alg} can be concluded. Ideally, a bound of $1+\epsilon$ for small ϵ is desired. In reality, we are happy with any constant bound. However, as will be seen in Section 4.4, even constant bounds are not achieved by some common algorithms.

In the next section, we will review NP-completeness results for problems related to layout. In Section 4.3, we will consider two subproblems of channel routing and show that they are both NP-complete. In Section 4.4 we will analyze a heuristic algorithm for one of the problems shown NP-complete in Section 4.3.

4.2 Placement and Routing: NP-complete Formulations

In this section we consider the complexity of the problems resulting from the decomposition of the layout problem into placement and routing.

4.2.1 The point model: some known results.

First consider the placement problem used for printed circuit boards and standard cells. Recall that components are modeled as points and total wire length is the quantity to be minimized by the layout. The quadratic assignment problem is one formulation of placement:

Quadratic Assignment Problem

Given: Locations 1 through m and distance matrix $\{d_{kh} | 1 \leq k,h \leq m\}$ between locations; components 1 through n and connection matrix $\{c_{ij} | 1 \leq i,j \leq n\}$ between components.

Find: A one-to-one mapping, p, of components to locations such that
COST(p) = sum over $i \neq j$ from 1 to n of $(c_{ij} d_{p(i)p(j)})$ is minimized.

Sahni and Gonzales [Sah76] prove that the parameterized quadratic assignment problem is NP-complete. In fact, they prove that unless NP = P, there is no approximation algorithm for quadratic assignment for which there is an $\epsilon > 0$ such that for all instances of the problem:

$$\mathrm{COST}(p_{alg})/\mathrm{COST}(p_{opt}) \leq 1 + \epsilon.$$

The proof does rely on instances of the problem with connection matrix values c_{ij} greater that ϵn. Consider only instances of the problem for which the c_{ij} are limited to a bounded range of values. The proof that the existence of an approximation algorithm for which the ratio of $\mathrm{COST}(p_{alg})$ to $\mathrm{COST}(p_{opt})$ is no more than $1 + \epsilon$ implies NP = P no longer holds for all ϵ! However, the restricted problem is NP-complete as long as the c_{ij} are allowed to take on the value 0 or 1.

The quadratic assignment problem is a formulation of the placement problem in which all point-to-point connections are specified. The cost c_{ij} represents the number of connections between components i and j. The distance d_{kh} is the estimate of the wire length needed to connect terminals at locations k and h. If a placement problem formulation in which an estimate of the length of wire needed to connect whole nets is used, the quadratic assignment problem is a special case in which all nets are of size two [Sah80]. Therefore, the quadratic assignment problem reduces to this formulation of the placement problem. It follows that this formulation of the placement problem in NP-complete.

Let us now turn to routing. If a Steiner tree interconnection pattern is desired for each net, then even finding the connection paths for one net is NP-complete. Formally:

The Steiner Tree Problem

Given: A set of points, P, in the plane with integer coordinates.

Find: A set of integer points, Y, such that the minimum length spanning tree of $P \cup Y$ is minimal over all sets of integer points containing P. One of two measures of distance may be used:
(i) The discretized Euclidean length: $\lceil((x_1-x_2)^2+(y_1-y_2)^2)^{\frac{1}{2}}\rceil$ where (x_1,y_1) and (x_2,y_2) are the points. The problem is then the Discretized Euclidean Steiner Tree Problem;
(ii) The rectilinear metric: $|x_1-x_2| + |y_1-y_2|$, giving the Rectilinear Steiner Tree Problem.

For either metric, the Steiner tree problem is NP-complete. If the standard Euclidean metric is used, the problem is NP-hard, but is not known to be in NP [Gar79].

The minimum spanning tree using either metric can be solved in polynomial time [Ah74]. For the rectilinear metric, the length of the minimum spanning tree is at most 3/2 the length of a minimal Steiner tree [Hw78]. Therefore, any algorithm to find minimum length spanning trees is a heuristic algorithm for finding minimum length Steiner trees with a worst case ratio of $length_{alg}$ over $length_{opt}$ of 3/2. A discussion of heuristic algorithms for the Rectilinear Steiner Tree Problem can be found in [Hw78].

The above two problems apply to the model of layout in which components are points and minimum length wiring is desire. Other problems related to this model are also NP-complete. Ting et. al. [Ti79] show that a via assignment problem encountered in their approach to routing is NP-complete. A summary of a number of NP-complete problems associated with this model can be found in [Sah80].

4.2.2 The rectangle model: a new result.

The above NP-complete results do not directly apply to the model of layout in which components are rectangles and minimum area is desired. We shall now prove that even when no interconnections are needed, the placement of rectangular components to minimize area is NP-complete. Since this is a special case of the layout problem when interconnections are required, the more general layout problem is NP-complete. The proof we present below does require some extra assumptions in addition to those given in the description of our model in Chapter 3. All components and the circumscribing rectangle determining the area must be oriented so that each of their sides is in the direction of one of two perpendicular axes. This does put some restrictions on the placements allowed but is consistent with the concept of "horizontal" and "vertical" being special directions which wires follow. We also restrict all points and dimensions to be integer valued so that the problem has discrete solutions.

Problem P1: Discrete layout with no interconnections

Given: A set of n rectangles and an integer, A. For $1 \leq i \leq n$, each rectangle r_i has dimensions h_i and w_i which are positive integers.

Question: Is there a placement of the rectangles on the plane with a cartesian coordinate system imposed so that:

(i) Each boundary is parallel to one of the coordinate system axes;

(ii) Corners of the rectangles lie on integer points in the plane;

(iii) No two rectangles overlap;

(iv) The boundaries of any two rectangles are separated by at least a unit distance;

(v) There is a rectangle in the plane which circumscribes the placed rectangles, has boundaries parallel to the axes, and is of area at most A. The boundary of the circumscribing rectangle is allowed to contain boundaries of placed rectangles.

Lemma 4.1: Problem P1 -- discrete layout with no interconnections -- is NP-complete.

Proof: Consider only placements for which the lowest leftmost corner of any rectangle is at (0,0). All other placements are just translations of these. The coordinates of the lower left corner of each rectangle and the orientation of the rectangle, i.e. whether the side of length h_i is in the x direction or the y direction, determine its position. Since the coordinates of each lower left corner can only take on integer values between 0 and $A-h_i-1$, there are at most A^2 choices for each point. A nondeterministic Turing machine can guess a possible placement and write it down. Given a placement, conditions (i) through (v) can be checked deterministically in polynomial time. This shows that P1 is in NP.

The proof that P1 is NP-hard is accomplished by reducing the Bin Packing Problem to it. Since the Bin Packing Problem is NP-complete [Gar79], this proves that P1 is NP-hard.

The Bin Packing Problem:

Given: A set of n items, each of size c_i a positive integer; also, positive integers B and C, the number of bins and bin capacity, respectively.

Question: Is there an assignment of items to bins so that for each k, $1 \leq k \leq B$, the sum of c_i over all items assigned to bin k does not exceed C?

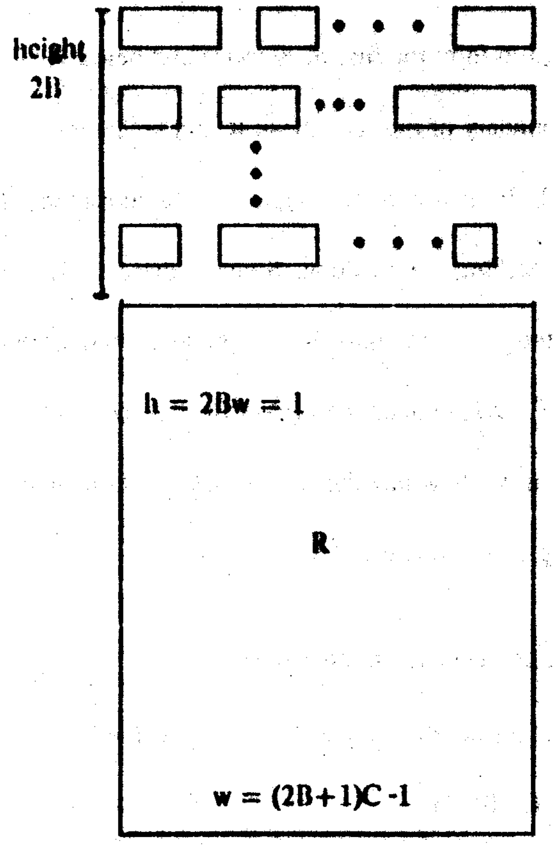
Given any instance of the Bin Packing Problem, we will construct an instance of P1 as follows. There will be n+1 rectangles. One, called R, will be of size h by w, where $w = (2B+1)C-1$ and $h = 2Bw+1$. The remaining rectangles will directly correspond to items. Rectangle r_i will have dimensions $h_i = (2B+1)c_i-1$ and $w_i = 1$. The bound on area will be $A = wh+2Bw$. Note that the length of the input to the Bin Packing Problem is $\Omega(n + \log C + \log B)$. The dimensions of the rectangles and A can be calculated taking no more than polynomial time in this length.

We must show that there is a placement satisfying conditions (i) through (v) if and only if there is an assignment of the items to B bins without exceeding capacity C in any bin. Given a bin packing, Figure 4.1 illustrates a satisfactory placement. We now argue that any placement satisfying (i) through (v) corresponds to a legal bin packing. Figure 4.2 will illustrate. Without loss of generality, let the side of rectangle R of dimension h be in the y direction. Let "left" denote towards lower numbers in the x direction and "right" denote towards higher numbers; let "above" denote towards higher numbers in the y direction and "below" denote towards lower numbers. At any point, the width of the layout in the x direction is strictly less than w+1. If not, then area $\geq h(w+1) = hw+2Bw+1 > A$. Therefore, all rectangles, r_i, must lie above or below R. None can extend a unit beyond the length h sides of R, so that for each rectangle, some line in the y direction intersects the rectangle and R. No rectangle, r_i, is oriented so that its long side is in the y direction. Otherwise, the dimension of the circumscribing rectangle in the y direction would be at least $h+(2B+1)c_i-1+1$, where the added 1 accounts for the separation between rectangle boundaries. Then:

$$\text{area} \geq w(h+(2B+1)c_i) \geq w(h+2B+1) > A \qquad \text{and (v) is not satisfied.}$$

We now know that all rectangles, r_i, are oriented so that their long sides are in the x direction. Any placement can be modified slightly without increasing the area so that the r_i form rows above and

Figure 4.1: A placement given a bin packing.



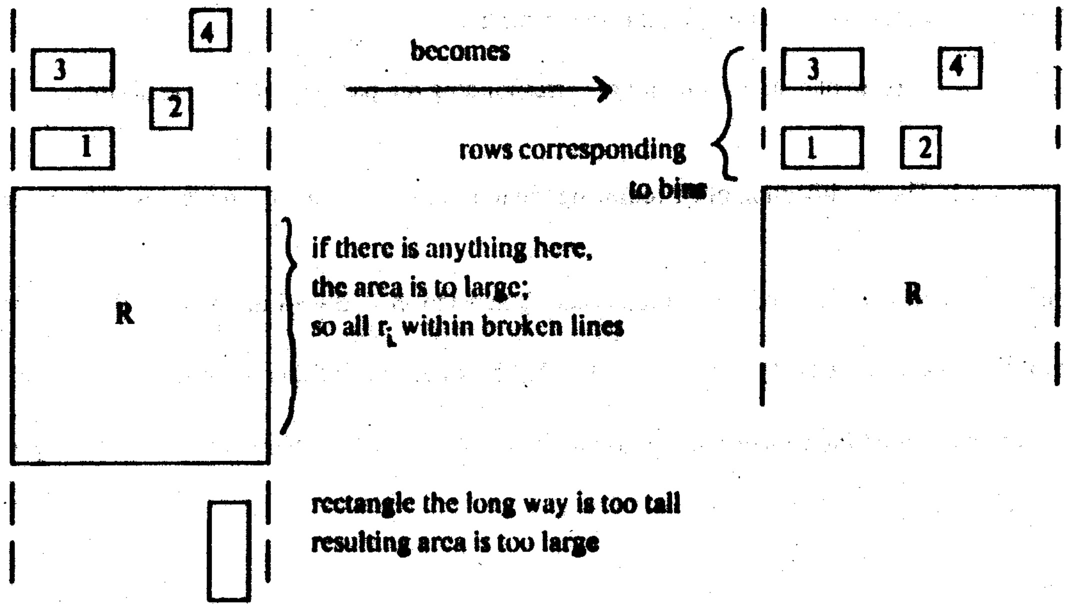
Each row contains rectangles corresponding to items in one bin. Adjacent rectangles are separated by one unit. Since the c_i of the bin sum to at most C, the length of the row is at most

$(\Sigma((2B+1)c_i-1)) + (\#$ of rectangles in the row -1)
rectangles
in the row

$\leq (2B+1)C - 1$

area = w(h+2B) = A

Figure 4.2: A packing given a placement.



becomes

rows corresponding to bins

if there is anything here,
the area is to large;
so all r_i within broken lines

rectangle the long way is too tall
resulting area is too large

there may be r_i s at the top and at the bottom below R. (Any r_i and r_j less than two units above or below R must be separated from each other by at least one unit to the left and right. These can be shifted to form the first rows above and below R. The next rows are formed analogously above the upper boundary of the row above R and below the lower boundary of the row below R. Figure 4.2 illustrates.) Each row of rectangles can be considered the packing of a bin. If the rows correspond to a legal bin packing, we are done. Suppose there are K rows. Then the dimension of the circumscribing rectangle in the y direction must be h+2K, since there must be a unit space separating rows from each other and R. If K > B, then area > w(h + 2B) = A, contradicting (v). Therefore, at most B bins are used. It remains for us to show that the sum of sizes c_i of the items in any bin is at most C. Each item corresponds to a rectangle with $h_i = (2B+1)c_i-1$.

$$\sum_{\text{rectangles in row}} ((2B+1)c_i-1) + \text{the number of rectangles in the row} - 1$$

$$\leq \text{the width of a row of rectangles} < w+1 = C(2B+1)$$

giving $\quad (2B+1)(\text{sum of } c_i \text{ for } r_i \text{ in row}) - 1 < C(2B+1)$

Therefore, $\quad (\text{sum of } c_i \text{ for } r_i \text{ in row}) < C + 1/(2B+1)$.

Since all c_i and C are positive integers, the above implies

$(\text{sum of } c_i \text{ in one bin under corresponding bin packing}) \leq C$ as desired. □

Corollary 4.1: The modification of P1 removing the minimum spacing requirement is NP-complete.

Proof: The same proof is used. Rectangle R has dimensions h = Bw+1 and w = C(B+1). For each i, r_i has dimensions $h_i = c_i(B+1)$ and $w_i = 1$. The NP-hardness part of this proof does not require that any of the dimensions be integers or that the rectangles be placed so that their corners are on integer points of the coordinate system. □

Lemma 4.1 is presented with spacing required between rectangles to closely mirror the problem in circuit layout. The dimension of each component can be increased by one unit to account for the required spacing implicitly. This will cause the circumscribing rectangle to be one unit too large in each dimension. When spacing is not explicitly required, the problem remains NP-complete even if the aspect ratio of the circumscribing rectangle is bounded.

Problem P2-α: Layout with bounded aspect ration and no interconnections

Given: A set of n rectangles and a positive number A. For $1 \leq i \leq n$, each rectangle, r_i, has dimensions h_i and w_i.

Question: Is there a placement of the rectangles on the plane with a cartesian coordinate system imposed so that:

(i) Each boundary is parallel to one of the coordinate system axes;

(ii) No two rectangles overlap;

(iii) There is a rectangle in the plane which circumscribes the placed rectangles, has boundaries parallel to the axes, is of area at most A, and has aspect ratio (long side)/(short side) at most α, where α is a rational number not less than one. The boundary of the circumscribing rectangle is allowed to contain boundaries of placed rectangles.

Lemma 4.2: Problem P2-α is NP-hard for any α a rational number not less than one.

Proof: The proof is a reduction from bin packing similar to the proof of Lemma 4.1. Given c_i for $1 \leq i \leq n$, C, and B, construct R with $w = \alpha C(B+1)$ and $h = w/\alpha - B$. Each r_i is of dimensions $h_i = \alpha c_i(B+1)$ and $w_i = 1$. The bound on area is $A = w^2/\alpha = \alpha C^2(B+1)^2$. The aspect ratio implies that the larger side of the desired circumscribing rectangle is at most $(\alpha A)^{1/2} = w$. Therefore, assuming R is oriented as in the proof of Lemma 4.1, none of the r_i can lie to the left or right of R. The rest of the proof is analogous to that for Lemma 4.1 and is left to the reader. □

Corollary 4.2: If the bound on aspect ratio, α, is allowed as an input for problem P2-α, the problem remains NP-hard.

Proof: The construction in the proof of Lemma 4.2 can be computed in time polynomial in the length of the representation of α; therefore, α can be an input. □

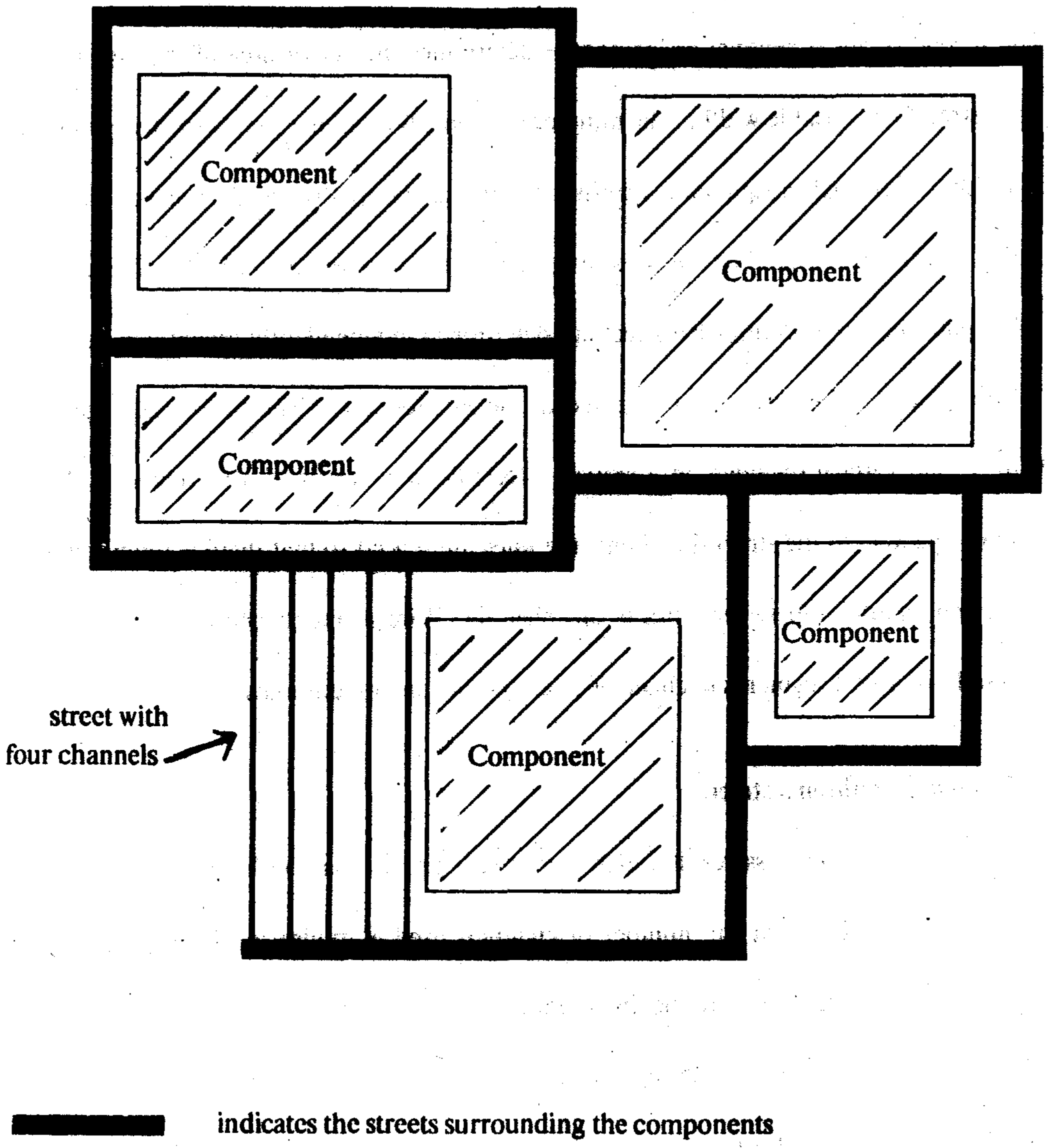
Lemmas 4.1 and 4.2 prove that the layout problem we are studying is NP-complete even in the degenerate case when only placement is required. Given a placement of components, is the routing problem NP-complete? We do not have a proof of a general NP-completeness result for routing. However, in the next section, we will present two NP-completeness results for subproblems encountered in channel routing.

4.3 NP-completeness in Channel Routing

In this section, we will prove that two problems encountered in a channel routing approach are NP-complete. Recall that in the channel routing approach, the routing area is divided into horizontal and vertical streets. Terminals lie along the sides of the streets. Each street is made up of a set of parallel channels in the direction of the street. Each channel is wide enough to account for the width of a wire and the required separation between wires. First the interconnection pattern of paths through the streets is chosen (street routing). Then, within each street, the actual routes of the path segments assigned to the street -- using the channels -- is determined (channel assignment). The goal is to minimize the overall layout area. (See Figure 4.3.)

The street routing problem can be represented as a graph problem. There will be a node in the graph for each position along a street at which there is are terminals. (Two terminals directly across from each other on a street are represented by one node.) There will also be a node in the graph for each intersection of streets. Each edge of the graph represents a portion of a street between two positions at which there are terminals or intersections. For each net, we wish to find a subgraph which is a tree whose nodes consist of the nodes representing terminals of the net and nodes representing street intersections. This is a Steiner tree problem on the graph. The intersection nodes in the graph are analogous to the

Figure 4.3: Streets and channels in the channel routing approach.



added nodes in the plane for a Rectilinear or Euclidean Steiner tree problem. However, we do not wish to find a minimum length Steiner tree in the graph for each net. Using a minimum length Steiner tree for each net does not necessarily yield a minimum area layout. The eventual area of a layout must be estimated when the Steiner tree for each net is being chosen.

The second phase -- channel assignment -- determines the actual area of the layout. We are assuming each street is of variable width. The number of channels used in a street directly corresponds to the street width. Channel assignment determines the actual paths in the plane realizing the interconnection pattern determined by street routing. The paths are composed of horizontal and vertical segments. The path segments within each street are determined independently except that the segments for paths which change streets must be connected at the intersections. The horizontal segments in a horizontal street (and vertical segments in a vertical street) lie in channels. Each channel is the region between two lines parallel to the street direction; the lines are spaced so that there is room for one wire width and required separation between wires in any channel. Wire segments perpendicular to the street direction are used to connect segments in channels to each other and to terminals.

4.3.1 Channel assignment within a street.

The first NP-completeness result which we present proves that, with certain restrictions, the routing of paths in a street so that the number of channels used is minimized is NP-complete. The restrictions are that (i) all terminals to be interconnected lie in the street (i.e. there are no street intersections to worry about), and (ii) each set of wires interconnecting one net uses exactly one channel. This problem is represented as follows.

Problem P3: Channel Assignment within a Street

Given: A line segment, S, containing equally spaced points numbered 1 through h, and a set of terminals, T. The line segment represents the street. Each terminal, t, is an ordered pair (i,b), where i is a number between 1 and h indicating the position of the terminal along the street and b is an element of {0,1} representing the side of the street on which the terminal lies. For any two terminals (i,b_1) and (j,b_2) with $b_1 = b_2$, $|i-j| \geq s$, where s is a positive integer chosen a priori. Integer s represents the wire width plus required separation. Also given are a collection of n disjoint sets of terminals, N_i for $1 \leq i \leq n$, representing nets, and a parameter k.

Question: For each net, N_i, let a_i be the position of the terminal of lowest position in N_i and z_i be the position of the terminal of highest position. Let $[a_i,z_i]$ denote the set of all points on S between a_i and z_i inclusive. Is there a mapping, ch, which assigns each net a number between 1 and k inclusive such that for any N_i and N_j, $1 \leq i,j \leq n$:

(i) If terminal $(x,0) \in N_i$ and terminal $(y,1) \in N_j$ and $|x-y|<s$, then $ch(N_j)<ch(N_j)$;

(ii) If $[a_i,z_i] \cap [a_j,z_j]$ is not empty, then $ch(N_i) \neq ch(N_j)$.

The mapping, ch, represents the assignment of a wire segment between points a_i and z_i to one of k channels for each net N_i. The interval $[a_i,z_i]$ is called the interval of net N_i. Segments perpendicular to the direction of the street are assumed to go from each terminal in the net N_i to the segment in the channel. The restriction that these segments must not overlap is represented as condition (i) above.

If there are no terminals satisfying the hypothesis of condition (i), i.e. condition (ii) is the only relevant condition, then Problem P3 is the interval coloring problem. The interval coloring problem is: given a finite set of intervals on a line and a positive integer, k, assign a color (positive integer) to each interval so that no two overlapping intervals have the same color and no more than k colors are used. Nets define intervals, and "channel" is just another name for "color". The interval coloring problem can be solved by a polynomial time algorithm [Gav72] [Has71]. The solution to this problem uses the same number of channels as the maximum over all points on S of the number of nets whose interval $[a_i,z_i]$ intersects the point. Therefore even if we allow wires for one net to use more than one channel, the solution found using one channel is optimal.

Without additional restrictions, the channel assignment problem stated as Problem P3 is NP-complete.

Lemma 4.3: Problem P3 is NP-complete.

Proof: In polynomial time, a nondeterministic Turing Machine can guess a mapping and check to see if (i) and (ii) are satisfied. Therefore, the problem is in NP.

We will reduce the circular arc coloring problem, an NP-complete problem [Gar], to Problem P3 to prove that P3 is NP-hard. The circular arc graph coloring problem is similar to the interval coloring problem except that arcs on a circle rather than intervals on a line are usc. Figure 4.4 illustrates.

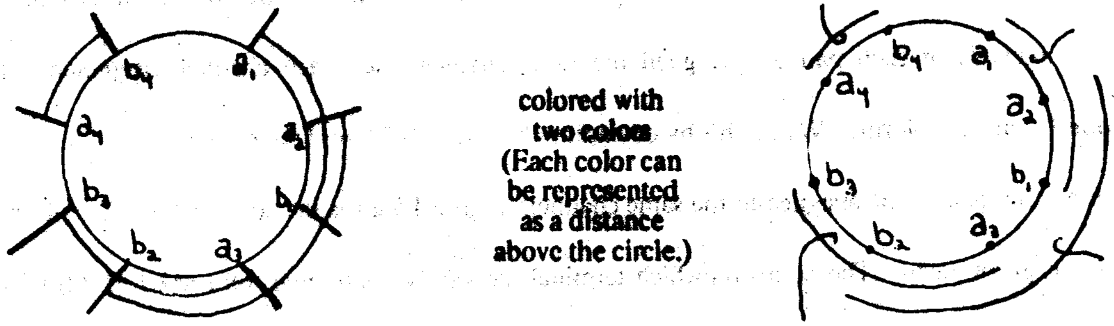
The Circular Arc Coloring Problem
Given: A finite set of arcs of a circle and a positive integer, k.

Question: Is there an assignment of colors numbered 1 though k to the arcs such that any two arcs which overlap are assigned different colors? Arcs which intersect only at their endpoints are not considered as overlapping.

Since arcs which intersect at endpoints are not overlapping, we may modify any set of arcs so that no arcs have endpoints in common (see Figure 4.4). The actual length of the arcs is irrelevant. Therefore, for n arcs, we can number the endpoints from 1 through 2n while traversing the circle in some direction. Each arc will be represented as an ordered pair, (i,j), listing the endpoints of the arc as encountered in the traversal of the circle.

Given an instance of the circular arc coloring problem with n arcs and k colors, we will produce an instance of the channel assignment problem with k channels and $(2n+c(2k-c+1))$ nets, where c is the number of arcs which contain arc (2n,1). Each net will contain exactly two terminals. Intuitively, we cut the circle between points 2n and 1 and stretch it out straight. This results in $n+c$ intervals, since c arcs have been cut in two. The n-2c intervals which do not have endpoints on the cut line will become the intervals of nets. Considering only these nets, any legal assignment of channels will be a legal assignment of colors to the corresponding arcs and vice versa.

Figure 4.4 Circular Arc coloring.

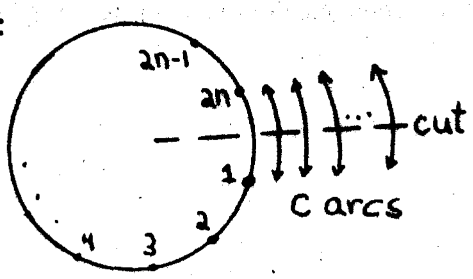


colored with two colors (Each color can be represented as a distance above the circle.)

arcs (a_1,b_1); (a_2,b_2); (a_3,b_3):(a_4,b_4) traversing clockwise.

Figure 4.5 Construction of a channel assignment problem given a circular arc coloring problem.

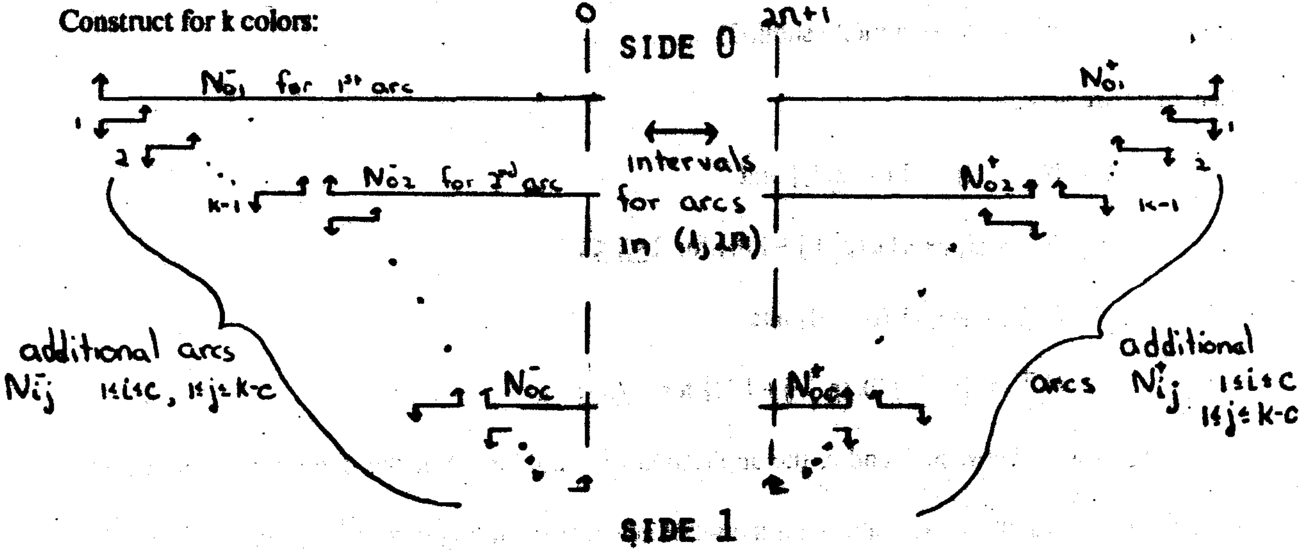
Given:



Key:

↲——↑ represents the interval of a net. Arrows point to the side of the street containing the terminal.

Construct for k colors:


0
SIDE 0
2n+1
N^-_{01} for 1st arc
N^+_{01}
1
2
k-1
N^-_{02} for 2nd arc
N^+_{02}
intervals for arcs in (1,2n)
additional arcs N^-_{ij} $1\le i\le c$, $1\le j\le k-c$
N^-_{0c}
N^+_{0c+1}
additional arcs N^+_{ij} $1\le i\le c$ $1\le j\le k-c$
SIDE 1

There are 2c intervals with endpoints on the cut line -- two intervals per arc. If we can insure that both intervals of each pair are assigned the same channel, then each channel assignment will correspond to an arc coloring. We do this by extending the intervals beyond the cut line and adding nets which force the pairs to be assigned to the same channel. Figure 4.5 gives the construction. The formal definition is given below. The points on which terminals lie will be numbered from $-c(k-(c-1)/2)+1$ to $2n+c(k-(c-1)/2)$ rather than from 1 to $2n+c(2k-c+1)$.

For each arc (a,z) which does not contain (2n,1), there will be a net {(a,0),(z,1)}. Order the arcs which do contain (2n,1) by increasing starting point. For the i^{th} such arc, there will be $2(k-i+1)$ nets -- half have terminals within the negatively numbered points and half have terminals within the points numbered above 2n. For arc (a_i, z_i), $1 \leq z_i < a_i \leq 2n$, the i^{th} arc containing (2n,1), the nets are defined as follows. For $1 \leq i \leq c$, let

$$\mathrm{sum}(i) = \sum_{j=0}^{i-1} (k-j) = i\,(k-½(i-1))$$

$$p_i^- = -\mathrm{sum}(c) + \mathrm{sum}(i-1)$$

$$p_i^+ = 2n + \mathrm{sum}(c) - \mathrm{sum}(i-1)$$

Then, there are nets:

$$N_{i0}^- = \{(p_i^-+1,0), (z_i,1)\} \text{ and}$$

$$N_{ij}^- = \{(p_i^-+j,1), (p_i^-+j+1,0)\} \text{ for } 1 \leq j \leq k-i;$$

$$N_{i0}^+ = \{(a_i,1), (p_i^+,0)\} \text{ and}$$

$$N_{ij}^+ = \{(p_i^+-j,0), (p_i^+-j+1,1)\} \text{ for } 1 \leq j \leq k-i.$$

For each i between 1 and c, the application of condition (i) in the definition of problem P3 results in two chains of k-i inequalities for the set of nets corresponding to the i^{th} arc containing (2n,1) -- one chain of inequalities $ch(N_{ij}^-) < ch(N^-_{ij+1})$ and one of inequalities $ch(N_{ij}^+) < ch(N^+_{ij+1})$ for $0 \leq j \leq k-i-1$. Therefore, nets N_{10}^- and N_{10}^+ must be assigned channel 1 if no more than k channels are to be used. Nets N_{20}^- and N_{20}^+ can be assigned channel 1 or 2 without violating condition (i), but their intervals overlap those for nets N_{10}^- and N_{10}^+, respectively. Therefore channel 2 is the only choice for nets N_{20}^- and N_{20}^+. Proceeding in this way, we see that nets N_{i0}^- and N_{i0}^+ must be assigned channel i if no more than k channels are to be used.

Given any channel assignment for the complete set of nets, we color the arcs which do not contain (2n,1) by the same numbered color as the corresponding channel. For the i^th^ arc which contains (2n,1), we use the number of the channel assigned to nets N_{i0}^- and N_{i0}^+. For any pair of arcs which overlap, there is at least one corresponding pair of nets whose intervals overlap. Therefore, any legal channel assignment corresponds to a legal coloring using the same number of colors as channel.

Given a coloring of the arcs using at most k colors, we can assign the nets to channels as follows. Permute the colors so that the i^{th} arc containing arc (2n,1) is assigned the i^{th} color. Assign nets N_{ij}^- and N_{ij}^+ to channel i+j, for $1 \leq i \leq c$ and $0 \leq j \leq k-i$. The remaining nets are assigned the same numbered channel as the color of the corresponding arc. Between points 1 and 2n, intervals overlap if and only if their corresponding arcs overlap. Elsewhere, no N_{ij}^- and N_{pq}^- for $i<p$ overlap except N_{i0}^- and N_{pq}^-. However, $ch(N_{i0}^-) = i < p+q = ch(N_{pq}^-)$, so no N_{ij}^- and N_{pq}^- which overlap are assigned the same channel. An analogous argument shows that the N_{ij}^+ are properly assigned to channels. Therefore, for each coloring, there is a legal channel assignment using the same number of channels as colors. □

Since the construction uses only nets with two terminals, we have:

Corollary 4.3: Problem P3 restricted to nets containing exactly two terminals is NP-complete.

4.3.2 Channel assignment with intersections.

The second problem which we prove NP-complete deals strictly with the ordering of paths in intersections so that the resulting street widths minimize the area. This problem is somewhat similar to the channel assignment problem in that if two paths are approaching an intersection from the same direction in a street, and one path needs to go left at the intersection and the other needs to go right, then they cannot share the same channel in the new street unless they are in the proper order when they reach the intersection. The problem will be modeled using a graph to represent the streets. Subgraphs which are trees will be used to represent the interconnection patterns resulting from street routing.

Let the street graph, S, be some subset of a two dimensional grid graph. Each node in S is labeled with integer coordinates (p,q). Each edge is either horizontal, i.e. between nodes (p,q) and (p+1,q), or vertical, i.e. between nodes (p,q) and (p,q+1). The graph S is partitioned into streets. Each street is a path in S using only vertical or only horizontal edges. A horizontal street goes between nodes (i,k) and (j,k) for some k and iKj; a vertical street goes between nodes (k,i) and (k,j) for some k and iKj. A node represents an intersection of two or more streets. An edge represents the portion of a street between two intersections. The interconnection pattern for each net is represented by a tree in S. Each tree, T, will be called a *net tree*. We would like to assign each occurrence of an edge in a net tree to a channel. Let ch be a mapping from each occurrence of an edge in a net tree to a positive integer. The integer indicates the number of the channel containing that edge in the street to which the edge belongs. We require:

(1) If edge e of S is in distinct trees T_1 and T_2, then the occurrence of e in T_1 is assigned to a different channel than the occurrence of e in T_2. (No overlapping wires.)

(2) If e_1 and e_2 are adjacent edges in a net tree T, and e_1 and e_2 belong to one street in S, then $ch(e_1) = ch(e_2)$. (A connection path cannot change channels within a street.)

(3) Suppose horizontal edges $e_1 = ((p-1,q),(p,q))$ and $e_2 = ((p,q),(p+1,q))$, for some p and q, belong to a street, s. Furthermore, suppose e_1 and e_2 belong to distinct net trees T_1 and T_2, respectively. If $ch(e_1) = ch(e_2)$, then:

(i) e_2 is not in T_1 and e_1 is not in T_2. (This follows from (1) and (2) above.)

(ii) Suppose that vertical edges ((p,q),(p,q+1)) and ((p,q-1),(p,q)) are in one street, and each of T_1 and T_2 contains at least one of the edges. Then $ch(x_1) < ch(x_2)$, where x_1 represents any occurrence in T_1 of either of the edges and x_2 represents any occurrence in T_2 of either of the edges. This condition insures that the wire segment represented by horizontal edge ((p-1,q),(p,q)) does not overlap the wire segment represented by horizontal edge ((p,q),(p+1,q)).

(4) Analogous to (3) but for vertical edges ((p,q-1),(p,q)) and ((p,q),(p,q+1)) in T_1 and T_2, respectively. If x_1 and x_2 are occurrences of horizontal edges ((p-1,q),(p,q)) and/or ((p,q),(p+1,q)) in T_1 and T_2, respectively, then $ch(x_1) < ch(x_2)$.

