ALGORITHMS FOR INTEGRATED CIRCUIT SIGNAL ROUTING

by

Alan Edward Baratz

B.S., University of California at Los Angeles
(1976)

S.M., Massachusetts Institute of Technology
(1979)

Submitted in Partial Fulfillment
of the Requirements for the Degree of

Doctor of Philosophy

at the

Massachusetts Institute of Technology

August 1981

Signature of Author [illegible]
Department of Electrical Engineering and Computer Science
August 31, 1981

Certified by Ronald L. Rivest
Ronald L. Rivest
Thesis Supervisor

Accepted by ______
Arthur C. Smith
Chairman, Department Committee

Algorithms for Integrated Circuit Signal Routing

by

Alan Edward Baratz

Submitted to the Department of Electrical Engineering and Computer Science
on August 31, 1981 in partial fulfillment of the requirements for
the Degree of Doctor of Philosophy

ABSTRACT

The integrated circuit layout problem broadly deals with the transformation of a circuit description, represented as a set of modules and a set of interconnections among these modules, into a set of conducting layer patterns that can be used to implement the circuit on a piece of silicon. If the modules are defined to be high level functional building blocks such as ALU's, PLA's, or RAM's, the layout process is typically divided into two phases; the placement phase and the routing phase. The placement phase deals with the assignment of each module to a distinct physical location on a routing plane such that no two modules overlap. The routing phase then involves the generation of a set of conducting layer patterns that can be used to implement the required interconnect.

This dissertation deals with the development of computer algorithms for the integrated circuit routing problem. We begin by describing an automated routing system, called the PI routing system, which has been developed to route custom nMOS circuit designs. The PI system employs a common two-step approach; global routing followed by region routing.

The remainder of this thesis deals with the development of "provably good" heuristic algorithms for various global routing and (rectangular) region routing problems. Most significantly, we formalize a new set of realistic wiring rules which are used to develop optimal algorithms for river routing with arbitrarily many conducting layers and near optimal algorithms for two-layer channel routing.

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

Key words: VLSI (Very Large Scale Integrated) circuit layout, integrated circuit signal routing, Steiner tree, channel routing, river routing, heuristic algorithm, algorithm analysis.

Acknowledgements

I would like to thank my thesis supervisor, Ronald Rivest, for his constant support and guidance. The opportunity to work with him has provided the single most important contribution to my graduate education. I would also like to thank my readers, Gary Miller and Christos Papadimitriou, for their direction in the course of this research. In addition, I would like to thank Ron Pinter for several interesting discussions about signal routing and for his important contributions to the original "overlap" wiring model; Charles Leiserson for suggesting Lemma 4.7; Paris Kanellakis, Andrew Moulton, and Flavio Rose for numerous enlightening discussions; and all those involved in the development of the PI system.

Finally, I am deeply grateful to all my friends and family for their understanding and support during the preparation of this thesis. I am most indebted to my wife, Raquel, for her continued faith and love throughout the past two years. Her selfless help in the preparation of the final manuscript made this thesis a reality.

Table of Contents

Chapter 1 - Introduction.

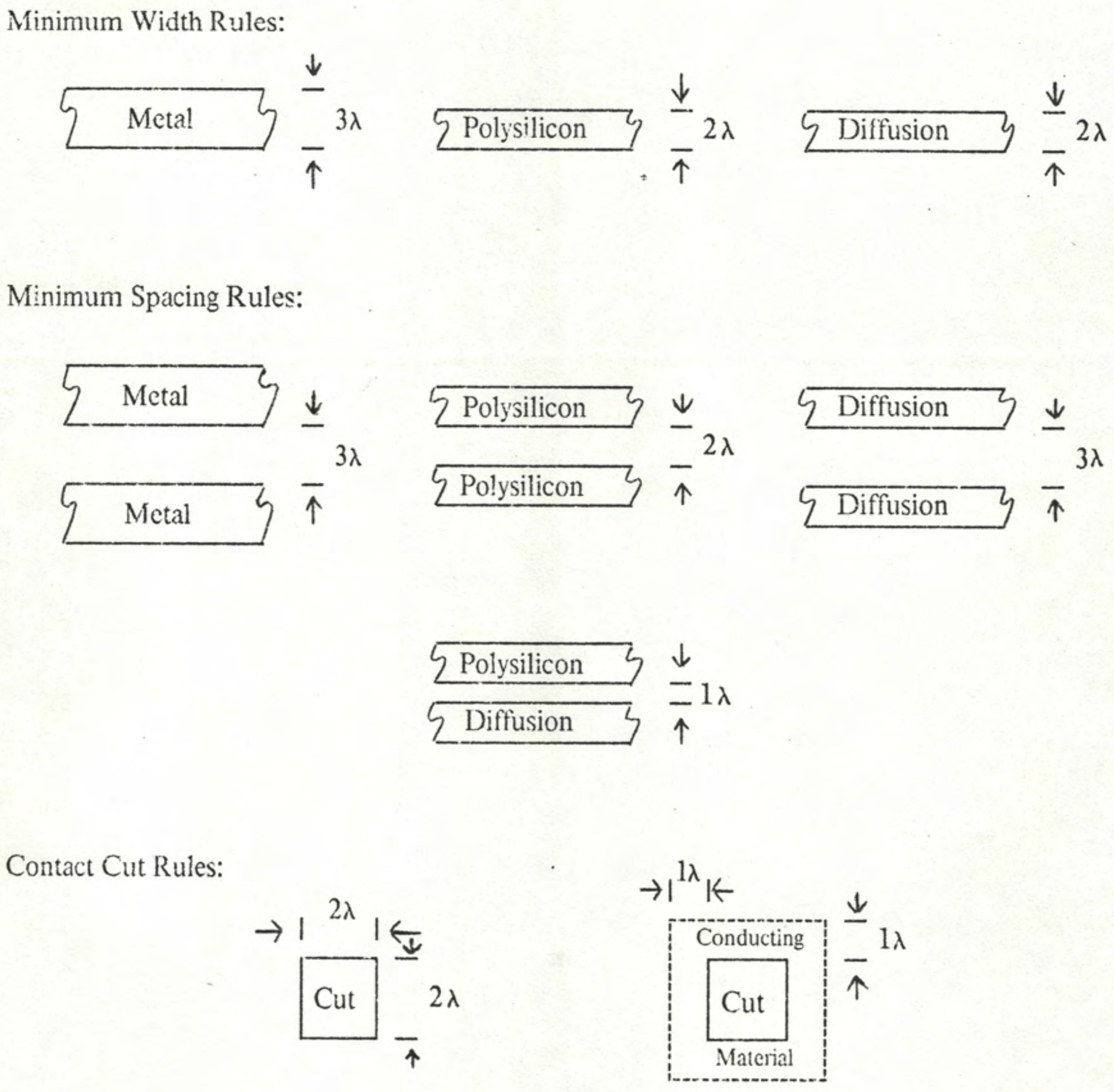
The recent development of LSI and VLSI circuit technology has greatly increased the need for automated integrated circuit (IC) design. In this thesis we shall investigate algorithms for IC layout. Integrated circuits are formed on silicon wafers by creating layers of different conducting substances (e.g., metal or polysilicon) in geometric patterns on the wafer [Mea79]. Electronic devices result from the interaction between overlapping regions (or *paths*) on different layers. *Interconnect lines* are simply paths on any one layer, between devices. The *IC layout problem* broadly deals with the transformation of a circuit description, represented as a set of modules and a set of interconnections among these modules, into a set of conducting layer patterns which can be used to implement the circuit on a piece of silicon. Such a set of geometric patterns is generally called a *circuit layout*. In an attempt to guarantee that most of the IC chips fabricated from a particular layout will satisfy all intended electrical properties, a set of *design rules* is followed in generating layouts. These design rules essentially consist of minimum width specifications for paths on each conducting layer and minimum separation specifications for pairs of paths on either the same or different conducting layers. The quality of a particular layout is measured in terms of the size and speed of the resulting circuit implementation. Obviously, minimum area and high speed are preferred.

As an example, consider the standard n-channel metal-oxide-semiconductor (nMOS) fabrication technology. Under this technology an integrated circuit is formed by creating three separate layers of conducting material on a silicon wafer. Proceeding from top to bottom, these layers are termed the *metal layer*, the *polysilicon layer*, and the *diffusion layer*. The metal layer is generally insulated from the polysilicon and diffusion layers. Thus paths on the metal layer can cross paths on the polysilicon or diffusion layers with no significant functional effect. Whenever a path on the polysilicon layer completely crosses a path on the diffusion layer, however, an nMOS *field effect transistor* is created. The only exception to these rules occurs when paths on two different layers cross a common contact cut. *Contact cuts* correspond to holes in the insulating material between conducting layers and thus they allow for electrical contact between paths on different layers. The design rules for nMOS are often extremely complex in practice. For our purposes, however, it is sufficient to consider the simplified Mead and Conway [Mea79] design rules summarized in Figure 1.1. Each of these design rules is expressed in terms of a single parameter, λ, called the *minimum feature size*. The typical minimum feature size for current technology is $\lambda = 1\mu m$.