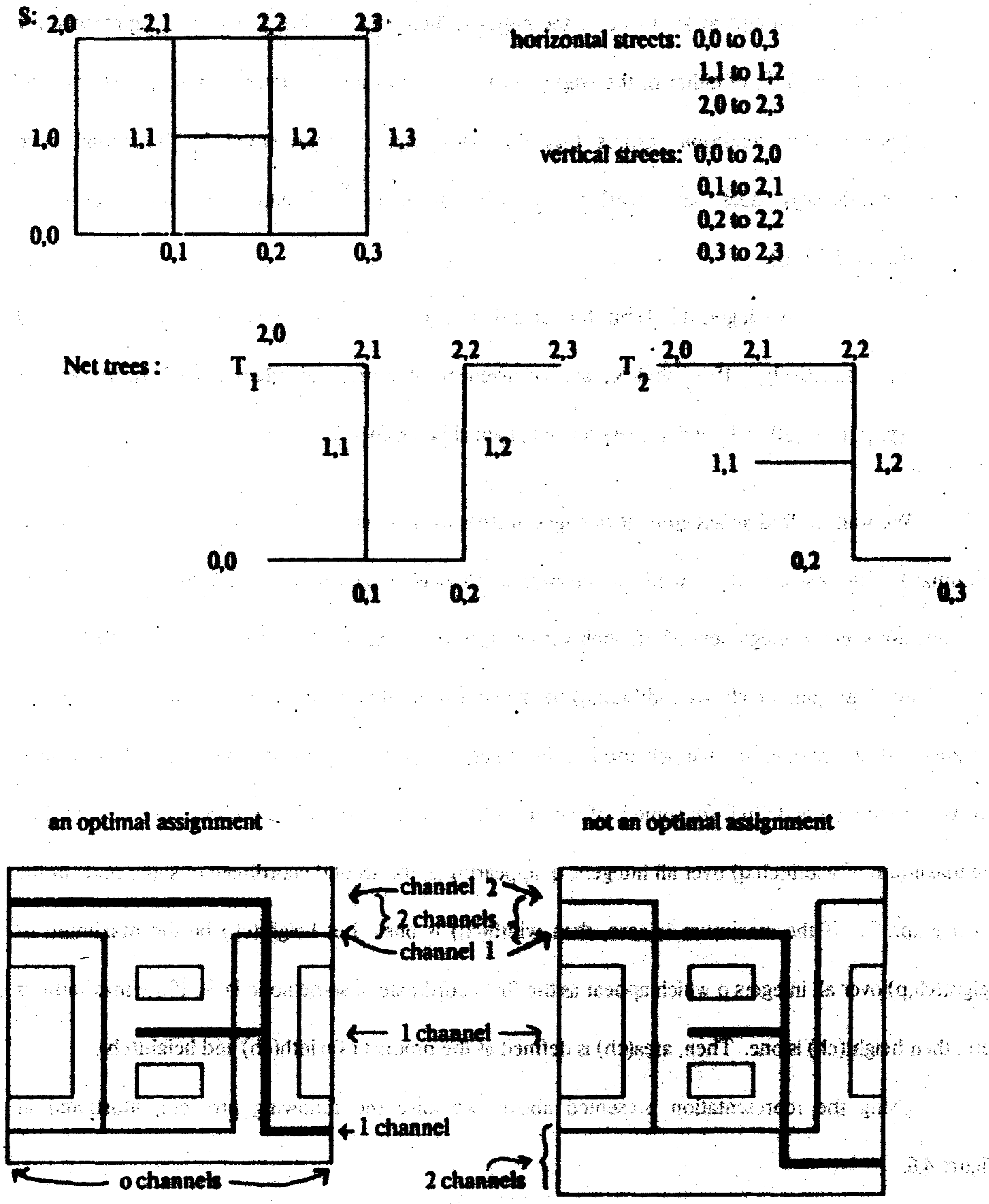
We want to find an assignment of edges in trees to channels so that the resulting overall area is minimized. The assignment is called *the intersection channel assignment* since the intersections induce the constraints on the assignment of channels within each street. Area will be measured as follows. For a given channel assignment, ch, let width(ch,q) be the sum over all vertical streets containing a node (i,q) for some i, of the number of channels used in the street; let height(ch,p) be the sum over all horizontal streets containing a node (p,j) for some j, of the number of channels used in the street. Let width(ch) be the maximum of width(ch,q) over all integers, q, appearing as the second coordinate of some node in the street graph S. If the maximum is zero, then width(ch) is one. Let height(ch) be the maximum of height(ch,p) over all integers p which appear as the first coordinate of some node in S. If the maximum is zero, then height(ch) is one. Then, area(ch) is defined as the product of width(ch) and height(ch).

Using the representation presented above, we have the following problem, illustrated in Figure 4.6.

Problem P4: The Intersection Channel Assignment Problem

Given: A street graph, S, partitioned into streets; a collection of net trees, T_i; and a positive integer, A.

Figure 4.6 The Intersection Channel Assignment Problem



Question: Is there an assignment, ch, of the occurrences of edges in net trees to channels in streets which satisfies conditions (1) through (4) above such that area(ch) $\leq$ A.

Lemma 4.4: The intersection channel assignment problem in NP-complete.

Proof: Conditions (1) through (4) and the area of an assignment can be checked by a nondeterministic Turing machine in polynomial time. Therefore, the problem is in NP.

We show that the problem in NP-hard by reducing 3-satisfiability [Gar79] to it:

3-Satisfiability

Given: A boolean expression composed of the conjunction of k clauses, c_i, for $1 \leq i \leq k$. Each clause is the disjunction of three distinct literals, where a literal is a boolean variable or its complement, i.e. $c_i = (y_{i1} \vee y_{i2} \vee y_{i3})$, where y_{ij} is x or $\neg x$ for some variable x.

Question: Is there an assignment of truth values to the boolean variables such that the expression is satisfied?

Given an instance of the 3-satisfiability problem with k clauses and v variables, we will construct an instance of the intersection channel assignment problem with 2k+1 horizontal streets, 2v+1 vertical streets and A = 2vk(3v+3). Let S be the (2v+1) by (2k+1) grid graph with nodes numbered from (0,0) through (2v,2k). Each path from (0,i) to (2v,i) is a horizontal street, for any i between 0 and 2k; each path from (i,0) to (i,2k) is a vertical street, for any i between 0 and 2v. Certain streets are associated with the clauses and variables of the boolean expression as follows:

(a) With each clause c_i, $1 \leq i \leq k$, associate the horizontal street from (0,2i-1) to (2v,2i-1) and name it C_i. The remaining horizontal streets are unnamed.

(b) For each variable x_j, $1 \leq j \leq v$, associate the vertical street from (j-1,0) to (j-1,2k), named X_j, and the vertical street from (2v-j+1,0) to (2v-j+1,2k), named X_{jc}.

(c) The one remaining vertical street, that from (v,0) to (v,2k), is named street M.

We construct two net trees for each variable. For variable x_j, the first net tree, T_j, contains the following edges:

(1) In street C_i, for each i from 1 to k: all edges on the path from (j-1,2i-1) to (2v-j+1,2i-1);

(2) In street X_j: the edge from (j-1,0) to (j-1,1) and for all even i between 1 and k-1 inclusive, the two edges on the path from (j-1,2i-1) to (j-1,2i+1). Also, if k is even, the edge from (j-1,2k-1) to (j-1,2k);

(3) In street X_{j0}: for all odd i between 1 and k-1 inclusive, the two edges on the path from (2v-j+1, 2i-1) to (2v-j+1, 2i+1). Also, if k is odd, the edge between (2v-j+1, 2k-1) and (2v-j+1, 2k);

(4) In street M: if variable x_j appears in clause c_i uncomplemented, then if i is even, the edge between (v,2i-1) and (v,2i); if i is odd, the edge between (v,2i-2) and (v,2i-1). If x_j appears in c_i complemented, then if i is even, the edge between (v,2i-2) and (v,2i-1); if i is odd, the edge between (v,2i-1) and (v,2i).

The second net tree for variable x_j, T_{j0}, contains the following edges:

(1)In street C_i, for each i from 1 to k: all edges on the path from (j-1,2i-1) to (2v-j+1,2i-1);

(2) In street X_j: for all odd i between 1 and k-1 inclusive, the two edges on the path from (j-1, 2i-1) to (j-1, 2i+1). Also, if k is odd, the edge between (j-1, 2k-1) and (j-1, 2k);

(3) In street X_{j0}: the edge from (2v-j+1,0) to (2v-j+1,1) and for all even i between 1 and k-1 inclusive, the two edges on the path from (2v-j+1,2i-1) to (2v-j+1,2i+1). Also, if k is even, the edge from (2v-j+1,2k-1) to (2v-j+1,2k);

(4) In street M: if variable x_j appears in clause c_i uncomplemented, then if i is odd, the edge between (v,2i-1) and (v,2i); if i is even, the edge between (v,2i-2) and (v,2i-1). If x_j appears

in c_i complemented, then if i is odd, the edge between (v,2i-2) and (v,2i-1); if i is even, the edge between (v,2i-1) and (v,2i).

The following observations are useful. For any coordinate p, call an edge from (p,2i-1) to (p,2i-2) an edge *down* from street C_i; call an edge from (p,2i-1) to (p,2i) an edge *up* from C_i. Each net tree T_j contains a path which begins with an edge in street X_j down from street C_1. The path goes through C_1 to street X_{j0}, goes up from C_1 to street C_2, and through C_2 to street X_j. The path continues snaking through the horizontal streets associated with the clauses, using street X_j to go from street C_i to street C_{i+1} when i is even, and using street X_{j0} to go from C_i to C_{i+1} when i is odd. Each net tree T_{j0} also contains a path which snakes through the C_i, but in the opposite direction: it begins with an edge in X_{j0} down from C_1; uses street X_{j0} to change from C_i to C_{i+1} when i is even; and uses street X_j to change when i is odd. For any net tree T_j or T_{j0}, if x_j appears uncomplemented in c_i, the edge in street M intersecting street C_i is in the same direction from C_i as the edge of the tree in street X_j; if x_j appears complemented, the edge in street M is in the same direction from street C_i as the edge of the tree in street X_{j0} (opposite to that in street X_j). Figure 4.7 gives an example of the construction.

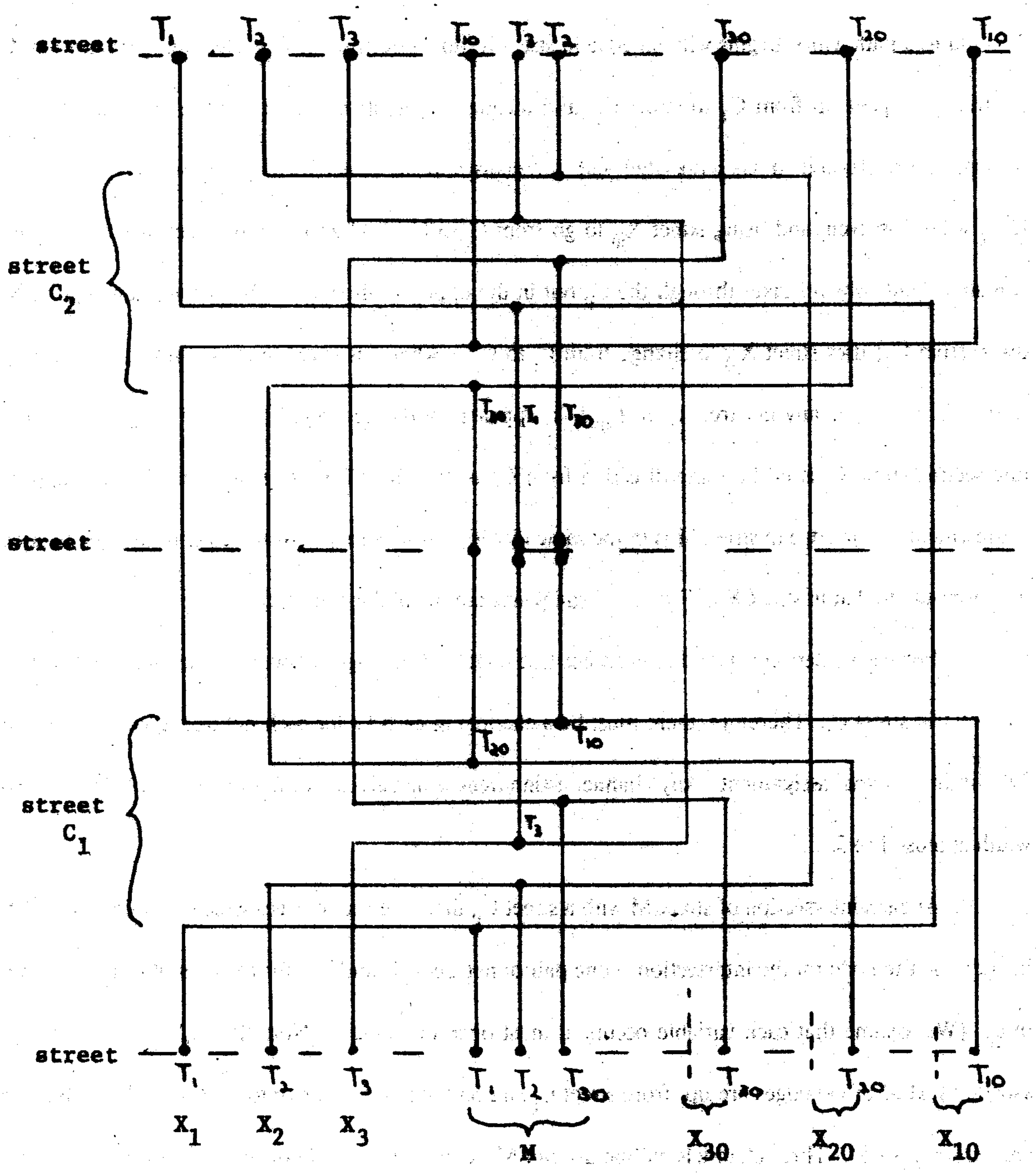
For each i between 1 and k, each net tree contains the edges between horizontal positions v-1 and v+1 in street C_i. Therefore, there must be a channel in each C_i for each net tree, giving a height of 2vk for any channel assignment. Any channel assignment which gives area at most 2vk(3v+3) must give width at most 3v+3.

At each intersection of street M with a street C_i, there are six net trees which contain edges of M incident on the node for the intersection -- one pair of net trees, T_j and T_{j0}, for each variable x_j appearing in c_i. (We assume that each variable occurs at most once in a clause. Note that $x \vee \neg x \vee y$ is always satisfied.) Half of the edges are up from street C_i and half are down. Therefore, at least three channels are required for M. Three channels will be used in M exactly when the boolean expression of interest is satisfiable.

Figure 4.7 Construction of the proof of Lemma 4.4.

Espression: $\underbrace{(x_1 \vee x_2 \vee \neg x_3)}_{C_1} \& \underbrace{(\neg x_1 \vee x_2 \vee x_3)}_{C_2}$

Truth assignment: $x_1 = x_2 = x_3 =$ true.



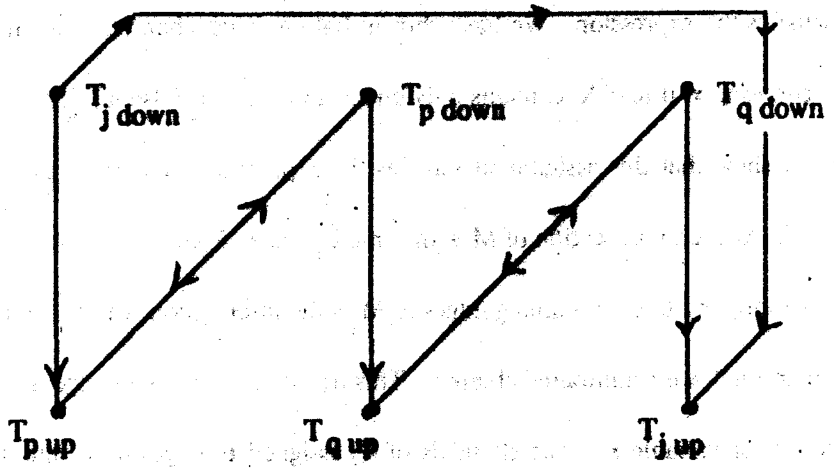
If the width is required to be at most $3v+3$, and street M contributes at least three, then the width contributed collectively by streets X_j and X_{j0} for $1 \leq j \leq v$, must be at most $3v$. Consider the intersections of streets X_j and X_{j0} with street C_i, for i odd. The edge from $(j-1,2i-2)$ to $(j-1,2i-1)$ in X_j belongs to T_j, and the edge from $(j-1,2i-1)$ to $(j-1,2i)$ in X_j belongs to T_{j0}. Therefore, if only one channel is used in X_j, the channel in C_i containing the edges of T_j must have a lower number than the channel in C_i containing the edges of T_{j0} (by condition (4) given for channel assignments). However, in X_{j0}, the edge from $(2v-j+1,2i-2)$ to $(2v-j+1,2i-1)$ belongs to T_{j0} and the edge from $(2v-j+1,2i-1)$ to $(2v-j+1,2i)$ belongs to T_j. If only one channel is used in X_{j0}, the channel in C_i containing edges of T_{j0} must have the lower number. Therefore, only one of X_j and X_{j0} can contain exactly one channel. If i is even, the edges which belong to T_j and T_{j0} are just interchanged from that above; we reach the same conclusion. Therefore, each pair of streets, X_j and X_{j0} contribute at least three channels to the width. However, if collectively at most 3v channels are contributed by streets X_j and X_{j0}, $1 \leq j \leq v$, at most three channels must be used by each pair. Note that T_j and T_{j0} "fit together" so that they can always share one channel in X_j or one channel in X_{j0}. In the other of X_j and X_{j0}, edges of T_j are assigned to one channel and edges of T_{j0} are assigned to a different channel. We will associate a channel assignment in which X_j has one channel with a value assignment in which x_j has value "true"; we will associate a channel assignment in which X_{j0} has one channel with a value assignment of "false" for x_j.

Given an assignment of truth values to the variables x_j such that the boolean expression is satisfied, the corresponding channel assignment giving width $3v+3$ is as follows. For each j between 1 and v, if x_j is true, one channel is used in X_j; if x_j is false, one channel is used in X_{j0}. Street M contains three channels. We first determine what trees will share channels at each intersection of M and some C_i. Note that only adjacent edges in a tree are required to use the same channel: edges of a tree which are in the same street but not part of the same path in the street are not required to use the same channel. Let x_j be a variable whose occurrence in c_i is true under the given assignment of truth values. The edges of T_j

and T_{j0} in M at C_i can share a channel. This follows from the fact that if x_j is uncomplemented in c_i, the edges are in the same direction as those in X_j, and the edges in X_j share a channel; if x_j is complemented, the edges are in the same direction as those in X_{j0}, and the edges in X_{j0} share a channel. However, other pairs of edges in M at C_i will not be able to share a channel if the corresponding literal in c_i is false. Therefore, let the edge for x_j in M down from C_i share with the edge up from C_i for a second variable x_p in c_i. Let the edge down for x_p share with the edge up for the third variable x_q. Finally, the edge down for x_q shares with the edge up for x_j. The new constraints on the assignment of channels in C_i induced by this sharing are consistent with the constraints from any sharing at the intersections of C_i with X_j, X_p, X_q, X_{j0}, X_{p0}, and X_{q0}. (See Figure 4.8.)

The edges in the horizontal streets between the C_i are not contained in any net tree. Therefore, the only condition relevant at the intersections of M with these streets is that a path in a net tree which goes through an intersection cannot change channels. There will be a path in a net tree through such an intersection if the same literal appears in clauses c_i and c_{i+1}, causing an edge up from C_i and down from C_{i+1} in one of T_j and T_{j0}. The edges are adjacent at (v,2i). The assignment to channels in M can be determined as follows. At the intersection of M with the street at horizontal position 0, arbitrarily assign each edge in a net tree incident on the intersection to a different channel among channels 1 through 3. These edges are edges down from street C_1. At C_1, the edges down have been assigned; assign the edges up so they share with the edges down as previously determined. Now consider the intersection of M with the street between C_1 and C_2. The channels for any tree paths from C_1 to C_2 have been determined at C_1; the remaining edges down from C_2 can be assigned arbitrarily. In this way, edges down from each C_i can be assigned channels at the intersection of M with the horizontal street before C_i so that paths do not change channels. Edges up are assigned at the intersection of M with C_i so that channels are shared properly. Thus, if there is any assignment of boolean variables satisfying the boolean expression, there is a channel assignment, ch, with width(ch) $= 3v+3$ and area(ch) $= 2vk(3v+3) \leq A$ as desired.

Figure 4.8 Compatibility of the constraints at the intersection of C_i with M and with X_j and X_{j0}.



"$T_{y\ down}$" represents the tree among T_y and T_{y0} which contains the edge down from C_i for variable x_y -- y equal to j, p, or q.

"$T_{y\ up}$" represents the tree containing the edge up.

$U \longrightarrow V$ indicates the channel in C_i containing edges of net tree U must have a lower number than that containing edges of net tree V.

$\longleftrightarrow$ indicates that the order could be either way.

For any choice of direction for the two $\longleftrightarrow$. edges, no cycle is created. Therefore the constraints are consistent.

Given a channel assignment with area(ch) $\leq$ A, we must show how to construct a boolean assignment which satisfies the expression. We have shown that any such channel assignment induces a truth assignment by choosing x_j true if X_j contains only one channel and x_j false if X_{j0} contains only one channel. It remains to show that this assignment satisfies the expression. We know that exactly three channels are used in M. At each intersection of M with some C_i, edges from two different net trees must share each channel. Among the trees containing edges in M at the intersection with C_i, consider the tree whose edges in C_i are in the lowest numbered channel. This tree must contain an edge in M down from C_i. Suppose the tree is for variable x_j. The channels of C_i assigned to edges in T_j and T_{j0} are in the proper order for the edges of T_j and T_{j0} in M to share a channel. If x_j appears uncomplemented in c_i, then the edges in M at C_i for T_j and T_{j0} are in the same direction as those in X_j at C_i. It must be street X_j which contains only one channel. Therefore x_j is true, and c_i is satisfied by literal x_j. If x_j appears complemented in c_i, then the edges in M at C_i for T_j and T_{j0} are in the same direction as those in X_{j0}. It must be street X_{j0} which contains only one channel. The value of x_j is false, and c_i is satisfied by literal $\neg x_j$. A literal satisfying each clause can be identified by looking at the intersection of M with the street for the clause. Therefore, there is an assignment of truth values to the variables such that the boolean expression is satisfied. □

In the construction used to prove Lemma 4.4, height(ch) is the same for any channel assignment. Therefore, we conclude:

Corollary 4.4: The modified Intersection Channel Assignment Problem in which one desires a channel assignment, ch, such that width(ch) $\leq$ D for some given integer, D, (or such that height(ch) $\leq$ D) is NP-complete.

Lemma 4.4 shows that even ignoring terminals, assigning channels is a difficult problem. In the next section we again consider channel assignment within one street. A heuristic algorithm is analyzed.

An example is constructed to show that the algorithm can be made to do arbitrarily badly with respect to the optimal solution.

4.4 Heuristics for Channel Assignment

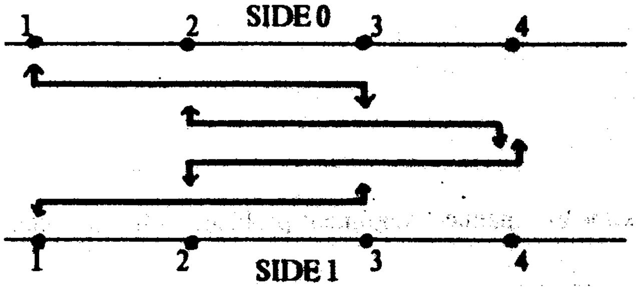
In this section, we discuss heuristics for the channel assignment problem within a street. We begin with a general discussion of various restrictions whose removal changes the optimal channel assignment. Then we present a heuristic algorithm and its analysis for Problem P3, the version of the problem proven NP-complete.

The ordering on $ch(N_i)$ required by condition (i) of the statement of Problem P3 can be represented using a directed graph, which we will call the *constraint* graph. There will be one node for each net. If $ch(N_i) < ch(N_j)$ is required under condition (i) of the statement of Problem P3, then there is an edge directed from the node for N_i to the node for N_j. It is possibile for the graph to be cyclic. In this case, there is no channel assignment for the problem as stated (P3). If each net can use more than one channel -- by using wire segments in the direction perpendicular to the street direction to connect segments in different channels -- then a channel assignment may exist. The segments perpendicular to the street direction are called *jogs*. Jogs used for different nets, like any other wire segments in the same direction for different nets, must be separated by the minimum spacing.

Even when jogs are allowed at any point along a street, the channel assignment problem may not have a solution. Figure 4.9 gives an example. However, if jogs are allowed anywhere between points on a street rather than only at the points, then the channel assignment problem is solvable in polynomial time [Ka79]. The model of a street used in [Ka79] differs slightly from the formulation used in P3. For this discussion, we only need note that, in [Ka79], terminals on opposite sides of the street are either at the same point along the street or are at points separated by at least minimum spacing.

Allowing jogs between points implies that an arbitrarily large number of segments

Figure 4.9 Channel assignment problem with no solution even if jogs are allowed.



Nets: $\{(1,0),(3,1)\} = N_1$ $\{(2,0),(4,1)\} = N_3$
$\{(1,1),(3,0)\} = N_2$ $\{(2,1),(4,0)\} = N_4$

If N_1 or N_2 uses a jog at 2 to change channels, neither N_3 nor N_4 can use a jog at 3:

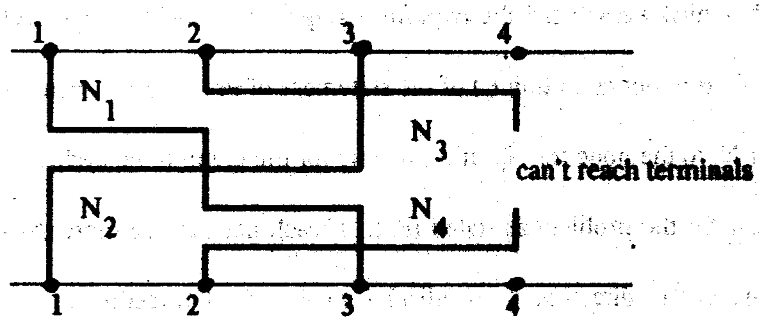
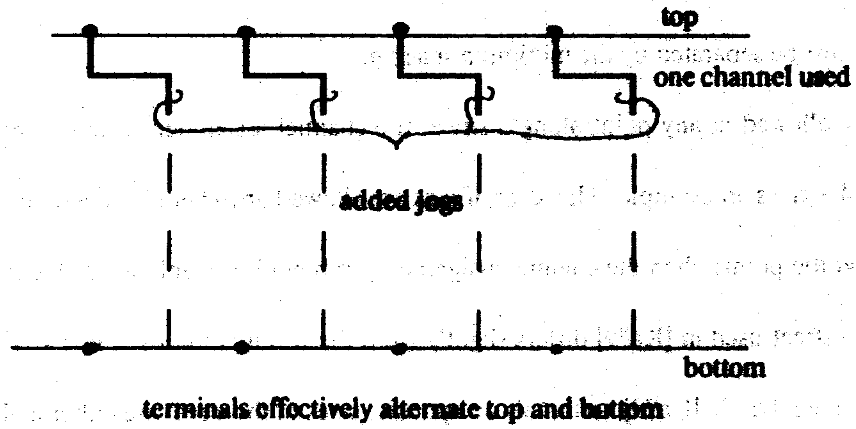


Figure 4.10 Inserting jogs between points to effectively eliminate terminals across from one another.



perpendicular to the street direction may be inserted between any two terminals. The length of the street -- the dimension of the street in the direction of the street -- must be variable, rather than just the street width being variable. The algorithm in [Ka79] minimizes only street width, not the area of the street. Let the *maximum overlap* of a set of nets be the maximum over all points on the street of the number of nets whose intervals overlap at the point. An optimal solution under the model in [Ka79] requires at most one more channel than the maximum overlap of the nets. Recall that if there are no constraints due to condition (i), the channel assignment problem P3 is solvable in polynomial time. Assume, as in [Ka79], that terminals are either at the same position along the street or have sufficient space between them. By using one channel and inserting a jog between every pair of consecutive terminal positions, we can effectively separate the terminals on opposite sides of the street so that there are no constraints under condition (i). (See Figure 4.10.) We at most double the street length. Then the number of channels needed (excluding the channel we used to create the effect of suitably spaced terminals) is the same as the maximum overlap of the nets. This implies that one never need to lengthen the street by more than double to get a channel assignment which uses within one channel of the minimum.

Since we are thinking of a street as running along the boundary of a component and we do not wish to think of each component as expandable,[1] allowing streets to lengthen is not satisfactory. An alternative is to use jogs in a street intersection. When a jog is needed, a segment which goes to the end of the street and out into the street intersection is used. In the intersection, the path can use a segment (the jog) in the perpendicular street to change to another channel in the original street, and reenter the region of the street it was previously in. This type of solution requires that perpendicular streets be available and that nets are allowed to have wire segments in two channels of a street at the same position along the street. Figure 4.11a illustrates. Such a routing not only affects the width of the street containing the

[1] In fact, some components are expandable [Jo79].

terminals being interconnected, but affects the width of the perpendicular street as well. Using this scheme, any collection of nets consisting of terminals in one street can be interconnected. Figure 4.11b shows the general connection pattern.

Let us return to our original channel assignment problem, problem P3. The proof of NP-completeness of problem P3 relies on the fact that jogs are not allowed. Figure 4.12 shows that even for instances of the problem derived from circular arc coloring, allowing jogs reduces the number of channels needed. We do not have a proof that the channel assignment problem with jogs in NP-hard, although the author believes that this is the case.

We now present a simple heuristic algorithm for problem P3 and analyze the quality of the solutions it produces.[1] This algorithm is the basic algorithm used in [Deu76] when jogs are not allowed. We will assume that the constraint graph produced by applying condition (i) does not have any cycles so that a solution without jogs does exist. We first define the *level* of a node in the constraint graph:

(1) All nodes which have no edges into them are on level 1.

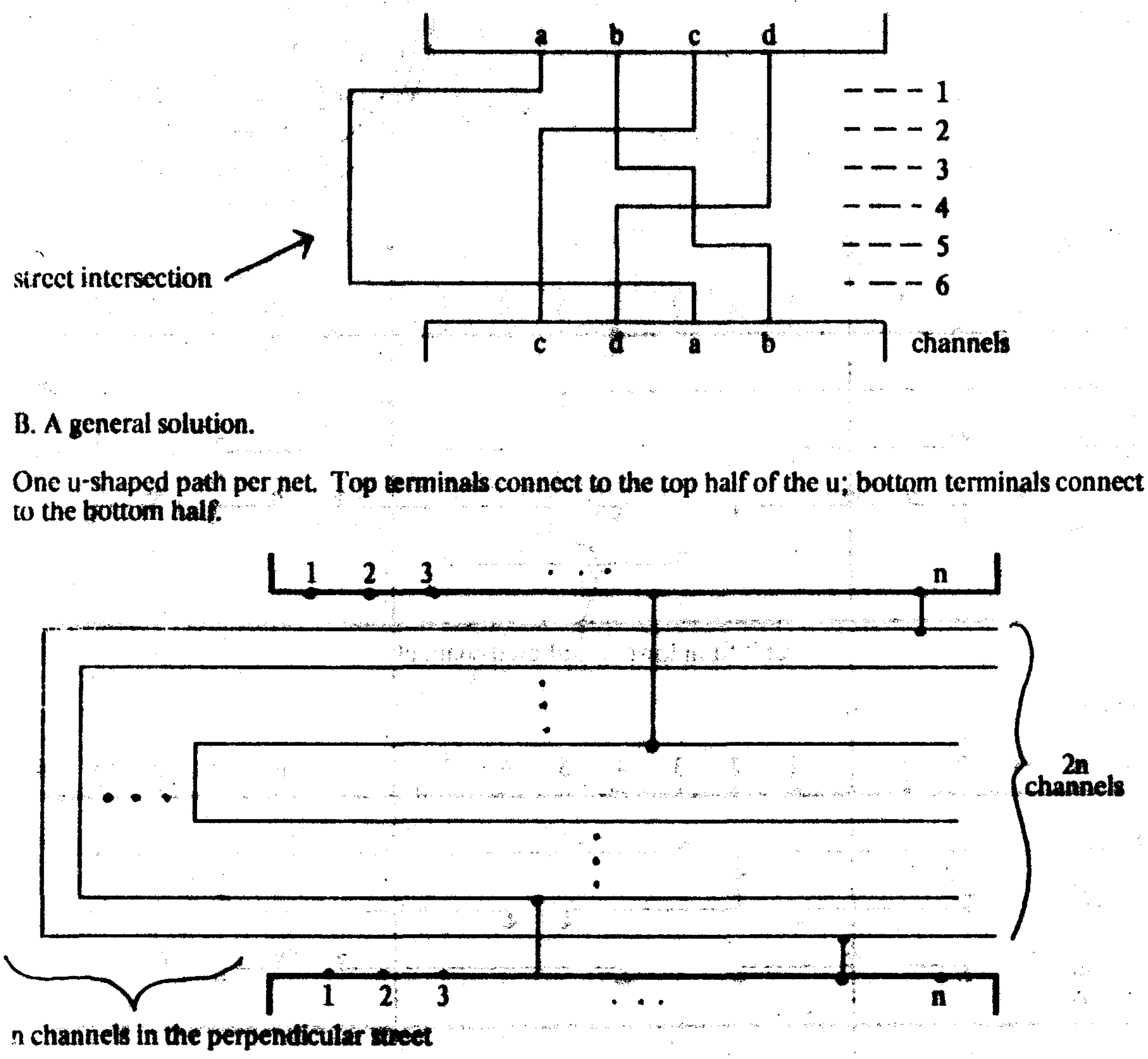
(2) Assuming that the nodes on levels 1 through k-1 are defined for k>1, a node is on level k if all edges entering it are from nodes on lower levels and it is not on a lower level.

Since the constraint graph is acyclic, the levels are well-defined. They can be computed by starting with level 1 nodes and following edges. We will say that a net is at level k if its node is at level k. The nodes from which there are edges to a given node are called *predecessors* of the given node, and the corresponding nets are called *predecessors* of the given net.

Order the nets in increasing order by the position, a_i, of their terminals of lowest position. If there are no constraints due to condition (i), then the following algorithm finds the optimal solution:

[1] This analysis is part of joint research with Errol Lloyd.

Figure 4.11 Problems requiring jogs.



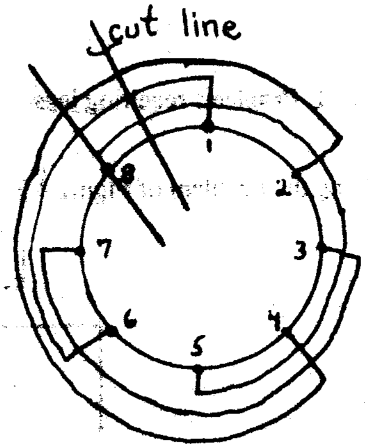
A. Solving the Problem of Figure 4.9

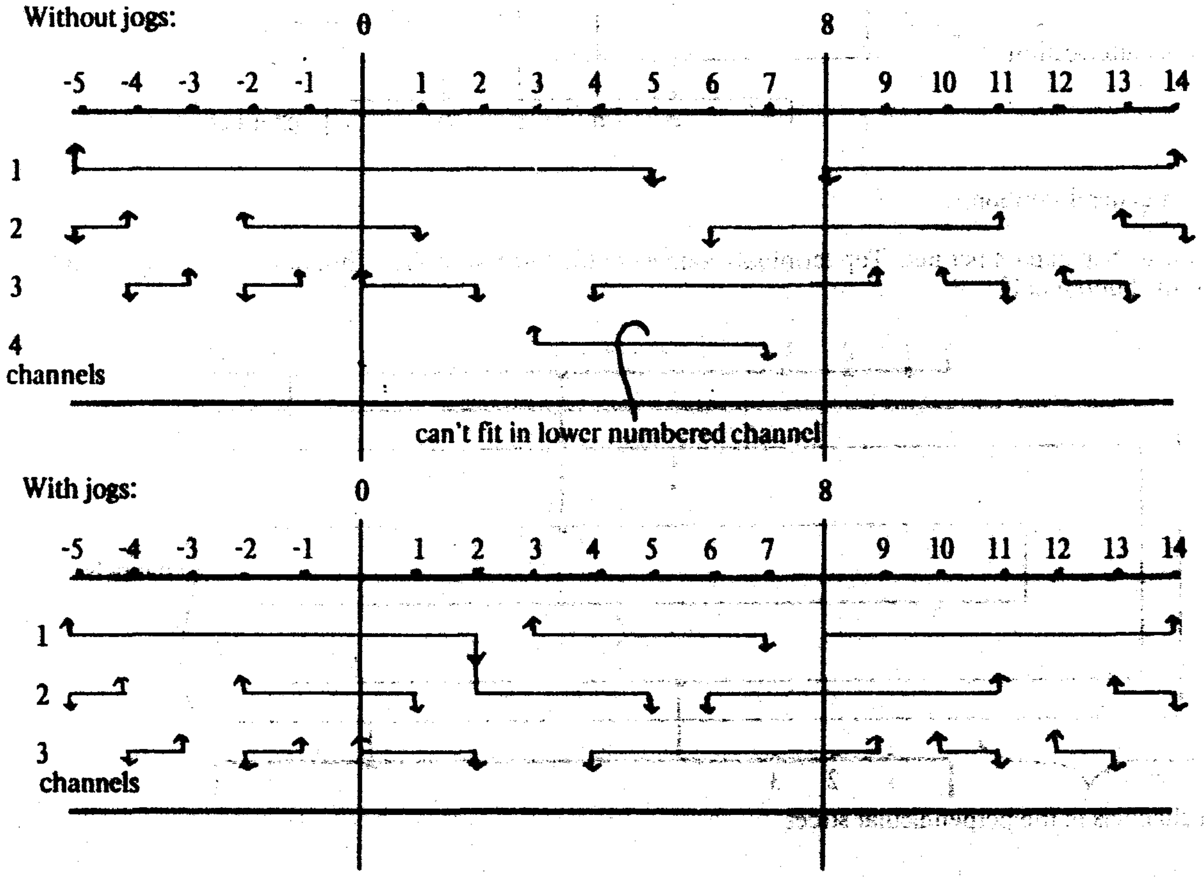
B. A general solution.

One u-shaped path per net. Top terminals connect to the top half of the u; bottom terminals connect to the bottom half.

Figure 4.12: Use of jogs when not necessary.

We look at a problem derived from a set of circular arcs: circular arcs (8,5), (3,7), (4,2), (6,1)





Low to High Fill [Has71]

```
Repeat for each channel, beginning with channel 1 and increasing the channel number by one
for each new channel until all nets are assigned:
    Choose the first unassigned net under the above ordering. Assign it to a new channel.
    Repeat until the channel is full:
        Choose the first unassigned net whose terminal of lowest position is at a
        higher position than the terminal of highest position in the last net to be
        assigned to the channel. Assign this net to the channel.
```

Claim [Has71]:[1] When there are no constraints due to condition (i), the above algorithm uses exactly the number of channels as the maximum overlap of the nets.

Proof: Look at the lowest point, a_0, in the interval of the first net, N, assigned to the last channel. Each lower numbered channel must contain a net whose interval contains point a_0. If there were a channel, k, which did not contain such a net, then the net N should have been placed in this channel. This follows because, in the net ordering, N is before the nets, if any, which were placed in channel k after point a_0. Therefore, all lower numbered channels are assigned nets whose intervals contain point a_0. The overlap at this point is equal to the number of channels used. □

Algorithm "Low to High Fill" will be the basis of the algorithm to assign nets to channels when constraints are present. Channels are again filled beginning with channel 1. At any point during execution of the algorithm, let the set, A, of *available* nets contain those nets which are unassigned and all of whose predecessors are assigned. "Low to High Fill" is modified to choose nets to be assigned only

[1] In [Has71], Hashimoto and Stevens use a different approach to prove the algorithm correct. They prove a different but equivalent claim. The proof given here is based on the proof used by Gavril [Gav72] to prove that his algorithm for chordal graph coloring is correct.

from set A (the nets are ordered as before) and to update A after every assignment.

Lemma 4.5: For an instance I of Problem P3, let d_k be the maximum over all points on the street of the number of nets at level k whose interval overlaps the point. Let h be the highest level. Then

(number of channels)$_{alg}$(I)/(number of channels)$_{opt}$(I) $\leq$ min(h, avg(d_k)) $\leq n^{1/2}$

where the algorithm is the modified Low to High Fill algorithm using set A, avg(d_k) is the average of the d_k for $1 \leq k \leq h$, and n is the number of nets in instance I.

Proof: We know that (number of channels)$_{opt}$(I) $\geq$ maximum of the d_k and h.

We prove that (number of channels)$_{alg}$(I) $\leq$ $\Sigma_{k=1}^{h}(d_k)$. Let C_1 denote the set of channels containing a net at level 1. For k>1, let C_k denote the set of channels with higher numbers than the last channel containing a net at level k-1 and which contain nets at level k. The number of channels used by the algorithm is the sum of the $|C_k|$ over all levels. For a particular k, C_k may be empty. If C_k is not empty, look at the first net at level k placed in the highest numbered channel in C_k; denote this net N_k. Let a_k be the lowest point in its interval. If any channel in C_k is not assigned a net whose interval contains a_k, then N_k should have been placed in this channel. This follows because N_k must have been in A when the channel was assigned -- all level k-1 nets are places before channels in C_k are assigned -- and N_k precedes in the net ordering whatever net, if any, was assigned to the channel after point a_k. Therefore, $|C_k| \leq d_k$.

We have:

$$\text{(number of channels)}_{alg}\text{(I)/(number of channels)}_{opt}\text{(I)} \leq \text{h(avg}(d_k))/\text{max(h,max}(d_k))$$
$$\leq \text{max(h,avg}(d_k)) \times \text{min(h,avg}(d_k))/\text{max(h,max}(d_k)) \leq \text{min(h,avg}(d_k))$$

The number of nets at level k is at least d_k. Therefore,

$$\Sigma_{k=1}^{h}(d_k) = h \times \text{avg}(d_k) \leq n \quad \text{and} \quad \text{min(h,avg}(d_k)) \leq n^{1/2}. \qquad \square$$

If the above bound on the worst case performance of the algorithm is accurate, the algorithm can do very badly. In fact, the bound is the same as would be obtained for an algorithm which applies

"Low to High Fill" on nets of a fixed level, one level at a time, requiring all nets on level k ($1 \leq k \leq h$) to be assigned to lower numbered channels than those of level $k+1$. Nonetheless, our bound is tight.

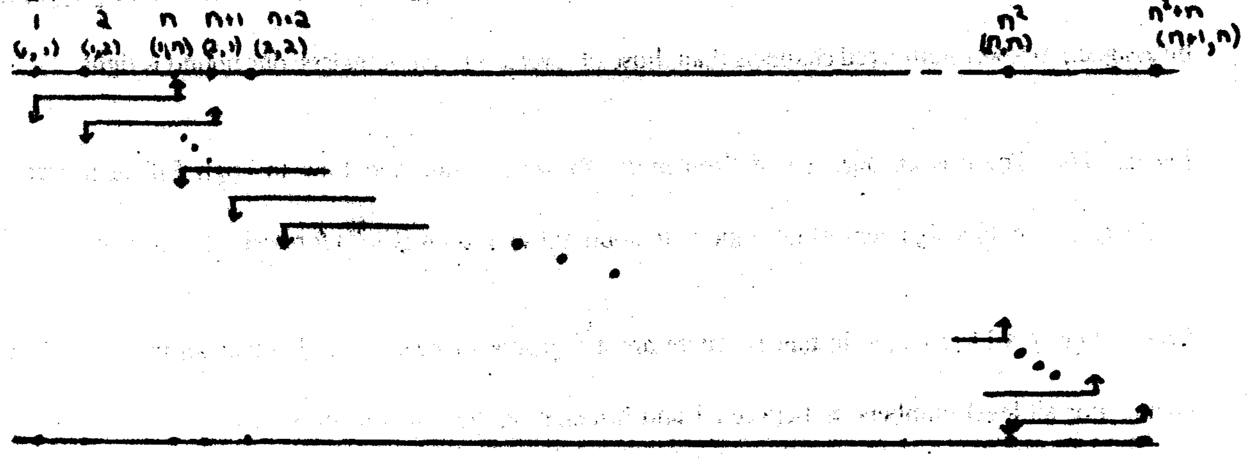
Lemma 4.6: There is an instance of Problem P3 for which modified Low to High Fill using set A produces a channel assignment whose ratio to the optimal solution is $\Omega(n^{1/4})$ for $avg(d_k) = h = n^{1/4}$.

Proof: Figure 4.13 gives the instance. There are $n^{1/4}$ groups of nets, each of which forms an $n^{1/4}$ level chain. For all level numbers, k, between 1 and h inclusive, the intervals of all nets on level k overlap, giving $d_k = n^{1/4}$. The algorithm assigns only one net to each channel. However, there is a channel assignment which does not assign nets to channels in a low to high order. Among nets on a level, some nets which are later in the net ordering are assigned to lower numbered channels than nets which come before them in the ordering. The nets which came first in the ordering can share channels with nets which are on higher levels than they are, but whose predecessors have already been assigned. Refer to the figure for details. □

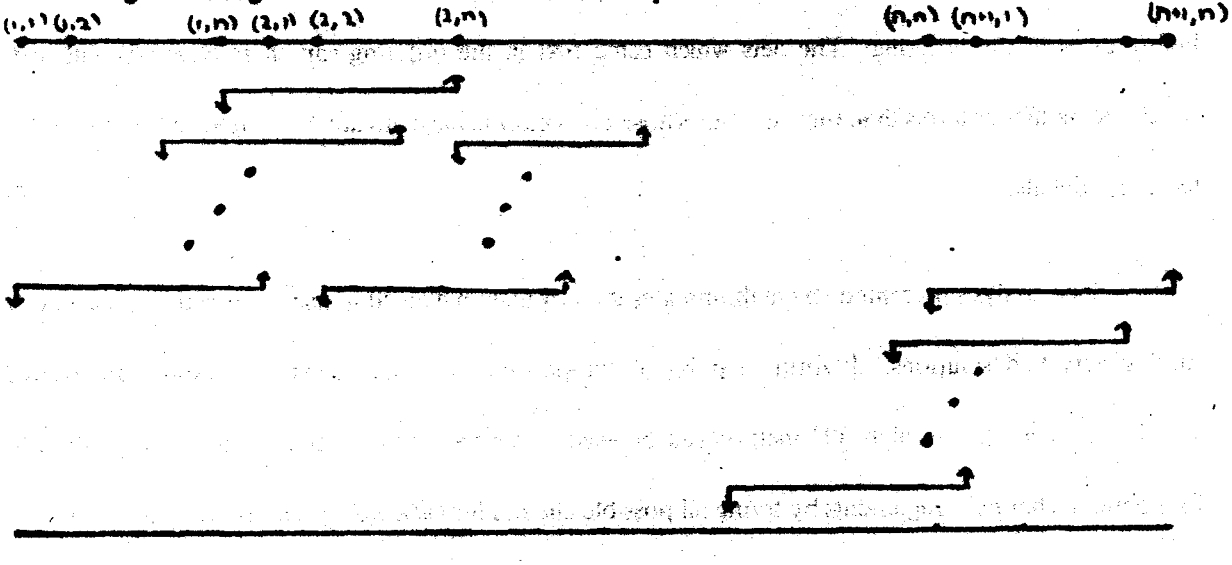
The analysis presented above illuminates the fact that an algorithm used in practice is capable of finding very bad solutions. It forms a point of comparison for more "clever" or more complicated algorithms. The "low to high fill" method can be used as the basis of an exponential time algorithm to find optimal channel assignments by trying all possible choices for each assignment rather than taking the first net in an ordering [Ker73].

In the next chapter, we present an algorithm for a special case of channel routing which is not NP-complete. The problem is to route interconnections among two point nets whose terminals lie on the outside of one rectangle. This problem is reminiscent of the circular arc coloring problem, which is NP-complete. However, the paths around the outside of the rectangle are allowed to change "color" as they go around corners, i.e. the order of paths need not be the same on adjacent sides of the rectangle. This order can change because horizontal and vertical wire segments are allowed to cross. Once the paths

Figure 4.13: Worst case for Low High Fill.



Nets: N_{ij} = {((i,j),1), ((i+1,j),0)} for $1 \leq i \leq n$ and $1 \leq j \leq n$
Nets N_{kj} for $1 \leq j \leq n$ overlap the interval ((k,n),(k+1,1)).
Condition (i) of P3 gives $ch(N_{ik}) < ch(N_{i+1,k})$ for $1 \leq i \leq n-1$.
Nets N_{kj} for $1 \leq j \leq n$ are on level k.
Low to High Fill assigns channels as shown above, but optimal solution is:



Total of 2n-1 channels.
$\text{channels}_{alg}/\text{channels}_{opt} = n^2/2n\text{-}1 = \Theta(n) = \Theta((\text{number of nets})^{1/2})$

have been routed through the four streets surrounding the rectangle, the channel assignment is four instances of the interval coloring problem. In the next chapter, we show how to do the street routing optimally.

Chapter 5: An Algorithm for Routing Terminals on a Rectangular Component

5.1 Preliminaries

We will now present an algorithm which finds an optimal solution to the following routing problem in polynomial time. Figure 5.1 presents an example.

The One Rectangle Routing Problem

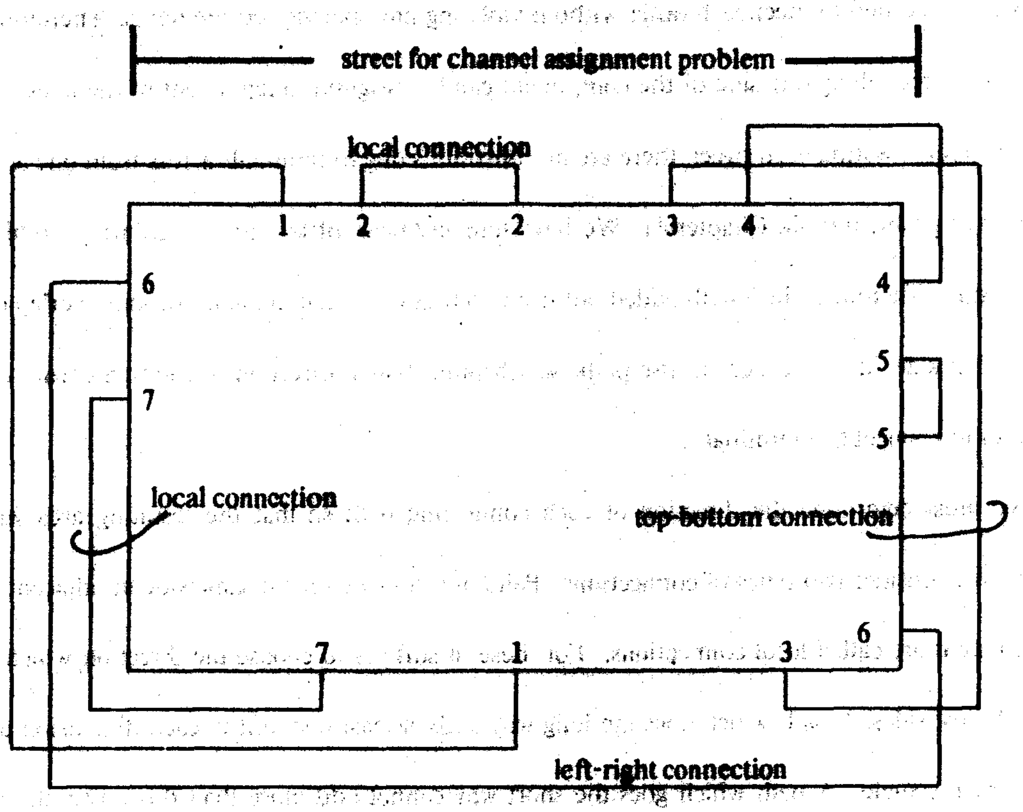
Given: a rectangular component with terminals around its outside edge. Terminals lie on positions which have at least unit spacing along the edge of the rectangle. The unit spacing represents the width of a wire plus the minimum spacing between wires. A list of nets, each containing a pair of terminals which must be connected, is given. Each terminal belongs to exactly one net.

Find: An optimal routing of the wires between pairs. Paths must be composed of line segments which are parallel to some side of the rectangle. Distinct paths may cross at right angles; however, parallel segments belonging to distinct paths must be separated by the minimum spacing. (We are assuming that there are two layers for interconnection. One layer is used for the line segments in one direction, and the other layer is used for line segments in the second direction.) All paths must lie outside the rectangular area of the component. An optimal routing is one which minimizes the area of the smallest rectangle which circumscribes the component and all routing paths. The sides of the circumscribing rectangle must be parallel to those of the component.

Place the rectangular component on a cartesian coordinate system so that its sides are parallel to the axes. Arbitrarily choose one axis direction to be horizontal and one vertical. Label the horizontal sides of the rectangle as top and bottom, the vertical sides as left and right.

Each pair of terminals can be connected either by a path which goes clockwise from one of the terminals or by a path which goes counterclockwise from the terminal. The directions of the connections determine a set of intervals which path segments will use along each side. Once the intervals to be used by each path along a side are determined, minimizing the length added out from the side is a channel

Figure 5.1: Example of the routing problem for one rectangular component.



Terminals are represented as points on the rectangle boundary.
Pairs of terminals with the same number must be connected.
An optimal solution is shown.
The height added to the top by this solution is 2 units; the height added to the bottom is 3 units.
The width added to the left side is 3 units; the width added to the right side is 2 units.

assignment problem. Vertical and horizontal segments which belong to one path can be extended beyond a corner as far as needed to meet each other without violating any spacing requirements. Therefore, the channels in the street along one side of the component can be assigned independent of the assignments within other streets. Within each street, there are no constraints due to terminals across from one another (condition (i) of problem P3 in Chapter 4). We have four instances of the interval coloring problem -- one for each side. Therefore, the length added out from each side is equal to the maximum overlap of the intervals on that side, and it suffices to use paths which only change direction to round a corner of the component or to connect to a terminal.

We must determine the direction of each connecting path so that the resulting area will be minimized. We consider two types of connections. Pairs of terminals on the same side or adjacent sides of the component are called local connections. For these, it suffices to choose the direction which goes around the fewest sides. A path which goes the long way adds at least one unit in each dimension to the circumscribing rectangle. A path which goes the short way cannot add more than this. Therefore it is never better to go the long way around.

The second type of connection contains those which go from the left side to the right side or the top to the bottom. The choice of direction for top-bottom connections is independent of the direction of left-right connections and vice versa, since regardless of the direction used for a top-bottom connection, one unit is added to the horizontal dimension of the circumscribing rectangle. Therefore, we have two instances of the following problem:

Top-Bottom Routing Problem

Given: Terminals on a top and a bottom, a set of top to bottom connections, and some local connections on these sides.

Find: A direction -- left or right -- for each top to bottom connection so that the resulting total vertical dimension is minimized. Each connection is made by going around to the left or the right as indicated by the direction.

In this description, connections which originally went from the top or bottom to an adjacent side are considered to be local connections which go to the very edge of the top or bottom. (The portion used to turn the corner is not of interest here.)

To solve the above problem, we first reduce it to a problem of assigning 0/1 values to elements of two vectors so that a matching on the vectors is maximized. The vectors represent the top and bottom terminals. The representation removes unnecessary information about local connections. The value of "0" or "1" represents the direction --left or right-- of the connection at a terminal. The matching identifies which segments will share the same channel in the final routing.

The major phases of the algorithm to solve the new assignment problem are as follows:

1. Partition each vector into regions within which matching can be localized. After assigning values within a region, the regions are recalculated. The algorithm iteratively assigns values within these regions until there is only one unassigned region for each of the top and bottom vectors.

2. For the remaining top region and bottom region, the algorithm assigns values from left to right along the top region, maintaining certain properties of the number of "0"s and "1"s in portions of the vectors. These properties guarantee a maximum matching for the vectors. It may happen that not all properties can be satisfied simultaneously when some element is assigned a value. At this point, we say a failure has occurred. When a failure occurs, the algorithm may still be able to determine an assignment which suffices to guarantee a maximum matching. However, it may be necessary to try both values for the assignment.

3. When the algorithm must try both choices for an assignment, it does not treat both choices equivalently. For the choice of value "1", the algorithm is applied recursively to complete the

assignment. For the choice of value "0", the algorithm is applied with modification. We assume that the choice of "0" leads to a better solution, not just as good a solution, as the choice of "1". Therefore, as soon as there is evidence that the "0" choice will lead to no better a solution, the algorithm may stop pursuing this choice. In particular, if the situation in which the algorithm must try both choices reoccurs, it will turn out that the second "0" choice leads to no better solution that the first "1" choice. This second "0" choice can be eliminated. Only the "1" choice is used, and the search done by the algorithm is thus bounded.

The technique of bounded search is crucial to the algorithm. Without the ability to bound the search, the algorithm would have exponential running time. The following sections described in more detail the algorithm outlined above. During this description, a large amount of notation will be introduced. The appendix to this chapter summarizes that notation.

5.2 Reduction to Maximizing Matchings

The Top-Bottom Routing Problem is reduced to the problem of assigning values within two vectors to maximize a matching. The vectors, T(1,m) and B(1,n), represent the terminals at the top and bottom, respectively, numbered from left to right. (We will drop the T and B when it is clear from context whether we are referring to a top or bottom terminal.) A value of "0" or "1" is used to represent the direction of the connection at each terminal. Define the value function VT: T(1,m) → {0,1,?} as follows:

VT(i) = 0 if the direction of the path from terminal i to its pair is to the left

1 if it is to the right

? if it is undetermined

Value function VB: B(1,n) → {0,1,?} is defined the same way.

Let p:T(1,m)∪B(1,n) → T(1,m)∪B(1,n)∪{*} be the pairing function. Terminal p(x) should be connected to terminal x. When p(x) = * (don't care), the terminal x is connected to a terminal on an adjacent side. Initially, the pairing function is known, and the value functions have 0 or 1 values only for the local connections. We would like to find VT and VB for which all directions are determined and are consistent with the initial values given. We will say that a value function V' is consistent with V if $V'(i)=V(i)$ whenever $V(i)\neq ?$. We also say that V' extends V. For any pair of value functions we require that $VT(i) = VB(j)$ if $p(T(i))=B(j)$.

We would like to balance paths to the left with those to the right on the top and bottom simultaneously. Let m_{VT} be a function matching terminals in T(1,m) of value "0" to terminals with higher index in T(1,m) and of value "1", i.e. matching paths to the left and right. The function is formally defined as a one-to-one partial function from T(1,m) to itself such that if $m_{VT}(i)=j$, then $VT(i)=0$, $VT(j)=1$, and $j > i$. Define m_{VB} similarly. For a particular VT and VB, let M_{VT} be the maximum over all matching functions m_{VT} of $|range(m_{VT})|$, i.e. M_{VT} is the maximum number of matches among T(1,m) for a given VT. Parameter M_{VB} is defined similarly.

We will reduce our routing problem to a problem of assigning "0"s and "1"s to maximize $M_{VT}+M_{VB}$. This is justified by the following lemma:

Lemma 5.1: Given a Top-Bottom Routing Problem with local connections represented by value functions VT_0 and VB_0, the problem of finding VT_f and VB_f under which all directions are defined and for which the vertical dimension of the circumscribing rectangle is minimized is equivalent to the problem of finding VT_f and VB_f such that $M_{VT_f}+M_{VB_f}$ is maximized over all VT and VB which map T and B to {0,1} and are consistent with VT_0 and VB_0.

Proof: The matching function m_{VT_f} corresponds to determining which horizontal segments above the top side will share the same channel. If $m_{VT_f}(i)=j$, then in some channel, the segment ending (going left to right) at terminal i is followed directly by the segment which starts at terminal j. Each channel begins with a segment running from the top left edge (a segment for which $VT_f(i)=0$ and $p(i) \in B(1,n)$ or $p(i)=*$) or with a segment which begins at a 1-valued terminal which is unmatched under m_{VT_f}. The rightmost segment in each channel is a segment running to the top right edge ($VT_f(i)=1$ and $p(i) \in B(1,n)$ or $p(i)=*$) or a segment ending at a 0-valued terminal which is unmatched under m_{VT_f}. Figure 5.2 illustrates possible configurations. The channels can be in any order above the side; therefore, we have found an assignment of segments to channels for each m_{VT_f}. The argument can be reversed so that we find a matching for each assignment to channels. There is a one-to-one correspondence between channel assignments and matchings giving:

number of channels used = number of segments going to left edge + number of unmatched "1"s
= number of segments going to right edge + number of unmatched "0"s

This gives: total vertical dimension corresponding to matching functions m_{VT_f} and m_{VB_f}
= ht. of rectangle (denoted h) + # channels at top + # channels at bottom
= h + # top segments going to left edge + # unmatched "1"s at top
+ # bottom segments going to right edge + # unmatched "0"s at bottom
= h + # top segments going to left edge + # "1"s at top - |range(m_{VT_f}|
+ # bottom segments going to right edge + # "0"s at bottom - |range(m_{VB_f}|

Note that: # "1"s at top = # top segments going to right edge + # top segments going to neither edge

and therefore: # top segments going to left + # "1"s at top = # top segments

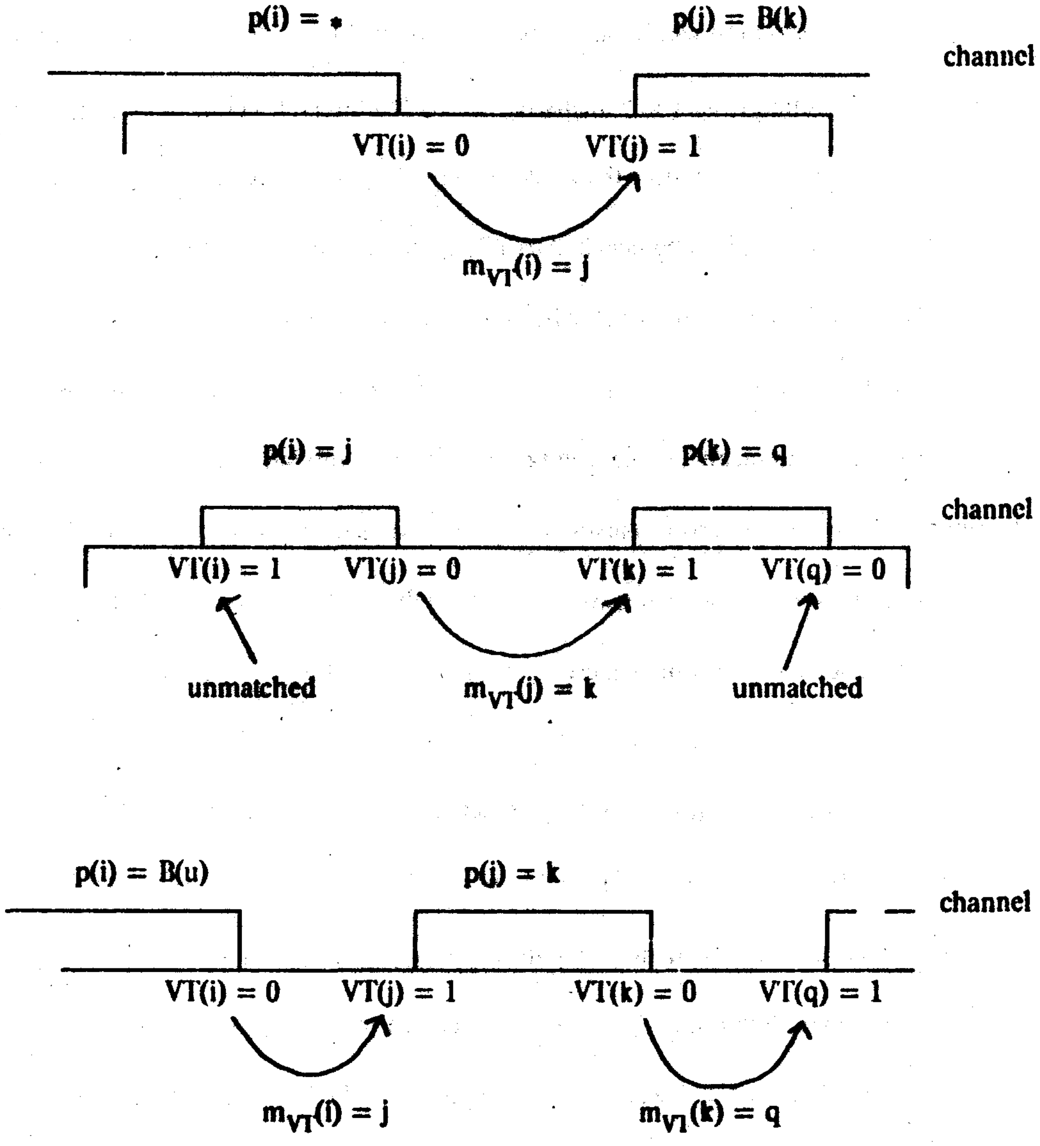
This gives: minimum total vertical dimension over all matching functions for VT_f and VB_f
= h + # top segments + # bottom segments - ($M_{VT_f} + M_{VB_f}$) = h + C - ($M_{VT_f} + M_{VB_f}$)

Where C = (# local connections top and bottom) + 2(# top-bottom connections) is a constant for any instance of the routing problem. Therefore, minimizing the height is equivalent to maximizing

$M_{VT_f} + M_{VB_f}$. □

We now present the various phases of the algorithm which finds value functions VT_f and VB_f

Figure 5.2: Possible configurations when filling channels.



maximizing $(M_{VT_f} + M_{VB_f})$.

5.3 Defining Regions

The following description will be in terms of T. An analogous development is assumed for B. Let T(x,y) denote the terminals from x to y inclusive. In any left interval, T(1,i), for any $i \leq m$, we would like to maintain the property that no more than half the terminals are 1-valued. This property must hold if every 1-valued terminal in (1,i) is to be matched to a 0-valued terminal with smaller index. If more that half the terminals in a left interval are 1-valued, some of these 1-valued terminals cannot be matched under any m_{VT}. If at least half the terminals are 1-valued, any ?-valued terminals in the interval should become 0-valued if we are to maximize the number of matches. Similarly, we want no more than half 0-valued terminals in any right interval to insure that these 0-valued terminals can be matched. Because of the local connections, it is not always possible to maintain these properties on any left or right interval. Instead, we define regions within which these bounds hold. Within these regions matching can be localized.

For a given value function, VT, let $ZEROS_{VT}(S) = \{i \in S | VT(i)=0\}$; $ONES_{VT}(S) = \{i \in S | VT(i)=1\}$; $UNDET_{VT}(S) = \{i \in S | VT(i)=?\}$ where $S \subseteq T$. Define the property OK-1(VT,x,y) (similarly OK-0) which is true if and only if $|ONES_{VT}(x,y)| \leq \lfloor \frac{1}{2}(y-x+1) \rfloor$. (We us the notation "$ONES_{VT}(x,y)$" rather than "$ONES_{VT}(T(x,y))$") Also define Full-1(VT,x,y) (similarly Full-0) which is true if and only if $|ONES_{VT}(x,y)| \geq \lceil \frac{1}{2}(y-x+1) \rceil$. Property Full-"b"(VT,x,y) is true when at least half the terminals in T(x,y) have value "b" under VT. Note that Full-1(VT,x,y) and OK-1(VT,x,y) can be simultaneously true only if $|ONES_{VT}(x,y)| = \frac{1}{2}(y-x+1)$, an integer. Full-1(VT,1,i) indicates that the terminals in $UNDET_{VT}(1,i)$ should become 0-valued if we are to maximize the number of matches.

We will now define the regions of T within which matching can be localized. Left-regions under VT are formed scanning T from 1 to m. A new region begins when the previous region has at least half

"1"s. Similarly, right-regions under VT are intervals of T scanning from m to 1 and counting "0"s.

First define functions l_{VT} and r_{VT} mapping T to itself:

$l_{VT}(1) = 1$

$l_{VT}(i) = i$ if Full-1(VT,l_{VT}(i-1),i-1)

$l_{VT}(i\text{-}1)$ otherwise, for $i > 1$

$r_{VT}(m) = m$

$r_{VT}(j) = j$ if Full-0(VT,i+1,r_{VT}(i+1))

$r_{VT}(i+1)$ otherwise, for $j < m$

The function l_{VT} induces an equivalence relation on T(1,m) under which two terminals i and j are equivalent if and only if $l_{VT}(i) = l_{VT}(j)$. The resulting equivalence classes are intervals of T(1,m); the lowest element of each is an element of the range of l_{VT}. These equivalence classes are called *left-regions* under VT. A new region begins when the previous region has at least half "1"s. Similarly, r_{VT} induces *right-regions* under VT which are intervals whose highest element is an element of the range of r_{VT}.

Lemma 5.2: (a)OK-1(VT,l_{VT}(i),i) unless $l_{VT}(i)=i$ and VT(i)=1

(b)OK-0(VT,i,r_{VT}(i)) unless $r_{VT}(i)=i$ and VT(i)=0.

Proof: We will prove (a). The proof for (b) is analogous.

If $l_{VT}(i)=i$ and VT(i)$\neq$1, then $|\text{ONES}_{VT}(i)|=0=\lfloor ½(i\text{-}i+1))\rfloor$ and OK-1(VT,i,i). If $l_{VT}(i)<i$, then not (Full-1(VT,l_{VT}(i-1),i-1)). In this case $l_{VT}(i)=l_{VT}(i\text{-}1)$ and

$|\text{ONES}_{VT}(l_{VT}(i),i)| \leq |\text{ONES}_{VT}(l_{VT}(i\text{-}1),i\text{-}1)|+1 < \lceil ½(i\text{-}l_{VT}(i))\rceil+1 = \lfloor ½(i\text{-}l_{VT}(i)+1)\rfloor+1$

and OK-1(VT,l_{VT}(i),i). □

On all the left-regions except possibly the last, at least half the terminals are 1-valued. Such regions are called *full* left-regions. We define a delimiter for the full regions. Let $L_{VT}=m$ if

Full-1(VT,$l_{VT}(m)$,m) and $l_{VT}(m)$-1 otherwise. Let $R_{VT}=1$ if Full-0(VT,1,$r_{VT}(1)$) and $r_{VT}(1)+1$ otherwise. Lemma 5.3 shows that the full left-regions can overlap the full right-regions only when all terminals in the interval of overlap are 0/1-valued.

Lemma 5.3: If $L_{VT} \geq R_{VT}$, then $|UNDET_{VT}(R_{VT},L_{VT})| = 0$, i.e. all directions are known on (R_{VT},L_{VT}).

Proof: Assume $L_{VT} \geq R_{VT}$. The proof counts the number of "0"s and "1"s necessarily in (R_{VT}, L_{VT}) using the definitions of R_{VT} and L_{VT}.

Case 1: $|ONES_{VT}(R_{VT},L_{VT})| \geq |ZEROS_{VT}(R_{VT},L_{VT})|$. Suppose L_{VT} is in range(r_{VT}). Then (R_{VT},L_{VT}) is a set of right-regions of T. Since Full-0 is true for any right-region of terminals greater than R_{VT}, it must be that Full-0(VT,R_{VT},L_{VT}). However, we have assumed that there are at least as many "1"s as "0"s in (R_{VT},L_{VT}). This gives

$$|ONES_{VT}(R_{VT},L_{VT})| = |ZEROS_{VT}(R_{VT},L_{VT})| = ½(L_{VT}-R_{VT}+1)$$

and no elements of (R_{VT},L_{VT}) have value "?" under VT.

We now suppose L_{VT} is not in the range of r_{VT}. Then

$$|ZEROS_{VT}(L_{VT}+1,r_{VT}(L_{VT}+1))| < \lceil ½(r_{VT}(L_{VT}+1)-(L_{VT}+1)+1)\rceil \qquad \text{and}$$

$r_{VT}(L_{VT}+1)=r_{VT}(L_{VT})$. Interval $(R_{VT},r_{VT}(L_{VT}))$ is a set of right-regions and Full-0 holds. This gives:

$$|ZEROS_{VT}(R_{VT},L_{VT})| = |ZEROS_{VT}(R_{VT},r_{VT}(L_{VT}))| - |ZEROS_{VT}(L_{VT}+1,r_{VT}(L_{VT}))|$$

$|ZEROS_{VT}(R_{VT},L_{VT})| > \lceil ½(r_{VT}(L_{VT})-R_{VT}+1)\rceil - \lceil ½(r_{VT}(L_{VT})-L_{VT})\rceil \geq \lfloor ½(L_{VT}-R_{VT}+1)\rfloor$, i.e. Full-0(VT,$R_{VT}$,$L_{VT}$) again implying no elements of (R_{VT},L_{VT}) have value "?" under VT.

Case 2: $|ONES_{VT}(R_{VT},L_{VT})| < |ZEROS_{VT}(R_{VT},L_{VT})|$ This case is proven in the same manner as case 1 with the roles of L_{VT} and left-regions interchanged with the roles of R_{VT} and right-regions. □

The matching within the full left-regions $(1,L_{VT})$ or the full right regions (R_{VT},m) can be maximized independent of the rest of T. Our algorithm uses this property to break up the problem.

Theorem 5.1 formally states the independence of the regions.

Theorem 5.1: Let VT_0 be a given initial value function defining R_{VT_0} and L_{VT_0}. For every VT consistent with VT_0 and every matching function m_{VT}, there is a matching function m'_{VT} agreeing with m_{VT} on 0-valued terminals in $(L_{VT_0}+1,m)$ but matching each 0-valued terminal in $(1,L_{VT_0})$ to a 1-valued terminal in $(1,L_{VT_0})$ and such that $|range(m'_{VT})| \geq |range(m_{VT})|$. Analogously, there is a matching function m''_{VT} agreeing with m_{VT} on 1-valued terminals in $(1,R_{VT_0}-1)$ but matching each 1-valued terminal in (R_{VT_0},m) to a 0-valued terminal in (R_{VT_0},m) and whose range is at least as large.

Proof: We will only prove the existence of m'_{VT}. The proof of the existence of m''_{VT} is similar. The proof is by induction on the number of left-regions in $(1,L_{VT_0})$.

Basis: $(1,L_{VT_0})$ contains at most one region.

If $L_{VT_0}=0$, then $(1,L_{VT_0})$ is empty. If $L_{VT_0}=1$, then $VT_0(1)=1$ and $(1,L_{VT_0})$ does not contain any "0"s. In either case, the theorem is trivially true.

If $L_{VT_0} > 1$, then $l_{VT_0}(L_{VT_0})=1$. Therefore, OK-1(VT_0,1,L_{VT_0}). We also have Full-1(VT_0,1,L_{VT_0}) and can conclude $|ONES_{VT_0}(1,L_{VT_0})| = ½L_{VT_0}$. For any value function, a maximum matching can be found by scanning T from 1 to m. Each time a "1" is encountered, it is matched to the lowest numbered yet unmatched "0". As long as there are no more "1"s than "0"s in an interval (1,i), there will be an yet unmatched "0" in (1,i-1) to match a 1-valued terminal i. In the case we are considering, OK-1(VT_0,1,i) holds for every $i \leq L_{VT_0}$. Therefore every 1-valued terminal can be matched to a 0-valued terminal if all ?-valued terminals in $(1,L_{VT_0})$ become 0-valued. Since there are ½L_{VT_0} 1-valued terminals in $(1,L_{VT_0})$ under VT_0, such a matching function would match all 0-valued terminals in $(1,L_{VT_0})$. For each 0-valued or ?-valued terminal under VT_0 in $(1,L_{VT_0})$, we can associate the 1-valued terminal to which it would match under the above matching. Given any value function VT consistent with VT_0 and any matching function, m_{VT}, the desired matching function m'_{VT} can be created by matching each 0-valued terminal in $(1,L_{VT_0})$ to the the 1-valued terminal it is associated with and using the matching defined by m_{VT} for 0-valued terminals in (L_{VT_0},m). Since m'_{VT} matches at least as many "0"s as m_{VT}, $|range(m'_{VT})|$ is at least as large as $|range(m_{VT})|$. (Recall that matching functions are one-to-one so that the domain of a matching function is the same size as its range.)

Induction: Consider the first left-region. By the argument used in the basis, the matching of the "0"s in this region can be restricted to the "1"s in the region. Therefore, we may remove the first region and consider the remaining terminals as a new problem. The left-regions of the remaining terminals are unchanged, and we apply the inductive assumption. The desired m'_{VT} uses the matching of "0"s in the first left-region to "1"s in the first left-region and the matching of "0"s in the remaining full left-regions to "1"s in the remaining full left-regions obtained by the inductive assumption. □

5.4 Assignment within Regions

Given Theorem 5.1, we can show that it suffices to consider functions VT consistent with VT_0 which assign all ?-valued terminals in $(1,L_{VT_0})$ the value "0" and all ?-valued terminals in (R_{VT_0},m) the value "1". This also holds for VB. Theorem 5.2 states this formally. The algorithm for assigning values begins by assigning these values to the originally ?-valued terminals and their pairs. When a conflict arises, i.e. $T(i)\in(1,L_{VT_0})$ and $B(j) = p(T(i))\in(R_{VB_0},n)$ or $T(i)\in(R_{VT_0},m)$ and $B(j) = p(T(i))\in(1,L_{VB_0})$, either choice can be made. We choose to define the algorithm so that assignments are made in the following order:

1. Make all ?-valued terminals in $(1,L_{VT_0})$ and their bottom pairs 0-valued.
2. Make all ?-valued terminals in (R_{VT_0},m) and their bottom pairs 1-valued.
3. Make all ?-valued terminals in $(1,L_{VB_0})$ and their top pairs 0-valued.
4. Make all ?-valued terminals in (R_{VB_0},n) and their top pairs 1-valued.

This order gives a preference to assignments dictated by the top regions. Executing each of 1 through 4 defines new functions VT and VB which extend VT_0 and VB_0 and which may be extended to value functions achieving the maximum matching $(M_{VT_f} + M_{VB_f})$. These new functions may induce new left-regions and right-regions. The algorithm repeatedly computes new L_{VT}, R_{VT}, L_{VB}, and R_{VB}, and applies one of 1 through 4, each in turn, until there are no full regions containing ?-valued terminals.

Theorem 5.2: Let VT and VB be any pair of value functions consistent with VT_0, VB_0, and the pairing function, p. If there are ?-valued terminals under VT_0 in $(1,L_{VT_0})$, then there are functions VT′ and VB′ consistent with VT_0, VB_0, and p, such that all ?-valued terminals in $(1,L_{VT_0})$ under VT_0 are 0-valued under VT′, and $M_{VT'} + M_{VB'} \geq M_{VT} + M_{VB}$. Similarly, there are value functions assigning ?-valued terminals in (R_{VT_0},m), $(1,L_{VB_0})$, or (R_{VB_0},n) the values "1", "0", or "1" respectively, without decreasing the sum of the matchings on T and B.

Proof: We will only prove the statement for fixed values in $(1,L_{VT_0})$. Proofs for the other intervals are analogous. Suppose there are ?-valued terminals in $(1,L_{VT_0})$ under VT_0. If these terminals are 0-valued under VT, we are done. Suppose there are such terminals which are not 0-valued under VT. Define value function VT′ to agree with VT on $(L_{VT_0}+1,m)$ but assign each ?-valued terminal under VT_0 in $(1,L_{VT_0})$ the value "0". Let m_{VT} be a maximum matching function for VT which matches each 0-valued terminal in $(1,L_{VT_0})$ under VT to a 1-valued terminal in $(1,L_{VT_0})$. By Theorem 5.1, such a m_{VT} must exist. Let $m_{VT'}$ be a matching function for VT′ which agrees with m_{VT} on $(L_{VT_0}+1,m)$ and matches each 0-valued terminal in $(1,L_{VT_0})$ under VT′ to a 1-valued terminal in $(1,L_{VT_0})$. Again, $m_{VT'}$ is guaranteed to exist by Theorem 5.1. Then,

$$|\mathrm{range}(m_{VT'})| - |\mathrm{range}(m_{VT})| = |ZEROS_{VT'}(1,L_{VT_0})| - |ZEROS_{VT}(1,L_{VT_0})| = d.$$

But, $M_{VT'} \geq |\mathrm{range}(m_{VT'})|$ and $M_{VT} = |\mathrm{range}(m_{VT})|$. Therefore, $M_{VT'} - M_{VT} \geq d$. The corresponding VB′ differs from VB by changing at most d "?"s or "1"s to "0"s. This can destroy at most d range values of any m_{VB}, giving $M_{VB} - M_{VB'} \leq d$. Therefore $M_{VT'} + M_{VB'} \geq M_{VT} + M_{VB}$. □

It remains to assign 0/1 values to any ?-valued terminals of $T(L_{VT_e}+1,R_{VT_e}-1)$ and $B(L_{VB_e}+1,R_{VB_e}-1)$, where VT_e and VB_e are the last extensions of VT_0 and VB_0 obtained by the above assignments. Intervals $T(L_{VT_e}+1,R_{VT_e}-1)$ and $B(L_{VB_e}+1,R_{VB_e}-1)$ may be empty. Given Theorem 5.1, we can maximize the sum of matches within $T(L_{VT_e}+1,R_{VT_e}-1)$ and $B(L_{VB_e}+1,R_{VB_e}-1)$ independent of the full regions. We remove the full regions and define a new T and B containing only the remaining terminals. We get a problem on vectors $T(1,m')$ and $B(1,n')$ with new pairing function, p', and new initial value functions, VT'_0 and VB'_0, such that

$$m' = R_{VT_e}-L_{VT_e}-1 \text{ and } n' = R_{VB_e}-L_{VB_e}-1,$$

$$VT'_0(i) = VT_e(L_{VT_e}+i) \text{ and } VB'_0(i) = VB_e(L_{VB_e}+i).$$

The pairing function, p', corresponds to the the old pairing function when both terminals of a pair are within $(L_{VT_e}+1,R_{VT_e}-1)$ and $(L_{VB_e}+1,R_{VB_e}-1)$. If a terminal within one of these intervals was originally paired with a terminal outside the intervals, then the new function pairs this terminal with * (don't care). Any terminal originally paired with a terminal outside these intervals is 0-valued or 1-valued under VT_e or VB_e. Since the pairing function is only needed for ?-valued terminals, changing the pairing of such a terminal to * does not change the problem. The new value functions, VT'_0 and VB'_0, induce exactly one left-region and one right-region on each of $T(1,m')$ and $B(1,n')$. None of these regions is full.