The IC layout problem can take on vastly different characteristics under various definitions of a module and an interconnection between modules. At one level of circuit layout, for example, modules might be defined to be high level functional building blocks such as ALU's, PLA's, or RAM's. At this level we would assume that layouts had already been generated for each of the modules and we would attempt to use them to construct a layout for an entire system. At lower levels of circuit layout, the modules might be defined as single storage cells, logic gates, or even transistors. Clearly the problems encountered in laying out an entire system, represented as a set of interconnected functional building blocks, will be very different from the problems encountered in laying out a single storage cell represented as a set of interconnected transistors. In this thesis we shall be concerned with the layout of entire systems represented as sets of interconnected functional building blocks. This is generally called the *custom IC layout problem*.

The input for an instance of the custom IC layout problem consists of a set of *module specifications* and a set of *signal net definitions*. For our purposes, a *module* is specified by a rectangular bounding box with fixed dimensions and an associated set of pins. Each *pin* specifies a fixed location along the perimeter of a module, at which an interconnect line (or *wire*) can be connected. A *signal net* is simply defined to be a set of pins which are to be electrically interconnected by wires. The nets are assumed to be pairwise disjoint. The layout problem then consists of placing the modules on a routing plane and generating the various conducting layer patterns needed to implement the specified nets. Conducting paths implementing signal nets are *not* allowed to pass over regions occupied by modules.

Due to the overall complexity of the custom IC layout problem, the solution process is usually divided into two phases; the *placement phase* and the *routing phase*. In the placement phase each of the modules is assigned a physical location on a routing plane such that no two modules overlap. The goal is to place the modules in a manner which utilizes minimum total area and yet allows enough space between modules to *efficiently* generate all the required



Mead-Conway Width and Spacing Rules

Figure 1.1

interconnect lines. A comprehensive survey of automatic placement techniques can be found in [Han72]. Once the modules have been placed, the conducting layer patterns implementing each of the nets must be generated. This constitutes the routing phase. The problem of generating such a set of geometric patterns is called the *integrated circuit signal routing problem*.

The classical approach to solving the IC signal routing problem is based on partitioning a routing plane into routing regions. A routing region is an area of the routing plane through which wiring may pass. This partition is then used as a road-map of the plane; to compute a global routing for each net. A global routing for a given net is a set of routing regions through which the wires implementing the net must pass. Once a global routing has been determined for each net, the wiring required *within* each routing region is generated.

This two-step approach to signal routing was first introduced in 1966 by R. Hitchcock [Hit69]. Hitchcock developed a *cellular routing* algorithm in which the routing plane is divided into a *regular* arrangement of cells (i.e., routing regions) through which conducting paths are found. These paths are computed, one at a time, by a breadth first search of the cell structure. As each path is computed, the exact locations of the points at which the path crosses from one cell to the next are determined. However, the path topology within each cell is not specified. Once all the desired paths have been found, the intra-cellular topologies are generated.

In 1971, A. Hashimoto and J. Stevens [Has71] proposed a modification of Hitchcock's procedure in which the actual locations of inter-region wire crossings not are specified until the intra-region routings are computed. Their system also incorporates a technique called *track assignment* for intra-region routing. The track assignment algorithm places all wiring on a rectilinear grid with vertical paths routed on one conducting layer and horizontal paths routed on another conducting layer. The goal is to assign the horizontal paths to *tracks* (i.e., grid rows) such that the number of tracks containing wiring is minimized.

The global routing technique developed by Hitchcock, combined with the track assignment algorithm of Hashimoto and Stevens, provides a very powerful mechanism for automatic signal routing [Hig80]. Several IC signal routing procedures based on these ideas have been developed over the last ten years [Hig80, Lau79, Per77, Pre78]. Each such procedure involves a series of global routing computations (one for each net) followed by a series of intra-region routing computations (one for each routing region). The global routing procedures essentially involve the computation of a Steiner tree in a graph representing the region structure. (Some systems allow regions to vary in size as a function of the global routing [Pre78] and other systems require that all regions be fixed in size [Hig80].) Each intra-region routing algorithm is based on a generalization of the Hashimoto and Stevens track assignment algorithm.

In Chapter 2 we shall describe a computer automated signal routing system, called the PI routing system, that has recently been developed by a group of students under the direction of Professor R. Rivest at the Massachusetts Institute of Technology Laboratory for Computer Science. The PI system follows the standard approach of dividing the solution process into a global routing phase and an intra-region routing phase. The region structure created by the PI system is composed of fixed-size rectangular regions. The global routing phase performs successive computations of approximately minimum weight Steiner trees in a graph representing the region structure. Once a global routing has been computed for each net, the PI system assigns fixed locations to the points at which associated wires must cross from one region to the next. This assignment is performed in a global manner, attempting to assign locations such that each of the resulting intra-region routing problems is simplified. The final step then consists of generating the wiring required within each routing region. The PI system employs a library of region routing algorithms for solving these problems.

Chapter 3 of this thesis will deal with theoretical aspects of the global routing problem. We will begin by formalizing the global routing problem as the problem of computing a Steiner tree in a graph. We will then use this formalization to develop efficient "near optimal" global routing algorithms.

Finally, Chapter 4 will deal with the problem of wire routing within a given rectangular region. We will first introduce a new realistic multiple layer grid-based wiring model. We will then use this model to construct polynomial-time algorithms for the well known river routing and channel routing problems. More specifically, we will present efficient polynomial-time algorithms for river routing with arbitrarily many conducting layers and "provably good" polynomial-time heuristic algorithms for two-layer channel routing.

Chapter 2 - The PI Routing System.

In this chapter we present a detailed description of the PI routing system. The PI system is a (LISP based) automated integrated circuit signal routing system. It was developed to free the IC designer from the tedious and time consuming task of determining the interconnect geometry required to implement the signal nets of a given circuit. Thus a designer need only specify a set of modules (along with their desired physical locations within some specified rectangular layout area), specify a set of signal nets defining the interconnect required among the modules, and then invoke the PI system. The goal was to develop a system that would generate layouts with quality equal to that of good "hand-drawn" layouts. In addition, the system was designed to provide quick turnaround and employ a simple and natural user interface. It is important to note, however, that the PI system was developed under the assumption of a *single metal layer* nMOS fabrication technology. Although great care was taken to parameterize all minimum width and spacing design rules, the system is highly sensitive to the fact that only a single metal layer is available for wiring. (Credit for the PI system must be given to a large number of people who were involved in its development: R. Rivest, A. Baratz, J. Batali, S. Bhatt, J. Byrd, D. Christman, C. Frank, A. Ghaznavi, A. Hanover, J. Koschella, E. Mayr, A. Moulton, R. Pinter, R. Rao, F. Rose, G. Roylance, S. Sur, A. Vijayvargia.)

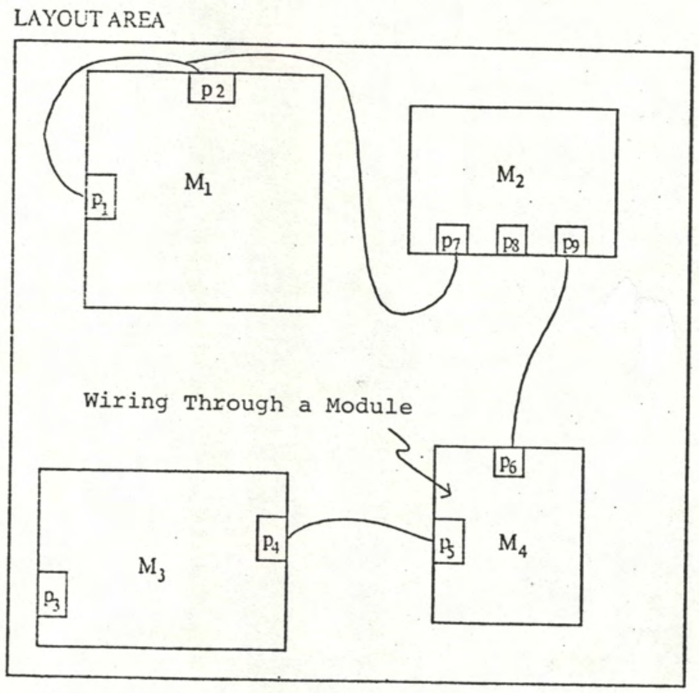
2.1 PI System Input.

We shall now define the structure of input to the PI system. In the process, we introduce some basic terminology that will be useful in describing the individual procedures comprising the system.

Input to the PI routing system consists of a desired rectangular bounding box for the completed layout, a set of *module specifications*, and a set of *signal net definitions*. Each *module* is specified by a rectangular bounding box with fixed dimensions and fixed location within the specified layout area. All rectangular regions are assumed to have orientation parallel to the *x*-*y* axes and thus their size, shape and location can be easily specified by giving the coordinates of

Pins:

p_1-metal
p_2-polysilicon
p_3-metal
p_4-metal
p_5-diffusion
p_6-metal
p_7-metal
p_8-metal
p_9-polysilicon



Terminals: $T_1=\{p_1\}$, $T_2=\{p_2\}$, $T_3=\{p_3,p_4\}$, $T_4=\{p_5,p_6\}$ $T_5=\{p_7,p_8\}$, $T_6=\{p_9\}$.

Nets: $N_1=\{T_1,T_2,T_5\}$, $N_2=\{T_3,T_4,T_6\}$.