The assignment to the new vectors is made using procedure SCAN-ASSIGN. The procedure is called with inputs $(T(1,m'),B(1,n'),p',VT'_0,VB'_0)$. This procedure scans the top vector, reassigning top-bottom pairs of ?-valued terminals the value "0" whenever this assignment would not create a bottom right interval, $B(s,n)$, with more that half 0-valued terminals. When such an interval would result, SCAN-ASSIGN tries to reassign the terminals the value "1". If this would create a top left interval, $T(1,q)$, with more than half 1-valued terminals, the procedure stops and returns "FAIL". The procedure which called SCAN-ASSIGN must handle the failure. If no failure occurs, SCAN-ASSIGN continues assigning until there is a full top right-region, i.e. a right interval with at least half "0"s. Any remaining

?-valued terminals are assigned the value "1". The procedure SCAN-ASSIGN is defined in Figure 5.3.

The following two lemmas and theorem prove the correctness of SCAN-ASSIGN(T(1,m),B(1,n),p,VT_0,VB_0) when it succeeds in assigning "0"s or "1"s to all ?-valued terminals in T(1,m) and B(1,n).

Lemma 5.4: Let VT_i and VB_i be the value functions assumed by variables VT and VB after considering terminal T(i) in either the first or second FOR loop during the execution of SCAN-ASSIGN(T(1,m),B(1,m),p,VT_0 ,VB_0). If VT_0 and VB_0 do not define any full regions on T or B, then for any i, $1 \leq i \leq m$, VT_i and VB_i satisfy the following properties:

For all k, $1 \leq k \leq m$, OK-1(VT_i,1,k)

For all k, $1 \leq k \leq n$, OK-0(VB_i,k,n)

Furthermore, if SCAN-ASSIGN(T(1,m),B(1,m),p,VT_0,VB_0) enters the second FOR loop, then for all value functions VT_j and VB_j, where T(j) is considered in this loop, $|ZEROS_{VT_j}(1,m)| = \lceil \frac{1}{2}m \rceil$.

Proof: We first prove the properties for functions defined in the first FOR loop by induction on i.

Basis: The properties are true for VT_0 and VB_0 by hypothesis.

Induction: By inductive assumption, the properties are true after considering terminal T(i-1), ie. after the (i-1)th execution of the loop. Function VT_i differs from VT_{i-1} at most for T(i). For this to affect OK-1(VT_i,1,k) for some k, the following must hold: $|ONES_{VT_{i-1}}(1,k)| = \lfloor \frac{1}{2}k \rfloor$, $k \geq i$, $VT_{i-1}(i) = ?$, and $VT_i(i) = 1$. However, if $|ONES_{VT_{i-1}}(1,k)| = \lfloor \frac{1}{2}k \rfloor$, then $VT_i(i)$ is not assigned "1". Therefore, for all k, OK-1(VT_i,1,k) holds. For OK-0(VB_i,k,n) not to hold for some k, it must be that: $|ZEROS_{VB_{i-1}}(k,n)| = \lfloor \frac{1}{2}(n-k+1) \rfloor$, $k \leq p(i)$, $VB_{i-1}(p(i)) = ?$, and $VB_i(p(i)) = 0$. However, if $|ZEROS_{VB_{i-1}}(k,n)| = \lfloor \frac{1}{2}(n-k+1) \rfloor$, then $VT_i(i)$ and $VB_i(p(i))$ are not assigned "0". Therefore, OK-0(VB_i,k,n) must hold for all k.

If SCAN-ASSIGN enters the second FOR loop, let h be the first value of j considered in this loop. Execution of the first FOR loop is completed because Full-0(VT_{h-1},q,m) holds for some q. We

Figure 5.3: Definition of SCAN-ASSIGN

SCAN-ASSIGN(T(1,m),B(1,n),p,VT_0,VB_0)

/Assigns 0/1 to ?-valued terminals in T and B where $L_{VT_0}=0$, $R_{VT_0}=m+1$, $L_{VB_0}=0$, $R_{VB_0}=n+1$, and no region is full./

```
    $VT := VT_0$ and $VB := VB_0$
    FOR i = 1 STEP 1 UNTIL $(\exists q)$Full-0(VT,q,m);
        IF VT(i)=? THEN
            IF $(\exists s)$ $p(i) \in B(s,n)$ and $|ZEROS_{VB}(s,n)| = \lfloor \frac{1}{2}(n-s+1) \rfloor$ THEN
                IF $(\exists q)$ $i \leq q \leq m$ and $|ONES_{VT}(1,q)| = \lfloor \frac{1}{2}q \rfloor$ THEN
                    RETURN (FAIL i,VT,VB)
                ELSE VT(i):=1 and VB(p(i)):=1
            ELSE VT(i):=0 and VB(p(i)):=0
    END
    FOR j = i STEP 1 THROUGH m
        IF VT(j)=? THEN VT(j):= 1 and VB(p(j)):=1
    END
    RETURN (SUCCESS,VT,VB)
END SCAN-ASSIGN
```

know that $h \geq 2$ and $q < h$ since VT_{h-1} agrees with VT_0 on $T(h,m)$ and for no k does Full-0(VT_0,k,m) hold. If $q \neq 1$, we know OK-1(VT_{h-1},1,q-1) holds by the first part of this proof. This implies that $|ZEROS_{VT_{h-1}}(1,q-1)| \geq \lceil \frac{1}{2}(q-1) \rceil$ since no terminals in $T(1,h-1)$ are ?-valued under VT_{h-1}. Then

$$|ZEROS_{VT_{h-1}}(1,m)| \geq \lceil \tfrac{1}{2}(q-1) \rceil + \lceil \tfrac{1}{2}(m-q+1) \rceil \geq \lceil \tfrac{1}{2}m \rceil.$$

However, $|ZEROS_{VT_{h-2}}(1,m)| < \lceil \frac{1}{2}m \rceil$; otherwise, the first loop would not have been repeated for h-1. Therefore, $|ZEROS_{VT_{h-1}}(1,m)| = \lceil \frac{1}{2}m \rceil$. The second FOR loop does not assign any "0"s. Therefore, for all VT_j, $h \leq j \leq m$, $|ZEROS_{VT_j}(1,m)| = \lceil \frac{1}{2}m \rceil$. Also, since OK-0($VB_{h-1}$,k,m) holds for any k, OK-0(VB_j,k,m) holds for any k.

It remains to show that OK-1(VT_j,1,k) holds for all k. Since any VT_j agrees with VT_{h-1} on T(1,h-1), OK-1(VT_j,1,k) must hold for $k < h$. Suppose that for some $k \geq h$ and some $j \geq h$, OK-1(VT_j,1,k) does not hold. Since the number of 1-valued terminals increases on each iteration, OK-1(VT_m,1,k) must also not hold. Since no terminal is ?-valued under VT_m, $|ZEROS_{VT_m}(1,m)| = \lceil \frac{1}{2}m \rceil$ implies $|ONES_{VT_m}(1,m)| = \lfloor \frac{1}{2}m \rfloor$; therefore $k \neq m$.

For any $i \geq h$, $\quad |ZEROS_{VT_m}(i,m)| = |ZEROS_{VT_0}(i,m)| < \lceil \frac{1}{2}(m-i+1) \rceil.$

This implies $\quad |ONES_{VT_m}(k+1,m)| > \lfloor \frac{1}{2}(m-(k+1)+1) \rfloor.$

Then $\quad |ONES_{VT_m}(1,k)| < \lfloor \frac{1}{2}m \rfloor - \lfloor \frac{1}{2}(m-k) \rfloor,$

giving $|ONES_{VT_m}(1,k)| < \lceil \frac{1}{2}k \rceil$, contradicting our assumption that OK-1(VT_m,1,k) does not hold. □

Lemma 5.5: For any call to SCAN-ASSIGN as in Lemma 5.4, the first FOR loop is completed with $i \leq m$.

Proof: If the lemma does not hold, then after $i = m$ in the first FOR loop:

$$|ZEROS_{VT_m}(1,m)| < \lceil \tfrac{1}{2}m \rceil.$$

However, by Lemma 5.4, $|ONES_{VT_m}(1,m)| \leq \lfloor \frac{1}{2}m \rfloor$, implying $|ZEROS_{VT_m}(1,m) \geq \lceil \frac{1}{2}m \rceil$. □

Theorem 5.3: Let VT_0 and VB_0 be initial value functions which induce no full regions on T and B. If SCAN-ASSIGN(T(1,m), B(1,n), p, VT_0, VB_0) returns (SUCCESS,VT_m,VB_m), then VT_m and VB_m maximize $M_{VT} + M_{VB}$ over all VT and VB consistent with VT_0 and VB_0.

Proof: Since OK-1(VT_m,1,k) holds for $1 \leq k \leq m$ and $|ONES_{VT_m}(1,m)| = \lfloor ½m \rfloor$, $M_{VT_m} = \lfloor ½m \rfloor$, the maximum possible. Since OK-0(VB_m,k,n) holds for $1 \leq k \leq n$, each 0-valued bottom terminal can be matched, and $M_{VB_m} = |ZEROS_{VT_m}(1,n)|$. If $M_{VB_m} = \lfloor ½n \rfloor$, then no larger matching is possible and we are done. Suppose $M_{VB_m} < \lfloor ½n \rfloor$. Let $M_{VB_m} = \lfloor ½n \rfloor - \Delta$, $\Delta > 0$. Let VT and VB be any other value functions consistent with p, VT_0, and VB_0. We must show that

$$M_{VT} + M_{VB} \leq M_{VT_m} + M_{VB_m} = \lfloor ½n \rfloor - \Delta + \lfloor ½m \rfloor.$$

Since the initially ?-valued terminals are assigned 0/1 values in top-bottom pairs, we have:

$$|ZEROS_{VT_0}(1,m)| - |ZEROS_{VB_0}(1,n)| = |ZEROS_{VT_m}(1,m)| - |ZEROS_{VB_m}(1,n)|$$
$$= \lceil ½m \rceil - \lfloor ½n \rfloor + \Delta \quad \text{and}$$

$$|ZEROS_{VT}(1,m)| - |ZEROS_{VT_0}(1,m)| = |ZEROS_{VB}(1,n)| - |ZEROS_{VB_0}(1,n)|.$$

Let $|ZEROS_{VT}(1,m)| = \lceil ½m \rceil + z$, where z is any integer. Then:

$$|ZEROS_{VB}(1,n)| = |ZEROS_{VT}(1,m)| - (|ZEROS_{VT_0}(1,m)| - |ZEROS_{VB_0}(1,n)|)$$
$$= \lceil ½m \rceil + z - (\lceil ½m \rceil - \lfloor ½n \rfloor + \Delta) = \lfloor ½n \rfloor + z - \Delta.$$

Case 1: $z < 0$. Then $M_{VT} + M_{VB} \leq |ZEROS_{VT}(1,m)| + |ZEROS_{VB}(1,n)|$

$$\leq \lceil ½m \rceil + z + \lfloor ½n \rfloor + z - \Delta < \lfloor ½m \rfloor + \lfloor ½n \rfloor - \Delta.$$

Case 2: $0 \leq z$. Then $M_{VT} + M_{VB} \leq |ONES_{VT}(1,m)| + |ZEROS_{VB}(1,n)|$

$$\leq \lfloor ½m \rfloor - z + \lfloor ½n \rfloor + z - \Delta = \lfloor ½m \rfloor + \lfloor ½n \rfloor - \Delta.$$ □

It remains for us to deal with a return of (FAIL k,VT,VB). Before describing how a failure is handled, we prove that for certain value functions, the value of a terminal can be switched from one of "0" or "1" to the other without decreasing $M_{VT}+M_{VB}$. These value functions are consistent with the initial value functions used as input to SCAN-ASSIGN and assign only "0"s and "1"s. They assign an imbalance of "0"s and "1"s in some left or right interval. We will use this ability to change the values of terminals to prove that the value functions returned by SCAN-ASSIGN do not assign too many or too few "0"s and "1"s to achieve an optimal matching.

Lemma 5.6: Let VT_0 and VB_0 be initial value functions which define no full regions on T and B. Let VT and VB be any function consistent with VT_0, VB_0, and p under which no terminal is ?-valued.

A) For any x, $1 \leq x \leq m$, if $|ONES_{VT}(1,x)| - |ZEROS_{VT}(1,x)| > 1$, then there is a terminal i in T(1,x) such that $VT_0(i)=?$, $VT(i)=1$, and the functions VT' and VB' obtained by changing the values of i and p(i) to "0" are such that $M_{VT'}+M_{VB'} \geq M_{VT}+M_{VB}$.

B) For any y, $1 \leq y \leq n$, if $|ZEROS_{VB}(y,n)| - |ONES_{VB}(y,n)| > 1$, then there is a terminal i in B(y,n) such that $VB_0(i)=?$, $VB(i)=0$, and the functions VT' and VB' obtained by changing the values of i and p(i) to "1" are such that $M_{VT'}+M_{VB'} \geq M_{VT}+M_{VB}$.

Proof: We will only prove A. The proof for B is analogous. Since $|ONES_{VT}(1,x)|-|ZEROS_{VT}(1,x)|>1$, there are at least two "1"s in T(1,x) unmatched under any matching function for VT.

Let there be a matching function achieving M_{VT} for which the terminal of lower index among two unmatched "1"s is ?-valued under VT_0. We will show by contradiction that such a matching function must exist. Given such a matching function, change the value of the terminal of lower index to "0", changing the value of its bottom pair as well. This defines VT' and VB'. The new "0" can match the second unmatched "1", giving $M_{VT'}=M_{VT}+1$. In B(1,n), at most one range value of any matching function has been destroyed; therefore, $M_{VB'} \geq M_{VB}-1$. It follows that $M_{VT'}+M_{VB'} \geq M_{VT}+M_{VB}$.

We now show that the matching function described above must exist. Figure 5.4 illustrates. Suppose that for any matching function achieving M_{VT}, at most one 1-valued terminal in T(1,x) which was initially ?-valued (i.e. under VT_0) is unmatched, and that if there is such a terminal, it is the terminal

Figure 5.4: The proof of Lemma 5.6.

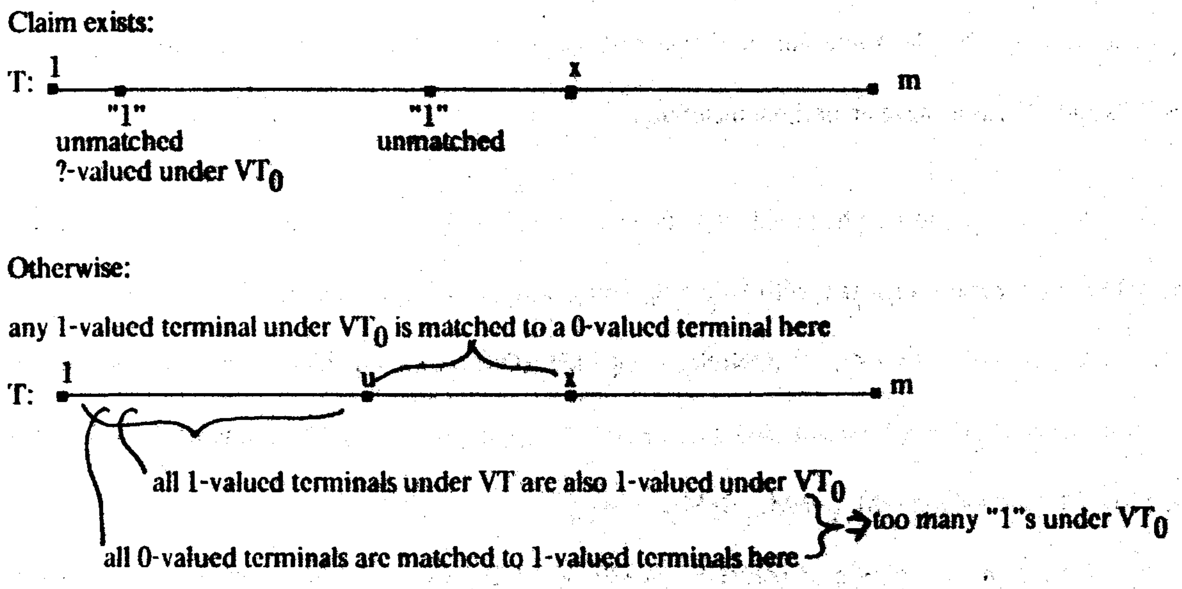
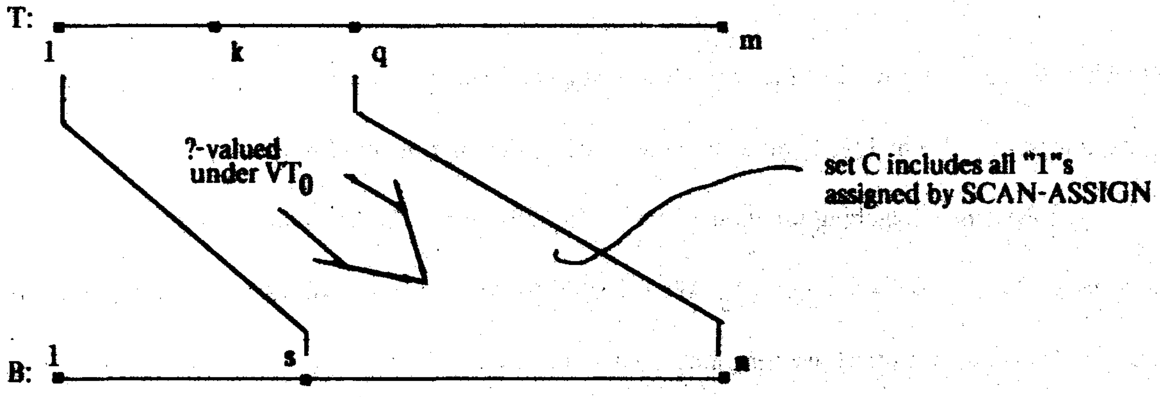

Claim exists:
T:
1
x
m
"1"
unmatched
?-valued under VT_0
"1"
unmatched
Otherwise:
any 1-valued terminal under VT_0 is matched to a 0-valued terminal here
T:
1
u
x
m
all 1-valued terminals under VT are also 1-valued under VT_0
all 0-valued terminals are matched to 1-valued terminals here
too many "1"s under VT_0

Figure 5.5: Definition of C.



of highest index among all unmatched "1"s in $T(1,x)$. Then, for any matching function achieving M_{VT}, there is at least one 1-valued terminal under VT_0 which is unmatched. Consider all matching functions achieving M_{VT} for which a maximum number of 1-valued terminals in $T(1,x)$ are matched. Among these, consider only those functions for which a maximum number of originally ?-valued, now 1-valued terminals in $T(1,x)$ are unmatched (either zero or one such terminals). For each of these, note the indices of any terminals in $T(1,x)$ which are unmatched and are 1-valued under VT_0. Choose the matching function for which the highest index, say u, is obtained. No initially ?-valued terminal in $T(1,u\text{-}1)$ which is 1-valued under VT is unmatched. Each 0-valued terminal in $T(1,u\text{-}1)$ must match an initially 1-valued terminal in $T(1,u\text{-}1)$. Otherwise, the matching function could be modified so that the offending 0-valued terminal matched u. If the offending 0-valued terminal had been unmatched, this would produce a larger matching; if it had matched a terminal in $T(x+1,m)$, this would produce a matching under which more 1-valued terminals in $T(1,x)$ are matched. If the offending terminal had matched an initially ?-valued terminal in $T(1,x)$, matching it to u would produce a matching with a larger number of unmatched 1-valued terminals which are ?-valued under VT_0; if it had matched a terminal in $T(u+1,x)$ which is 1-valued under VT_0, this would produce a matching with an unmatched terminal which is 1-valued under VT_0 and of higher index than u. Any of these possibilities contradict our choice of matching function. There can be no matched 1-valued terminals in $T(1,u\text{-}1)$ which are initially ?-valued nor any unmatched terminals of this type. All 1-valued terminals in $T(1,u\text{-}1)$ are initially 1-valued. Since all 0-valued terminals in $T(1,u\text{-}1)$ must match 1-valued terminals in $T(1,u\text{-}1)$, we have:

$$|ZEROS_{VT}(1,u\text{-}1)| \leq |ONES_{VT}(1,u\text{-}1)| = |ONES_{VT_0}(1,u\text{-}1)|$$

Since no terminals are ?-valued under VT, this implies $|ONES_{VT_0}(1,u\text{-}1)| \geq \lceil ½(u\text{-}1) \rceil$, contradicting the hypothesis that there are no full regions under VT_0. □

5.5 Handling a "failure"

Let us suppose that SCAN-ASSIGN returns (FAIL k,VT,VB). We will now describe what is known about T, B, and their value functions after SCAN-ASSIGN returns. We will use this information to handle the failure. Procedure SCAN-ASSIGN returns the value, k, of loop variable i when SCAN-ASSIGN encountered the failure. Therefore we know that VT_{k-1} and VB_{k-1} are the value functions returned. There is a $q \geq k$ such that $|ONES_{VT_{k-1}}(1,q)| = \lfloor ½q \rfloor$. We will use the smallest such q. There is an s satisfying the following three properties: (i) $p(k) \geq s$, (ii) $|ZEROS_{VB_{k-1}}(s,n)| = \lfloor ½(n-s+1) \rfloor$, and (iii) each terminal in T(1,k-1) which has been assigned the value "1" by SCAN-ASSIGN is paired with a terminal in B(s,n), i.e. $ONES_{VT_{k-1}}(1,k-1)-ONES_{VT_0}(1,k-1) \subseteq p(B(s,n))$. This last property follows from the fact that SCAN-ASSIGN only assigns $VT(i)=1$ when, at the time, $(\exists s)(p(i) \in B(s,n)$ and $|ZEROS_{VB}(s,n)| = \lfloor ½(n-s+1) \rfloor$. Therefore, we can associate an s_i with each ?-valued T(i) made 1-valued. Once $|ZEROS_{VB_i}(s_i,n)| = \lfloor ½(n-s_i+1) \rfloor$ for some VB_i, the number of "0"s in the interval does not change in later iterations. Therefore, choosing the smallest of any such s_i and the s_k associated with the failure, we have an index, s, which satisfies all three properties. In fact, we use the largest s which satisfies all three properties. Note that $VT_{k-1}(k) = VB_{k-1}(p(k)) = ?$. We also know that each terminal in B(s,n) which is assigned a "0" or a "1" value by SCAN-ASSIGN must be paired with a terminal in T(1,k-1), since no initially ?-valued terminals in T(k,m) or their bottom pairs have been reassigned.

Let C be the set of initially ?-valued terminals in T(1,q) whose pairs are in B(s,n) (Figure 5.5). For an optimal assignment at the top, all remaining ?-valued terminals in T(1,q) should become "0"s. For an optimal assignment at the bottom, all remaining ?-valued terminals in B(s,n) should become "1"s. Obviously, on C these are conflicting goals. We wish to find 0/1 assignments for the remaining ?-valued terminals in C (including k) for which some extension will achieve the maximum sum of matches. To do this, we first prove that we need only consider value functions which assign a number of "0"s or "1"s in C within a certain range. Let:

$$c_1 = |ONES_{VT_{k-1}}(C)|$$
$$c_0 = |ZEROS_{VT_{k-1}}(C)|$$
$$c_? = |UNDET_{VT_{k-1}}(C)|$$

To simplify the statement of the lemmas to follow, let "assume the standard state after SCAN-ASSIGN returns 'FAIL'" mean: "Assuming VT_0 and VB_0 are initial value functions which define no full regions on T and B, let SCAN-ASSIGN(T(1,m), B(1,n), p, VT_0, VB_0) return (FAIL k, VT_{k-1}, VT_{k-1}). Let q, s, C, c_0, c_1, and $c_?$ be as we have defined them above."

Lemma 5.7: Assume the standard state after SCAN-ASSIGN returns "FAIL". Let VT and VB be arbitrary value functions consistent with VT_0, VB_0, and p under which there are no ?-valued terminals. If $c_?>1$, then for any x, $1 \leq x \leq c_?-1$, there are value functions VT′ and VB′ also consistent with VT_0, VB_0, and p under which there are no ?-valued terminals such that:

$$M_{VT'} + M_{VB'} \geq M_{VT} + M_{VB} \quad \text{and} \quad |ONES_{VT'}(C)| = c_1 + x$$

If $c_?=1$, there are such functions VT′ and VB′ for which $c_1 \leq |ONES_{VT'}(C)| \leq c_1+1$.

Proof: If s > 1, let D be the set of initially ?-valued terminals in T(1,q) whose pairs are in B(1,s-1).

Then: $|ONES_{VT}(1,q)| = |ONES_{VT}(C)| + |ONES_{VT}(D)| + |ONES_{VT_0}(1,q)|$

Note that $|ONES_{VT_{k-1}}(1,q)| = |ONES_{VT_0}(1,q)| + |ONES_{VT_{k-1}}(C)| = \lfloor ½q \rfloor$

Case 1 $|ONES_{VT}(C)| \geq c_1 + x$, where if $c_?=1$, $x=1$.

We use induction on $|ONES_{VT}(C)| + |ONES_{VT}(D)|$.

Basis: $|ONES_{VT}(C)| + |ONES_{VT}(D)| = c_1 + x$. Then $|ONES_{VT}(C)| = c_1 + x$ as desired.

Induction: Assume that the lemma holds if:

$$c_1+x \leq |ONES_{VT}(C)| + |ONES_{VT}(D)| < c_1+x+i, \; i>0.$$

When $|ONES_{VT}(C)| + |ONES_{VT}(D)| = c_1+x+i$, then:

$$|ONES_{VT}(1,q)| = c_1 + x + i + |ONES_{VT_0}(1,q)| = \lfloor \tfrac{1}{2}q \rfloor + x + i \geq \lfloor \tfrac{1}{2}q \rfloor + 2$$

If $|ONES_{VT}(C)| = c_1 + x$, we are done. Otherwise, we use Lemma 5.6-A.

$$|ONES_{VT}(1,q)| - |ZEROS_{VT}(1,q)| \geq \lfloor \tfrac{1}{2}q \rfloor + 2 - (\lceil \tfrac{1}{2}q \rceil - 2) \geq 3$$

Therefore, by Lemma 5.6-A, there are VT' and VB' such that $M_{VT'} + M_{VB'} \geq M_{VT} + M_{VB}$ and

$$|ONES_{VT'}(C)| + |ONES_{VT'}(D)| = |ONES_{VT}(C)| + |ONES_{VT}(D)| - 1.$$

By inductive assumption, there are VT″ and VB″ such that $M_{VT''} + M_{VB''} \geq M_{VT'} + M_{VB'}$ and $|ONES_{VT''}(C)| = c_1 + x$.

Case 2 $|ONES_{VT}(C)| \leq c_1 + x$, where if $c_? = 1$, $x = 0$.

Then $|ZEROS_{VT}(C)| \geq c_0 + c_1 + c_? - (c_1 + x)) = c_0 + c_? - x$. If $q < m$, let E be the set of initially ?-valued terminals on B(s,n) whose pairs are in T(q+1,m). Let p(C) denote the bottom pairs of C.

$$|ZEROS_{VB}(s,n)| = |ZEROS_{VB}(p(C))| + |ZEROS_{VB}(E)| + |ZEROS_{VB_0}(s,n)|$$

We know $|ZEROS_{VB_{k-1}}(s,n)| = |ZEROS_{VB_{k-1}}(p(C))| + |ZEROS_{VB_0}(s,n)| = \lfloor \tfrac{1}{2}(n-s+1) \rfloor$.

We use induction on $|ZEROS_{VB}(p(C))| + |ZEROS_{VB}(E)|$.

Basis: $|ZEROS_{VB}(p(C))| + |ZEROS_{VB}(E)| = c_0 + c_? - x$. Then $|ZEROS_{VB}(p(C))| = c_0 + c_? - x$, implying $|ONES_{VB}(p(C))| = c_1 + x$ as desired.

Induction: Assume the lemma holds for:

$$c_0 + c_? - x \leq |ZEROS_{VB}(p(C))| + |ZEROS_{VB}(E)| < c_0 + c_? - x + i, \text{ for } i > 0.$$

When $|ZEROS_{VB}(p(C))| + |ZEROS_{VB}(E)| = c_0 + c_? - x + i$:

$|ZEROS_{VB}(s,n)| = c_0 + c_? - x + i + |ZEROS_{VB_0}(s,n)| = \lfloor \tfrac{1}{2}(n-s+1) \rfloor + c_? - x + i \geq \lfloor \tfrac{1}{2}(n-s+1) \rfloor + 2$.

If $|ZEROS_{VB}(p(C))| = c_0 + c_? - x$, then $|ONES_{VT}(C)| = c_1 + x$ as desired. Otherwise, we use Lemma 5.6-B.

$$|ZEROS_{VB}(s,n)| - |ONES_{VB}(s,n)| \geq \lfloor \tfrac{1}{2}(n-s+1) \rfloor + 2 - (\lceil \tfrac{1}{2}(n-s+1) \rceil - 2) \geq 3.$$

By Lemma 5.6-B, there are VT' and VB' such that $M_{VB'} + M_{VB'} \geq M_{VB} + M_{VB}$ and

$$|ZEROS_{VB'}(p(C))| + |ZEROS_{VB'}(E)| = |ZEROS_{VB}(p(C))| + |ZEROS_{VB}(E)| - 1.$$

By inductive assumption, there are VT″ and VB″ such that $M_{VT''} + M_{VB''} \geq M_{VT'} + M_{VB'}$ and $|ONES_{VT''}(C)| = c_1 + x$. □

We now know that there are value functions achieving the maximum sum of matches which assign at least as many "1"s in C as SCAN-ASSIGN has assigned. However, we do not know if the particular choices for 1-valued terminals will lead to an optimal solution. We will now show that except for possibly one terminal which we can easily find, SCAN-ASSIGN has made good choices. Note that the distinction made between $c_? = 1$ and $c_? > 1$ as in Lemma 5.7 is necessary. When $c_? > 1$, regardless of the extensions of VT_0 and VB_0 being considered, there are enough necessarily unmatched "1"s in T(1,q) or unmatched "0"s in B(s,n) so that we can trade off top matches against bottom matches. When $c_? = 1$, we know that an optimal assignment will have at least one unmatched "1" on T(1,q) or one unmatched "0" on B(s,n). However, we cannot trade these against each other because when a "0" is unmatched in B(s,n), the extra "0" in T(1,q) may match a "1" in T(q+1,m). Similarly, an extra "1" in B(s,n) may match a "0" in B(1,s-1). The algorithm must try both choices -- leaving one unmatched "1" in T(1,q) or one unmatched "0" in B(s,n).

The terminal which may have been incorrectly assigned is the smallest numbered 1-valued terminal in B(s,n) which is ?-valued under VB_0. Assume this is terminal i. Given two terminals, u and v, with $u < v$ in T and $p(u) < p(v)$ in B, assigning "0" to u and "1" to v is always preferable to assigning "1" to u and "0" to v. This is because any terminal which v or p(v) can match when they are 0-valued, u or p(u), respectively, can match when they are 0-valued; any terminal which can be matched to u or p(u) when they are 1-valued, can be matched to v or p(v), respectively, when they are 1-valued. Therefore, assigning "0" to u and p(u) and "1" to v and p(v) gives at least as large a matching on each of T and B as the opposite assignment. Therefore, if terminal i and its pair are smaller numbered than p(k) and k, respectively, then it is better to have terminal i 0-valued and terminal k 1-valued than vice versa. However, the preferred assignment is not consistent with VB_{k-1} and VB_{k-1}. We define new value functions which maintain the important properties of VT_{k-1} and VB_{k-1}, but allow the preferred assignment.

Definition(see Figure 5.6): Assume the standard state after SCAN-ASSIGN returns "FAIL". Let i be the terminal of smallest index in B(s,n) which is ?-valued under VB_0 and 1-valued under VB_{k-1}. Certainly, $p(i)<k$. If $i<p(k)$, let h = i; otherwise, let h = p(k). By definition of h, all 1-valued terminals in C are paired with terminals in B(h,n). Define VT_{fx} and VB_{fx} as follows. If h = p(k), then $VT_{fx} = VT_{k-1}$ and $VB_{fx} = VB_{k-1}$. If h ≠ p(k), then VT_{fx} and VB_{fx} agree with VT_{k-1} and VB_{k-1} except at h, k, p(h), and p(k), where:

$$VT_{fx}(k) = 1 \qquad VT_{fx}(p(h)) = ?$$

$$VB_{fx}(p(k)) = 1 \qquad VB_{fx}(h) = ?$$

Note that OK-1(VT_{fx},1,x) holds for all x, $1 \leq x \leq q$, and OK-0(VB_{fx},y,n) holds for all y, $s \leq y \leq n$. The "standard state after SCAN-ASSIGN returns 'FAIL'" will now include h, VT_{fx}, and VB_{fx} as just defined.

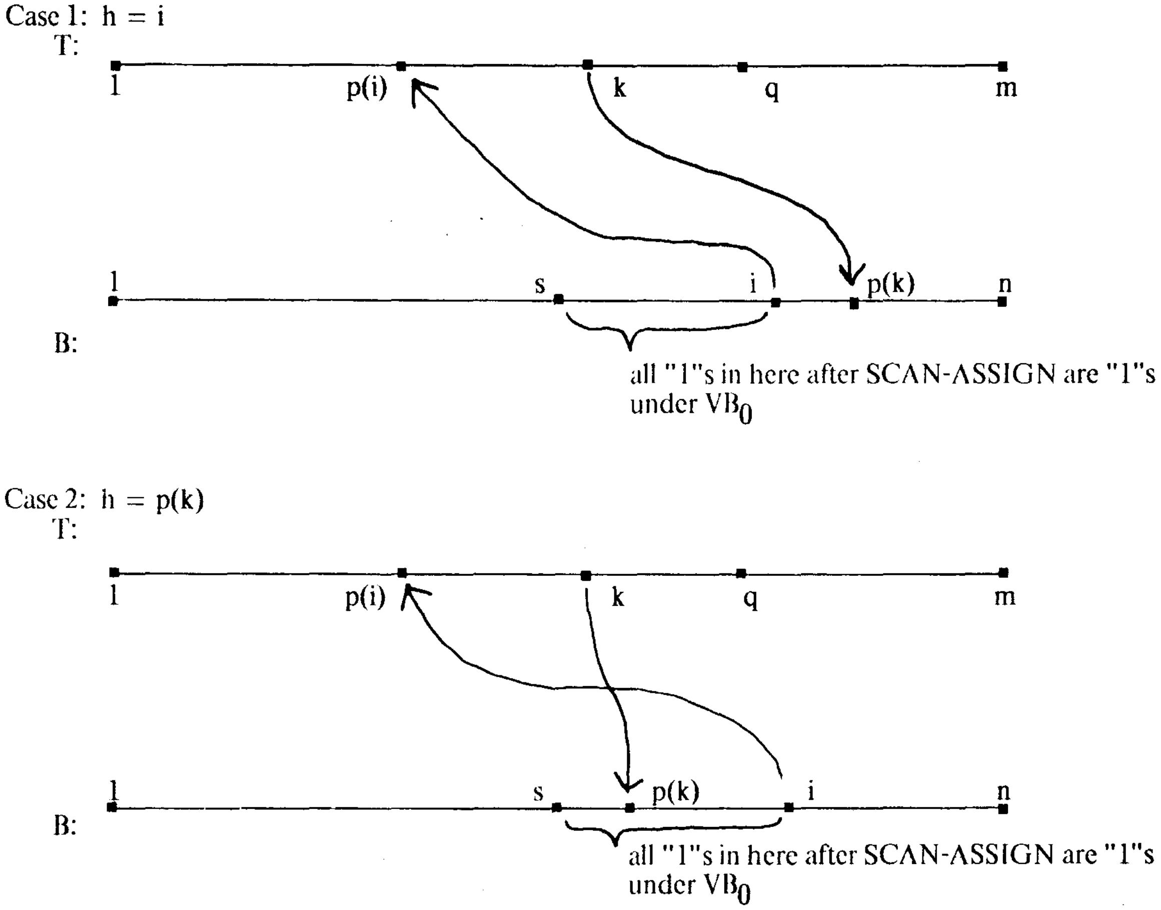
We will now prove that VT_{fx} and VB_{fx} are satisfactory extensions of VT_0 and VB_0, i.e. they will lead to a pair of value functions which achieve the maximum matching $M_{VT_f} + M_{VB_f}$. Lemma 5.8 gives properties of extensions of VT_{fx} and VB_{fx} which will be needed to prove that it is sufficient to consider only these extensions.

Lemma 5.8: Assume the standard state after SCAN-ASSIGN returns "FAIL". Let VT and VB be extensions of VT_{fx} and VB_{fx}.

A. If there is a ?-valued terminal in T(p(h),q) under VT_{fx} which is 1-valued under VT, then, for any 1-valued terminal, i, in T(1,q) under VT, there is a matching function m_{VT} achieving M_{VT} which matches each 0-valued terminal in T(1,q) to a 1-valued terminal in T(1,q) and leaves i unmatched.

B. If there is a ?-valued terminal in B(s,h) under VB_{fx} which is 0-valued under VB, then, for any 0-valued terminal, j, in B(s,n) under VB, there is a matching function m_{VB} achieving M_{VB} which matches

Figure 5.6: Definition of h.



Case 1: $h = i$

all "1"s in here after SCAN-ASSIGN are "1"s under VB_0

Case 2: $h = p(k)$

all "1"s in here after SCAN-ASSIGN are "1"s under VB_0

each 1-valued terminal in B(s,n) to a 0-valued terminal in B(s,n) and leaves j unmatched.

Proof of A: If suffices to show that $|ZEROS_{VT}(x,q)| < |ONES_{VT}(x,q)|$ for all x in T(1,q). If this is true, then we can produce a matching on T(1,q) which matches each "0" in T(1,q) to a "1" in T(1,q) and does not match a given 1-valued terminal i. We define the matching function by scanning T(1,q) from q to 1 and matching each "0" encountered to a yet unmatched "1" of higher index other than i. The fact that $|ZEROS_{VT}(x,q)| < |ONES_{VT}(x,q)|$ guarantees that we will find such an unmatched "1" for each "0". This matching can be extended to a matching which achieves M_{VT} by letting the matching agree with any matching function achieving M_{VT} on "0"s in T(q+1,m).

For any x in T(1,q), we will show that $|ZEROS_{VT}(x,q)| < |ONES_{VT}(x,q)|$ by showing that more than half the terminals in T(x,q) are 1-valued under VT. Suppose $1 \leq x \leq p(h)$. Then, by hypothesis:

$$|ONES_{VT}(x,q)| \geq |ONES_{VT_{fx}}(x,q)| + 1.$$

For $x = 1$, $|ONES_{VT_{fx}}(1,q)| = \lfloor \frac{1}{2}q \rfloor$. For $x > 1$, we know that OK-1(VT_{fx},1,x-1) holds. Therefore,

$$|ONES_{VT_{fx}}(x,q)| \geq \lfloor \tfrac{1}{2}q \rfloor - \lfloor \tfrac{1}{2}(x-1) \rfloor \geq \lfloor \tfrac{1}{2}(q-x+1) \rfloor \text{ and}$$

$$|ONES_{VT}(x,q)| \geq \lfloor \tfrac{1}{2}(q-x+1) \rfloor + 1, \text{ as desired.}$$

Suppose $p(h) < x \leq q$. We claim that $|ONES_{VT_{fx}}(1,x-1)| < \lfloor \frac{1}{2}(x-1) \rfloor$. If true, this gives:

$$|ONES_{VT}(x,q)| \geq |ONES_{VT_{fx}}(x,q)| > \lfloor \tfrac{1}{2}q \rfloor - \lfloor \tfrac{1}{2}(x-1) \rfloor \geq \lfloor \tfrac{1}{2}(q-x+1) \rfloor \text{ as desired.}$$

If $x > k$, then $|ONES_{VT_{fx}}(1,x-1)| = |ONES_{VT_{k-1}}(1,x-1)| < \lfloor \frac{1}{2}(x-1) \rfloor$. Otherwise, we would have chosen x-1 as q, but $x \leq q$. If $x \leq k$, then since $x > p(h)$, $p(h) \neq k$. In this case,

$$|ONES_{VT_{fx}}(1,x-1)| = |ONES_{VT_{k-1}}(1,x-1)| - 1$$

since $VT_{fx}(p(h)) = ?$ and $VT_{k-1}(p(h)) = 1$ and the value functions agree everywhere else on T(1,k-1). Since OK-1(VT_{k-1},1,x-1) holds, $|ONES_{VT_{fx}}(1,x-1)| < \lfloor \frac{1}{2}(x-1) \rfloor$.

Proof of B: It suffices to show that $|ONES_{VB}(s,y)| < |ZEROS_{VB}(s,y)|$ for all y in B(s,n) so that all 1-valued terminals in B(s,n) can be matched to 0-valued terminals in B(s,n) without using j. Produce the matching by scanning from s to n. As in the proof of A, we prove that more than half the terminals in $B(s,y)$ are 0-valued. For $y \geq h$, by hypothesis:

$$|ZEROS_{VB}(s,y)| \geq |ZEROS_{VB_{fx}}(s,y)| + 1.$$

We know $|ZEROS_{VB_{fx}}(s,n)| = \lfloor ½(n-s+1) \rfloor$ and, if $y < n$, OK-0(VB_{fx}, $y+1,n$) holds. Therefore:

$$|ZEROS_{VB_{fx}}(s,y)| \geq \lfloor ½(n-s+1) \rfloor - \lfloor ½(n-(y+1)+1) \rfloor \geq \lfloor ½(y-s+1) \rfloor \text{ and}$$

$$|ZEROS_{VB}(s,y)| \geq \lfloor ½(y-s+1) \rfloor + 1, \text{ as desired.}$$

For $y < h$, $B(y+1,n)$ contains $p(k)$ and all 1-valued terminals in C under VB_{k-1}. Therefore, if $|ZEROS_{VB_{k-1}}(y+1,n)| = \lfloor ½(n-y) \rfloor$, we would have chosen $y+1$ as s, but $y \geq s$. Therefore:

$$|ZEROS_{VB_{fx}}(y+1,n)| = |ZEROS_{VB_{k-1}}(y+1,n)| \leq \lfloor ½(n-y) \rfloor \text{ and}$$

$$|ZEROS_{VB}(s,y)| \geq |ZEROS_{VB_{fx}}(s,y)| > \lfloor ½(n-s+1) \rfloor - \lfloor ½(n-y) \rfloor \geq \lfloor ½(y-s+1) \rfloor. \quad \square$$

Now that we have Lemma 5.8, we can prove a two part theorem which, when combined with Lemma 5.7, proves that in our search for an optimal pair of value functions, it is sufficient to consider only value functions consistent with extensions of VT_{fx} and VB_{fx}. If $c_? > 1$, the extensions give all terminals on $T(1,q)$ and $B(s,n)$ "0" or "1" values; if $c_? = 1$, the terminals h and $p(h)$ are the only terminals in these intervals whose values are not fixed.

Theorem 5.4-A: Assume the standard state after SCAN-ASSIGN returns "FAIL". Let VT and VB be value functions consistent with VT_0, VB_0 and p for which no terminal is ?-valued and such that $|ONES_{VT}(C)| = c_1 + c_? - 1$. There are extensions, VT_{ex} and VB_{ex} of VT_{fx} and VB_{fx} consistent with p and such that:

(i) no terminals are ?-valued

(ii) all ?-valued terminals in $T(1,q)$ under VT_{fx} with pairs in $B(1,s-1)$ are 0-valued under VT_{ex}

(iii) all ?-valued terminals in $B(s,n)$ under VB_{fx} with pairs in $T(q+1,m)$ are 1-valued under VB_{ex}

(iv) if $c_? > 1$, all ?-valued terminals in C under VT_{fx} except p(h) are 1-valued under VT_{ex} ($c_?$-1 of them), and VT_{ex}(p(h)=0

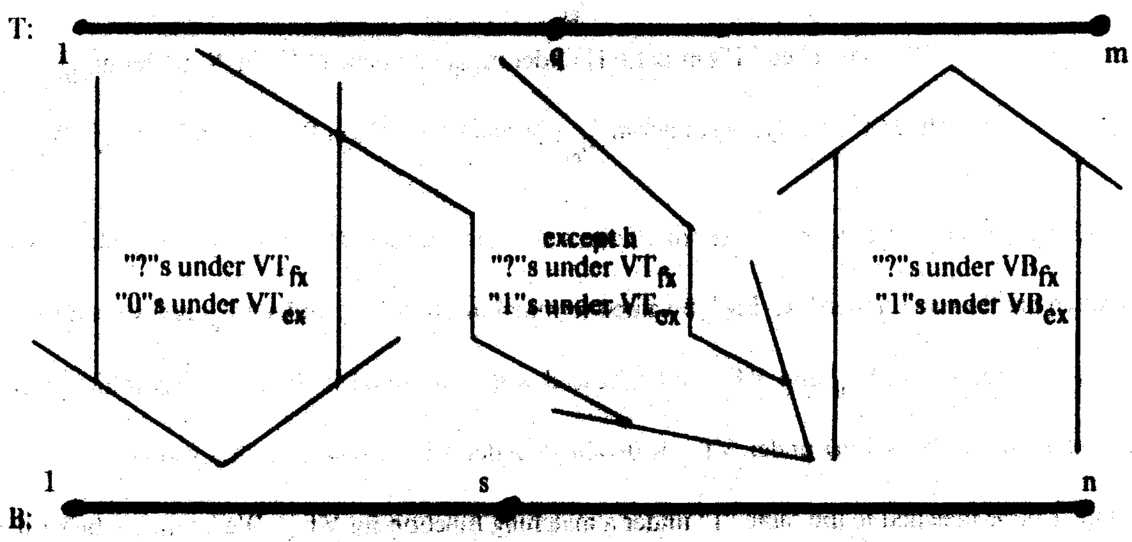
(v) $M_{VT_{ex}} + M_{VB_{ex}} \geq M_{VT} + M_{VB}$.

Proof: Properties (ii) through (iv) define all valued of VT_{ex} and VB_{ex} on T(1,q) and B(s,n) except those of p(h) and h when $c_? = 1$. If $c_? = 1$, we will begin by letting VT_{ex}(p(h)) = VB_{ex}(h) = 0. This may be changed. The values of terminals in T(q+1,m) and B(1,s-1) whose pairs are in B(s,n) and T(1,q) are also defined by ii and iii. Note that ii, iii, and iv deal with disjoint sets of terminals so no conflicts arise. Figure 5.7 illustrates. We still must specify the values under VT_{ex} and VB_{ex} of terminal pairs with one terminal of each pair in T(q+1,m) and the other in B(1,s-1). For ?-valued terminals under VT_{fx} and VB_{fx} of this type, VT_{ex} and VB_{ex} agree with VT and VB.

We prove $M_{VT_{ex}} + M_{VT_{ex}} \geq M_{VT} + M_{VB}$ by comparing the maximum matchings in various intervals and combining. Let m_{VB} be a matching function achieving M_{VB} and m_{VT} be a matching function achieving M_{VT}.

I. Consider B(1,s-1). Let S_1 be the set of ?-valued terminals in B(1,s-1) under VB_0 with pairs in T(1,q) which are 1-valued under VB. The only terminals in B(1,s-1) which are ?-valued under VB_0 and 1-valued under VB_{ex} are those which are ?-valued under VB_{fx} with pairs in T(q+1,m). This follows from the fact that any ?-valued terminal under VB_0 which is 1-valued under VB_{fx} is in B(s,n), and any ?-valued terminal under VB_{fx} with a pair in T(1,q) is 0-valued under VB_{ex}. Functions VB_{ex} and VB agree on all ?-valued terminals in B(1,s-1) under VB_0 with pairs in T(q+1,m). Therefore, every 1-valued terminal in B(1,s-1) under VB_{ex} is 1-valued under VB. Each of these terminals which is matched under m_{VB} can be matched to the same 0-valued terminal under a matching function for VB_{ex}. Let $m_{VB_{ex}}$ be a matching function for VB_{ex} which matches "1"s in B(1,s-1) as m_{VB} does. The matching under $m_{VB_{ex}}$ on B(s,n) will be defined later. We have:

Figure 5.7: Assignments under Theorem A4-A.



$|$matched "1"s in B(1,s-1) under $m_{VB_{ex}}$ $| =$

$|$matched "1"s in B(1,s-1) under $m_{VB}|$ - $|$matched "1"s in S_1 under $m_{VB}|$

$|$matched "1"s in B(1,s-1) under $m_{VB_{ex}}$ $| \geq |$matched "1"s in B(1,s-1) under $m_{VB}| - |S_1|$.

II. Consider T(q+1,m). Let S_0 be the set of ?-valued terminals under VT_0 in T(q+1,m) with pairs in B(s,n) which are 0-valued under VT. Each 0-valued terminal in T(q+1,m) under VT_{ex} is 0-valued under VT_0 or is ?-valued under VT_0 and VT_{fx} and is paired with a terminal in B(1,s-1). Therefore, every 0-valued terminal in T(q+1,m) under VT_{ex} is 0-valued under VT. Every such "0" which is matched under m_{VT} can be matched to the same "1" under a matching function for VT_{ex}. We let $m_{VT_{ex}}$ be such a matching function on T(q+1,m).

$|$matched "0"s in T(q+1,m) under $m_{VT_{ex}}$ $| =$

$|$matched "0"s in T(q+1,m) under $m_{VT}|$ - $|$matched "0"s in S_0 under $m_{VT}|$

$|$matched "0"s in T(q+1,m) under $m_{VT_{ex}}$ $| \geq |$matched "0"s in T(q+1,m) under $m_{VT}| - |S_0|$.

III. Consider B(s,n). Since $VB_{ex}(h) = 0$, Lemma 5.8-B holds. Under VB_{ex}, each 1-valued terminal in B(s,n) can be matched to a 0-valued terminal in B(s,n) while leaving any particular 0-valued terminal unmatched. Complete the matching $m_{VB_{ex}}$ defined on B(1,s-1) in I by such a matching on B(s,n) which leaves h unmatched. We know that:

$|ONES_{VT}(C)| = |ONES_{VT_{ex}}(C)| = c_1+c_?-1$ and

$|$1-valued terminals in B(s,n) under VB_{ex} which are ?-valued under VB_0 and have pairs in T(q+1,m)$|$

= $|$terminals in B(s,n) which are ?-valued under VB_0 and have pairs in T(q+1,m)$|$

= $|$1-valued terminals in B(s,n) under VB which are ?-valued under VB_0 and have pairs in T(q+1,m)$|$

+ $|S_0|$

Then, $|$matched "1"s in B(s,n) under $m_{VT_{ex}}$ $| = |ONES_{VB_{ex}}(s,n)| = |ONES_{VB}(s,n)| + |S_0|$

$\geq |$matched "1"s in B(s,n) under $m_{VB}| + |S_0|$.

IV. Consider T(1,q). We must consider $c_? = 1$ and $c_? > 1$ separately.

For $c_? > 1$, there are other terminals in $\mathrm{UNDET}_{VT_{fx}}(C)$ beside h. These terminals must lie in T(k+1,q), since no terminal except h in T(1,k) is ?-valued under VT_{fx}. Since $k \geq p(h)$ and these terminals are 1-valued under VT_{ex}, Lemma 5.8-A applies. Under VT_{ex}, every "0" in T(1,q) can be matched to a "1" in T(1,q). Complete the matching $m_{VT_{ex}}$ defined on T(q+1,m) in II with such a matching on T(1,q). We know that:

$$|\mathrm{ZEROS}_{VT}(C)| = |\mathrm{ZEROS}_{VT_{ex}}(C)| = c_0 + 1;$$

|0-valued terminals in T(1,q) under VT_{ex} which are ?-valued under VT_0 and have pairs in B(1,s-1)|

= |?-valued terminals in T(1,q) under VT_0 with pairs in B(1,s-1)|

= |0-valued terminals in T(1,q) under VT which are ?-valued under VT_0 and have pairs in B(1,s-1)|

+ $|S_1|$.

Then |matched "0"s in T(1,q) under $m_{VT_{ex}}$| = $|\mathrm{ZEROS}_{VT_{ex}}(1,q)| = |\mathrm{ZEROS}_{VT}(1,q)| + |S_1|$

$\geq$ |matched "0"s in T(1,q) under m_{VT}| + $|S_1|$.

Combining I through IV gives:

$$M_{VT_{ex}} + M_{VB_{ex}} \geq |\mathrm{range}(m_{VT_{ex}})| + |\mathrm{range}(m_{VB_{ex}})| \geq |\mathrm{range}(m_{VT})| + |\mathrm{range}(m_{VB})| = M_{VT} + M_{VB}.$$

For $c_? = 1$, since all ?-valued terminals in T(1,q) under VT_{fx} are 0-valued under VT_{ex}, $\mathrm{ONES}_{VT_{ex}}(1,q) = \mathrm{ONES}_{VT_{fx}}(1,q)$. Since OK-1($VT_{fx}$,1,x) holds for all x in T(1,q), all $\lfloor \frac{1}{2}q \rfloor$ "1"s in T(1,q) can be matched to "0"s in T(1,q) under VT_{ex}. The definition of $m_{VT_{ex}}$ on T(1,q) is such a matching.

|matched "0"s on T(1,q) under $m_{VT_{ex}}$| $= \lfloor \frac{1}{2}q \rfloor = |\mathrm{ZEROS}_{VT_{ex}}(1,q)| - (\lceil \frac{1}{2}q \rceil - \lfloor \frac{1}{2}q \rfloor)$

$\geq |\mathrm{ZEROS}_{VT}(1,q)| + |S_1| - (\lceil \frac{1}{2}q \rceil - \lfloor \frac{1}{2}q \rfloor)$

$\geq$ |matched "0"s in T(1,q) under m_{VT}| + $|S_1|$ $-(\lceil \frac{1}{2}q \rceil - \lfloor \frac{1}{2}q \rfloor)$.

(i) If q is even, the above implies, with I through III, that $M_{VT_{ex}} + M_{VB_{ex}} \geq M_{VT} + M_{VB}$ as desired.

(ii) If any of the inequalities deduced in I through IV is strict, we have

$$M_{VT_{ex}} + M_{VB_{ex}} > M_{VT} + M_{VB} - (\lceil \tfrac{1}{2}q \rceil - \lfloor \tfrac{1}{2}q \rfloor), \text{ implying}$$

$$M_{VT_{ex}} + M_{VB_{ex}} \geq M_{VT} + M_{VB} \text{ as desired.}$$

Assume (i) and (ii) do not hold. If $|S_1| = 0$, then

$$\lfloor \tfrac{1}{2}q \rfloor = |\text{matched "0"s in } T(1,q) \text{ under } m_{VT}| - 1$$

It follows that under m_{VT} more than half the terminals in T(1,q) are matched "0"s. This can only be true if an element of $ZEROS_{VT}(1,q)$ matches an element of $ONES_{VT}(q+1,m)$ under m_{VT}. Each 1-valued terminal in T(q+1,m) under VT is 1-valued under VT_{ex}. Also, any 1-valued terminal in T(q+1,m) matched under $m_{VT_{ex}}$ is matched to a "0" in T(q+1,m) and is matched to the same "0" by m_{VT}. Therefore, there must be an unmatched 1-valued terminal under $m_{VT_{ex}}$ in T(q+1,m). We can extend $m_{VT_{ex}}$ so that the unmatched "0" in T(1,q) matches this "1". Then,

$$|\text{matched "0"s in } T(1,q) \text{ under } m_{VT_{ex}}| = |\text{matched "0"s in } T(1,q) \text{ under } m_{VT}| \text{ and}$$

$$M_{VT_{ex}} + M_{VB_{ex}} \geq M_{VT} + M_{VB}.$$

We now suppose that $|S_1| > 0$. We know that

$$|\text{matched "1"s in } B(1,s\text{-}1) \text{ under } m_{VB_{ex}}| = |\text{matched "1"s in } B(1,s\text{-}1) \text{ under } m_{VB}| - |S_1|. \text{ Also}$$

$$|ZEROS_{VB_{ex}}(1,s\text{-}1)| = |ZEROS_{VB}(1,s\text{-}1)| + |S_1|. \text{ Therefore}$$

$$|ZEROS_{VB_{ex}}(1,s\text{-}1)| - |\text{matched "1"s in } B(1,s\text{-}1) \text{ under } m_{VB_{ex}}| =$$

$$|ZEROS_{VB}(1,s\text{-}1)| + |S_1| - |\text{matched "1"s in } B(1,s\text{-}1) \text{ under } m_{VB}| + |S_1| \geq 2.$$

There must be an unmatched "0" in B(1,s-1) under $m_{VB_{ex}}$, since no "0" in B(1,s-1) matches a "1" in B(s,n) under this matching function. When $m_{VB_{ex}}$ was defined on B(s,n) in III, h was left unmatched. Modify VT_{ex} and VB_{ex} so that $VT_{ex}(p(h)) = VB_{ex}(h) = 1$. On $ZEROS_{VT_{ex}}(q+1,m)$, $m_{VT_{ex}}$ is unchanged. On T(1,q), Lemma 5.8-A now applies and we can define $m_{VT_{ex}}$ so that all 0-valued terminals on T(1,q) are matched to 1-valued terminals on T(1,q).

$$|\text{matched "0"s in } T(1,q) \text{ under } m_{VT_{ex}}| = \lfloor \tfrac{1}{2}q \rfloor \text{ as before.}$$

On $ZEROS_{VB_{ex}}(1,s\text{-}1)$, $m_{VB_{ex}}$ is extended so that a previously unmatched "0" is matched to h. On $ZEROS_{VB_{ex}}(s,n)$, $m_{VB_{ex}}$ is unchanged. Therefore

$M_{VT_{ex}} + M_{VB_{ex}} \geq |range(m_{VT_{ex}})| + |range(m_{VB_{ex}})| = |range(m_{VT})| + |range(m_{VB})| = M_{VT} + M_{VB}$.

Note that we have changed VT_{ex} and VB_{ex} so that p(h) and h are 1-valued. □

Now that we have Theorem 5.4-A, we know how to handle "failures" when $c_? > 1$. By Lemma 5.7, there are value functions VT_f and VB_f consistent with p, VT_0 and VB_0 for which $|ONES_{VT_f}(C)| = c_1 + c_? - 1$ and which achieve the maximum of $M_{VT} + M_{VB}$ over all functions consistent with p, VT_0 and VB_0. Given these functions, we apply Theorem 5.4-A to deduce that there are extensions of VT_{fx} and VB_{fx} consistent with p and satisfying (i) through (iv) of Theorem 5.4-A which achieve $M_{VT_f} + M_{VB_f}$. Therefore, for all terminals which are 0-valued or 1-valued under VT_{fx} and VB_{fx}, we can fix their values to be those under VT_{fx} and VB_{fx}. For all terminals whose values are dictated by (ii) through (iv) of Theorem 5.4-A, we can fix their values as dictated. We now have new initial value functions under which no terminal in T(1,q) or B(s,n) is ?-valued. We are guaranteed that there are value functions which achieve $M_{VT_f} + M_{VB_f}$ and which are consistent with these new initial value functions. Some terminals within T(q+1,m) and B(1,s-1) may have acquired 0/1 values under the new initial value functions. We can recursively apply the algorithm, beginning with computation of left-regions and right-regions, to find extensions of the new initial value functions which maximize the sum of matches on T(1,m) and B(1,n). The number of ?-valued terminals has decreased, since at least h and p(h) have changed from ?-valued to 0-valued.

When $c_? = 1$, we first need a modified version of Theorem 5.4-A. When $c_? = 1$, Lemma 5.7 only guarantees that there are functions achieving the maximum matching, $M_{VT_f} + M_{VB_f}$, with $|ONES_{VT_f}(C)|$ equal to c_1 or $c_1 + 1$. Theorem 5.4-A can only be applied if $|ONES_{VT_f}(C)| = c_1$. Therefore, we prove a modified version, Theorem 5.4-B, for the case that $|ONES_{VT_f}(C)| = c_1 + 1$.

Theorem 5.4-B: Assume the standard state after SCAN-ASSIGN returns "FAIL". Let $c_?=1$. Let VT and VB be value functions consistent with VT_0, VB_0 and p for which no terminal is ?-valued and such that $|ONES_{VT}(C)| = c_T+1$. There are extensions, VT_{ex} and VB_{ex} of VT_{fx} and VB_{fx} consistent with p and such that:

(i) no terminals are ?-valued,

(ii) all ?-valued terminals in T(1,q) under VT_{fx} with pairs in B(1,s-1) are 0-valued under VT_{ex},

(iii) all ?-valued terminals in B(s,n) under VB_{fx} with pairs in T(q+1,m) are 1-valued under VB_{ex},

(iv) $M_{VT_{ex}} + M_{VB_{ex}} \geq M_{VT} + M_{VB}$.

Proof: We need to modify the proof to Theorem 5.4-A so that the roles of intervals T(1,q) and B(s,n) are interchanged. Functions VT_{ex} and VB_{ex} are defined as before except that we begin with $VT_{ex}(p(h)) = VB_{ex}(h) = 1$.

I. In B(1,s-1), as for Theorem 5.4-A.

II. In T(q+1,m), as for Theorem 5.4-A.

III. In B(s,n). Lemma 5.8-B does not hold, but $ZEROS_{VB_{ex}}(s,n) = ZEROS_{VB_{fx}}(s,n)$. Therefore, OK-0($VB_{ex}$,y,n) holds for all y in B(s,n). All $\lfloor ½(n-s+1) \rfloor$ "0"s in B(s,n) can be matched to "1"s in B(s,n) under VB_{ex}. As in the proof of Theorem 5.4-A:

$$|ONES_{VB}(s,n)| + |S_0| = |ONES_{VB_{ex}}(s,n)| \text{ and}$$

$$|\text{matched "1"s on B(s,n) under } m_{VB_{ex}}| = |ONES_{VB_{ex}}(s,n)| - (\lceil ½(n-s+1) \rceil - \lfloor ½(n-s+1) \rfloor)$$

$$\geq |\text{matched "1"s on B(s,n) under } m_{VB}| + |S_0| - (\lceil ½(n-s+1) \rceil - \lfloor ½(n-s+1) \rfloor)$$

IV. Lemma 5.8-A does hold and we can define $m_{VT_{ex}}$ so that p(h) is left unmatched and

$|\text{matched "0"s on } T(1,q) \text{ under } m_{VT_{ex}}| = |ZEROS_{VT_{ex}}(1,q)|$

$$= |ZEROS_{VT}(1,q)| + |S_1| \geq |\text{matched "0"s in } T(1,q) \text{ under } m_{VT}| + |S_1|$$

We have $M_{VT_{ex}} + M_{VB_{ex}} \geq M_{VT} + M_{VB} - (\lceil \frac{1}{2}(n-s+1)\rceil - \lfloor \frac{1}{2}(n-s+1)\rfloor)$. If n-s+1 is even or any of the inequalities in I through IV are strict, we have the desired result. Assume not. If $|S_0| = 0$, then

$$\lfloor \tfrac{1}{2}(n-s+1)\rfloor + 1 = |\text{matched "1"s on } B(s,n) \text{ under } m_{VB}|.$$

There must be a 0-valued terminal in B(1,s-1) which matches a 1-valued terminal in B(s,n) under m_{VB}. This 0-valued terminal under VB is also 0-valued under VB_{ex} and unmatched under $m_{VB_{ex}}$. We can modify $m_{VB_{ex}}$ so that the unmatched "1" in B(s,n) matches this "0", giving

$$M_{VT_{ex}} + M_{VB_{ex}} \geq M_{VT} + M_{VB}.$$

If $|S_0| > 0$, we modify VT_{ex} and VB_{ex}. We have:

$$|ONES_{VT_{ex}}(q+1,m)| - |\text{matched "0"s in } T(q+1,m) \text{ under } m_{VT_{ex}}|$$

$$= |ONES_{VT}(q+1,m)| + |S_0| - (|\text{matched "0"s in } T(q+1,m) \text{ under } m_{VT}| - |S_0|) \geq 2$$

We conclude that there is an unmatched 1-valued terminal in T(q+1,m) under $m_{VT_{ex}}$. Let $VT_{ex}(p(h)) = VB_{ex}(h) = 0$. Now p(h) can match the unmatched 1-valued terminal in T(q+1,m) and $|range(m_{VT_{ex}})|$ is increased by 1. In B(s,n), Lemma 5.8-B now applies and we still have

$$|\text{matched "1"s in } B(s,n) \text{ under } m_{VB_{ex}}| = \lfloor \tfrac{1}{2}(n-s+1)\rfloor$$

We conclude that $M_{VT_{ex}} + M_{VB_{ex}} \geq M_{VT} + M_{VB}$. □

Given Theorem 5.4-B in addition to Theorem 5.4-A, we can deduce that there are value functions consistent with VT_{fx} and VB_{fx} and satisfying (ii) and (iii) of Theorem 5.4-A/B which achieve the maximum of $M_{VT} + M_{VB}$ over all functions consistent with p, VT_0 and VB_0. However, we do not know what the value of p(h) and h should be. If the terminal pair is assigned "0", there is an extra "0" at the top which may be matched to some terminal in T(q+1,m), but one 0-valued terminal in B(s,n) will be unmatched. If the pair is assigned "1", the opposite is true. Therefore, we do not know which to choose until we have completed assignments within T(q+1,m) and B(1,s-1). Since p(h) and h may be the only

?-valued terminals in T(1,q) and B(s,n) under VT_0 and VB_0, we cannot use a recursive application of the algorithm. Therefore, the algorithm must try both possibilities for h and p(h). If we were to apply the algorithm recursively for each choice, the running time could be exponential in the number of terminals. Therefore, we use a modification of the procedures we have described so far. The main procedure, which given VT_0 and VB_0 finds VT and VB which maximize $M_{VT} + M_{VB}$, is called MAX-MATCH. The modified version is called BETTER-MATCH.

When we are trying value "1" for h and p(h), we will recursively apply MAX-MATCH, beginning with the computation of left-regions and right-regions; on T(1,m) and B(1,n) for initial value functions which extend VT_{fx} and VB_{fx} as dictated by properties (ii) and (iii) of Theorem 5.4-A/B and under which p(h) and h are 1-valued. When we are trying value "0" for h and p(h), we use initial valued functions which are the same as for the first choice except that p(h) and h are 0-valued. However, in this case, BETTER-MATCH is used. Procedure BETTER-MATCH does not look for a pair of value functions consistent with the new initial value functions and which maximizes $M_{VT} + M_{VB}$ over all consistent function pairs. Rather, the procedure looks for a pair of consistent functions which maximizes $M_{VT} + M_{VB}$ over all consistent function pairs and achieves a better matching than any function pair consistent with the initial value functions when p(h) and h are 1-valued. Consider all function pairs which maximize the sum of matchings on T and B over all functions consistent with p, VT_{fx} and VB_{fx}. If under one of these, h and p(h) are 1-valued, then a function pair achieving a better matching with p(h) and h 0-valued does not exist. Looking only for better matchings allows us to bound the search space of the algorithm so that exponential running time is avoided. Note that we could equally well have chosen to look for maximizing matchings under the "0" choice and better matchings under the "1" choice. Lemma 5.9 is used to bound the search.

Lemma 5.9 (see Figure 5.8): Assume the standard state after SCAN-ASSIGN returns "FAIL". Let $c_7=1$. Let VT_{ex0} and VB_{ex0} be value functions consistent with VT_{fx}, VB_{fx}, p, and (ii) and (iii) of Theorem 5.4-A/B under which p(h) and h are 0-valued. If under some matching function $m_{VB_{ex0}}$ which achieves $M_{VB_{ex0}}$, there is an unmatched "0" in B(1,s-1), then there are valued functions VT_{ex1} and VB_{ex1} consistent with all of the above and under which p(h) and h are 1-valued such that

$$M_{VT_{ex1}} + M_{VB_{ex1}} \geq M_{VT_{ex0}} + M_{VB_{ex0}}.$$

Proof: Let VT_{ex1} and VB_{ex1} agree with VT_{ex0} and VB_{ex0} everywhere except at p(h) and h, where $VT_{ex1}(p(h)) = VB_{ex1}(h) = 1$. Define $m_{VT_{ex1}}$ to assign exactly those matches which are assigned by $m_{VT_{ex0}}$ and do not involve p(h), where $m_{VT_{ex0}}$ achieves $M_{VT_{ex0}}$. Then:

$$M_{VT_{ex1}} \geq |\mathrm{range}(m_{VT_{ex1}})| \geq |\mathrm{range}(m_{VT_{ex0}})| - 1 = M_{VT_{ex0}} - 1$$

Since VB_{ex0} is an extension of VB_{fx} and $VB_{ex0}(h)=0$, Lemma 5.8-B applies. Let $m'_{VB_{ex0}}$ be a matching function for VB_{ex0} which agrees with $m_{VB_{ex0}}$ for each range value in B(1,s-1) but matches each "1" in B(s,n) to a "0" in B(s,n) while leaving h unmatched. Any "0" in B(1,s-1) which was unmatched under $m_{VB_{ex0}}$ is unmatched under $m'_{VB_{ex0}}$.

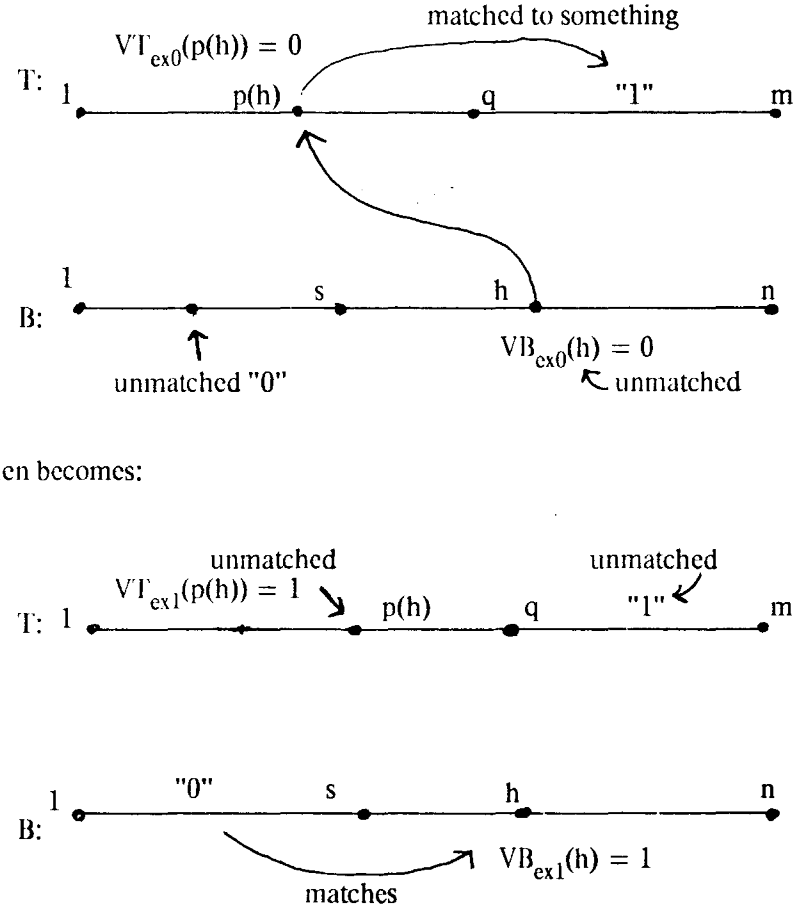
$$|\mathrm{range}(m'_{VB_{ex0}})| = |\text{matched "1's on B(1,s-1) under } m_{VB_{ex0}}| + |\mathrm{ONES}_{VB_{ex0}}(s,n)|$$
$$= |\mathrm{range}(m_{VB_{ex0}})| = M_{VB_{ex0}}$$

Let $m_{VB_{ex1}}$ be a matching function for VB_{ex1} which assigns all matches that $m'_{VB_{ex0}}$ assigns, and, in addition, matches h to an unmatched 0-valued terminal in B(1,s-1). Then

$$M_{VB_{ex1}} \geq |\mathrm{range}(m_{VB_{ex1}})| = |\mathrm{range}(m'_{VB_{ex0}})| + 1 = M_{VB_{ex0}} + 1 \text{ and}$$
$$M_{VT_{ex1}} + M_{VB_{ex1}} \geq M_{VT_{ex0}} + M_{VB_{ex0}}$$ □

Lemma 5.9 allows us to modify the assignment procedures described so far while pursuing the 0-valued choice for p(h) and h. While pursuing this choice, suppose that MAX-MATCH would make an assignment giving a new pair of value functions, either after computing left-regions and right-regions or

Figure 5.8: Configuration in proof of Lemma 5.9.



If:

Then becomes:

after processing a return of "FAIL" from a call of SCAN-ASSIGN. Suppose we can deduce that there is a pair of value functions, VT_{f0} and VB_{f0}, consistent with pairing function p and the new value functions, which maximizes $M_{VT}+M_{VB}$ over all such consistent functions but for which a matching function achieving $M_{VB_{f0}}$ leaves a "0" in B(1,s-1) unmatched. Then BETTER-MATCH need not consider any pair of functions consistent with the new value functions. None will induce better matchings than the functions found when pursuing the 1-valued choice for p(h) and h. This true because Lemma 5.9 can be applied to VT_{f0} and VB_{f0} to give functions yielding as good a sum of matchings under which h and p(h) are 1-valued.

If the pair of new value functions which has been rejected is one of two choices after a call on SCAN-ASSIGN which returned "FAIL", then this choice is eliminated. BETTER-MATCH never needs to pursue two choices. If the pair of new value functions is defined by assignment to left-regions and right-regions, then by Theorem 5.2, any pair of value functions maximizing $M_{VT}+M_{VB}$ over value functions consistent with p and the new value functions also maximizes $M_{VT}+M_{VB}$ over value functions consistent with p and the old value functions. Since there is such a function under which a "0" in B(1,s-1) can be left unmatched, no function consistent with the old valued function will produce a sum of matchings better that that produced under the 1-valued choice for h and p(h). Procedure BETTER-MATCH need not pursue functions consistent with the old value functions either. Therefore, BETTER-MATCH returns "(null,null)", indicating that the 0-valued choice for h and p(h) will not yield a larger sum of matchings than the 1-valued choice. For the same reason, BETTER-MATCH returns "(null,null)" if $c_?>1$ when SCAN-ASSIGN returns "FAIL".

The main procedure, MAX-MATCH, is presented in Figure 5.9. Our original routing problem is solved by calling MAX-MATCH(T(1,m),B(1,n),p,VT_0,VB_0), where VT_0 and VB_0 represent the local connections. The modified procedure, BETTER-MATCH, used when processing a 0-valued choice for p(h) and h, is presented in Figure 5.10. Suppose BETTER-MATCH(T(1,m),B(1,n),p,VT_0,VB_0) returns

Figure 5.9: The main procedure.

```
MAX-MATCH(T(1,m),B(1,n),p,$VT_0$,$VB_0$)

/Finds extensions of $VT_0$ and $VB_0$ which maximize $M_{VT}+M_{VB}$ over all such extenstions/

        VT: = $VT_0$ and VB: = $VB_0$
        Compute $L_{VT}$, $L_{VB}$, $R_{VT}$, and $R_{VB}$
        DO WHILE there are any ?-valued terminals in T(1,$L_{VT}$), T($R_{VT}$,m), B(1,$L_{VB}$), and B($R_{VB}$,n)
                FOR each ?-valued terminal x in T(1,$L_{VT}$) DO
                        VT(x): = 0 and VB(p(x)): = 0
                END
                Compute $R_{VT}$, and $R_{VB}$                    /assigning "0"s only affects right-regions/
                FOR each ?-valued terminal x in T($R_{VT}$,m) DO
                        VT(x): = 1 and VB(p(x)): = 1
                END
                Compute $L_{VT}$ and $L_{VB}$                     /assigning "1"s only affects left-region/
                FOR each ?-valued terminal y in B(1,$L_{VB}$) DO
                        VB(y): = 0 and VT(p(y)): = 0
                END
                Compute $R_{VT}$, and $R_{VB}$
                FOR each ?-valued terminal y in B($R_{VB}$,n) DO
                        VB(y): = 1 and VT(p(y)): = 1
                END
                Compute $L_{VT}$ and $L_{VB}$
        END
        IF there are any ?-valued terminals in T($L_{VT}$+1,$R_{VT}$-1) and B($L_{VB}$+1,$R_{VB}$-1) THEN DO
                m': = $R_{VT}$-$L_{VT}$-1
                n': = $R_{VB}$-$L_{VB}$-1
                VT' is such that VT'(x) = VT($L_{VT}$+x)
                VB' is such that VB' = VB($L_{VB}$+x)
                p' maintains pairs corresponding to those under p when both terminals of the pair are
                    in T(1,m') and B(1,n'). For other terminals in T(1,m') and B(1,n'), p' assigns "*"
                (STATUS, $VT_r$, $VB_r$): = SCAN-ASSIGN(T(1,m'),B(1,n'),p',VT',VB')
                IF STATUS = "FAIL k" THEN DO
                        VT': = $VT_r$ and VB': = $VB_r$
                        Calculate q, s, h, and $c_r$
                        IF p'(h)≠k THEN VT'(k): = 1 and VB'(p'(k)): = 1
                        FOR all x in T(1,q) other that p'(h) and k DO
                                IF VT'(x) = ? THEN
                                        IF p'(x) is in B(1,s-1) THEN
                                                VT'(x): = 0 and VB'(p'(x)): = 0
                                        ELSE VT'(x): = 1 and VB'(p'(x)): = 1
                        END
```

```
            FOR all y in B(s,n') DO
                IF VB'(y)=? and p'(y) is in T(q+1, m') THEN
                    VB'(y):=1 and VT'(p'(y)):=1
            END
            IF $c_?$ > 1 THEN DO
                VB'(h):=0 and VT'(p'(h)):=0
                ($VT_r$,$VB_r$):= (MAX-MATCH(T(1,m'), B(1,n'),p',VT',VB'))
            END
            IF $c_?$ = 1 THEN DO
                VB'(h):=1 and VT'(p'(h)):=1
                ($VT_{r1}$,$VB_{r1}$):=MAX-MATCH(T(1,m'),B(1,n'),p',VT',VB'))
                VB'(h):=0 and VT'(p'(h)):=0
                ($VT_{r0}$,$VB_{r0}$):=BETTER-MATCH(T(1,m'),B(1,n'),s-1,p',VT',VB'))
                IF $VT_{r0}$=null OR $M_{VT_{r0}}+M_{VB_{r0}} \leq M_{VT_{r1}}+M_{VB_{r1}}$ THEN
                    ($VT_r$,$VB_r$):=($VT_{r1}$,$VB_{r1}$)
                ELSE ($VT_r$,$VB_r$):=($VT_{r0}$,$VB_{r0}$)
            END
        END
        FOR each x in T($L_{VT}$+1,$R_{VT}$-1) DO
            VT(x):= $VT_r$(x-$L_{VT}$)
        END
        FOR each y in B($L_{VB}$+1,$R_{VB}$-1) DO
            VB(x):= $VB_r$(x-$L_{VB}$)
        END
    END
    RETURN(VT,VB)
END MAX-MATCH
```

Figure 5.10: The modified procedure for 0-valued choices.

```
BETTER-MATCH(T(1,m),B(1,n),d,p,$VT_0$,$VB_0$)

/If can guarantee that a pair of extensions of $VT_0$ and $VB_0$ which maximize $M_{VT}+M_{VB}$ over all such
pairs of extensions allow an unmatched 0-valued terminal in B(1,d) under a maximum matching, then
returns (null,null). Otherwise, returns a pair of extensions of $VT_0$ and $VB_0$. If d < n, it is assumed that
B(d+1,n) has no ?-valued terminals under $VB_0$ and that matchings on B(d+1,n) and B(1,d) can be
maximized independent of each other./

    VT: = $VT_0$ and VB: = $VB_0$
    Compute $L_{VT}$, $L_{VB}$, $R_{VT}$, and $R_{VB}$
    IF there is a right-region in B(1,d) containing exactly one terminal THEN
            RETURN(null,null)
    DO WHILE there are any ?-valued terminals in T(1,$L_{VT}$), T($R_{VT}$,m), B(1,$L_{VB}$), and B($R_{VB}$,n)
            FOR each ?-valued terminal x in T(1,$L_{VT}$) DO
                    IF p(x) is in B($R_{VB}$,d) THEN RETURN (null,null)
                    ELSE VT(x): = 0 and VB(p(x)): = 0
            END
            Compute $R_{VT}$, and $R_{VB}$
            IF there is a right-region in B(1,d) containing exactly one terminal THEN
                    RETURN(null,null)
            FOR each ?-valued terminal x in T($R_{VT}$,m) DO
                    VT(x): = 1 and VB(p(x)): = 1
            END
            Compute $L_{VT}$ and $L_{VB}$
            FOR each ?-valued terminal y in B(1,$L_{VB}$) DO
                    VB(y): = 0 and VT(p(y)): = 0
            END
            Compute $R_{VT}$, and $R_{VB}$
            IF there is a right-region in B(1,d) containing exactly one terminal THEN
                    RETURN(null,null)
            FOR each ?-valued terminal y in B($R_{VB}$,n) DO
                    VB(y): = 1 and VT(p(y)): = 1
            END
            Compute $L_{VT}$ and $L_{VB}$
    END
    IF there are any ?-valued terminals in T($L_{VT}+1$,$R_{VT}-1$) and B($L_{VB}+1$,$R_{VB}-1$) THEN DO
            m': = $R_{VT}-L_{VT}-1$
            n': = $R_{VB}-L_{VB}-1$
            VT' is such that VT'(x) = VT($L_{VT}+x$)
            VB' is such that VB'(x) = VB($L_{VB}+x$)
            p' maintains pairs corresponding to those under p when both terminals of the pair are
                in T(1,m') and B(1,n'). For other terminals in T(1,m') and B(1,n'), p' assigns "*"
            (STATUS, $VT_r$, $VB_r$): = SCAN-ASSIGN(T(1,m'),B(1,n'),p',VT',VB')
```

```
        IF STATUS = "FAIL,k" THEN DO
            $VT' := VT_r$ and $VB' := VB_r$
            Calculate q, s, h, and $c_?$
            IF $c_? > 1$ THEN RETURN (null,null)
            IF $c_? = 1$ THEN DO
                FOR all x in T(1,q) DO
                    IF $VT'(x) = ?$ THEN
                        IF $p'(x)$ is in B(1,s-1) THEN
                            $VT'(x) := 0$ and $VB'(p'(x)) := 0$
                        ELSE $VT'(x) := 1$ and $VB'(p'(x)) := 1$
                      /Only k and $p'(k)$ are assigned by the ELSE statement/
                END
                FOR all y in $B(s,n')$ DO
                    IF $VB'(y) = ?$ and $p'(y)$ is in $T(q+1,m')$ THEN
                        $VB'(y) := 1$ and $VT'(p'(y)) := 1$
                END
                $(VT_r, VB_r) :=$ (BETTER-MATCH($T(1,m'), B(1,n'), n', p', VT', VB'$))
                IF $VT_r$ = null THEN RETURN (null,null)
            END
        END
        FOR each x in $T(L_{VT}+1, R_{VT}-1)$ DO
            $VT(x) := VT_r(x-L_{VT})$
        END
        FOR each y in $B(L_{VB}+1, R_{VB}-1)$ DO
            $VB(x) := VB_r(x-L_{VB})$
        END
    END
    RETURN(VT,VB)
END BETTER-MATCH
```

VT_f and VB_f which are not "null". Functions VT_f and VB_f may not maximize $M_{VT}+M_{VB}$ over all value functions consistent with p, VT_0 and VB_0. However, if $M_{VT_f}+M_{VB_f}$ is better that the maximum sum of matches for the 1-valued choice, then $M_{VT_f}+M_{VB_f}$ does maximize $M_{VT}+M_{VB}$. Lemma 5.10 states the property of $M_{VT_f}+M_{VB_f}$ which allows us to make this conclusion.