Module Specifications and Signal Net Definitions

Figure 2.1

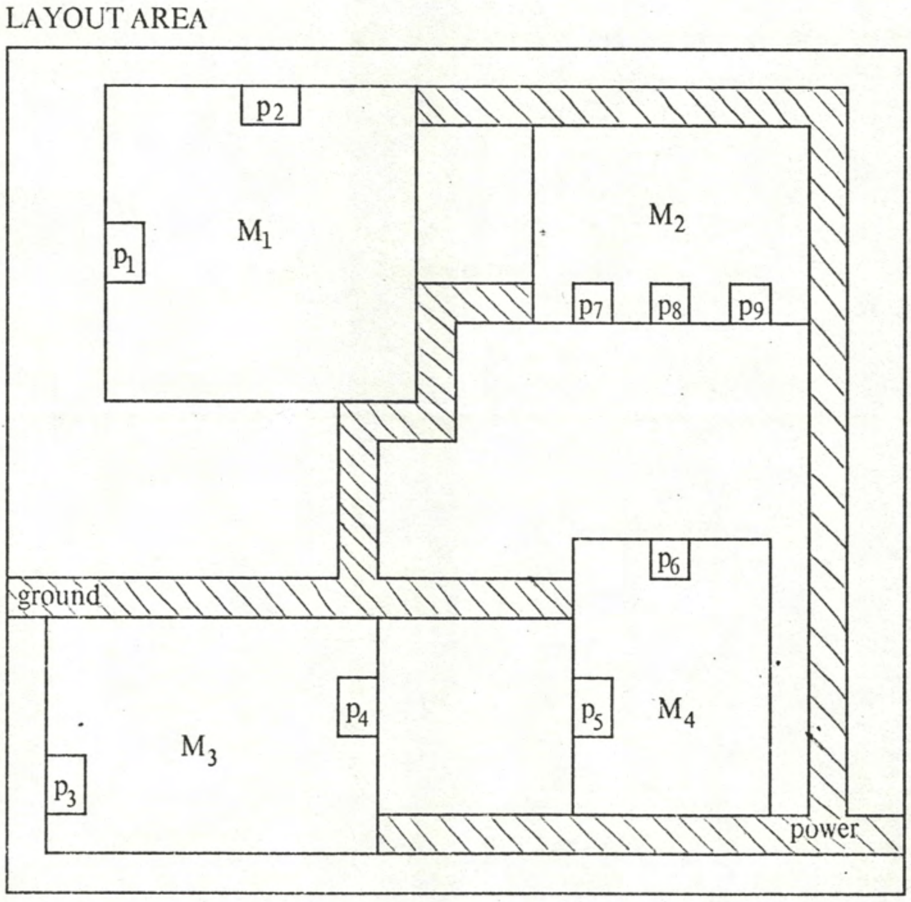
their lower-left and upper-right corners. Each module also has an associated set of physical connection points called pins. A *pin* is simply a rectangular region of fixed size and location adjacent to the perimeter of a module. Each pin also has an associated conducting layer. Thus a pin specifies an area within a module at which a wire on a particular conducting layer can be connected. The pins associated with any given module are then grouped into a set of logical connection points called terminals. A *terminal* is simply a set of pins on a module which are all electrically interconnected within the module (i.e., electrically equivalent). In practice, a terminal will commonly contain only one pin. A set of module specifications is depicted in Figure 2.1.

A *signal net* is now simply defined to be a set of terminals which must be interconnected by wires. Making a connection to a terminal simply requires making a connection to any one of the pins belonging to the terminal. Notice that this input structure allows electrically equivalent pins to be selected for interconnect as a function of an optimal routing. In addition, it allows the possibility of *wiring through* a module as illustrated in Figure 2.1.

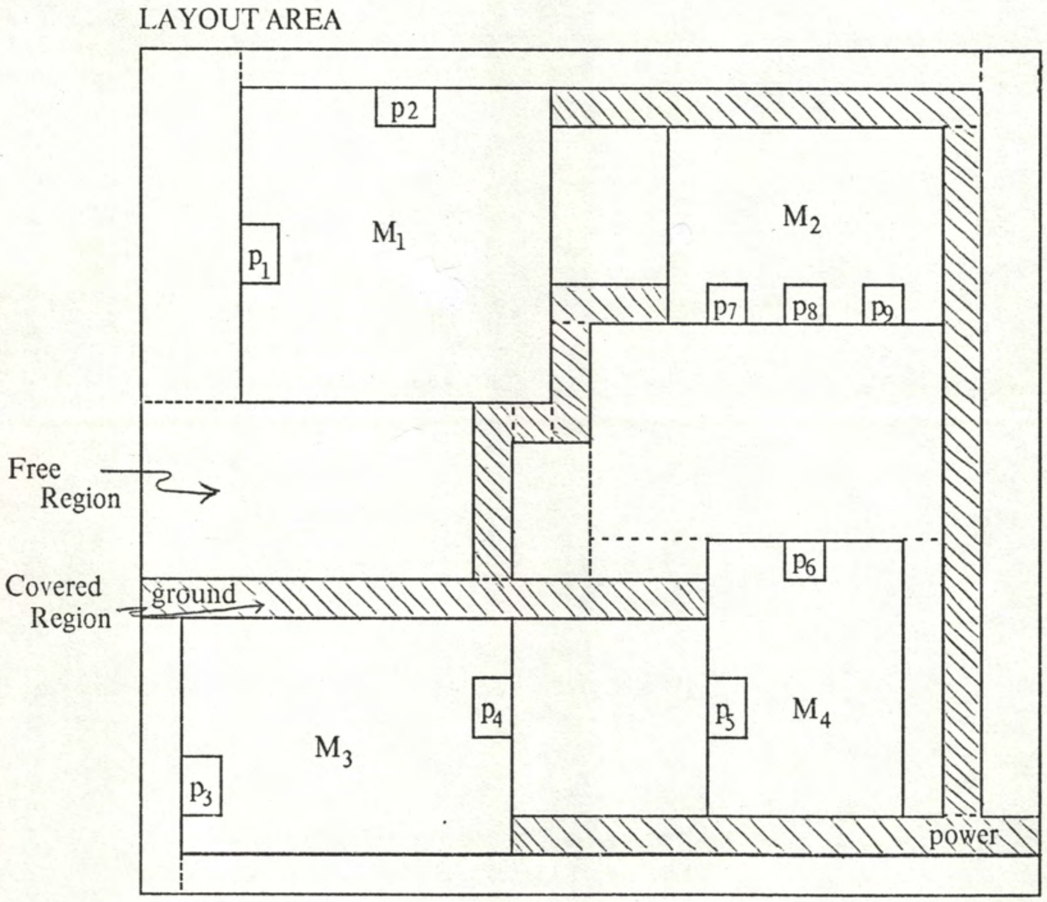
Finally, we should mention that the current implementation of the PI system does not handle the routing of power and ground nets. Thus these nets must be hand-wired and specified as part of the input to the system. The input of a power or ground net simply consists of a set of specifications (size, shape, and location) of metal layer rectangular regions. A simple PI system input example is depicted in Figure 2.2. We are now ready to describe the overall organization of the system.

2.2 PI System Organization.

In this section we present a brief overview of the PI routing system and each of its individual procedures. The PI routing system is organized into four phases; *routing region definition*, *global routing*, *crossing placement*, and *intra-region routing*. In the first phase (routing region definition), the layout area outside the modules is divided into non-overlapping rectangular *routing regions* as shown in Figure 2.3. This division is performed such that each resulting region is either disjoint from the area occupied by the power and ground rectangles or



PI System Input Diagram
Figure 2.2



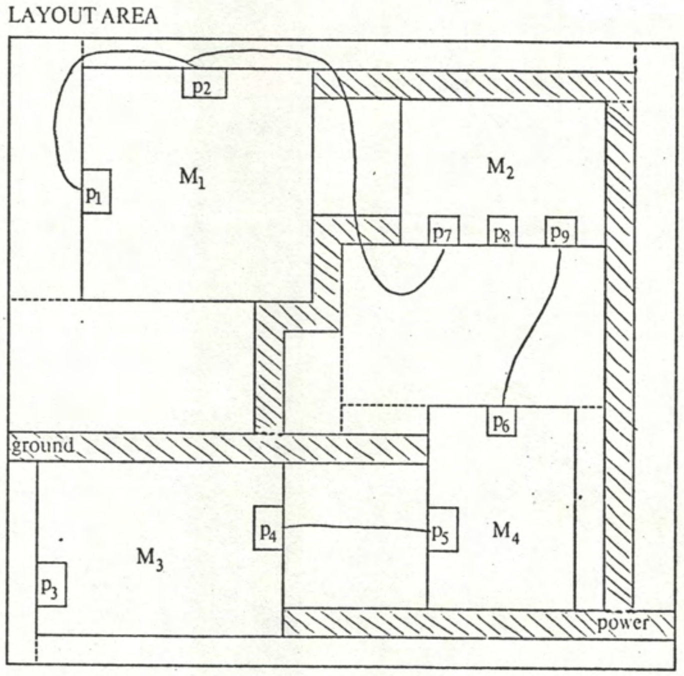
Routing Region Definition
Figure 2.3

contained within the area occupied by the power and ground rectangles. Routing regions of the former type are called *free regions* and those of the latter type are called *covered regions*. Each routing region defines a portion of the layout area through which wiring may pass. Notice that wiring located within a covered region must not lie on the metal layer.