Lemma 5.10: Let VB_0 and d be such that if $d<n$, no terminal in $B(d+1,n)$ is ?-valued, $|ZEROS_{VB_0}(d+1,n)| = \lfloor \frac{1}{2}(n-d) \rfloor + 1$, and for any 0-valued terminal j in $B(d+1,n)$, each 1-valued terminal in $B(d+1,n)$ can be matched to a 0-valued terminal in $B(d+1,n)$ while leaving j unmatched. If BETTER-MATCH($T(1,m),B(1,n),d,p,VT_0,VB_0$) returns "(null,null)", then there is a pair of value functions consistent with p, VT_0 and VB_0 maximizing $M_{VT}+M_{VB}$ over all such consistent functions and for which there is a matching function achieving the maximum matching on B which does not match all "0"s in $B(1,d)$. If BETTER-MATCH($T(1,m),B(1,n),d,p,VT_0,VB_0$) returns "($VT_f,VB_f$)" with VT_f and VB_f not "null", then there are no ?-valued terminals under VT_f and VB_f, and if there are VT and VB consistent with p, VT_0 and VB_0 for which $M_{VT}+M_{VB} > M_{VT_f}+M_{VB_f}$, then there are VT_g and VB_g consistent with p, VT_0 and VB_0 for which $M_{VT_g}+M_{VB_g} \geq M_{VT}+M_{VB}$ and under which some matching function achieving M_{VB_g} leaves a "0" in $B(1,d)$ unmatched.

Proof: By induction on the number of recursive calls to BETTER-MATCH before the call being considered returns.

Basis: no recursive calls. By Theorem 5.2, any extensions of VT_0 and VB_0 defined in the WHILE loop, which assigns within full regions, can be extended to functions maximizing $M_{VT}+M_{VB}$ over all functions consistent with p, VT_0 and VB_0. We know that if $d<n$, $d+1$ must be the end of a full right-region. Otherwise, we would have:

$$|ZEROS_{VB_0}(d+1,r(d+1))| < \lceil \tfrac{1}{2}(r(d+1)-d) \rceil, \text{ which implies}$$

$$|ONES_{VB_0}(d+1,r(d+1))| > \lfloor \tfrac{1}{2}(r(d+1)-d) \rfloor.$$

In this case, not all "1"s in $B(d+1,r(d+1))$ could be matched to "0"s in $B(d+1,n)$, contradicting part of the hypothesis. It follows that $d \geq R_{VB_0}$. Let VB and VT be equal to VB_0 and VT_0 or extensions of VB_0 and VT_0 defined in the WHILE loop. If under VB there is a right-region containing exactly one terminal, say j, then:

$$|ZEROS_{VB}(j,d)| = |ZEROS_{VB}(j+1,d)|+1 \geq \lceil ½(d-j)\rceil+1 = \lfloor ½(d-j+1)\rfloor+1$$

For any extension of VB, there will be a matching function maximizing the matching on B which matches all "1"s in $B(d+1,n)$ to "0"s in $B(d+1,n)$ and which leaves a "0" in $B(j,d)$ unmatched. Therefore, "(null,null)" is returned correctly when there is a right-region containing exactly one terminal.

If under some VT defined as above there is a ?-valued terminal in $T(1,L_{VT})$ with its pair in $B(R_{VB},d)$, then by Theorem 5.2, there are extensions, VT_{ex} and VB_{ex}, of VB and VT which assign this terminal the value "0" and maximize the sum of matchings on T and B over all extensions of VT and VB. Then:

$$|ZEROS_{VB_{ex}}(R_{VB},d)| \geq \lceil ½(d-R_{VB}+1)\rceil+1.$$

There is a matching function achieving $M_{VB_{ex}}$ which matches all "1"s in $B(d+1,n)$ to "0"s in $B(d+1,n)$ and leaves a "0" in $B(R_{VB},d)$ unmatched. Therefore, "(null,null)" is returned correctly.

Let VT_w and VB_w be the final extensions of VT_0 and VB_0 upon exiting the WHILE loop. If no terminals in $T(L_{VT_w}+1,R_{VT_w}-1)$ and $B(L_{VB_w}+1,R_{VB_w}-1)$ are ?-valued, then VT_w and VB_w maximize $M_{VT}+M_{VB}$ over all functions consistent with p, VT_0 and VB_0. There are no functions consistent with p, VT_0 and VB_0 for which $M_{VT}+M_{VB} > M_{VT_w}+M_{VB_w}$, and (VT_w,VB_w) is correctly returned. If there are ?-valued terminals, let $T(1,m')$ and $B(1,n')$ be intervals $T(L_{VT_w}+1,R_{VT_w}-1)$ and $B(L_{VB_w}+1,R_{VB_w}-1)$ when renumbered. Let VT'_w and VB'_w be the restrictions of VT_w and VB_w to these intervals. Let VT_r and VB_r be a pair of valued functions which maximizes the sum of matchings on $T(1,m')$ and $B(1,n')$ over all functions consistent with p', VT'_w and VB'_w. Theorem 5.1 guarantees that the function pair which agrees with VT_r and VB_r on $T(L_{VT_w}+1,R_{VT_w}-1)$ and $B(L_{VB_w}+1,R_{VB_w}-1)$ and agrees with VT_w and VB_w

elsewhere maximizes the sum of matchings on T and B over all functions consistent with p, VT_w and VB_w. When SCAN-ASSIGN returns (SUCCESS,VT_{sc},VB_{sc}), VT_{sc} and VB_{sc} maximize the sum of matchings on T(1,m′) and B(1,n′) and do not leave any terminals ?-valued. Therefore, BETTER-MATCH correctly returns functions which maximize $M_{VT}+M_{VB}$ over all functions consistent with p, VT_0 and VB_0.

If SCAN-ASSIGN returns (FAIL,k,VT_{sc},VB_{sc}) and $c_7>1$, then by Lemma 5.7 and Theorem 5.4-A, there is a terminal s and a pair of functions, VT_r and VB_r, which maximizes the sum of matchings on T(1,m′) and B(1,n′) over all functions consistent with p′, VT_w' and VB_w' and for which

$$|ZEROS_{VB_r}(s,n')| = \lfloor \tfrac{1}{2}(n'-s+1)\rfloor+1.$$

Any matching function for VB_r leaves a "0" in B(s,n′) unmatched. The functions on T(1,m) and B(1,n) which agree with VT_r and VB_r on $T(L_{VT_w}+1,R_{VT_w}-1)$ and $B(L_{VB_w}+1,R_{VB_w}-1)$ and agree with VT_w and VB_w elsewhere maximize $M_{VT}+M_{VB}$ over all functions consistent with p, VT_0 and VB_0. Given these functions, there is a maximum matching on B which matches each "1" in $B(R_{VB_w},n)$ to a "0" in $B(R_{VB_w},n)$ (invoking Theorem 5.1) and leaves a "0" in $B(s+L_{VB_w},R_{VB_w}) \subseteq B(1,d)$ unmatched. In this case, "(null,null)" is returned correctly.

Induction: Let VT_x and VB_x agree with the extensions of VT_w' and VB_w' defined when $c_7=1$ upon the return of SCAN-ASSIGN(T(1,m′),B(1,n′),p′,VT_w',VB_w') except at p′(h) and h where $VT_x(p'(h))$ = $VB_x(h)$ = ?. Let VT_{x0} and VB_{x0} be the extensions of VT_x and VB_x making p′(h) and h 0-valued and VT_{x1} and VB_{x1} be the extensions making them 1-valued. The value functions used for the recursive call to BETTER-MATCH are VT_{x1} and VB_{x1}. By Lemma 5.7 and Theorems 5.4-A and 5.4-B, any function pair maximizing $M_{VT}+M_{VB}$ over all functions consistent with p′, VT_x and VB_x maximizes $M_{VT}+M_{VB}$ over all functions consistent with p′, VT_w' and VB_w'. For any function pair consistent with VT_{x0} and VB_{x0}, there are $\lfloor \tfrac{1}{2}(n'-s+1)\rfloor+1$ "0"s in B(s,n′), where s is defined under the standard state after SCAN-ASSIGN returns "FAIL". One such "0" must be unmatched under any matching.

If the recursive call to BETTER-MATCH using T(1,m'), B(1,n'), n', p', VT_{x1} and VB_{x1} returns "(null,null)", then by inductive assumption, there is a pair of valued functions which maximizes the sum of matchings on T(1,m') and B(1,n') over all functions consistent with p', VT_{x1} and VB_{x1} and for which a maximum matching of B(1,n') leaves a "0" in B(1,n') unmatched. Either this pair of functions also maximizes $M_{VT}+M_{VB}$ over all function pairs consistent with p', VT_x and VB_x or a function pair for which p'(h) and h are 0-valued maximizes $M_{VT}+M_{VB}$ over these functions. In either case, there is a value function pair which maximizes $M_{VT}+M_{VB}$ over all functions consistent with p', VT'_w and VB'_w and for which a "0" in B(s,n') is unmatched under some maximum matching for B(1,n'). As for the case when $c_? > 1$ in the proof of the basis above, "(null,null)" is correctly returned by the original call to BETTER-MATCH.

Suppose the recursive call to BETTER-MATCH returns VT_r and VB_r which are not "null". These functions are consistent with VT'_w and VB'_w since VT_{x1} and VB_{x1} are consistent with VT'_w and VB'_w. Given any pair of value functions on T(1,m') and B(1,n'), let the *expansions* of these functions denote those functions on T(1,m) and B(1,n) which agree with the given functions on $T(L_{VT_w}+1,R_{VT_w}-1)$ and $B(L_{VB_w}+1,R_{VB_w}-1)$ and agree with VT_w and VB_w elsewhere. The original call to BETTER-MATCH returns VT_f and VB_f which are the expansions of VT_r and VB_r. Since no terminal is ?-valued under VT_r and VB_r, none is ?-valued under VT_f and VB_f. Suppose there is a pair of functions, VT and VB, consistent with p, VT_0 and VB_0 for which $M_{VT}+M_{VB} > M_{VT_f}+M_{VB_f}$. By Theorem 5.2, there are functions VT_1 and VB_1 consistent with p, VT_w and VB_w for which $M_{VT_1}+M_{VB_1} \geq M_{VT}+M_{VB}$. Since pairs VT_1, VB_1 and VT_f,VB_f are both consistent with VT_w and VB_w, we can use Theorem 5.1 to deduce that the restrictions of these functions to $T(L_{VT_w}+1,R_{VT_w}-1)$ and $B(L_{VB_w}+1,R_{VB_w}-1)$ must be such that

$$M_{VT'_1}+M_{VB'_1} > M_{VT_r}+M_{VB_r},$$

where VT'_1 and VB'_1 are the restrictions of VT_1 and VB_1. By Lemma 5.7 and Theorems 5.4-A and 5.4-B, there are functions, VT'_2 and VB'_2, consistent with p', VT_x and VB_x and assigning no "?"s such that:

$$M_{VT'_2}+M_{VB'_2} \geq M_{VT'_1}+M_{VB'_1} > M_{VT_r}+M_{VB_r}.$$

Case 1: Suppose VT'_2 and VB'_2 are consistent with VT_{x0} and VB_{x0}. For VB'_2, there is an unmatched "0" under any matching function on $B(1,n')$. Let VT_2 and VB_2 be the expansions of VT'_2 and VB'_2. Then $M_{VT_2}+M_{VB_2} \geq M_{VT_1}+M_{VB_1}$, and there is a matching function achieving M_{VB_2} which leaves a "0" in $B(L_{VB_w}+1,R_{VB_w}-1) \subsetneq B(1,d)$ unmatched. Functions VT_2 and VB_2 are the desired VT_g and VB_g.

Case 2: Suppose VT'_2 and VB'_2 are consistent with VB_{x1} and VB_{x1}. By inductive assumption, there are functions VT'_g and VB'_g consistent with p', VB_{x1} and VB_{x1} for which $M_{VT'_g}+M_{VB'_g} \geq M_{VT'_2}+M_{VB'_2}$ and under which some matching function achieving $M_{VB'_g}$ leaves a "0" in $B(1,n')$ unmatched. Since VB_{x1} and VB_{x1} are extensions of VT'_w and VB'_w, VT'_g and VB'_g are consistent with VT'_w and VB'_w. Let VT_g and VB_g be the expansions of VT'_g and VB'_g. They are consistent with VT_0 and VB_0. We have:

$$M_{VT_g}+M_{VB_g} \geq M_{VT_2}+M_{VB_2} \geq M_{VT_1}+M_{VB_1}.$$

Under VB_g, there is a matching function achieving M_{VB_g} which leaves a "0" in $B(L_{VB_w}+1,R_{VB_w}-1)$, a subset of $B(1,d)$, unmatched. Thus, VT_g and VB_g are the desire functions. □

Lemma 5.10 completes the technical development needed to verify the correctness of the algorithm. Theorem 5.5 states the correctness of the main procedure, MAX-MATCH.

Theorem 5.5: Let VT_0 and VB_0 be value functions consistent with pairing function p and such that the only ?-valued terminals are members of top-bottom pairs. Then MAX-MATCH(T(1,m),B(1,n),p, VT_0,VB_0) returns functions VT_f and VB_f consistent with p, VT_0 and VB_0 which maximize $M_{VT}+M_{VB}$ over all such consistent functions.

Proof: The correctness follows from the development presented in this chapter. The formal argument is by induction on the number of recursive calls; it is similar to that used in proving Lemma 5.10 except that verifying the correct return of "(null,null)" is not necessary. We only present the argument used when $c_? = 1$ after a call to SCAN-ASSIGN. It is under this condition that BETTER-MATCH is called by MAX-MATCH and Lemma 5.10 is needed in the proof.

Assume the standard state after the call to SCAN-ASSIGN in MAX-MATCH returns "FAIL". Let VT_x and VB_x be the extensions of the functions returned by SCAN-ASSIGN under which all terminals in T(1,q) and B(s,n') except h and p'(h) are 0-valued or 1-valued. Let VT_{x0} and VB_{x0} extend VT_x and VB_x so that p'(h) and h are 0-valued, and VT_{x1} and VB_{x1} extend them so that p'(h) and h are 1-valued. By inductive assumption, the functions VT_{r1} and VB_{r1} returned by MAX-MATCH(T(1,m'), B(1,n'),p',VT_{x1},VB_{x1}) maximize $M_{VT}+M_{VB}$ over all functions consistent with p', VT_{x1} and VB_{x1}.

Suppose BETTER-MATCH(T(1,m'),B(1,n'),s-1,p',VT_{x0},VB_{x0}) returns "(null,null)". Then there is a pair of functions VT_{ex0} and VB_{ex0} consistent with VT_{x0} and VB_{x0} maximizing $M_{VT}+M_{VB}$ over all functions consistent with p', VT_{x0} and VB_{X0} for which there is a matching function achieving $M_{VB_{ex0}}$ which does not match all "0"s in B(1,s-1). By Lemma 5.9, there are value functions VT_{ex1} and VB_{ex1} consistent with p', VT_{x1} and VB_{x1} such that $M_{VT_{ex1}}+M_{VB_{ex1}} \geq M_{VT_{ex0}}+M_{VB_{ex0}}$. Then:

$$M_{VT_{r1}}+M_{VB_{r1}} \geq M_{VT_{ex1}}+M_{VB_{ex1}} \geq M_{VT_{ex0}}+M_{VB_{ex0}},$$

and VT_{r1} and VB_{r1} maximize $M_{VT}+M_{VB}$ over all functions consistent with p', VT_x and VB_x. The expansions of VT_{r1} and VB_{r1} maximize $M_{VT}+M_{VB}$ over all functions consistent with p, VT_0 and VB_0.

Suppose BETTER-MATCH(T(1,m'),B(1,n'),s-1,p',VT_{x0},VB_{x0}) returns VT_{r0} and VB_{r0} which are not "null". Suppose VT_{r0} and VB_{r0} do not maximize $M_{VT}+M_{VB}$ over all functions consistent with p', VT_{x0} and VB_{x0}. Let VT_{ex0} and VB_{ex0} be value functions consistent with p', VT_{x0} and VB_{x0} which do maximize $M_{VT}+M_{VB}$ over all such functions. Then, $M_{VT_{ex0}}+M_{VB_{ex0}} > M_{VT_{r0}}+M_{VB_{r0}}$. By Lemma 5.10, there are functions VT_g and VB_g consistent with p', VT_{x0} and VB_{x0} for which $M_{VT_g}+M_{VB_g}$ $\geq M_{VT_{ex0}}+M_{VB_{ex0}}$ and for which a matching function achieving M_{VB_g} leaves a "0" in B(1,s-1) unmatched. By Lemma 5.9, there are value functions VT_{g1} and VB_{g1} consistent with p', VT_{x1} and VB_{x1} such that:

$$M_{VT_{g1}} + M_{VB_{g1}} \geq M_{VT_{g}} + M_{VB_{g}} \geq M_{VT_{ex0}} + M_{VB_{ex0}} \text{ Therefore,}$$

$$M_{VT_{r1}} + M_{VB_{r1}} \geq M_{VT_{g1}} + M_{VB_{g1}} \geq M_{VT_{ex0}} + M_{VB_{ex0}} > M_{VT_{r0}} + M_{VB_{r0}}.$$

Functions VT_{r1} and VB_{r1} maximize $M_{VT}+M_{VB}$ over all functions consistent with p', VT_x and VB_x and are correctly expanded and returned by MAX-MATCH.

If VT_{r0} and VB_{r0} do maximize $M_{VT}+M_{VB}$ over all functions consistent with p', VT_{x1} and VB_{x1}, then the maximum of $M_{VT_{r0}}+M_{VB_{r0}}$ and $M_{VT_{r1}}+M_{VB_{r1}}$ maximizes $M_{VT}+M_{VB}$ over all functions consistent with p', VT_x and VB_x. The pair of functions achieving the maximum (with ties broken in favor of VT_{r1} and VB_{r1}) are correctly expanded and returned by MAX-MATCH. □

5.6 Running Time of the Algorithm

We will find an upper bound on the running time of the algorithm without making assumptions about the details of the data structures used to implement the algorithm. We will assume that each of the following takes one step: addition to a counter, testing the value of a scalar variable, and assignment to a scalar variable.[1] Assignment to a function variable is assumed to be assignment to one scalar variable for each domain value of the function; this allows a function to be represented as an array. The number of steps taken by the algorithm will be measured as a function of the number of terminals $(m+n)$ and the number of ?-valued terminal pairs under the initial value functions. Recall that the algorithm assumes that all ?-valued terminals are members of top-bottom pairs. Let u denote the number of ?-valued pairs.

Procedure SCAN-ASSIGN does not use any other procedures, including recursive calls to itself.

[1] This follows the unit cost model for random access machines of Aho, Hopcroft, and Ullman [Ah74]. The logarithmic cost model, which takes into account the number of bits required to represent a number, would multiply any running time by $\log_2(m+n)$.

The procedure considers each terminal in T at most once. For each ?-valued terminal it considers, it must test the sizes of sets of 0-valued terminals and 1-valued terminals and make an assignment when possible. If the sets are recomputed for each ?-valued terminal considered, the number of steps used to process one ?-valued terminal is at most $k(m+n)$ for some constant k. The stopping test of the first FOR loop must also be computed, but only when an assignment has been made. This test also requires computing the size of a set of terminals. Let $SC(m,n,u,a_s)$ denote the maximum number of steps taken by SCAN-ASSIGN on any m top terminals and n bottom terminals with u ?-valued terminal pairs when a_s ?-valued terminals pairs are assigned by SCAN-ASSIGN. Then

$$SC(m,n,u,a_s) \leq k_1(a_s+1)(m+n), \quad \text{for some constant } k_1.$$

If $a_s < u$, SCAN-ASSIGN returns "FAIL" and (a_s+1) is the number of ?-valued terminal pairs considered. Let $SC(m,n,u)$ denote the maximum number of steps taken by SCAN-ASSIGN on any m top terminals and n bottom terminals with u ?-valued terminal pairs.

$SC(m,n,u) \leq$ maximum over all $a_s \leq u$ of $SC(m,n,u,a_s) \leq k_1(u+1)(m+n) = \mathcal{O}(u(m+n))$.

Procedure BETTER-MATCH may use procedure SCAN-ASSIGN and a recursive call to itself. First BETTER-MATCH computes the full left-regions and right-regions and assigns to ?-valued terminals within them. This is done in the WHILE loop. After assigning within any of $(1,L_{VT})$, (R_{VT},m), $(1,L_{VB})$, and (R_{VB},n), some regions must be recomputed. This takes $\mathcal{O}(m+n)$ steps. Once regions are computed, testing for the conditions under which "(null,null)" is returned takes only a constant number of steps. Suppose a_w initially ?-valued terminals are assigned 0/1 values in the WHILE loop. Then its execution, including the initial computation before entering it, takes at most $k_2(a_w+1)(m+n)$ steps, for some constant k_2.

After the WHILE loop is completed, if any ?-valued pairs remain, SCAN-ASSIGN is called. Preparation for calling SCAN-ASSIGN takes $\mathcal{O}(m'+n')$ steps to assign to VT', VB', and p'. If SCAN-ASSIGN returns "SUCCESS", then merging the functions that SCAN-ASSIGN returns with the functions on the full regions to create the functions returned by BETTER-MATCH takes $O(m'+n')$ steps. If SCAN-ASSIGN returns "FAIL", it takes $O(m'+n')$ steps to assign to VT' and VB' and calculate q, s, h, and c_7. If $c_7=1$, BETTER-MATCH is called recursively, returning functions or "null"s. If the recursive call returns functions, then the processing required to return their expansions takes $O(m'+n')$ steps.

Let $B(m,n,u)$ be the maximum number of steps taken by any call to BETTER-MATCH with m top terminals, n bottom terminals, and u initially ?-valued pairs. Such a call to BETTER-MATCH under which a_w initially ?-valued pairs are assigned in the WHILE loop and a_s pairs are assigned by SCAN-ASSIGN takes at most

$$k_2(a_w+1)(m+n) + SC(m',n',u\text{-}a_w,a_s) + B(m',n',u\text{-}a_w\text{-}a_s\text{-}1) + k_3(m'+n') \text{ steps}$$

for some constant k_3, where $B(m,n,-1)=0$ by definition for any m and n. Using the bound for SC gives:

$$\leq k_2(a_w+1)(m+n) + k_1(a_s+1)(m'+n') + k_3(m'+n') + B(m',n',u\text{-}a_w\text{-}a_s\text{-}1) \text{ steps.}$$

We know $1\leq m'\leq m$ and $1\leq n'\leq n$. For $k_4=k_1+k_2+k_3$, the number of steps is bounded above by

$$k_4(a_w+a_s+1)(m+n) + B(m',n',u\text{-}a_w\text{-}a_s\text{-}1). \qquad \text{expression 1}$$

Then $B(m,n,u) \leq$ (maximum over all $m'\leq m$, $n'\leq n$, and $0\leq a_w+a_s\leq u$ of expression 1)

Order all triples (m,n,u), for $m\geq 1$, $n\geq 1$, and $u\geq 0$, lexicographically. We prove by induction on this ordering that for all such triples, $B(m,n,u) \leq k_5(u+1)(m+n)$, for some constant k_5. The basis, (1,1,0), is trivial. For any triple, $(m,n,0)$, $B(m,n,0) \leq k_4(m+n)$, so $k_5\geq k_4$ suffices. Let the proposition be true for any triple lexicographically smaller that (m,n,u) where $u>0$. We know that $1\leq m'\leq m$ and $1\leq n'\leq n$ whenever BETTER-MATCH is recursively called. When $a_w+a_s<u$, $0\leq u\text{-}a_w\text{-}a_s\text{-}1\leq u\text{-}1$, and we can use the inductive assumption. When $a_w+a_s=u$, $u\text{-}a_w\text{-}a_s\text{-}1 = -1$ but $k_5(u\text{-}a_w\text{-}a_s)(m'+n') = 0$ is still an upper bound on the contribution of the recursive call to BETTER-MATCH, since no recursive call is made. Therefore:

$$\text{expression 1} \leq k_4(a_w+a_s+1)(m+n) + k_5(u\text{-}a_w\text{-}a_s)(m'+n')$$

$$\text{expression } 1 \leq k_5(a_w + a_s + 1)(m+n) + k_5(u - a_w - a_s)(m+n)$$

$$\text{using } k_5 \geq k_4 \text{ and } m' + n' \geq m+n$$

$$\leq k_5(u+1)(m+n)$$

and $B(m,n,u) \leq k_5(u+1)(m+n) = O(u(m+n))$.

We now turn to MAX-MATCH. The bound on the number of steps used in the WHILE loop is of the same order as that for BETTER-MATCH, by the same argument. Procedure MAX-MATCH does not do the testing to return "(null,null)" which BETTER-MATCH does, but this only adds a constant per assignment to the number of steps used by BETTER-MATCH, and only affects the constant for MAX-MATCH's bound. The pre-processing for SCAN-ASSIGN and the post-processing when SCAN-ASSIGN returns "SUCCESS" have the same bound on the number of steps used as that for BETTER-MATCH. However, if SCAN-ASSIGN returns "FAIL", a recursive call to MAX-MATCH and possibly a call to BETTER-MATCH are used. The pre-processing and post-processing for these calls take $O(m'+n')$ steps. Let $M(m,n,u)$ be the maximum number of steps used by any call to MAX-MATCH with m top terminals, n bottom terminals, and u initially ?-valued pairs. Then for constants k_6 and k_7

$$M(m,n,u) \leq \text{maximum over all } m' \leq m,\ n' \leq n, \text{ and } 0 \leq a_w + a_s \leq u \text{ of}$$

$$k_6(a_w+1)(m+n) + S(m',n',u-a_w,a_s) + B(m',n',u-a_w-a_s-1)$$

$$+ M(m',n',u-a_w-a_s-1) + k_7(m'+n') \qquad \text{expression 2}$$

We now prove by induction on the ordering of triples that

$$M(m,n,u) \leq k_8(u+1)^2(m+n) \text{ for some constant } k_8.$$

The basis and cases where u = 0 are straightforward if $k_8 \geq k_6 + k_7$. Suppose the proposition is true for all triples lexicographically less than (m,n,u) where u > 0. Then

$$\text{expression } 2 \leq k_6(a_w+1)(m+n) + k_1(a_s+1)(m'+n') + k_5(u-a_w-a_s)(m'+n')$$

$$+ k_8(u-a_w-a_s)^2(m'+n') + k_7(m'+n')$$

$$\leq k_9(u+1)(m+n) + k_8(u-a_w-a_s)^2(m+n) \qquad \text{for } k_9 = k_6 + k_1 + k_5 + k_7$$

Letting $k_8 \geq k_9$, we have

$$\text{expression } 2 \leq k_8(u^2+u+1)(m+n) \leq k_8(u+1)^2(m+n)$$

and $\quad M(m,n,u) \leq k_8(u+1)^2(m+n) = \mathcal{O}(u^2(m+n)) = \mathcal{O}((m+n)^3)$

We have shown that our algorithm runs in a number of steps at worst proportional to the cube of the number of terminals.

5.7 Summary

In this chapter, we have presented an algorithm which routes the connections between pairs of terminals located on the outside of a rectangular component. The routing assumes horizontal and vertical wires are on separate layers and uses minimum area. The problem is reduces to an assignment problem on vectors. Procedure MAX-MATCH solves the problem on vectors using procedures BETTER-MATCH and SCAN-ASSIGN. Procedures SCAN-ASSIGN and BETTER-MATCH have $\mathcal{O}((m+n)^2)$ running time and MAX-MATCH has $\mathcal{O}((m+n)^3)$ running time. Procedure MAX-MATCH is actually used twice in routing terminals on a rectangle--once for top-bottom connections and once for left-right connections. Before using MAX-MATCH, local connections are assigned directions in a predetermined manner. Assigning the local connections uses $\mathcal{O}(t)$ computation steps, where t is the number of terminals on the rectangle. Each call to MAX-MATCH uses $\mathcal{O}(t^3)$ steps. Therefore, given a rectangular component with t terminals around its boundary, the problem presented in Section 5.1 can be solved in polynomial time $\mathcal{O}(t^3)$.

Appendix to Chapter 5: Notation

The notation used in Chapter 5 is listed here, with definitions, in order of appearance.

T(1,m) denotes top terminals 1 through m.

B(1,n) denotes bottom terminals 1 through n.

VT: $T(1,m) \rightarrow \{0,1,?\}$ and VB: $B(1,n) \rightarrow \{0,1,?\}$ indicate the directions of connections from top and bottom terminals, respectively, as follows:

VT / VB (i) = 0 if the direction of the path from terminal i to its pair is to the left
1 if it is to the right
? if it is undetermined

$p: T(1,m) \cup B(1,n) \rightarrow T(1,m) \cup B(1,n) \cup \{*\}$ is the pairing function indicating what terminals should be connected.

For the following definitions, there are analogous definitions for B and VB:

m_{VT}: $T \rightarrow T$ and is the function matching terminals in T(1,m) of value "0" to terminals with higher index in T(1,m) and of value "1", given value function VT. If $m_{VT}(i) = j$, then $VT(i) = 0$, $VT(j) = 1$, and $i < j$.

M_{VT} is the maximum over all matching functions, m_{VT}, for a value function VT, of $|range(m_{VT})|$.

$ZEROS_{VT}(S) = \{i \in S | VT(i)=0\}$ for a value function VT and $S \subseteq T$.

$ONES_{VT}(S) = \{i \in S | VT(i)=1\}$ for a value function VT and $S \subseteq T$.

$UNDET_{VT}(S) = \{i \in S | VT(i)=?\}$ for a value function VT and $S \subseteq T$.

$ONES_{VT}(x,y)$ is simplified notation for $ONES_{VT}(T(x,y))$, etc.

OK-1(VT,x,y) (similarly OK-0) is true if and only if $|ONES_{VT}(x,y)| \leq \lfloor ½(y-x+1) \rfloor$.

Full-1(VT,x,y) (similarly Full-0) is true if and only if $|ONES_{VT}(x,y)| \geq \lceil ½(y-x+1) \rceil$.

l_{VT}:T → T and r_{VT}:T → T are used to define left-regions and right-regions on T under a given value function VT:

$l_{VT}(1) = 1$
$l_{VT}(i) = i$ if Full-1(VT,l_{VT}(i-1),i-1)
l_{VT}(i-1) otherwise, for $i > 1$

$r_{VT}(m) = m$
$r_{VT}(j) = j$ if Full-0(VT,i+1,r_{VT}(i+1))
r_{VT}(i+1) otherwise, for $j < m$

Left-regions are defined as the equivalence classes induced by the equivalence relation on T(1,m) under which two terminals i and j are equivalent if and only if $l_{VT}(i) = l_{VT}(j)$.

Right-regions are the equivalence classes induced by the equivalence relation using r_{VT}.

L_{VT} is defined to equal m if Full-1(VT,l_{VT}(m),m) and to equal l_{VT}(m)-1 otherwise. (1,L_{VT}) are the full left-regions of T under VT.

R_{VT} is defined to equal 1 if Full-0(VT,1,r_{VT}(1)) and to equal $r_{VT}(1)+1$ otherwise. (R_{VT},m) are the full right-regions of T under VT.

The following definitions are made after SCAN-ASSIGN returns "(FAIL k, VT_{k-1}, VB_{k-1})":

q is the smallest $i \geq k$ such that $|ONES_{VT_{k-1}}(1,i)| = \lfloor \frac{1}{2} i \rfloor$.

s is the smallest j satisfying the following three properties:

(i) $p(k) \geq j$;
(ii) $|ZEROS_{VB_{k-1}}(j,n)| = \lfloor \frac{1}{2}(n-j+1) \rfloor$;
(iii) each terminal in T(1,k-1) which has been assigned the value "1" by SCAN-ASSIGN is paired with a terminal in B(j,n).

C is the set of initially ?-valued terminals in T(1,q) whose pairs are in B(s,n).

$c_1 = |ONES_{VT_{k-1}}(C)|$.

$c_0 = |ZEROS_{VT_{k-1}}(C)|$.

$c_? = |\mathrm{UNDET}_{VT_{k-1}}(C)|$.

h is defined as follows: Let i be the terminal of smallest index in B(s,n) which is ?-valued under VB_0 and 1-valued under VB_{k-1}. Certainly, $p(i)<k$. If $i<p(k)$, then $h = i$; otherwise, $h = p(k)$.

VT_{fx} and VB_{fx} are defined as follows: If $h = p(k)$, then $VT_{fx} = VT_{k-1}$ and $VB_{fx} = VB_{k-1}$. If $h \neq p(k)$, then VT_{fx} and VB_{fx} agree with VT_{k-1} and VB_{k-1} except at h, k, p(h), and p(k), where:

$$VT_{fx}(k) = 1 \qquad VT_{fx}(p(h)) = ?$$

$$VB_{fx}(p(k)) = 1 \qquad VB_{fx}(h) = ?.$$

Chapter 6 Discussion of the Algorithm

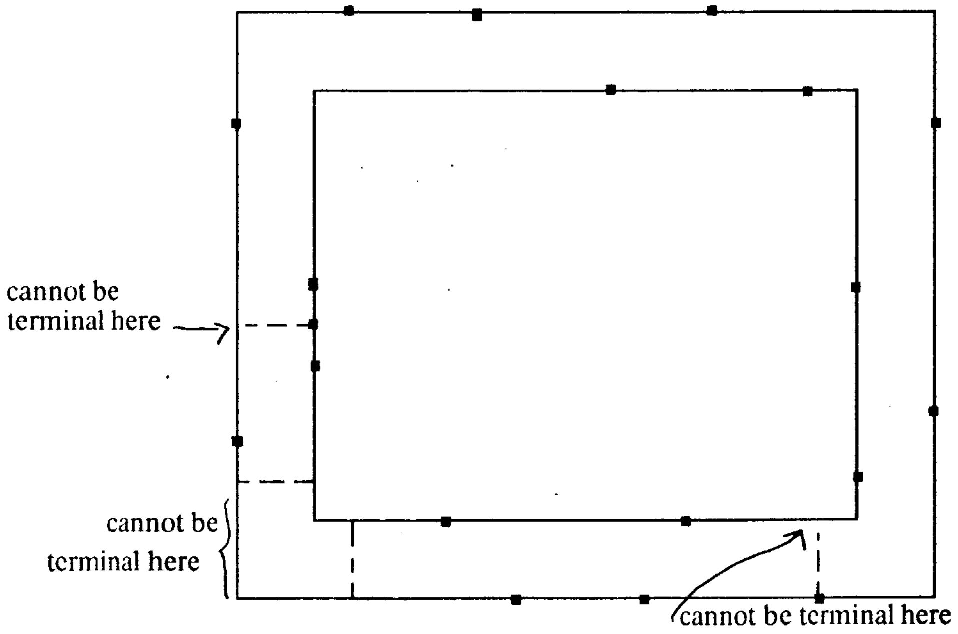
6.1 Removing Assumptions

In this chapter we make some observations about the algorithm we have just presented. The algorithm is of the channel routing variety. The streets to be used by each connection path are chosen by the algorithm. Only one choice is considered for local connections. For top-bottom and left-right connections, the procedure MAX-MATCH makes the decision. Three assumptions are crucial to the working of the algorithm: (1) only pairs of terminals need to be connected; (2) there are only four routing areas -- one parallel to each side of the rectangle; (3) the segments from each terminal to the routing area (perpendicular to the routing direction in this area) cannot conflict regardless of their lengths.

We do not necessarily need to have terminals around the outside of one rectangle as long as the above assumptions are satisfied. In fact, terminals may also lie along a rectangular boundary circumscribing the rectangular component, i.e. on the opposite side of the routing area, as long as these terminals can be projected on the rectangle to obtain an instance of the original problem (see Figure 6.1.). This configuration might be found when connecting terminals of a functional component or set of functional components to bonding pads. The bonding pads lie along the outside edges of the chip, and the routing regions lie between these pads and the rest of the integrated circuit. Minimizing the area used for the interconnections minimizes the size of the chip.

When the third assumption above is removed, we have already seen in Chapter 4 that the resulting routing problem is NP-complete even for one routing channel with terminals along its side. When either of assumptions (1) or (2) is removed, our division into local connections and opposite side connections no longer limits the choice of directions for each connection. Consider the problem for one rectangular component when it may be necessary to interconnect three or more terminals. When three terminals are involved, there are three choices instead of two for the type of path used to interconnect the

Figure 6.1: Alternate configuration of terminals for our algorithm.



■ indicates a terminal

terminals. This can be seen by dividing the boundary of the rectangle into three pieces -- each piece goes between adjacent terminals as the boundary is traversed. Any path which follows two adjacent pieces of the boundary can be used to connect the terminals. When the terminals are on three different sides, two of the possible path types use more sides that the third. However, we cannot eliminate these possibilities. It is not true that the third choice is always as good as the other two. Figure 6.2-A gives a counterexample. Connecting these terminals involves three sides of the rectangle. Therefore, we have lost the independence of the two dimensions which we had when routing top-bottom and left-right connections. If three terminals are only on two sides, even the path type using all four sides cannot be eliminated. This is shown in Figure 6.2-B. It is an open problem whether the one component routing problem can be solved in polynomial time when sets of three or more terminals need to be interconnected.

Consider retaining the restriction that only pairs of terminals are interconnected but allowing terminals to lie on one of two rectangular components which are placed side by side. Ignore for the moment any connection which involves terminals which lie on the boundaries of the street between the two rectangles. The resulting routing problem is similar to the one component routing problem except that paths can take a "short cut" through the street between the two rectangles. However, this option adds many choices of paths to be used to interconnect terminals. The top and bottom sides are no longer independent of the left and right sides. Figure 6.3 illustrates the difficulties presented by the short cut. The complexity of this routing problem is also open.

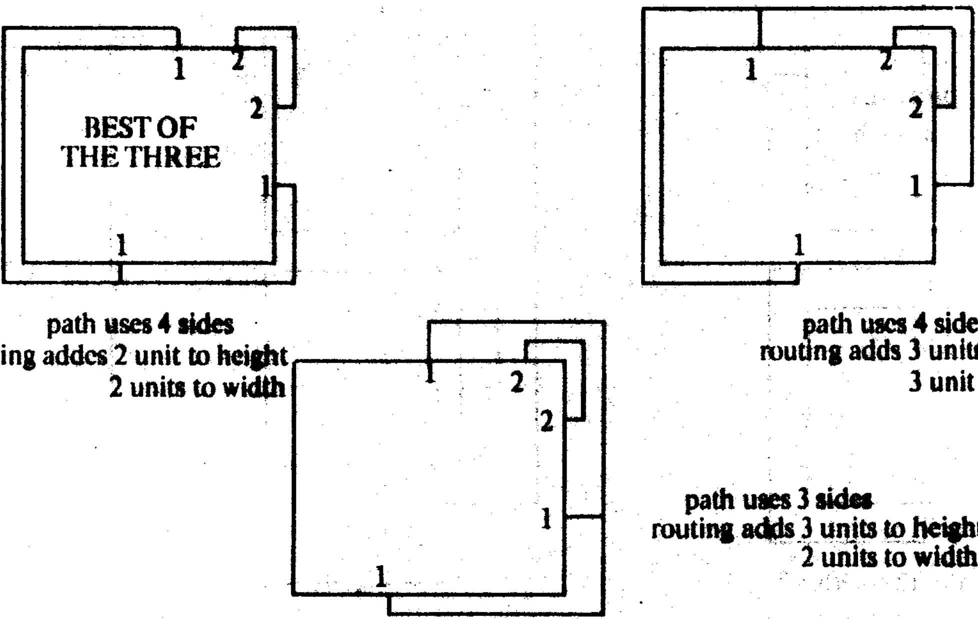
6.2 A Special Case for Procedure MAX-MATCH

The procedure MAX-MATCH, with its sub-procedures, solves the optimization problem defined in Section 5.2. We now prove that when all terminals are ?-valued under the initial value functions, there is always an assignment of "0"s and "1"s such that the sum of maximum matchings is

Figure 6.2: The One Component Routing Problem with more than two terminals in nets.

Terminals with the same number should be connected.

A: Three terminals on three sides -- three choices.

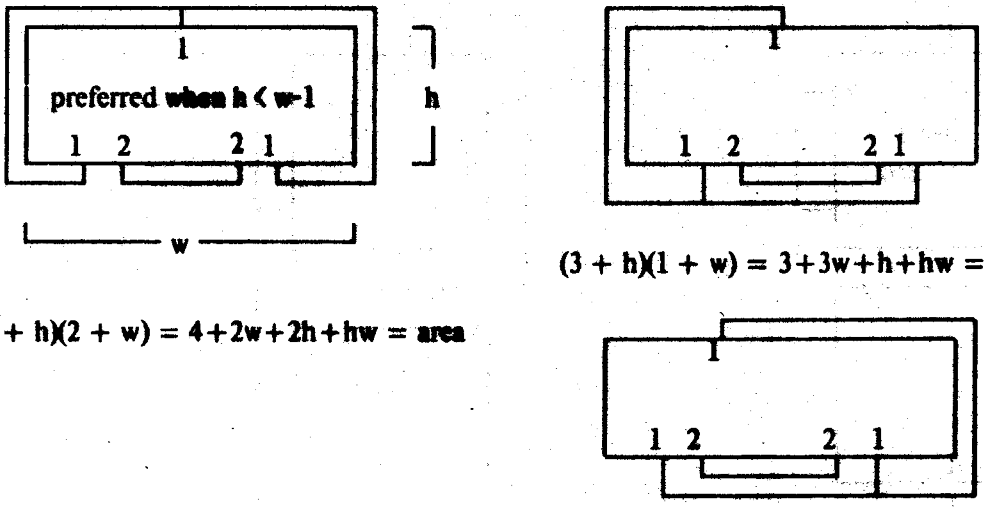


path uses 4 sides
routing addes 2 unit to height
2 units to width

path uses 4 sides
routing adds 3 units to height
3 unit to width

path uses 3 sides
routing adds 3 units to height
2 units to width

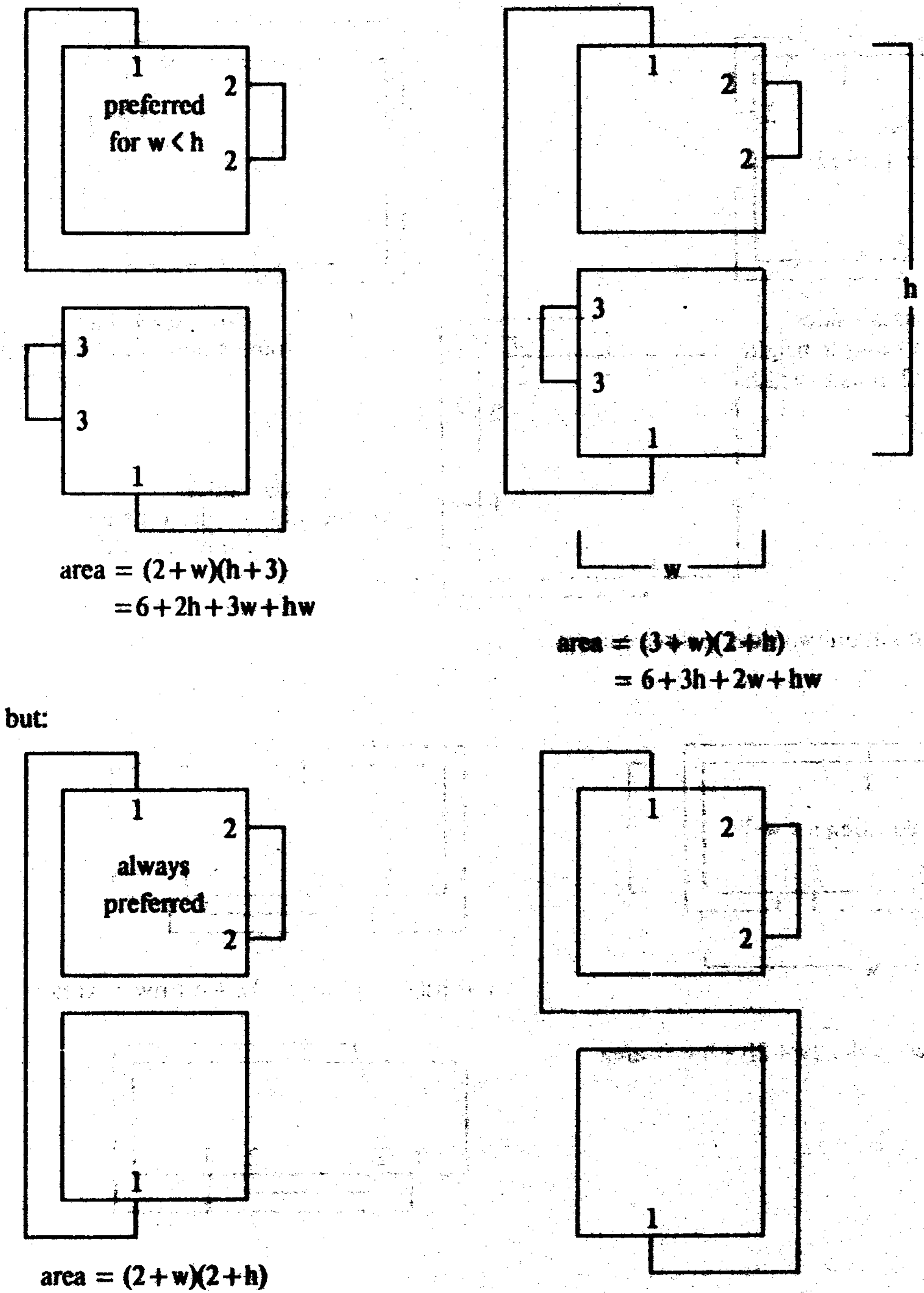
B: Three terminals on two sides -- three choices.



(2 + h)(2 + w) = 4+2w+2h+hw = area

(3 + h)(1 + w) = 3+3w+h+hw = area

Figure 6.3: Extending the routing problem of Chapter 5 to two components.



area = $(2+w)(h+3)$
$=6+2h+3w+hw$

area = $(3+w)(2+h)$
$= 6+3h+2w+hw$

but:

area = $(2+w)(2+h)$

area = $(2+w)(3+h)$

within one of the upper bound obtained when the specific pairing of top terminals to bottom terminals is disregarded. We always assume that all ?-valued terminals are members of top-bottom pairs under p. When all terminals are ?-valued under the initial value functions, the number of top terminals, m, must equal the number of bottom terminals, n.

Lemma 6.1 Let VT_0 and VB_0 assign "?"s to all terminals in T and B. Then MAX-MATCH($T(1,m),B(1,m),p,VT_0,VB_0$) returns VT_f and VB_f such that $M_{VT_f} + M_{VB_f} \geq m-1$.

Proof: Initially, there are no full regions and SCAN-ASSIGN is called. If SCAN-ASSIGN returns (SUCCESS,VT_r,VB_r), then $\lceil ½m \rceil$ terminals are 0-valued in each of T and B. We know OK-0(VB_r,1,m) holds. Therefore, m is even. The matchings give $M_{VT_r} + M_{VB_r} = ½m + ½m = m$.

Suppose SCAN-ASSIGN returns (FAIL,k,VT_r,VB_r). Let q, s, h, and $c_?$ be as defined in Chapter 5. All terminals in T(k,m) are ?-valued under VT_r. Therefore, $q = k$, $c_? = 1$, and k is odd. We know:

$$|ONES_{VT_r}(1,k-1)| = |ONES_{VB_r}(s,m)| = \lfloor ½k \rfloor = ½(k-1)$$

and $\lfloor ½(m-s+1) \rfloor = |ZEROS_{VB_r}(s,m)| \leq |ZEROS_{VT_r}(1,k-1)| = ½(k-1)$ (a)

Also, $\lfloor ½(m-s+1) \rfloor + ½(k-1) = |ZEROS_{VB_r}(s,m)| + |ONES_{VB_r}(s,m)| \leq (m-s+1)-1$

implying $½(k-1) \leq \lceil ½(m-s+1) \rceil -1$ (b).

Combining (a) and (b) gives:

$$\lfloor ½(m-s+1) \rfloor = ½(m-s) = ½(k-1) \text{ and } |ZEROS_{VB_r}(s,m)| = |ZEROS_{VT_r}(1,k-1)|.$$

Either $VB_r(s) = 1$, or $s = p(k)$. Otherwise, $|ONES_{VB_r}(s+1,m)| = |ONES_{VB_r}(s,m)| = ½(m-s)$ and B(s+1,m) contains p(k) and the pairs of all "1"s in T(1,q). Therefore, we would have chosen s+1 rather than s. In either of the two possible cases, s = h.

Let VT_{x1} and VB_{x1} be the value functions defined when the "1" value is tried for h. We know $|ZEROS_{VB_{x1}}(s+1,m)| = ½(m-s)$ and OK-0(VB_{x1},x,m) holds for all x in B(s,m). Therefore, B(s+1,m) is a set of full right-regions with no ?-valued terminals. All ½(m-s) "0"s in B(s+1,m) can be matched. Unless s = 1 or s = 2, interval B(1,s) contains one left-region and one right-region, neither of which are full. Since $|ONES_{VT_{x1}}(1,k)| = \lceil ½k \rceil$ and OK-1(VT_{x1},1,x) holds for all x in T(1,k-1), T(1,k) is a set of full right-regions in which ½(k-1) "0"s can be matched to "1"s in T(1,k). When s = 1, in which case k = m, we have

$$M_{VT_{x1}} + M_{VB_{x1}} = ½(m-1) + ½(m-1) = m-1$$

and the desired result follows. When s > 1, T(k+1,m) contains one left-region and one right-region, neither of which are full. If s = 2, B(1,2) is a full left-region and B(1) is ?-valued under VB_{x1}. Terminals B(1) and p(B(1)) = T(m) are assigned the value "0" by the recursive call to MAX-MATCH. Terminal B(1) can match B(2), but T(m) is unmatched. The resulting maximum matchings sum to ½(m-2)+1+½(m-2) = m-1 as desired. If s > 2, let T(1,m′) be the renumbering of T(k+1,m). Procedure SCAN-ASSIGN is called for T(1,m′) and B(1,s) using the appropriate restrictions of VT_{x1} and VB_{x1}.

If the second call to SCAN-ASSIGN returns (SUCCESS,VT_{r1},VB_{r1}), then $|ZEROS_{VB_{r1}}(1,s)| = |ZEROS_{VT_{r1}}(1,m')| = \lceil ½m' \rceil$ and $M_{VT_{r1}} + M_{VB_{r1}} = \lfloor ½m' \rfloor + \lceil ½m' \rceil = m'$. Combining this with the matchings on the full regions gives

$$M_{VT_{f1}} + M_{VB_{f1}} = m' + ½(m-s) + ½(k-1) = m-k + k-1 = m-1$$

where VT_{f1} and VB_{f1} are the expansions of VT_{r1} and VB_{r1} which agree with VT_{x1} and VB_{x1} on T(1,k) and B(s+1,m).

If the second call to SCAN-ASSIGN returns (FAIL,k′,VT′,VB′), let q′, s′, and h′ be the special terminals for this call. Under the initial value functions for this call to SCAN-ASSIGN, all terminals in T(1,m′) and all terminals in B(1,s-1) are ?-valued and s is 1-valued. Therefore, q′ = k′, k′ is odd, and

$$|ONES_{VB'}(s',s-1)| = |ONES_{VT'}(1,k'-1)| = ½(k'-1)$$

$$|ZEROS_{VB'}(s',s)| = \lfloor ½(s-s'+1) \rfloor \le |ZEROS_{VT'}(1,k'-1)| = ½(k'-1).$$

Also, $|ONES_{VB'}(s',s)| + |ZEROS_{VB'}(s',s)| = ½(k'-1) + 1 + \lfloor ½(s-s'+1) \rfloor \le (s-s'+1)-1.$

Therefore, $\lfloor M(m'+1) \rfloor \leq M(l'-1)$ and $\lfloor M(m'+1) \rfloor \leq m' - M(l'-1) - 1$,

implying $2\lfloor M(m'+1) \rfloor \leq m'-1$, [illegible] SCAN-ASSIGN must return "SUCCESS".

We have shown that the maximum of $M_{VT}+M_{VB}$ over all functions consistent with VT_{s1} and VB_{s1} is m-1. Since this is a lower bound for the maximum of $M_{VT_f}+M_{VB_f}$ over all functions consistent with VT_f and VB_f, we have $M_{VT_f}+M_{VB_f} \geq m-1$ [illegible]. Note that if the first call to SCAN-ASSIGN returns "FAIL", it is [illegible]. In this case, m-1 is also an upper bound and $M_{VT_f}+M_{VB_f} = m-1$. □

The problem solved by SCAN-ASSIGN is a combinatorial optimization problem which need not be considered a routing problem. As such, Lemma [illegible] shows that under the given conditions, the solution is within one of the minimum possible for any instance of the problem with vectors of size m. Interpreting the problem as a routing problem, [illegible] that when the only type of connections are top-bottom connections, that the amount of [illegible] only varies by one unit over all pairings of [illegible] when terminal pairs are diagonal to each other. This result is somewhat surprising when applied to routing since it implies that as long as all connections are top-bottom pairs or left-right pairs, no rearrangement of terminals within each side will improve by more than one unit the amount added to each dimension of the rectangle.

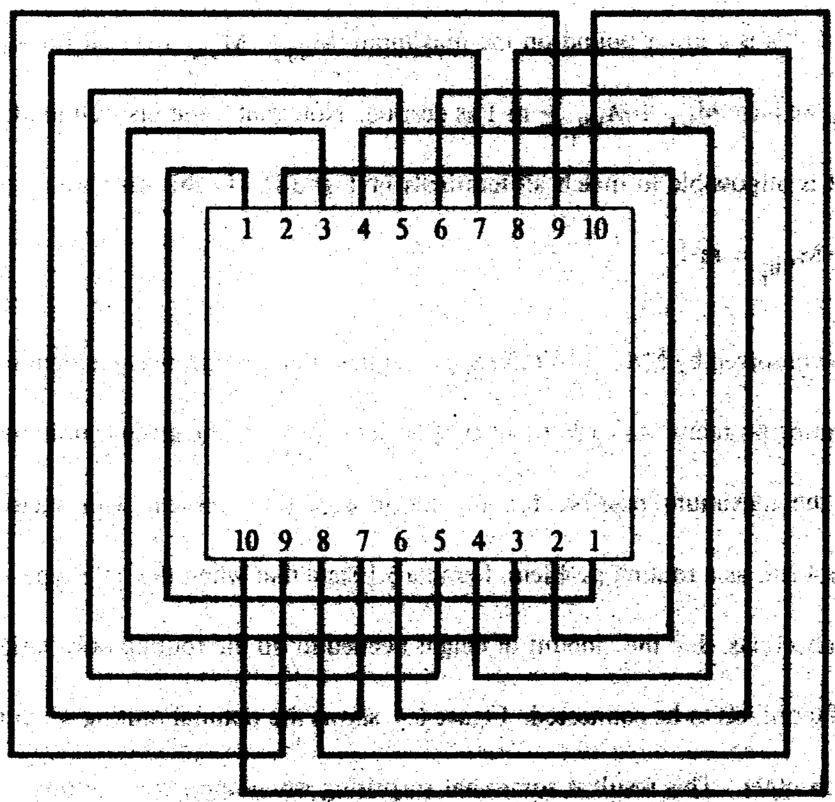
6.3 The Use of Layers

6.3.1 Removing the separate layer - separate direction restriction.

The algorithm presented in Chapter 5 assumes a routing scheme in which only horizontal and vertical wire segments are used and for which horizontal and vertical segments are on different layers. The use of only vertical and horizontal wire segments for routing around a rectangle is reasonable, since the paths will follow the boundary of the rectangle. The assumption that vertical and horizontal wires are

Figure 6.4: The One Component Routing Problem with no local connections.

Terminals labeled with the same number should be connected.



on different layers simplifies the [illegible] may be [illegible]. However, it can lead to larger area. We will give an upper bound on the [illegible] when the restriction on segments in the horizontal and vertical directions [illegible] layers for horizontal and vertical segments is removed. In this case, the layer of each wire segment must be specified. Two unconnected wires may overlap in the planar representation if they are on different layers. We will consider the routing problem [illegible] with the freedom to assign any wire segment to either of two layers. Segments on different layers may be connected at any points where they intersect when layers are superimposed. Segments on the same layer which belong to distinct connections cannot intersect.

Lemma 6.2: The height added along a side of the rectangle for routing when horizontal and vertical wires are on different layers is at most twice as much as that added when both horizontal and vertical wires can be on either of two layers.

Proof: Given a routing which does not use separate layers for horizontal and vertical directions, we will construct a routing which does use separate layers and uses only twice the height above each side. Consider the segments adjacent to the top side and parallel to it. The modifications for the other sides are analogous. For each channel containing segments, there will be two channels containing segments in the new routing: one for segments originally on the first layer and one for segments originally on the second layer. Consider any two horizontal segments which are connected to each other by a connection between layers, a vertical segment, or a combination of the two. The projections of these segments on the horizontal axis must overlap. If the projection of one segment contains the projection of the other, the shorter one be eliminated. Otherwise, the segment which ends later (to the right) and everything to its right exchanges channels with everything to the right of the segment which ends earlier. Since the projection of the segment which ends later does not contain the projection of the other segment, it must start later. The exchange causes the two originally connected segments to overlap but no other segments to overlap. (See Figure 6.5) In this way, we can modify the routing so that there is only one horizontal segment for each connection. This segment must lie above the terminal or terminals to which it needs to be connected since some segment originally did, and segments have not been shortened, just merged. The vertical segments which connect to terminals can be added on the vertical layer. Also, after the modification has been made to each side, segments which must connect at the corners can be lengthened or shortened to do so without causing conflicts. □

This upper bound on the amount of height added by using separate layers for the two directions is actually achievable for a very simple routing problem shown in Figure 6.6.

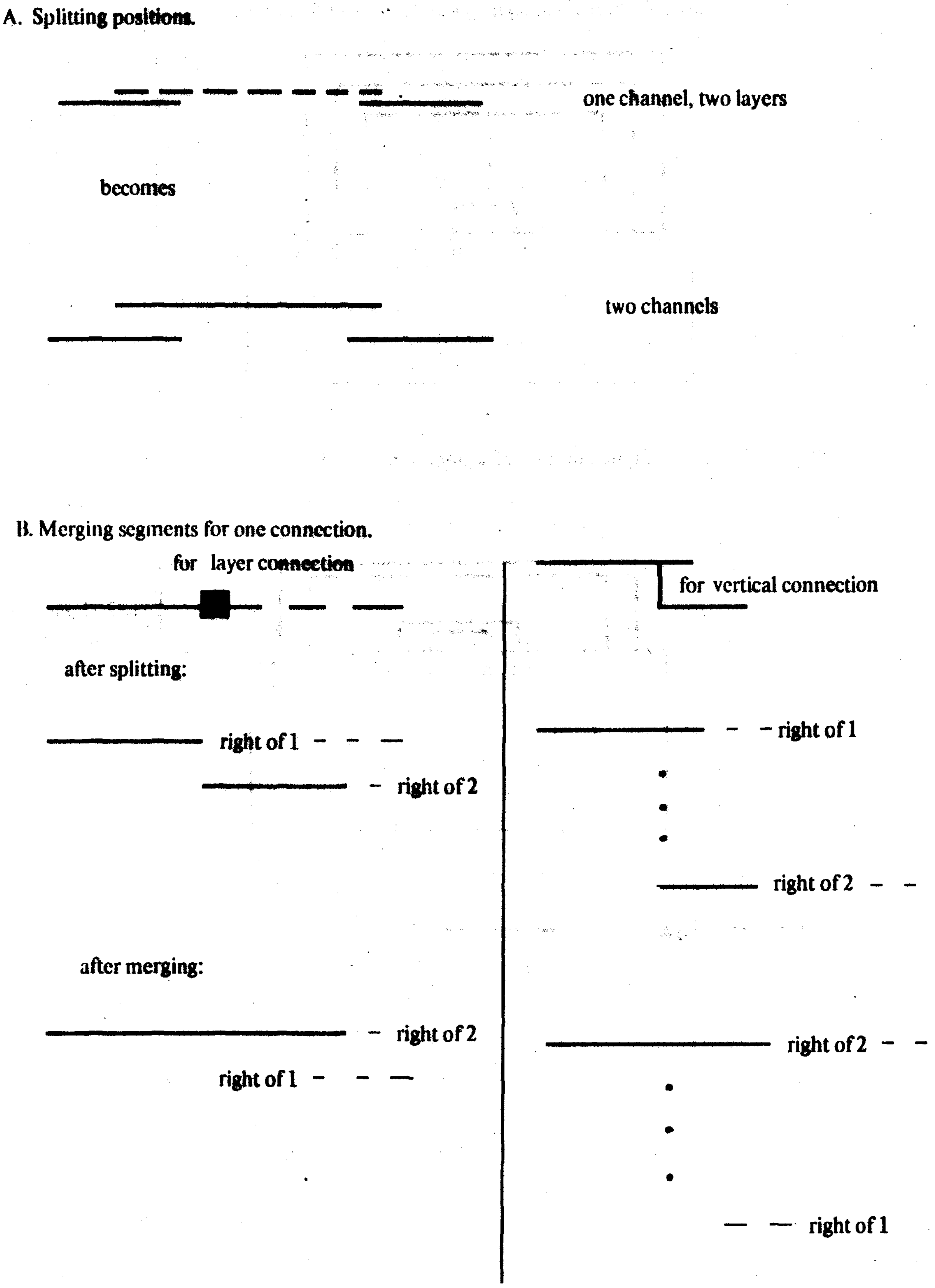
The argument used to prove Lemma 6.2 holds for any region above a component side as long as all wires are either perpendicular or parallel to the side, segments of arbitrary length extending from different terminals perpendicular to the side will not intersect, and segments which run parallel to the side can be extended at the ends of the side without causing conflicts. Any number of terminals can be interconnected, not only pairs.

6.3.2 Routing around a rectangle using more than two layers.

Let us now consider the One Component Routing Problem when there are more than two layers for interconnection. To take advantage of the extra layers while retaining the limitation to horizontal and vertical wire segments, some parallel segments must lie one on top of another. The connection of segments on different layers is no longer trivial. When two segments on different layers must connect, no segment intersecting the connection point can lie on any layer between the layers of the connecting segments without also being connected to them. The simplest way of avoiding unwanted connections is to put segments which connect on adjacent layers.

Suppose that we do wish to retain the "separate direction -- separate layer" strategy for more

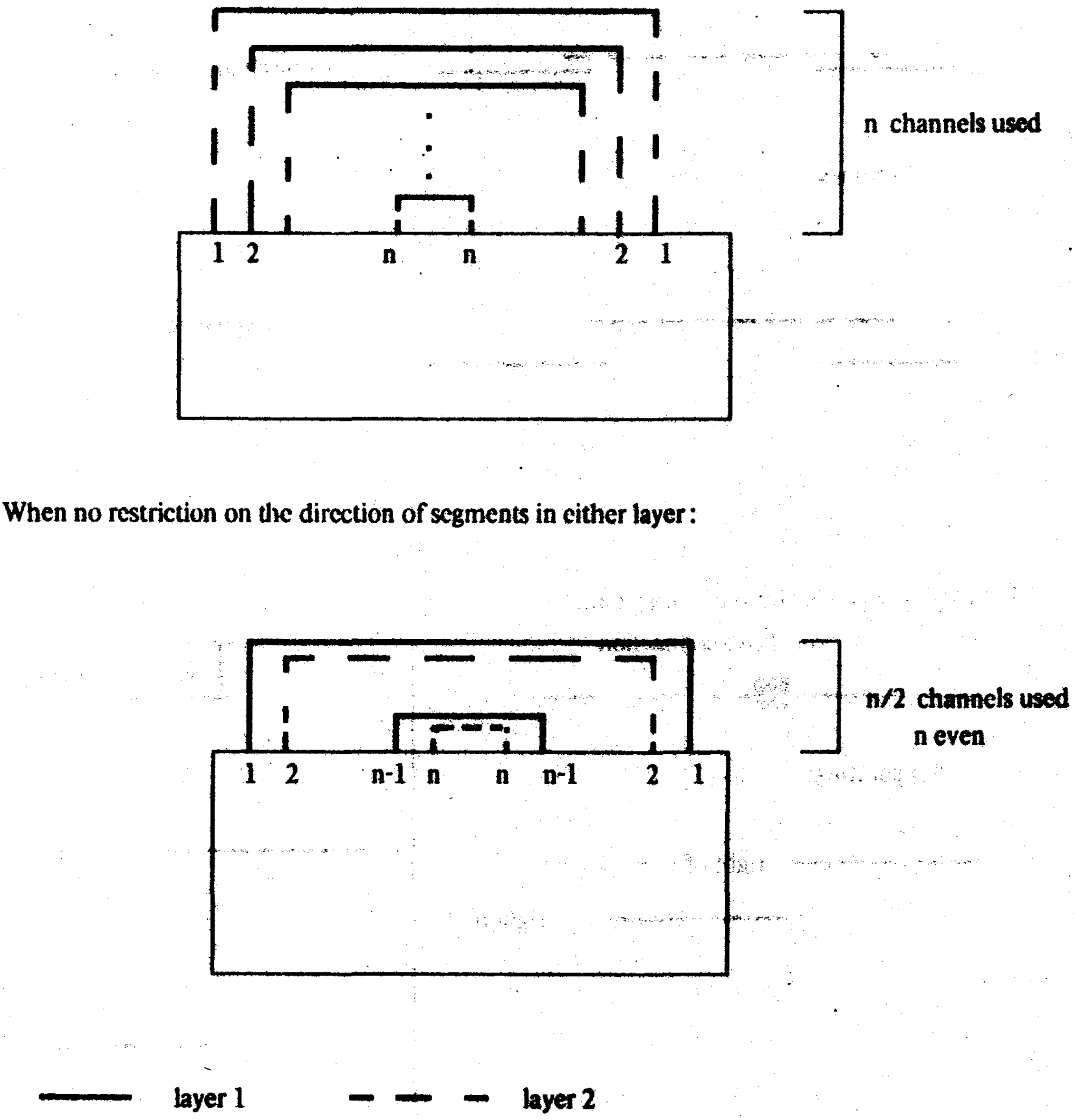
Figure 6.5: Construction for Lemma 6.2.



A. Splitting positions.

B. Merging segments for one connection.

Figure 6.6: Instance of One Component Routing Problem realizing upper bound of Lemma 6.2.



When one layer for horizontal segments and one layer for vertical segments;

When no restriction on the direction of segments in either layer:

than two layers. We will let all odd numbered layers be for horizontal segments and all even numbered layers be for vertical segments. Each layer is adjacent to layers containing wires in the other direction. Let "track" denote a particular layer in a channel. When there is only one layer for each direction, the number of channels used for routing along a side of the component is the same as the number of tracks used. When there are H layers for segments in each direction, the number of channels required is as little as 1/H of the number of tracks required. More than this number of channels may be needed to allow segments to connect at corners properly. The connections at corners place constraints on the layers used. All layers in a channel may not be usable. It is important to observe that there is no longer guaranteed to be an optimal routing in which each path uses at most one segment running along each side of the rectangular component. Permitting a path to use more than one segment along a side allows the path to change layers. In this way, layers may be better utilized. Figure 6.7 illustrates.

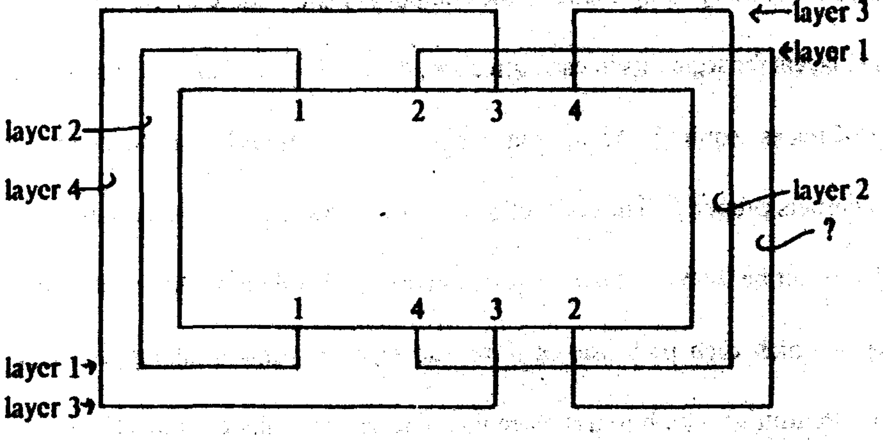
Let us suppose we do wish to restrict the routing to one segment along each side per path. We can choose which segments will share the same track, e.g. using the matching function of Chapter 5. Given any such choice, we must assign the resulting sets of segments to channels and layers in the channels so that the connections at corners are properly made. A legal assignment minimizing the area of the layout is desired. We have broken up the assignment of path segments to tracks into two phases: determining what segments will share the same track, and determining what track they will share. When there is only one layer for each direction, the second part of this problem is trivial -- any assignment will do.

If we require that all pairs of connecting segments be on adjacent layers, then we can model the constraints imposed by the connections between segments using a directed graph. The graph contains one node for each set of segments sharing a track. Let a node representing a set of segments above the top of the component be called a *top node*. Similarly, the terms *bottom node, left node*, and *right node* denote nodes representing sets of segments on the indicated sides. There is an edge between two nodes

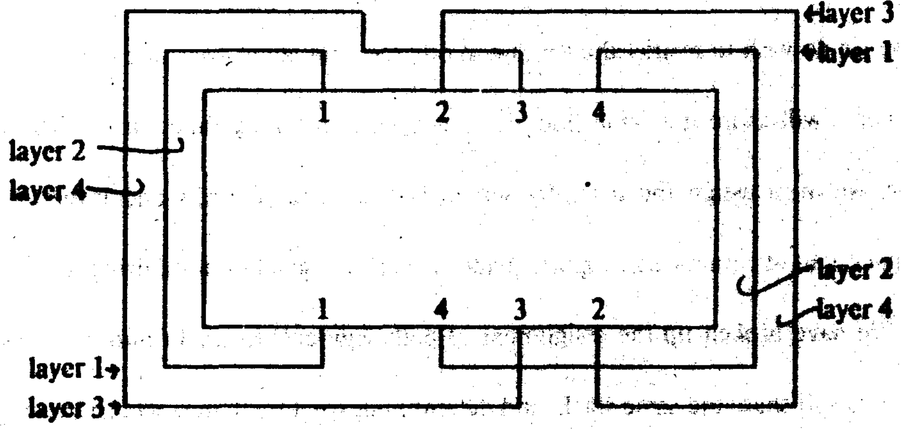
Figure 6.7: Layer assignment for the One Component Routing Problem.

It is no longer true that one channel per street for each net is always optimal.

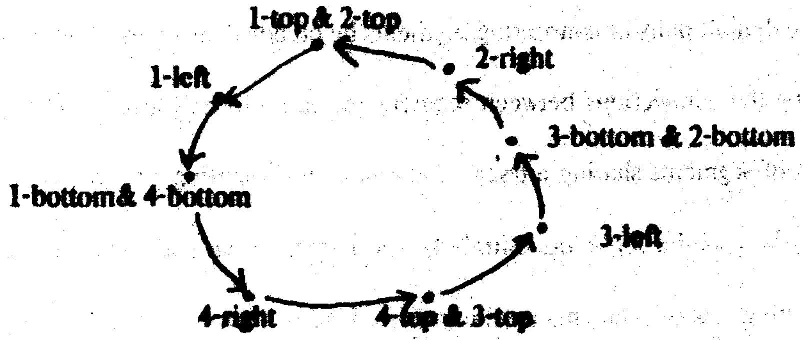
Given 4 layers, no assignment of layers uses only one channel on each side. One attempt is shown.



Versus a successful assignment



Graph representation of track connections for first track assignment



when the sets of segments represented by the nodes contain segments which must be connected at a corner. Edges are directed as follows: from a top node to a left node, representing a connection in the top left corner; from a left node to a bottom node, representing a connection in the bottom left corner; from a bottom node to a right node, representing a connection in the bottom right corner; from a right node to a top node, representing a connection in the top right corner. The graph consists of disjoint chains and cycles. Figure 6.7 shows the graph for the segments of that example. Since connections are always between adjacent layers, it suffices to determine the layer for each set of segments. The actual channel used by each set of segments can be arbitrarily chosen, just as they could for only two layers. The only restriction is that there be only one set of segments on each layer in any channel. Therefore, we wish to assign a layer number to each node of the graph so that left and right nodes have odd numbers and top and bottom nodes have even numbers. Adjacent nodes in the graph must be numbered with adjacent numbers. For any given side, each occurrence of a number will be in a different channel. We wish to find a numbering which minimizes the area of the corresponding layout.

Let us now consider graphs consisting only of disjoint cycles. We restrict our attention to these graphs to illustrate how cyclic constraints can be handles without worrying about graphs with different numbers of tracks on different sides, which can occur if chains are present. A method of assigning layers to nodes in a graph consisting only of cycles is as follows. Choose some cycle; begin with some top node of this cycle. Assign layer 1 to this node. Moving in both directions away from this node, assign the nodes along the cycle -- one node in each direction at each step -- in the following pattern. Going in the direction of the edges (clockwise), beginning with i=1, assign in consecutive steps: i+1 to the next left node; i to the next bottom node; i+1 to the next right node; i+2 to the next top node. Increase i by two and repeat the pattern. Going opposite to the direction of the edges (counterclockwise), beginning with i=1, in consecutive steps assign: i+1 to the next right node; i to the next bottom node; i+1 to the next left node; i+2 to the next top node. Increase i by two and repeat the pattern. The first node reached by both directions will either be a top or a bottom node. Since these are being assigned the same number at the same step, the cycle is completed properly. If the step assigning 2H to a right node clockwise and a left node counterclockwise is reached before the cycle is finished, a pattern of descending numbers is used. For clockwise, beginning with i=2H-1, assign: i to the next top node; i-1 to the next left node; i to the next bottom node; i-1 to the next right node. Decrease i by two and repeat. For counterclockwise, beginning with i=2H-1, assign: i to the next top node; i-1 to the next right node; i to the next bottom node; i-1 to the next left node. Decrease i by two and repeat. Again, if the step assigning 2 to a right node clockwise and a left node counterclockwise is reached before the cycle is finished, the original pattern of ascending numbers is repeated. By repeating the two patterns through increasing and decreasing numbers alternately, a cycle of any size can be labeled. (See Figure 6.8). After the first cycle is finished, a second cycle can be labeled beginning within the next step of the pattern which assigns to a top or bottom node. If the next step which assigns to one of the two sides assigns to a top node, then an arbitrary top node of the new cycle is chosen as the starting point. If the step assigns to a bottom node, then an arbitrary bottom node is chosen as the starting point. In this way, each cycle can be assigned in turn. Each pass through increasing or decreasing labels uses one channel on each side. At most one layer is unused in any channel. One pass through increasing numbers assigns numbers to H groups of four nodes. Each group consists of consecutive top, left, bottom, and right nodes on a cycle. One pass through decreasing numbers assigns numbers to H-1 such groups. Therefore, the number of channels used on any side under this assignment method is at most

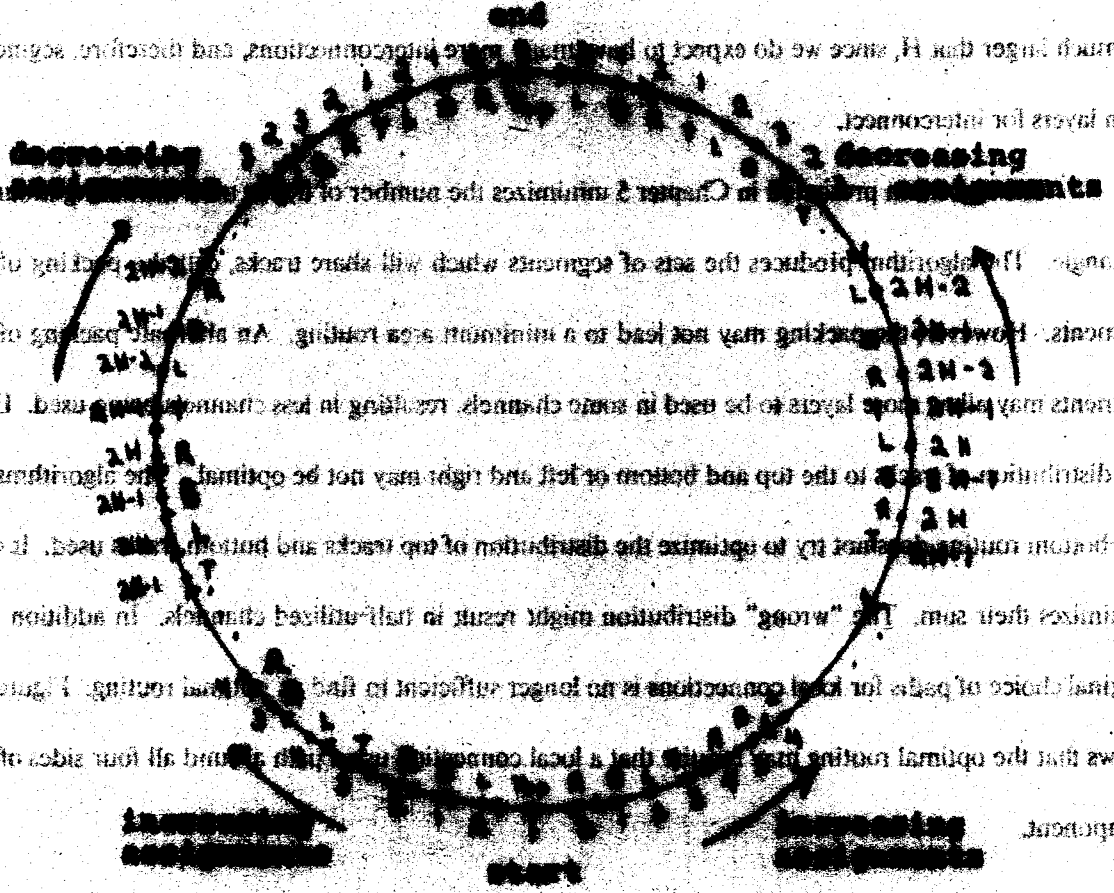
$$4 \times \lceil \Sigma_{cycles}(\tfrac{1}{4}n(cycle) + \tfrac{1}{4})/2H\text{-}1 \rceil, \qquad \text{equation 1}$$

where n(cycle) is the number of groups of four nodes in the cycle. Note that when a graph consists of only cycles, it contains the same number of nodes for each side. Therefore equation 1 is equal to

$$4 \times \lceil \tfrac{1}{4}(n_s + \text{the number of cycles})/2H\text{-}1 \rceil,$$

where n_s is the number of nodes for each side. The number of channels used on any side is at least 1/H

Figure 6.8: Labeling [illegible]



of the number of nodes for that side. Therefore, the ratio of the number of channels used on a side by the assignment technique to the the number of channels used on a side by an optimal assignment is at most $4H/2H\text{-}1 + 4H/n_s$. For $2 \leq H < 1/12n_s$, this ratio is less than three. It is reasonable to expect n_s to be much larger that H, since we do expect to have many more interconnections, and therefore, segments, than layers for interconnect.