The set of routing regions forms a road-map of the layout area which can be used to determine a global routing for each net. This global routing process comprises the second phase of the PI system and simply consists of assigning each net to the routing regions through which its implementation must pass. The goal is to route the nets in a manner which minimizes some estimate of total interconnect length and at the same time avoids over-congestion in any routing region. When the global routing phase has been completed, each net will have been assigned to a set of routing regions and thus each routing region will have been assigned a set of nets. We will then know for each routing region which wires must pass through the region and which region-to-region border edges these wires must cross. Notice that the exact locations at which wires cross region-to-region edges is not yet specified (see Figure 2.4).

The third phase of the PI system (crossing placement) then determines a fixed location for the point at which each wire must cross a region-to-region edge. The goal is to place the crossings so that the remaining wiring problem within each region will be as simple as possible. More specifically, the crossing locations are determined in a manner which attempts to minimize the number of crossovers that will eventually be required between electrically distinct conducting paths. In addition, an attempt is made to place crossings so that total wire length is minimized and the number of straight-across runs is maximized.

The result of crossing placement will be a decomposition of the remaining problem into a set of well defined subproblems. Each subproblem will consist of a rectangular routing region with fixed dimensions and a set of physical connection points with fixed locations along the perimeter of the region. In addition, the physical connection points will be partitioned into disjoint subsets that are to be interconnected (see Figure 2.5). The remaining problem is to determine the interconnect geometry required within each rectangular region. The goal is to maximize usage of the metal conducting layer while minimizing total path length. This constitutes the fourth and final phase of the PI routing system.

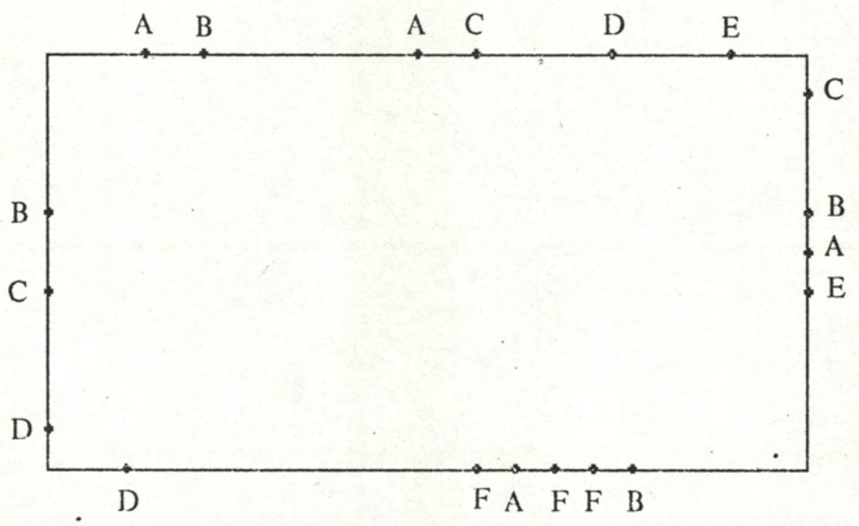


Terminals: $T_1=\{p_1\}$, $T_2=\{p_2\}$, $T_3=\{p_3,p_4\}$, $T_4=\{p_5,p_6\}$, $T_5=\{p_7,p_7\}$, $T_6=\{p_9\}$

Nets: $N_1=\{T_1,T_2,T_5\}$, $N_2=\{T_3,T_4,T_6\}$.

Global Routing
Figure 2.4

Routing Region



Partition: {A,B,C,D,E,F}

Intra-Region Routing Problem
Figure 2.5

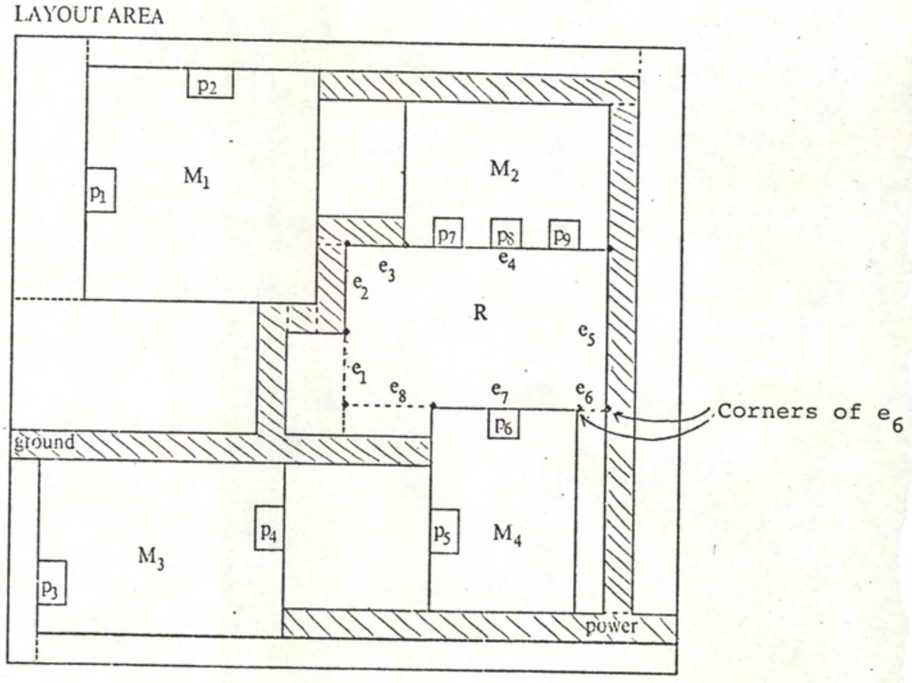
If the PI routing system should fail at any point during execution, the designer must modify the input placement to allow more wiring area and then restart the system. In practice, several such interations may be required. (We note that algorithms are currently being implemented to automate this feedback loop.)

Finally, we should mention that the PI system uses *geometric hashing* as a general technique for organizing local searches: The entire layout area is divided into square sections currently of size 70λ (in Mead-Conway [Mea79] units) on a side. For each section there is a data structure listing all the objects of a given type (module, routing region, crossing, etc...) which lie within that section. This technique is invisible to the user, but provides for fast searches for objects in the neighborhood of other objects.

The remaining sections will constitute a detailed description of the procedures implementing each of the four routing phases.

2.3 Routing Region Definition.

The first phase of the PI routing system involves decomposing an input layout area into disjoint rectangular regions such that each region is either a free routing region, a covered routing region, or a module. This decomposition is defined by a set of *edges* that separate the routing regions from each other and from the modules. Each edge is a horizontal or vertical line segment which borders no more than two distinct routing regions or modules. The endpoints of an edge are called *corners*. Two distinct edges can intersect only at a corner and thus a corner can be the endpoint of at most four distinct edges; two vertical and two horizontal. A corner is said to be *exposed* if it is the endpoint of exactly one vertical edge and exactly one horizontal edge (neither of which lies on the boundary of the layout area). Figure 2.6 illustrates the fact that a routing region may be enclosed by an arbitrary number (greater than three) of edges.



Routing Region R Is Defined By 8 Edges:

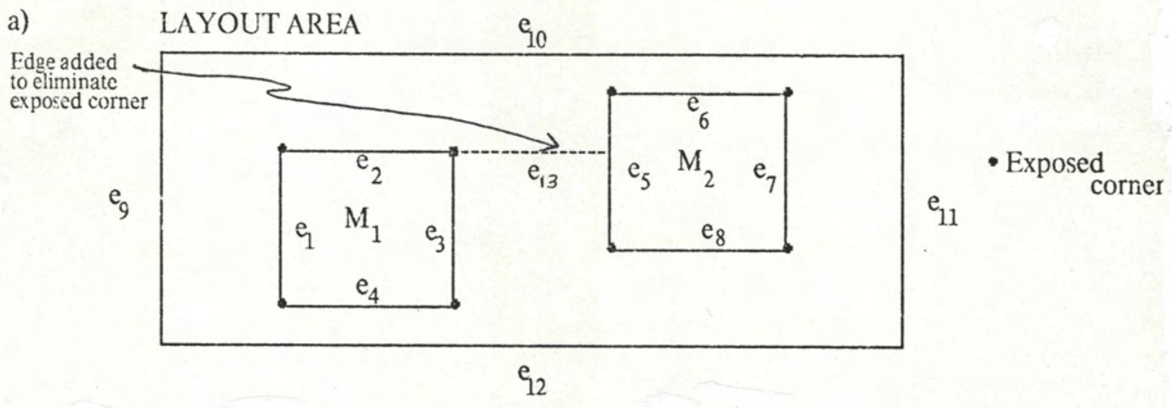
$e_1, e_2, e_3, e_4, e_5, e_6, e_7, e_8$

Examples of Edges and Corners
Figure 2.6

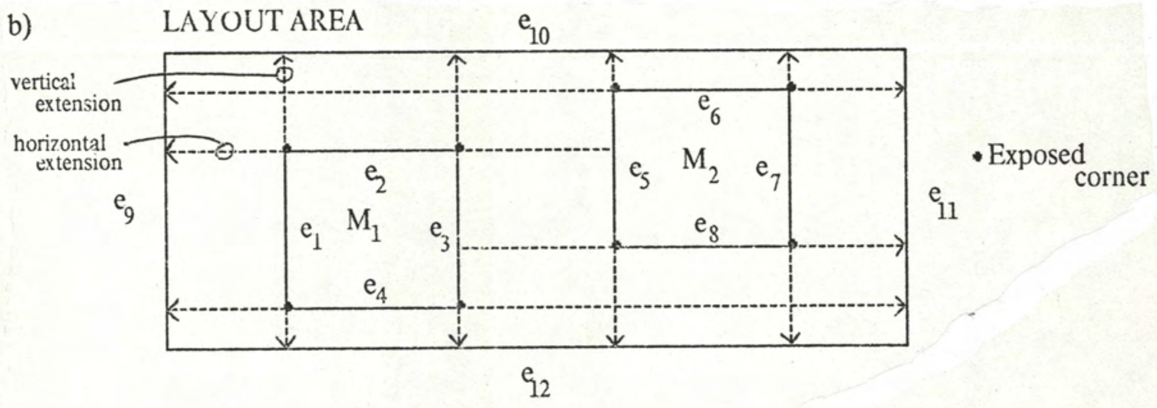
Since the time complexity of the global routing phase of the PI system is directly related to the number of routing regions composing the layout area, we use a decomposition algorithm which produces large "natural looking" routing regions (a criterion initially proposed by F. Rose). This algorithm basically attempts to minimize the total length of edges defining the routing regions.