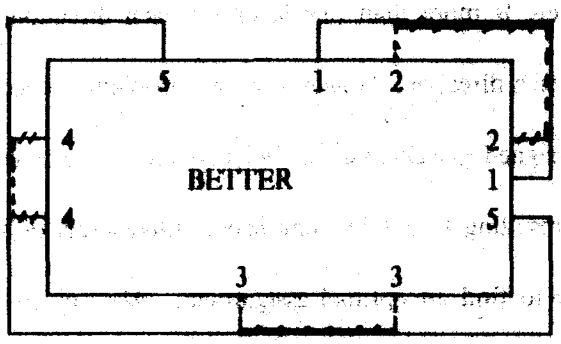
The algorithm presented in Chapter 5 minimizes the number of tracks used in routing around a rectangle. The algorithm produces the sets of segments which will share tracks, called a packing of the segments. However, the packing may not lead to a minimum area routing. An alternate packing of the segments may allow more layers to be used in some channels, resulting in less channels being used. Even the distribution of tracks to the top and bottom or left and right may not be optimal. The algorithms for top-bottom routing does not try to optimize the distribution of top tracks and bottom tracks used. It only minimizes their sum. The "wrong" distribution might result in half-utilized channels. In addition, our original choice of paths for local connections is no longer sufficient to find an optimal routing. Figure 6.9 shows that the optimal routing may require that a local connection use a path around all four sides of the component.

6.4 Summary

The algorithm presented in Chapter 5 finds an optimal routing when certain restrictions are placed on the problem and on the allowed routing paths. In this chapter, we have discussed the repercussions of removing some of these restrictions. When we allow solutions which use only two layers but allow horizontal and vertical segments on both layers or which use more than two layers, the algorithm no longer always finds an optimal solution. In these cases, the algorithm may be considered a heuristic algorithm. The algorithm limits the range of solutions it considers but finds an optimal solution within this limited set. When two layers are used for both horizontal and vertical segments, we have

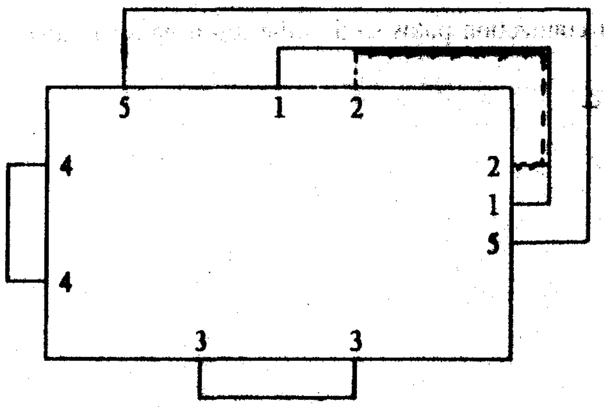
Figure 6.9: The One Component Routing Problem with more than two layers.

Connection for 5 goes around 4 sides.



adds 2 units to horizontal dimension
adds 2 units to vertical dimension

Connection for 5 goes around 2 sides.



adds 3 units to horizontal dimension
adds 3 units to vertical dimension

——— horizontal layer 1
-+-+- horizontal layer 2
| vertical layer 1
⋮ vertical layer 2

shown in Section 6.3 that the amount of length added to each dimension by the routing produced by the algorithm is never more that twice that added by the optimal routing. When separate layers are used for horizontal and vertical segments but there is more than one layer for each direction, the algorithm minimizes the number of tracks used in each direction. However, a new problem is encountered. Each set of segments sharing a track must be assigned to a channel and layer so that connections at corners can be made properly and the area of the resulting layout in minimized. Given sets of segments to be assigned, we do not know an algorithm to find an optimal assignment. Also, given a collection of segments, we do not know how to pack them into tracks so that the resulting area, rather than the resulting number of tracks, is minimized. Finally, given a One Component Routing Problem, we do not know how to choose directions for connection paths so that the resulting area, rather than the resulting number of tracks used, is minimized.

Chapter 7 Summary and Directions for Research

In this thesis, we have examined problems encountered in circuit layout from the perspective of complexity theory. Our goal has been to gain better understanding of the problems and discover better techniques for their solution. We have presented three proofs of NP-completeness -- one for an orthogonal layout problem with rectangular components of arbitrary size, one for the channel assignment problem in a street, and one for the intersection channel assignment problem, i.e. channel assignment over all streets and intersections. We also have analyzed a common heuristic algorithm for channel assignment within a street. This analysis shows that the ratio of the number of channels used in the solution produced by the algorithm to the optimal number is not bounded by any constant. This algorithm and its analysis provide a reference point against which other algorithms can be compared. The development of better algorithms for channel assignment remains a topic for research.

The proof of NP-completeness of the channel assignment problem holds only when each net is required to use at most one channel, i.e. channel assignment without jogs. The complexity of the channel assignment problem with jogs remains open. The author conjectures that the problem is NP-complete. In practice, jogs are allowed within a street. Therefore, the analysis of known channel assignment algorithms which use jogs and the development of better algorithms are useful areas of research. The lower bounds on optimal channel assignments without jogs do not apply when jogs are allowed. Analysis of solutions when jogs are allowed is necessary to determined useful lower bounds.

In Chapter 5, we have presented an algorithm which finds the optimal channel routing for a special case of the layout problem in polynomial time. Among other requirements, the layout problem must involve only one rectangular component, and all nets must contain exactly two terminals. We have shown in Chapter 6 that when any restrictions on the problem are removed, the algorithm is no longer guaranteed to find the optimal solution. Assumptions about satisfactory routing paths no longer hold. When nets are allowed to have more than two terminals but all other restrictions of Chapter 5 hold, the complexity of the layout problem is open. A modification of the algorithm presented in Chapter 5 or a completely different algorithm may find an optimal solution in polynomial time. The author has been unable to find such an algorithm or produce a proof of NP-hardness. An alternate relaxation of the restrictions allows two components which lie side by side, but does not allow terminals on the adjacent sides. The restriction to two terminal nets is retained. The ability of paths to run between the two rectangles is added. This version of the problem has also defied analysis. More generally, the techniques used in the algorithm may be used as heuristics for the general routing problem. Further research is necessary to determine the quality of these techniques as heuristics.

An interesting combinatorial problem is suggested by the routing problem of Chapter 5. Circular arc coloring and routing around a rectangle resemble each other except that, for the rectangle, paths can go one of two ways around the rectangle and paths can change "color" at any corner. Suppose we extend circular arc coloring so that "arcs" can go around the circle in either direction. More precisely, we modify the circular arc coloring problem as follows. Pairs of points on a circle are given. The problem is to choose one of the two arcs determined by each pair and color the chosen arcs so that the number of colors used is minimized. Since circular arc coloring is NP-complete, we suspect that this modification is as well.

We have chosen to use a model which represents components as rectangles and wires as objects with uniform width and uniform interwire spacing. The design rules for integrated circuit design [Mel80] are more complicated. Widths and spacings vary from layer to layer. Under the model we use, the widest of the required widths and spacings must be use. This simplification removes the complications of particular technologies from the topological and geometric aspects of the layout problem. Problems which vary significantly from our model, such as power and ground wiring, require separate algorithms. We assume that no topological information about the desired layout is given with the problem specification. This is in contrast with the stick diagram approach reviewed in Chapter 2, which has topological information as part of the input. Both techniques have been found useful in practice, but lead to different problems. The study of algorithms for problems encountered in alternate approaches to layout from that in this thesis is an area of complementary research.

In Chapter 3, in addition to presenting the model we have used, we discussed a graph theoretic model for layout. This graph theoretic model has proven very useful in obtaining bounds on the area required by interconnection patterns represented by various classes of graphs. There are many open problems in this area of research. Two open problems of great interest are to determine tight upper and lower bounds on the area needed to embed an arbitrary planar graph and on the area needed to embed a shuffle exchange graph [Lei80], [Th80]. An alternate cost measure to those presented in Chapter 3 has recently been proposed by Storer [St80]. He also proves the NP-hardness of optimally embedding a graph in the grid both under the cost measures defined in Chapter 3 and under his measure.

In the problems we have considered, placement and routing are restricted to be orthogonal. Research is needed on the cost of this assumption. How much space is lost by allowing only horizontal and vertical wires? How much is gained by adding a third direction for wires, e.g. 45 degree wires. Tompa [To80] has shown that for one street containing terminals on both sides such that all connections are of the form "connect the i^{th} terminal on one side to the i^{th} terminal on the second side", the optimal routing uses arcs of circles as wire paths. We have also assumed that horizontal and vertical wires lie on different layers. We have indicated how much this assumption costs for the problem of Chapter 5. How much this assumption costs in general is an open problem. When there are only two layers for interconnect, running two wires in parallel, one on top of the other, on separate layers across a chip separates the two sides of the chip. This suggests that restricting each layer to wires in one direction is desirable.

At present, most integrated circuit designs are done with only two layers for interconnect. However, in the future, more layers may be available. When more than two layers are available restricting routing to two orthogonal directions may be very costly in terms of area of layouts. Should more directions be introduced and one direction be used for each layer? Should the "different layer, different direction" technique be abandoned completely? For more than two layers, a major area of research is to determine the restrictions which simplify the routing problem sufficiently to allow development of good algorithms and which result in good layouts in relation to the layouts produced without the restrictions.

The model we have used considers only the area of layouts. In practice, other characteristics of circuit layouts, such as circuit timing and power consumption, are of interest. Research is needed into ways of modeling these parameters. Very few algorithms have been developed which take these parameters into account in any way. We would also like to have a less restrictive model of components. The extension of the model to include logically and physically equivalent terminals was discussed in Chapter 3. We would also like the ability to define variable size components -- components which could stretch or shrink depending on the requirements of the layout. A very useful concept is that of semi-restricted regions for routing. These would be regions where some but not all layers are free to be used by wires. This concept in combination with stretchable cells could allow wires in certain layers to run over cells after the cells have been stretched to make room for the wires. These concepts have been used by some systems, but new representations and algorithms exploiting these representations are needed.

In the thesis, we have separated placement and routing in the conventional fashion. The interaction of placement and routing is very complicated. One area of research is the development of algorithms which do not separate placement and routing, or which at least allow placement and routing to interact more. The graph theoretic techniques mentioned in Section 2.2 do consider placement and routing simultaneously. However, these models do not represent the geometry of the components. We have seen in Chapter 4 that packing rectangles in minimum area is in itself NP-complete. Therefore, the geometry of components adds a significant complication to the problem. When placement and routing are separated, it is not clear whether it is important to get a good geometric packing or a packing which facilitates routing. At one extreme, a placement which is a good packing of components as rectangles can be sought. This, of course, is NP-complete. At the other extreme, the placement can be governed by the desired interconnections between components with little or no regard for how the rectangles fit together. One area for future research is the study of the tradeoffs between the two aspects of placement. Good criteria for judging whether a placement does facilitate the routing are needed. The criteria currently used most often estimate wire length, but not routing area.

We have intended our model to be independent of the particular technology of interest, although we have used the design rules of [Me80] for nMOS as our guide. These design rules are intended to scale down as the fabrication technology becomes capable of creating smaller and smaller objects. Therefore, our model should remain useful despite the fact that actual parameters of designs change frequently.

In this thesis, we have been concerned with the quality of algorithms to solve problems encountered in layout. The criteria of complexity theory are not the only criteria useful in judging an algorithm to be used in practice. Algorithms should be easy to use and maintain. They should allow some interaction with the human designer. The algorithm presented in Chapter 5 finds optimal solutions. Therefore, interaction with the designer is not as important as it is for heuristic algorithms, where the designer's insight is an important input. Nevertheless, the algorithm of Chapter 5 can be written to allow the designer to predetermine the paths of some wires. It will find the optimal routing using these paths. Another criterion for judging algorithms is the number of good solutions which an algorithm can produce efficiently. An algorithm which can present a designer with several good choices will be preferable to one which presents one good solution, unless the one solution is substantially better than any of the choices. Even among algorithms which find optimal solutions, one which obtains more than one optimal solution in approximately the same running time may be preferred. Criteria other than the optimization criteria may be used to choose among optimal solutions. For example, the algorithm of Chapter 5 is biased towards assigning "0"s to left top terminals. A dual version of the algorithm biased towards assigning "1"s to bottom right terminals might also be run to find an alternate solution. Complexity analysis can assist in the development of better algorithms for layout problems. However, the usability of an algorithm must also be considered in evaluating new algorithms for layout design.

Appendix: Basic Definitions

In this thesis, the concepts of graph theory will frequently be used to describe problems. All definitions can be found in [Liu68]. We review them here.

A *graph*, G = (V,E), consists of a finite set of *nodes* (or *vertices*), V, and a set of *edges*, E. Each edge, e = (v_1,v_2), is a pair of nodes. Edge e is said to be *incident* on vertices v_1 and v_2; vertices v_1 and v_2 are the *endpoints* of e. Vertices v_1 and v_2 are *adjacent* to each other. If the edges are ordered pairs, the graph is *directed*; otherwise, it is *undirected*. In a directed graph, an edge (v_1,v_2) goes *from* (or *out of*) v_1 *to* (or *into*) v_2. If a node has d edges incident to it, then the node is of *degree* d.

A *path* in a graph is a sequence of edges:

$$(v_1,v_2), (v_2,v_3), (v_3,v_4), ..., (v_n,v_{n+1}).$$

The nodes on the path are the nodes v_1, v_2, ..., v_{n+1}. The path is of length n. Nodes v_1 and v_{n+1} are the endpoints of the path; the path goes from v_1 to v_{n+1}. If no node appears in the sequence of edges more than once, the path is said to be *simple*. We will always mean "simple path" when we say "path". A path is a *cycle* if $v_1 = v_{n+1}$. A simple cycle is a path on which only $v_1 = v_{n+1}$ appears more than once. A graph is *acyclic* if there are no cycles in the graph. An acyclic undirected graph is called a *tree*. A *subgraph*, S = (V_S,E_S), of a graph, G = (V_G,E_G), is a graph whose set of nodes, V_S, is a subset of V_G and whose set of edges, E_S, is a subset of E_G containing only edges whose endpoints are in V_S.

A graph, G, is *planar* if there is a mapping from the nodes of G to points in the plane and from edges of G to curves in the plane such that:

(1) The endpoints of the curve corresponding to an edge are the images of the endpoint of the edge;

(2) No two curves intersect at any point other than their endpoints;

(3) A curve does not intersect the image of a node unless that node is an endpoint of the edge corresponding to the curve.

Informally, a planar graph is one which can be drawn in the plane without crossing edges.

An m by n two-dimensional *grid graph* is an undirected graph with node set:

$$V = \{(i,j) \mid i \text{ and } j \text{ are positive integers, } 1 \leq i \leq m \text{ and } 1 \leq j \leq n\}$$

and edge set:

$$E = \{(v_1,v_2) \mid v_1 = (i,j) \text{ and } v_2 = (i,j+1) \text{ for } 1 \leq i \leq m \text{ and } 1 \leq j \leq n-1$$
$$\text{or } v_1 = (i,j) \text{ and } v_2 = (i+1,j) \text{ for } 1 \leq i \leq m-1 \text{ and } 1 \leq j \leq n\}.$$

To indicate the growth rate of functions when discussing the performance of an algorithm, we need the following notation. Given real-valued functions f and g on the same domain, f is $O(g)$ if there is a positive constant, c, such that for all but a finite number of domain values, $f(x) \leq cg(x)$. The O notation is used when discussing upper bounds on a function. A statement of the form "the running time is $O(g(n))$" gives an upper bound on the running time. When discussing lower bounds, the Ω notation is used. Given real-valued functions f and g on the same domain, f is $\Omega(g)$ if there is a positive constant, c, such that for all but a finite number of domain values, $f(x) \geq cg(x)$. Then "the running time is $\Omega(g(n))$" is stating a lower bound on the running time.

The above definitions are of the concepts most frequently used in this thesis. Other definitions are presented as needed.

Annotated Bibliography

[Ab72] Abel, Luther C., "On the Ordering of Connections for Automatic Wire Routing," *IEEE Transactions on Computers*, Vol. C-21, No. 11, Nov. 1972, pp. 1227-1233.

Presents experimental evidence that the success of routing one connection at a time (point-to-point) on one layer using a maze router is independent of the order in which connections are attempted. Performance is measured in terms of the total of the rectilinear distances between the endpoints of the connections successfully completed. The maze router includes a heuristic to avoid running a path next to a row of terminals and blocking all the terminals.

[Abs80] Abelson, H.; Andreae, P., "Information Transfer and Area-Time Tradeoffs for VLSI Multiplication," *Communications of the ACM*, Vol. 23, No. 1, Jan. 1980, pp. 20-23.

[Agu77] Agule, B.J.; Lesser, J.D.; Ruehli, A.E.; Wolff, P.K. Sr., "An Experimental System for Power/Timing Optimization of LSI Chips," *Proceedings of the Fourteenth Design Automation Conference*, IEEE, June 1977, pp. 147-152.

Presents (with [Ru77]) a system to produce layouts of integrated circuits when power and timing are considered. Given a circuit, a layout minimizing the total power required to drive the circuit while satisfying timing constraints is desired.

[Ah74] Aho, A.; Hopcroft, J.; Ullman,J., *The Design and Analysis of Computer Algorithms*, Addison-Wesley Publishing Co., Reading, Mass., 1974.

A good text on the analysis of algorithms.

[Ay79] Ayres, Ron, "Silicon Compilation -- A Hierarchical Use of PLAs", *Proceedings of the Sixteenth Design Automation Conference*, IEEE, June 1979, pp. 314-326.

Presents a method of converting a specification of a function in synchronous logic (a high level language) to an implementation of the function on a chip using one or more PLAs.

[Ba79] Baker, B.S.; Coffman, E.G. Jr.; Rivest, R.L., "Orthogonal Packing in Two Dimensions," Department of Electrical Engineering and Computer Science TRCS79-1, U. of California at Santa Barbara (also to appear in *SIAM J. on Computing*).

Presents the analysis of a heuristic algorithm for a restricted placement problem for rectangles. Rectangles have a predetermined orientation.

[Be66] Berge, C., *The Theory of Graphs and its Applications*, John Wiley & Sons, Inc., New York, 1966.

[Br76] Brinkmann, Mlynski, "Computer-Aided Chip Minimization for IC-Layout", *Proceedings of the IEEE International Symposium on Circuits and Systems*, April 1976, pp. 650-653.

Presents a method of placing rectangular components on a rectangle. The goal is to minimize unused space. The method is to divide the large rectangle into smaller rectangles, and modify the small rectangles to the size of the actual components.

[Ch77] Cho, Y.E.; Korenjak, A.J.; Stockton, D.E., "Floss: An Approach to Automated Layout for High-Volumn Designs,", *Proceedings of the Fourteenth Design Automation Conference*, IEEE, June 1977, pp. 138-141.

Describes FLOSS, a program for the automatic packing of the hand sketch of a circuit layout.

[Des72] *Design Automation of Digital Systems, Volumn 1: Theory and Techniques*, Melvin Breuer, ed. Prentice-Hall, Inc. Englewood Cliffs, N.J. 1972.

Reviews various problems within design automation. Logic synthesis, simulation and test generation are covered as well as all phases of layout. A good introduction, but addresses primarily printed circuit layout.

[Deu76] Deutsch, D.N., "A 'Dogleg' Channel Router," *Proceedings of the Thirteenth Design Automation Conference*, IEEE, 1976, pp. 425-433.

Describes a channel assignment algorithm when jogs are allowed.

[Fe76] Feller, A., "Automatic Layout of Low-Cost Quick-Turnaround Random-Logic Custom LSI Devices," *Proceedings of the Thirteenth Design Automation Conference*, IEEE, June 1976, pp. 79-85.

Presents the standard cell approach used by RCA. Components, called cells are placed in rows.

[Gar79] Garey, M.R.; Johnson, D.S., *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W.H. Freeman and Co., San Francisco, 1979.

Gives a good presentation of the theory of NP-completeness. Techniques for proving problems NP-complete and methods of dealing with NP-complete problems are described. A list of over 300 NP-complete problems is given in the appendix.

[Gar] Garey, M.R.; Johnson, D.S.; Miller, G.L.; Papadimitriou, C.H., "The Complexity of Coloring Circular Arcs and Chords", *SIAM Journal on Algebraic and Discrete Methods*, to appear.

[Gav72] Gavril, F., "Algorithms for Minimum Coloring, Maximum Clique, Minimum Covering by Cliques, and Maximum Independent Set of a Chordal Graph", *SIAM Journal of Computing*, Vol. 1, No. 2, June 1972, pp.180-187.

The class of chordal graphs includes the class of interval graphs. Channel assignment under certain conditions and interval graph coloring are equivalent (see Chapter 4).

[Gi76] Gibson, D.; Nance, S.; "SLIC – Symbolic Layout of Integrated Circuits," *Proceedings of the Thirteenth Design Automation Conference*, IEEE, 1976, pp.434-440.

Presents a system under which designers use symbols to represent mask topologies. A design specified by symbols closely corresponds to a full drawing. A symbolic design is automatically transformed into a full geometric specification.

[Gu79] Gupta, U.; Lee, D.; Leung, J., "An Optimal Solution for the Channel-Assignment Problem", *IEEE Transactions on Computers*, Vol. C-28, No. 11, Nov. 1979, pp. 807-810.

Presents an algorithm with $O(n\log n)$ running time for solving the channel assignment problem when there are no constraints due to terminals across from one another. The algorithm is generalized to solve a two-dimensional problem.

[Han72] Hanan, M.; Kurtzberg, J., "A Review of the Placement and Quadratic Assignment Problems," *SIAM Review*, Vol. 14, No. 2, April 1972, pp. 324-342.

A review of algorithms for the problem of placing points to optimize some function and a special case, called the quadratic assignment problem. A summary of experiments done with some programs is given.

[Han76] Hanan, M.; Wolff, P.K.; Agule, B.J., "Some Experimental Results on Placement Techniques," *Proceedings of the Thirteenth Design Automation Conference*, IEEE, June 1976, pp. 214-224.

Presents the results of an empiracal study of placement algorithms. Both random and constructive initial placements are combined with various iterative improvement techniques.

[Has71] Hashimoto, A.; Stevens, J., "Wire Routing By Optimizing Channel Assignment within Large Apertures," *Proceedings of the Eighth Design Automation Workshop*, IEEE, 1971, pp. 155-169.

This paper introduced the channel routing technique. The general method is outlined and an algorithm for channel assignment when there are no constraints due to terminals is given.

[He78] Heller, W.; Mikhail, W.; Donath, W., "Prediction of Wiring Space Requirements for LSI," *Journal of Design Automation and Fault-Tolerant Computing*, Vol. 2, 1978, pp. 117-144.

Presents a method of estimating the space required for wiring using a grid and an array of locations. A stochastic model is used.

[Hi69] Hightower, David, "A Solution to Line-Routing Problems on the Continuous Plane", *Proceedings of the Sixth Design Automation Workshop*, SHARE, ACM, and IEEE, 1969, pp.1-24.

This paper introduces the escape line technique for finding a path between two points.

[Hi74] Hightower, David, "The Interconnection Problem: A Tutorial," *Computer*, Vol.7, No.4, April 1974, pp. 18-32.

A survey of the routing problem, primarily for printed circuit boards. Discusses pin assignment, wire list determination, layering, ordering, and "wire layout" (routing as we have discussed it). Includes a good summary of the major routing algorithms. Has a large bibliography.

[Hit69] Hitchcock, R.B., "Cellular Wiring and the Cellular Modeling Technique", *Proceedings of the Sixth Design Automation Workshop*, SHARE, ACM, and IEEE, 1969, pp.25-41.

Introduces the cellular approach to routing.

[Hs79] Hsueh, Min-Yu; Pederson, Donald, "Computer-Aided Layout of LSI Circuit Building-Blocks", *Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS)*, 1979, PP.474-477.

This paper describes CABBAGE, a program which converts a schematic stick representation of an integrated circuit layout into a full geometric representation. Once a geometric representation is obtained, the program compacts the circuit layout.

[Hw78] Hwang, F.K., "The Rectilinear Steiner Problem," *Journal of Design Automation and Fault-Tolerant Computing*, Vol. 2, No. 4, Oct. 1978, pp. 303-310.

Presents a review of results on the Rectilinear Steiner Problem.

[Jo79] Johannsen, D., "Bristle Blocks: A Silicon Compiler", *Proceedings of the Sixteenth Design Automation Conference*, IEEE, June 1979, pp. 310-313.

Presents a system for the design of LSI circuits base on a particular chip architecture. A hierarchical design approach is used. A number of representations are exploited.

[Ka79] Kawamoto, T.; Kajitani, Y., "The Minimum Width Routing of a 2-Row 2-Layer Polycell-Layout", *Proceedings of the Sixteenth Design Automation Conference*, IEEE, June 1979, pp. 290-296.

Presents a polynomial time algorithm to find a minimum channel assignment when jogs are allowed anywhere, i.e. when the street length can expand.

[Kel77] Kelly, Michael; Smith, Robert, "Analytical and Experimental Analysis of Routing Algorithms", *Proceedings of the Eleventh Asilomar Conference on Circuits, Systems, and Computers*, IEEE, Nov. 1977, pp.362-369.

Presents an empirical study of maze, line search, and channel routers. Data is difficult to interpret.

[Ker73] Kernichan, B.; Schweikert, D.; Persky, G., "An Optimal Channel-Routing Algorithm for Polycell Layouts of Integrated Circuits," *Proceedings of the Tenth Design Automation Workshop*, IEEE, 1973, pp. 50-59.

Presents an algorithm which finds an optimal channel assignment. The algorithm searches through all possible solutions and has exponential running time.

[Ko69] Kodres, U.R., "Logic Circuit Layout," *Proceedings of the Joint Conference on Mathematical and Computer Aids to Design*, ACM/SIAM/IEEE, Oct. 1969, pp. 165-191.

A survey paper which uses a graph theoretic model for circuit layout. Other subproblems of the layout problem in addition to placement and routing are discussed.

[Ku79] Kuh, E.S.; Kashiwabara, T.; Fujisawa, T., "On Optimal Single-Row Routing," *IEEE Transactions on Circuits and Systems*, Vol CAS-26, No. 6, June 1979, pp. 361-368.

See entry for [So74].

[La79] Lauther, Ulrich, "A Min-Cut Placement Algorithm for General Cell Assemblies Based on a Graph Representation", *Proceedings of the Sixteenth Design Automation Conference*, IEEE, June 1979, pp.1-10.

Describes a method of placement of rectangular components of arbitrary size. A square of the same area as the total area of the components is partitioned into pieces of the same area as the individual components. In this way, the relative positions of the components in the layout are determined.

[Lee61] Lee, C. Y., "An Algorithm for Path Connections and Its Applications", *IRE Transactions on Electronic Computers*, VEC-10, September 1961, pp.346-365.

Presents an algorithm for finding the optimal path between two point in a graph using any of several optimality criteria. This algorithm is the foundation of the maze routing technique.

[Lei80] Leiserson, Charles, "Area-Efficient Graph Layouts (for VLSI)" to appear in *Proceedings of the Twenty-First Annual Symposium on Foundations of Computer Science*, IEEE, Oct. 13-15, 1980.

Presents upper bounds on the area needed to embed graphs in the grid. The bounds are for graphs belonging to classes with a separator theorem.

[Liu68] Liu, C.L., *Introduction to Combinatorial Mathematics*, McGraw-Hill Book Co., New York, 1968.

A presentation of graph theory concepts can be found here.

[Lo79] Loosemore, K.J., "Automatic Layout for Integrated Circuits", *Proceedings of the IEEE International Symposium on Circuits and Systems*, 1979, pp.665-668.

This paper describes an automatic layout system developed for the GABLIC design automation system. Components are variable size rectangles. They are placed in horizontal regions (more flexible that rows) with streets for routing in between. The components can be placed above the routing area independently, so that less area is wasted.

[Me80] Mead, Carver; Conway, Lynn, *Introduction to VLSI Systems*, Addison-Wesley Publishing Co., Reading, Mass., 1980.

This text presents the fundamentals of designing digital systems in nMOS technology. A systems approach rather than an electronics approach is taken. This is an excellent reference for anyone who wishes to learn how to design chips.

[Na78] Nan, N.; Feuer, M. "A Method for the Automatic Wiring of LSI Chips," *Proceedings of the IEEE International Symposium on Circuits and Systems*, May 1978, pp. 11-16.

Uses the channel routing technique of dividing routing into global and local problems.

[No76] Noto, Richard; Wagner, Barry, "Computer-Aided Design for Complex Circuits", US Army Electronics Research and Development Command Technical Report DELET-TR-76-1398-F.

Presents a program which contains a method for placing and routing critical paths, whose delay must be minimized. The program produces output pin delay characteristics as well as a layout.

[Per73] Persky, G.; Gummel, H.K., "Grafos -- A Symbolic Routing Language," *Proceedings of the Tenth Design Automation Workshop*, IEEE, 1973, pp. 173-181.

Presents a language for specifying connections. The designer symbolically represents a routing as a program specifying what line segments should be drawn between what points. The program realizes the specification.

[Per77] Persky, G.; Deutsch, D.N.; Schweikert, D.G., "LTX -- A Minicomputer-Based System For Automated LSI Layout", *Journal of Design Automation and Fault-Tolerant Computing*, Vol. 1, No. 3, May 1977, pp. 217-255.

Presents Bell Laboratory's system for integrated circuit design using standard cells. Many interesting algorithms have been developed for this system. They are outlined in this paper.

[Pi77] Pippenger, N., "Superconcentrators," *SIAM Journal on Computing*, Vol. 6, No. 2, June 1977, pp. 298-304.

[Po79] Porter, Edwin, "An Automatic Layout System for VLSI", *Proceedings of the Eighteenth IEEE Computer Society International Conference (COMPCON)*, IEEE, February 1979, pp. 9-14.

Describes the DIAL system, a system intended for VLSI layout. The system is designed to be completely automatic, i.e. to produce VLSI layouts without human assistance.

[Pr78] Preas, B.T.; Gwyn, C.W., "Methods for Hierarchical Automatic Layout of Custom LSI Circuit

Masks," *Proceedings of the Fifteenth Design Automation Conference*, June 1978, pp. 206-212.

Presents a layout system which works with components which are rectangles of arbitrary size. A hierarchical approach to circuit design is used.

[Pr79] Preas, B.T.; vanCleemput, W.M., "Placment Algorithms for Arbitrarily Shaped Blocks," *Proceedings of the Sixteenth Design Automation Conference*, IEEE, June 1979, pp. 474-480.

Describes a method of exhaustively searching for an optimal placement of rectangles of arbitrary size. Also presents a heuristic initial placement algorithm. An iterative improvement algorithm is discussed briefly.

[AsC] *Proceedings of the Asilomar Conferences on Circuits, Systems, and Computers*, sixth (1972) through thirteenth (1979). (Formerly *Asilomar Conference on Circuits and Systems*, first (1967) through fifth (1971).) 1968, 1977, 1978 copywrites by IEEE, remaining copywrites by Asilomar Conference.

These conferences contain sessions on design automation.

[DAC] *Proceedings of the Design Automation Conferences*, IEEE, twelvth (1975) through seventeeth (1980). (Formerly the *Design Automation Workshops*. Proceedings of second (1965) through eleventh (1973) available.) Sponsored by SHARE from 1965 through 1971. Joint sponsorship of ACM with SHARE and/or IEEE from 1967 through 1980.

Major conference on design automation systems and techniques. Many examples of working design automation systems can be found.

[CAS] *Proceedings of the IEEE International Symposia on Circuits and Systems*, 1974 through 1980 (formerly *International Symposium on Circuit Theory*), IEEE.

These symposia usually contain several sessions on design automation.

[Ru77] Ruehli, A.E.; Wolff, P.K. Sr.; Goertzel, G., "Analytical Power/Timing Optimization Technique for Digital System," *Proceedings of the Fourteenth Design Automation Conference*, IEEE, June 1977, pp. 142-146.

Presents (with [Agu77]) a system to produce layouts of integrated circuits when power and timing are considered. Given a circuit, a layout minimizing the total power required to drive the circuit while satisfying timing constraints is desired.

[Sah80] Sahni, S.; Bhatt, A., "The Complexity of Design Automation Problems," *Proceedings of the Seventeenth Design Automation Conference*, IEEE, June 1980, pp. 402-411.

Presents a survey of NP-hard problems related to design automation. The layout problems discussed use a point model for components.

[Sah76] Sahni, S.; Gonzalez, T., "P-Complete Approximation Problems," *Journal of the ACM*, Vol. 23, No. 3, July 1976, pp. 555-565.

It is shown that, for several NP-complete optimization problems, no polynomial-time algorithm can find solutions which are guaranteed to have objective function values within a constant multiple of the optimal unless NP = P. Quadratic assignment is one of the problems.

[Sav79] Savage, J.E., "Area-Time Tradeoffs for Matrix Multiplication and Related Problems in VLSI Models," Brown University Department of Computer Science Technical Report No. CS-50, 1979.

[Sc76] Schweikert, Daniel, "A 2-Dimensional Placement Algorithm for the Layout of Electrical Circuits," *Proceedings of the Thirteenth Design Automation Conference*, IEEE, 1976, pp.408-416.

The algorithm presented uses constructive initial placement and improvement by pairwise exchange.

[So74] So, H.C., "Some Theoretical Results on the Routing of Multilayer, Printed-Wiring Boards," *Proceedings of the IEEE International Symposium on Circuits and Systems*, 1974, pp. 296-303.

Presents a decomposition of the routing problem for multi-layer printed circuit boards, resulting in several instances of a routing problem for one row of points -- the single-row, singe-layer routing problem. In this problem, space on either side of the row and in between the points can be used by wire paths. Sufficient conditions for routability of a collection of nets are studied. Theoretical research on the single-row, single-layer routing problem has been continued by several people. See [Ku79, Ti76, Ti78, Ts79].

[St80] Storer, J.A., "The Node Cost Measure for Embedding Graphs on the Planar Grid," *Proceedings of the Twelfth Annual ACM Symposium on Theory of Computing*, 1980, pp. 201-210.

[Su73] Sutherland, Ivan; Oestreicher, Donald, "How Big Should a Printed Circuit Board Be?" *IEEE Transactions on Computers*, May 1973, pp.537-542.

Presents an theory for predicting what size printed circuit boards should be for successful circuit layout.

[Ti78] Ting, B.S.; Kuh, E.S., "An Approach to the Routing of Multilayer Printed Circuit Boards," *Proceedings of the IEEE International Symposium on Circuits and Systems*, May 1978, pp.902-911.

Presents an method which is an extension of that of So [So74].

[Ti79] Ting, B.S.; Kuh, E.S.; Sangiovanni-Vincentelli, A., "Via Assignment Problem in Multilayaer Printed Circuit Boards," *IEEE Transactions on Circuits and Systems*, Vol. CAS-26, No. 4, April 1979, pp. 261-271.

Formalizes the via assignment problem resulting from the approach of So [So74]. The NP-completeness of the formalized problem is proven. Heuristics are discussed.

[Ti76] Ting, B.S.; Kuh, E.S.; Shirakawa, I., "The Multilayer Routing Problem: Algorithms and Necessary and Sufficient Conditions for the Single-Row Single-Layer Case," *IEEE Transactions on Circuits and Systems*, Vol. CAS-23, No. 12, Dec. 1976, pp. 768-778.

See entry for [So74].

[Th80] Thompson, C.D., "A Complexity Theory for VLSI," Ph.D. thesis, Department of Computer Science, Carnegie-Mellon University, 1980.

Presents a new measure of the complexity of a function in terms of the chip area and computing time of a VLSI realization of the function. This thesis has initiated a large amount of research using the measure.

[To80] Tompa, Martin, "An Optimal Solution to a Wire-Routing Problem," *Proceedings of the Twelfth Annual ACM Symposium on Theory of Computing*, 1980, pp. 161-176.

Addresses the problem of connecting terminals in order across a street, i.e. the leftmost terminal on one side is connected to the leftmost on the other side, etc.

[Ts79] Tsukiyama, S.; Kuh, E.S.; Shirakawa, I., "An Algorithm for Single-Row Routing with Prescribed Street Congestion," *Proceedings of the IEEE International Symposium on Circuits and Systems*, 1979, pp. 466-469.

See entry for [So74].

[Val75] Valiant, L., "On Non-Linear Lower Bounds in Computational Complexity," *Proceedings of the Seventh Annual ACM Symposium on Theory of Computing*, 1975, pp. 45-53.

Introduces superconcentrators.

[Val79] Valiant, L., "Universality Considerations in VLSI Circuits," results presented at the IEEE International Conference on Information Theory, Grignano, Italy, June 25-29, 1979.

[van76] van Cleemput, W.M., "Mathematical Models for the Circuit Layout Problem," *IEEE Transactions on Circuits and Systems*, Vol. CAS-23, No. 12, Dec 1976, pp. 759-767.

Contains a brief summary of various models of circuits using graphs. A model developed by the author is presented.

[van] van Cleemput, W.M. *Computer Aided Design of Digital Systems -- A Bibliography*, Computer Science Press, Inc. Vol.s I and II (1976), Vol. III (1978).

A large bibliography of papers on any topic related to design automation, organized by topics. Includes author and keyword indices. Vol. I covers publications through Dec. 1974; Vol II. covers Jan. 1975 through May 1976; Vol. III cover May 1976 through Oct. 1977. The bibliography is not annotated.

[Wi77] Williams, J.D., "STICKS -- A New Approach to LSI Design," M.S. Thesis, Dept. of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, June 1977.

Presents an program to convert hand-sketched stick diagrams of circuits into layouts which satisfy design rules. Layouts are also compacted without changing relative positions of components.

Biographical Note

The author was born on June 26, 1952 in Middletown, Connecticut, where she spent her childhood. Following her graduation from Middletown High School, she attended Cornell University, enrolling in the College of Arts and Sciences. She received her A.B. degree cum laude in physics and with distinction in all subjects from Cornell in June 1974. In the following September, she began her graduate work in the Department of Electrical Engineering and Computer Science at Massachusetts Institute of Technology. She joined the Theory of Computation research group in the Laboratory for Computer Science, where she worked under the supervision of Ronald Rivest. She completed her M.S. and E.E degrees in June 1977.

The author is married to Michael Lipkowitz. She will continue her residence in Massachusetts while joining the Computer Science faculty of Brown University as a visiting assistant professor.