The routing region definition algorithm, developed by R. Rivest and R. Rao, is a simple "greedy" type algorithm based on the observation that every exposed corner must be eliminated by the addition of at least one incident edge as shown in Figure 2.7a. The edge set is initialized to contain exactly those line segments defining the boundary of the total layout area, the boundaries of the modules, and the boundaries of the power and ground regions. Notice that each module gives rise to four exposed corners. The algorithm then proceeds by generating for each exposed corner both a horizontal and a vertical extension from that corner outward to the first point of intersection with an existing edge (see Figure 2.7b). The shortest of all these extensions is then added to the edge set. Let a and b denote the corners of this new edge. Clearly each of these corners must be located on at least one other edge; edge e_a and edge e_b respectively. Now if a is not located at an endpoint of e_a, then edge e_a is replaced by two new edges having a as a common corner as shown in Figure 2.7c. Similarly, if b is not located at an endpoint of e_b, then edge e_b is replaced by two new edges having b as a common corner. (Notice that at most one of these two cases can occur.) This entire process is then iterated on the remaining exposed corners.

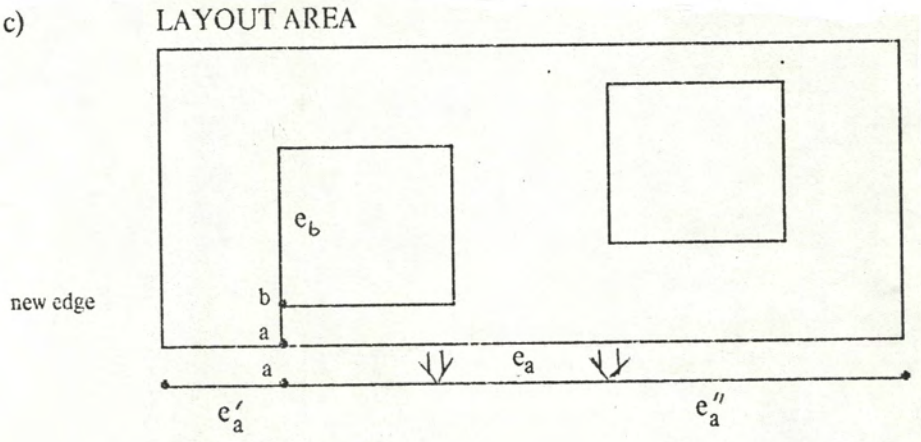
It should now be noted that this algorithm is only a heuristic for computing a decomposition with minimum total edge length. In fact, there are examples for which this algorithm will compute a decomposition with total edge length much larger than the minimum possible (see Figure 2.8.) Nonetheless, the algorithm is extremely efficient and tends to produce very natural looking routing regions. For completeness we remark that the minimum length decomposition problem has not been proven NP-complete and yet there exists no known polynomial time algorithm for solving it.



a)
LAYOUT AREA
Edge added to eliminate exposed corner
e_{10}
e_6
M_2
e_5
e_7
e_8
e_2
e_{13}
e_1
M_1
e_3
e_4
e_9
e_{11}
e_{12}
• Exposed corner

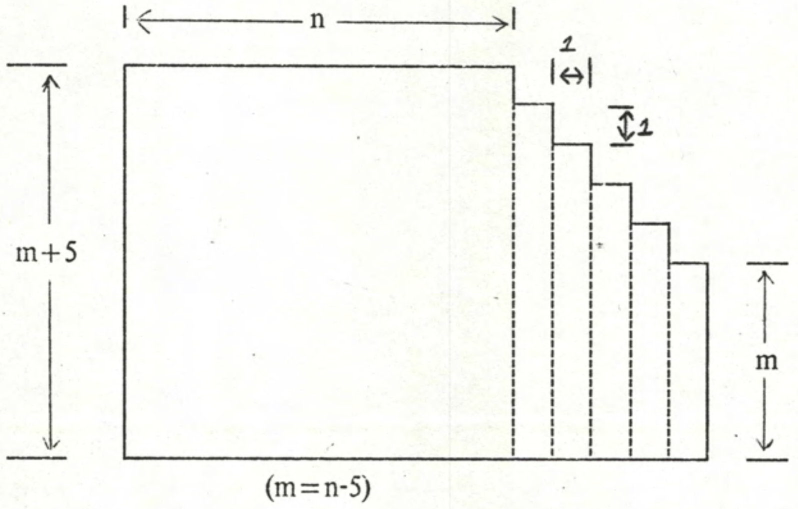


b)
LAYOUT AREA
e_{10}
vertical extension
horizontal extension
e_6
M_2
e_5
e_7
e_8
e_2
e_1
M_1
e_3
e_4
e_9
e_{11}
e_{12}
• Exposed corner

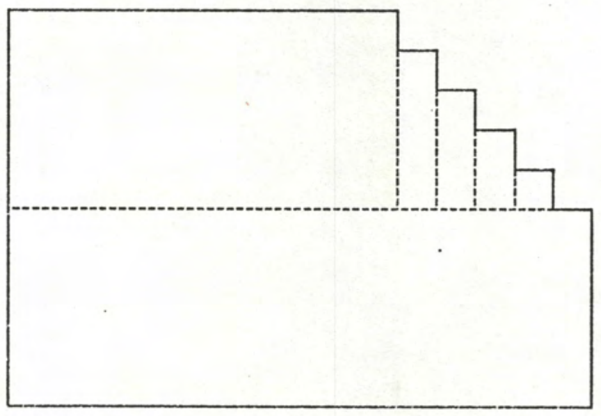


c)
LAYOUT AREA
e_b
new edge
b
a
a
e_a
e'_a
e''_a

Figure 2.7



Decomposition Computed by Heuristic: Length of Added Edges=5m+10



Alternate Decomposition: Length of Added Edges=n+14=m+19

Pathological Example for Region Decomposition Algorithm
Figure 2.8

Finally, the last step in the routing region definition phase consists of identifying all *narrow* routing regions (i.e., those which are so narrow that wires can only run straight across the region) and then refining the decomposition so that all covered and narrow routing regions are enclosed by exactly four edges. This refinement is accomplished in four scans across the layout area; one from left to right, one from right to left, one from top to bottom, and one from bottom to top. During the first scan (left-to-right), the algorithm searches for any corner a, located on the left side of a covered or narrow routing region r, such that there are two vertical edges incident on a and bordering r. When such a corner is found, the algorithm creates a new (horizontal) edge with corners a and b such that b is located on a vertical edge e bordering the right side of r. If b is not located at an endpoint of e, then the edge e is replaced by two new edges having b as a common corner. Thus the region r is divided into two distinct routing regions. Once this division is complete, the algorithm continues its left-to-right scan, searching for another corner with the specified properties. The remaining three scans are then performed in an analogous manner. We now note that this refinement is performed to simplify the global routing phase of the system.

2.4 Global Routing.

The global routing phase of the PI system utilizes the previously defined region structure to compute a global routing for each net. This computation simply involves the assignment of each net to the set of routing regions that its corresponding wires will traverse. The set of all edges that will be crossed by the wires implementing a given net is concurrently recorded. The data structure recording that the implementation of a net will cross a given edge is called a *crossing*. In the global routing phase the actual position at which a wire will cross an edge is unspecified. This location is determined later, in the crossing placement phase.

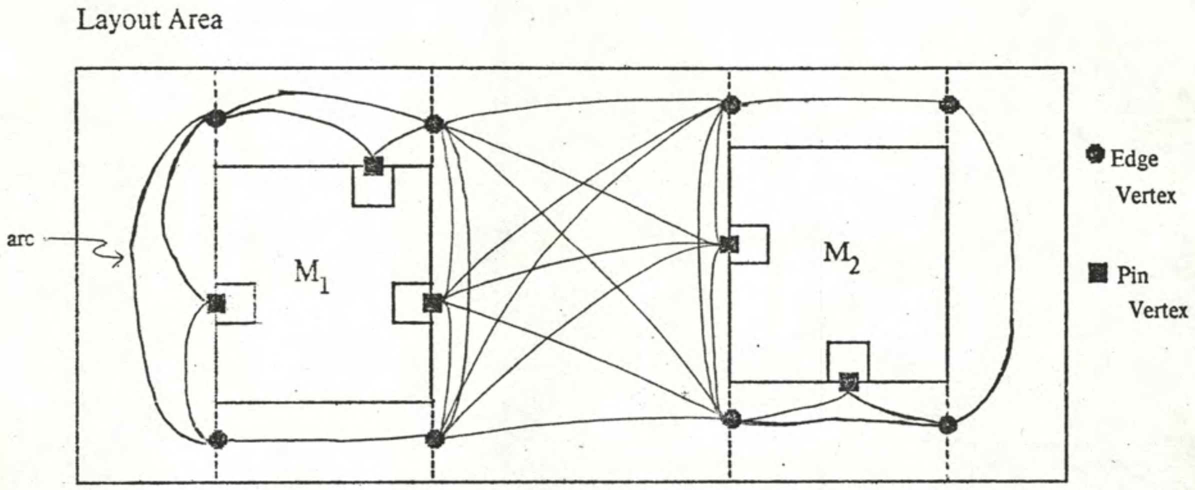
A maximal connected portion of the interconnect lines implementing a given net, which lies entirely within a given routing region, is called a *strand* of that net. It is possible for a given net to be represented by more than one strand within a given routing region. The data structure representing a strand contains a pointer to the net of the strand, a pointer to the routing region containing the strand, and a list of crossings representing points on the perimeter of the associated routing region that are to be interconnected by the strand. If a strand lists more than two crossings, the corresponding interconnect geometry will form a tree-like pattern.

The global routing phase processes the nets sequentially, roughly in order of increasing net interconnect length. (Actually, the nets are processed in order according to the length of the perimeter of the bounding box of the net's pins, divided by the number of terminals comprising the net; so a multi-terminal net may be routed early.) The processing of any given net then proceeds by creating a graph structure representing the search to be performed. The vertices of this graph are the mid-points of all the region-to-region edges plus points representing the pins of the net currently being processed. Each pin is generally represented by the mid-point of its side adjacent to the perimeter of the module on which it lies. (Recall that a pin is a rectangular region located within a module.) Two vertices are adjacent (i.e., connected by an arc) in the graph if and only if they lie on a common routing region (see Figure 2.9). A distance function which attempts to estimate the amount of wire needed to connect two adjacent vertices is then defined on the arcs in the graph. More precisely, the distance associated with an arc in the graph is defined to be the Manhattan distance between its endpoints plus penalty values for turning corners (i.e., *jogging*), traversing a covered routing region, or reaching a vertex representing an edge that is nearly *full* of crossings. (Recall that each edge has fixed length and thus there is an upper bound on the number of wires that can cross it.) In the current implementation these penalty values are defined as follows:

1) Each required jog (i.e., direction change) results in a penalty of 5λ.

2) Traversing a covered routing region results in a penalty of 5λ times the length of the traversal.

3) Reaching a vertex representing an edge that is nearly full of crossings results in a penalty of $A/4^f$, where A denotes one-half the length of the perimeter of the layout area and f denotes the number of wires that can still cross the edge.



The Global Routing Graph Structure
Figure 2.9

Once the graph structure has been generated for a net, a heuristic algorithm is employed to determine an approximately minimum weight connected subgraph which contains a set of vertices representing at least one pin from each terminal of the net. The arcs of this subgraph will correspond to the routing regions to be traversed by the wires implementing the net and the vertices of this subgraph will correspond to the edges to be crossed by these wires. Notice that the distance function is defined such that this algorithm attempts to compute a global routing which minimizes interconnect length and avoids congestion. The details of the heuristic algorithm for computing an approximately minimum weight connected subgraph will be presented in Chapter 3.

At the completion of the global routing phase, the PI system will have determined for each net:

1) A set of routing regions through which the interconnect lines implementing the net must pass.
2) A set of crossings recording the edges which these interconnect lines must cross.
3) A set of strands specifying the crossings representing points which must be interconnected within each region.

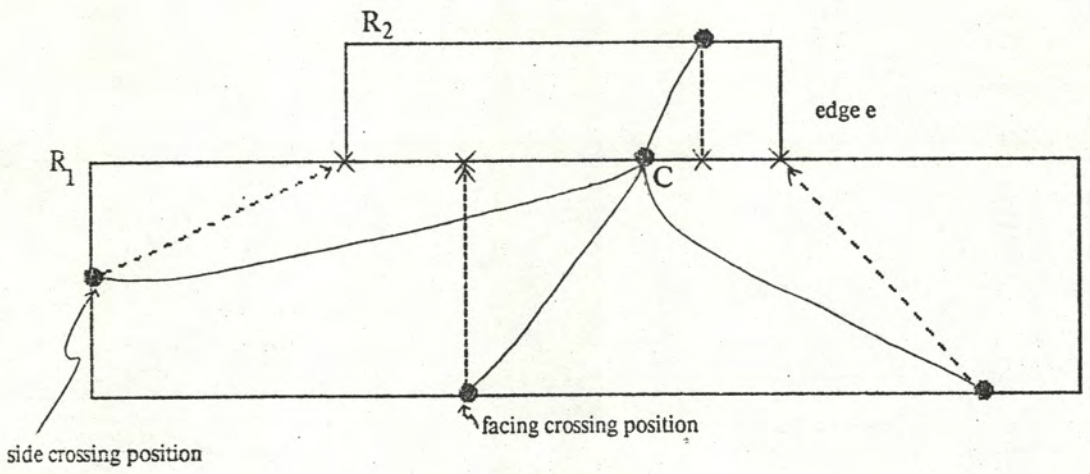
Recall, however, that as yet there is no specific location associated with a crossing. This is the task of the next phase.

2.5 Crossing Placement.

The crossing placement phase of the PI routing system assigns a fixed location to each crossing. The basic approach is to organize the crossings in a *global* manner, attempting to generate crossing placements that will maximize the probability of success during the intra-region routing phase. An important advantage of this explicit crossing placement phase is that it cleanly decomposes the remaining problem into a set of independent region routing problems.

Each edge of the routing region structure initially contains a number of crossings to be placed. A crossing representing an edge enclosing a module is associated with a pin. Thus such crossings are assigned the location of the mid-point of the side of the pin adjacent to the associated edge. All other crossings (i.e., crossings representing edges separating two routing regions) are called *floating* crossings. Each floating crossing belongs to exactly two distinct strands: one for each routing region adjacent to the edge associated with the crossing.

The procedure used to place the floating crossings is an iterative relaxation method, developed by R. Rivest, where each edge is considered in turn and the crossing positions on that edge are adjusted. Initially each floating crossing is assigned the location of the mid-point of the edge with which it is associated. At each step a new position for a crossing c on an edge e is computed as the weighted average of the old position of c on e, the mid-point of e, and the projections onto edge e of the current positions of the other crossings belonging to the strands containing c. The projections of crossing positions which are closer to the old position of crossing c on edge e are weighted heavier than the projections of crossing positions farther away. In addition, the projections of crossing positions *facing* the edge e are weighted heavier than the projections of crossing positions off to the side of e (see Figure 2.10).



Crossing Placement
Figure 2.10

After a new position is computed for each crossing on an edge, the positions are *discretized* so that they lie on a global grid whose spacing equals the standard *track-center-to-track-center* distance (currently set to 7λ). This discretization process simply involves adjusting each crossing position in turn, moving it to the closest unoccupied grid position. The crossings are adjusted in order of decreasing *placement force*, where the force of a placement is simply the total weight involved in computing the current crossing position.

The only exception to this crossing placement procedure occurs when two crossings belong to a common strand across a covered or narrow channel. In this case the two crossings are treated as a unit for the purposes of crossing placement. Thus we are guaranteed that such crossings will always be assigned location directly opposite each other.

Finally, this simple crossing placement procedure is iterated over all edges several times. The result is a crossing placement which tends to yield instances of the intra-region routing problem requiring a minimum amount of jogging and wire crossover.

2.6 Intra-Region Routing.

When the PI system begins the intra-region routing phase, all that remains is to solve a set of well-defined independent *intra-region routing problems*. Each such problem will consist of a rectangular region with fixed dimensions and a set of physical connection points (i.e., crossing positions) with fixed locations along the perimeter of the region. In addition, a set of strands will specify wiring required internal to the region. The problem then consists of generating a layout for the routing region which implements all the required interconnect. Care must be taken, however, not to route conducting paths too close to the border of the region as this might interfere with the wiring in an adjacent routing region. The only exception occurs when a wire is running to the edge of a region to make a connection to a crossing position.

The PI system takes a somewhat novel approach to the intra-region routing problem by employing a *library* of region routing procedures, each with different philosophies and capabilities. Currently two such routers have been implemented: a "quick and dirty" router and a slower, more sophisticated router. Empirically, the "quick router" succeeds on approximately 80% of the routing regions. Whenever the quick router fails, we depend on the "powerful" router to solve the problem.

Before performing the region routing, each region is assigned a *preferred direction* (vertical or horizontal) which indicates the direction in which the metal layer wiring will be run; wires running in the other direction will be on the polysilicon or diffusion layer whenever they cross a metal wire. The preferred direction is chosen in a way which attempts to maximize the amount of metal layer wiring that will be generated.

The quick region router employs a three step solution process. In the first step a simple set of heuristics is used to determine the basic geometry of the desired layout. No consideration is given to conducting layers or design rules at this point. The goal is simply to generate a set of geometric patterns that will interconnect the crossing positions specified by each strand. The second step of the quick router then consists of assigning a conducting layer to each interconnect path. This layer assignment is performed such that all paths running in the preferred direction are assigned the metal layer and paths running in the other direction are assigned the metal layer whenever possible. The final step performed by the quick router is a complete design rule check of the layout generated. If the layout passes the design rule check, the region is marked as routed and the system moves on to another region. If the layout fails the design rule check, it is passed on to the powerful router. The details of the quick router will be presented in Chapter 4.

The powerful region router, developed by D. Christman and J. Koschella, uses a coordinate transformation so it can *view* the region with the preferred direction running from left to right. The basic components of this router are:

1) A *jiggler* which can be used to adjust the track positions of the wires as they enter on the left or right.

2) A recursive *slice router* that successively wires a sequence of vertical *slices* into which the region is divided. A given slice may contain connections to crossing positions on the top and/or bottom of the region.

The approach is then to route the region in a left to right manner. The details of this procedure are presented in [Kos81]. In addition, several algorithms with the same flavor will be discussed in Chapter 4.

2.7 Summary.

The PI routing system is a four phase integrated circuit signal routing system. The basic approach taken by PI is a fairly common one in which the routing problem is solved by first generating a global routing for each signal net and then generating a set of small detailed local routings for various regions of the layout area. However, the PI system is unique in that the data structure representing the layout area consists of a set of *fixed* two-dimensional rectangular routing regions. Thus we always know the exact size and shape of each routing region. Furthermore, we always know the exact length of any edge separating two routing regions. This provides a greater ability to avoid congestion in any one area of the layout.

Another notable ingredient of the PI system is the separate phase for determining the exact positions at which wires must cross edges. This results in a completely independent set of region routing problems and thus it simplifies the last phase of the system.

The final important ingredient of the PI system is the notion of a library of region routers. The goal is to develop a set of very efficient routers that each have a high probability of success on a specific type of region routing problem. The proper router can then be selected for any given problem.

In conclusion, empirical results show that the first two phases of the PI system (i.e., region routing definition and global routing) produce fairly high quality results. The iterative improvement algorithm implementing the crossing placement phase, however, is much too sensitive to the values assigned to various parameters in the weighting function. It is believed that a better approach would be one which is more combinatorial in nature. The goal would then simply be to place crossings such that the number of required wire crossovers is minimized. Finally, the library approach to region routing seems to be a great time saver. The real success of this approach, however, is dependent upon the development of a comprehensive theory for region routing.

Chapter 3 - Global Routing.

The most common approach to computing a global routing for a given net consists of searching a graph structure representing all existing paths across a layout area. The edges of this graph are generally assigned weights so that the weight of a path in the graph corresponds to the quality of a conducting path through the layout area. For each net, a global routing is generated by computing an approximately minimum weight connected subgraph containing the vertices corresponding to the pins for the net. Such a subgraph is called a *Steiner tree* [Gar79].

In this chapter we begin by formalizing the notion of a Steiner tree and the problem of computing an optimal Steiner tree in a graph. We then describe and analyze two "provably good" heuristic Steiner tree algorithms. The first of these algorithms, due to C. El-Arbi [EA78], is guaranteed to compute a solution having total weight no more than twice optimal. The second algorithm has the interesting property that its performance is related to the number of Steiner vertices in an optimal solution to the given problem. Finally, we present a description of the Steiner tree algorithm used in the PI system. This algorithm is essentially an efficient implementation of the algorithm developed by El-Arbi.

3.1 The Steiner Tree Problem: A Model for Global Routing.

In this section we shall formalize the notion of a Steiner tree and the problem of computing an optimal Steiner tree in a graph. We begin by defining some standard graph-theoretic notation. (The informed reader may wish to proceed directly to Definition 3.5.)

Definition 3.1:

An *undirected weighted graph* $G=(V,E,w)$ consists of a finite set of *vertices* $V=\{v_1,\ldots,v_n\}$, a set of *edges* $E\subseteq\{\{v_i,v_j\}\mid v_i,v_j\in V \text{ and } v_i\neq v_j\}$, and a *weight function* $w{:}E\rightarrow\mathbb{R}$ which maps E into the set of non-negative real numbers. Two vertices $v_i\in V$ and $v_j\in V$ are said to be *adjacent* in G if $\{v_i,v_j\}\in E$. Further, for any vertex $v_i\in V$ we define the *degree* of v_i in G, denoted $deg_G(v_i)$, to be the number of distinct vertices adjacent to v_i in G:

$$deg_G(v_i)=|\{\{v_i,v_j\}\mid\{v_i,v_j\}\in E\}|.$$

For any undirected weighted graph $G=(V,E,w)$, if $e=\{u,v\}$ is an edge in E, then e is said to *connect* u and v in G. Furthermore, u and v are called the *endpoints* of e in G.

Definition 3.2:

Let $G=(V,E,w)$ denote an undirected weighted graph. A *path* p, from vertex u_1 to vertex u_k in G, is a sequence of distinct vertices and edges $p=u_1e_1u_2e_2\ldots u_{k-1}e_{k-1}u_k$ $(k\geq 2)$ such that $u_i\in V$ for each $1\leq i\leq k$, $e_i\in E$ for each $1\leq i\leq k\text{-}1$, and $e_i=\{u_i,u_{i+1}\}$ for each $1\leq i\leq k\text{-}1$. The vertices u_1 and u_k are called the endpoints of p and p is said to connect u_1 and u_k in G. The *weight* of a path p in G is defined to be:

$$W(p)=\Sigma\{w(e)\mid e\in E \text{ is an edge in } p\}.$$

Definition 3.3:

Let $G=(V,E,w)$ denote an undirected weighted graph. A *tree* $T=(V_T\subseteq V,\ E_T\subseteq E)$ in G is a connected subgraph of G such that the removal of any edge in the subgraph will leave it disconnected. Any vertex $v\in V_T$ such that $deg_T(v)=1$ is called a *leaf* in T. Further, for any $V'\subseteq V$ we say that T *spans* V' in G if $V'\subseteq V_T$. A tree which spans V in G is called a *spanning tree* of G. Finally, the weight of a tree T is defined to be:

$$W(T)=\Sigma\{w(e)\mid e\in E_T\}.$$

Occasionally we will speak about a path in a tree or the weight of a path in a tree. A path in a tree is simply a path in the underlying graph composed entirely of vertices and edges in the tree. Similarly, the weight of a path in a tree is defined relative to the weight function in the underlying graph. We are now ready to formally define the notion of a Steiner tree and the problem of computing an optimal Steiner tree in a graph.

Definition 3.4:

Let $G=(V,E,w)$ denote an undirected weighted graph and let $D\subseteq V$ denote any subset of the vertices in G with $|D|>1$. A *Steiner tree* T_D with respect to D in G is simply a tree in G that spans D and has leaves from only the set D. A *Steiner vertex* in T_D is then defined to be any vertex v in T_D such that $deg_{T_D}(v)>2$ and $v\notin D$. Finally, a *minimum weight Steiner tree* with respect to D in G is defined to be any Steiner tree with respect to D in G whose weight is minimum among all such Steiner trees.

Definition 3.5:

Let $G=(V,E,w)$ denote an undirected weighted graph, let $D \subseteq V$ denote any subset of the vertices in G with $|D|>1$, and let T_D denote a Steiner tree with respect to D in G. Further, let p' denote any path from a vertex u_1 to a vertex u_k in T_D such that each of u_1 and u_k is either a Steiner vertex in T_D or an element of the set D and every other vertex u' in p' satisfies the constraints $deg_{T_D}(u')=2$ and $u' \notin D$. Then the path p' is called a *basic path* in T_D.

Figure 3.1 illustrates the notions of a Steiner tree, a basic path in a Steiner tree, and a Steiner vertex in a Steiner tree. An instance of the Steiner tree problem can now be easily defined as follows.

Definition 3.6:

An instance of the *Steiner tree problem* is simply an ordered pair, $STP=(G,D)$, in which:

1) $G=(V,E,w)$ denotes an undirected weighted graph.

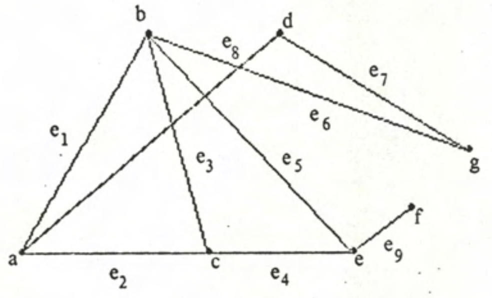
2) $D \subseteq V$ denotes a subset, with $|D|>1$, of the vertices in G.

Each vertex $d \in D$ is called a *distinguished vertex* in G. An *optimal solution* to such a problem instance consists of a minimum weight Steiner tree with respect to D in G.

The problem of computing an optimal solution to any Steiner tree problem given as input is known in the literature as the *Graph Steiner Tree Problem* [Gar79]. This problem has been shown to be NP-hard [Gar79] and thus there is little hope of developing an efficient (i.e., polynomial-time) algorithm for solving it. In the next section we will discuss two efficient heuristic algorithms for computing a "near optimal" solution to a Steiner tree problem. Before describing these algorithms we require the following definition.

Definition 3.7:

Let $G=(V,E,w)$ denote an undirected weighted graph and let $D \subseteq V$ denote any subset of the vertices in G. The *path graph* for D in G is then defined to be the undirected weighted graph $G_D=(D,E_D,w_D)$ such that:



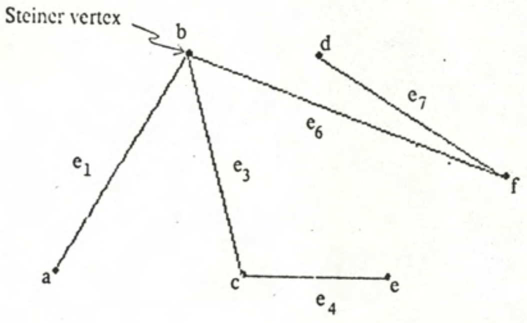
G=(V,E,w), where

$V=\{a,b,c,d,e,f,g\}$

$E=\{e_1,e_2,e_3,e_4,e_5,e_6,e_7,e_8,e_9\}$

$w(e_i)=1$ for all i $(1\leq i\leq 9)$

$D=\{a,d,e,g\}$



$T_D=(V_{T_D},E_{T_D})$, where

$V_{T_D}=\{a,b,c,d,e,g\}$

$E_{T_D}=\{e_1,e_3,e_4,e_6,e_7\}$

The path $p=b\ e_3\ c\ e_4\ e$ is a basic path in T_D.

An Example of a Steiner Tree
Figure 3.1

1) $E_D=\{\{d_i,d_j\} \mid d_i,d_j \in D \text{ and } d_i \neq d_j\}$.

2) $w_D(\{d_i,d_j\})$ is set equal to the weight of any minimum weight path from d_i to d_j in G.

Notice that each edge in a path graph can be associated with some minimum weight path in the underlying graph.

3.2 Heuristic Algorithms for the Steiner Tree Problem.

In this section we shall consider two heuristic Steiner tree algorithms which are guaranteed to compute a solution with total weight no more than twice optimal. The first of these algorithms, developed in 1977 by C. El-Arbi [EA78], is based on a simple spanning tree computation in a path graph. The second algorithm attempts to apply the same spanning tree computation to a path graph that has been augmented with several vertices corresponding to Steiner vertices in an optimal solution. The search for efficient heuristic Steiner tree algorithms is justified by the fact that the Steiner tree problem has been shown to be NP-hard even under various restricted graph topologies and weight functions [Gar79].

Algorithm 3.1: (El-Arbi [EA78])

Let $STP=(G,D)$ denote any instance of the Steiner tree problem with $G=(V,E,w)$. Algorithm 3.1 then computes a Steiner tree T_D with respect to D in G as follows:

1) Construct the path graph G_D for D in G.

2) Compute a minimum weight spanning tree T_S of the path graph G_D.

3) For each edge $e=\{d_i,d_j\}$ in T_S, let p_e denote a minimum weight path between d_i and d_j in G. Furthermore, let $P_S=\{p_e \mid e \text{ is an edge in } T_S\}$ and let $R \subseteq V$ denote the set of all vertices belonging to a path in P_S. Then let G_R denote the subgraph of G induced by the vertex set R and compute a minimum weight spanning tree T_R of G_R. (Notice that $D \subseteq R$ and thus T_R spans D in G_R.)

4) Construct a Steiner tree T_D from T_R by repeatedly deleting from T_R any vertex (along with its incident edge) which has degree 1 in T_R and does not belong to the set D.

Notice that the tree T_R will have weight no greater than the weight of T_S since the construction of G_R insures that there exists a spanning tree of G_R with weight at most equal to the weight of T_S. Thus it must also be the case that the Steiner tree T_D will have weight at most equal to the weight of T_S. We will now demonstrate an upper bound on the weight of T_D by proving an upper bound on the weight of T_S. This bound is based on the observation that any minimum weight Steiner tree, T_{opt}, for D in G can be decomposed into a set of (possibly overlapping) paths in G which correspond to a spanning tree of G_D with weight at most $2 \cdot W(T_{opt})$.

Lemma 3.1:

Let $G=(V,E,w)$ denote an undirected weighted graph, let $D \subseteq V$ denote any subset of the vertices in G with $|D|>1$, and let T_D denote any Steiner tree with respect to D in G. Further, let $a \in D$ and $b \in D$ denote any two distinct leaves in T_D and let p' denote the unique path from a to b in T_D. Then there exists a set of paths $P=\{p_1,\ldots,p_{|D|-1}\}$ in T_D such that:

1) Each $p \in P$ denotes the unique path from some vertex $d_i \in D$ to some other vertex $d_j \in D$ in T_D.

2) The graph $G'=(D,E')$, where $E'=\{\{d_i,d_j\} \mid d_i$ and d_j are the two endpoints of some path in $P\}$, forms a tree (i.e., a connected graph such that the removal of any edge will leave it disconnected).

3) Each edge in T_D belongs to at most two distinct paths in P and each edge in p' belongs to at most one path in P.

Proof: [Induction on $|D|=m$]

For $m=2$, let $D=\{d_1,d_2\}$ and assume without loss of generality that $a=d_1$ and $b=d_2$. Then it must be the case that T_D denotes a path between a and b in G. Thus $P=\{T_D\}$ must clearly satisfy the conditions of the lemma.

We now assume that Lemma 3.1 holds for $m=k$ ($k \geq 2$) and we consider the case $m=k+1$. For $m=k+1$ there are two possibilities; either T_D has exactly two leaves or T_D contains more than two leaves. If T_D has exactly two leaves, assumed without loss of generality to be $a=d_1 \in D$ and $b=d_{k+1} \in D$, then T_D denotes a path between a and b in G. We now assume (wlog) that T_D is a path from a to b and let $(a=d_1, d_2, \ldots, d_{k+1}=b)$ denote the ordered sequence of all vertices of D along the path T_D. Further, let $P=\{p_1, \ldots, p_k\}$ denote the set of k paths in T_D such that p_i denotes the unique path from d_i to d_{i+1} in T_D (see Figure 3.2a). Then clearly the set P satisfies the conditions of the lemma.

Finally, if T_D contains more than two leaves then let $c \in D$ denote any leaf in T_D such that $c \neq a$ and $c \neq b$. Further, let q denote the basic path from c to some vertex v in T_D. Now consider the set $D'=D\text{-}\{c\}$ and the Steiner tree $T_{D'}$ generated by removing from T_D all the edges and vertices (with the exception of v) belonging to q (see Figure 3.2b). By the induction hypothesis, there exists a set $P'=\{p_1, \ldots, p_{k-1}\}$ of paths in $T_{D'}$ which satisfy the conditions of the lemma for G, D', $T_{D'}$, a, and b. Now if $v \in D$, then simply let $P=P' \cup \{q\}$. Clearly, the set P satisfies the conditions of the lemma for T_D. However, if v denotes a Steiner vertex in T_D then there must exist at least one path $p_i \in P'$ such that v belongs to p_i. In this case we generate the required set P for T_D by simply replacing p_i in P' with a pair of paths as shown in Figure 3.2c.

□

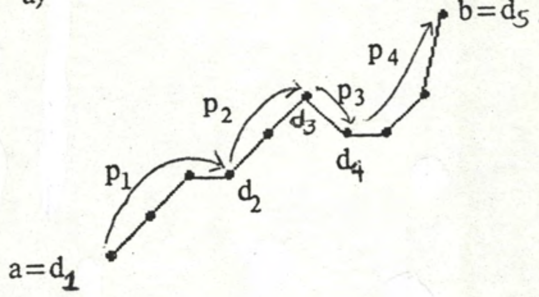
Theorem 3.1: (El-Arbi [EA78])

Let $STP=(G,D)$ denote any instance of the Steiner tree problem with $G=(V,E,w)$. Further, let T_{opt} denote a minimum weight Steiner tree with respect to D in G and let T_{alg} denote a Steiner tree generated by Algorithm 3.1 for STP. If we now let l denote the number of distinct leaves in T_{opt}, then:

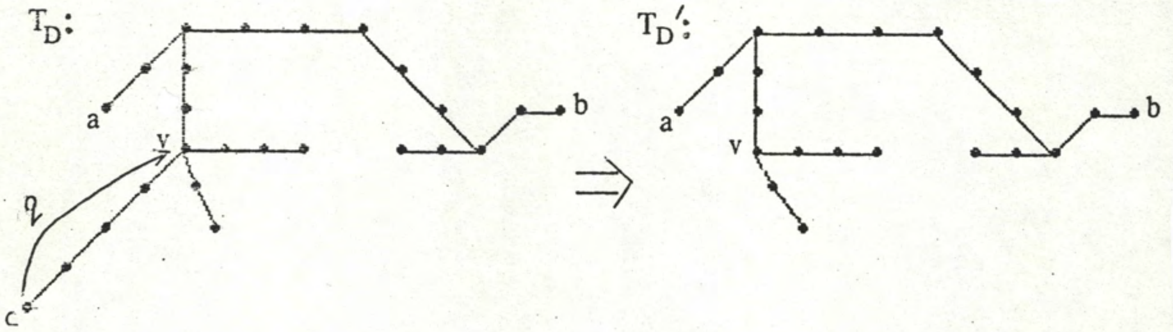
$$W(T_{alg})/W(T_{opt}) \leq 2(1\text{-}1/l) \leq 2(1\text{-}1/|D|).$$

Proof:

Let $a \in D$ and $b \in D$ denote two distinct leaves in T_{opt} such that the weight of the path p' from a to b in T_{opt} is maximized. Then let $(a=d_1, d_2, \ldots, d_l=b)$ denote the ordered sequence of all leaves in T_{opt} as visited in a preorder traversal of T_{opt} (c.f., [Aho74]); beginning at a and a)



b)



c)

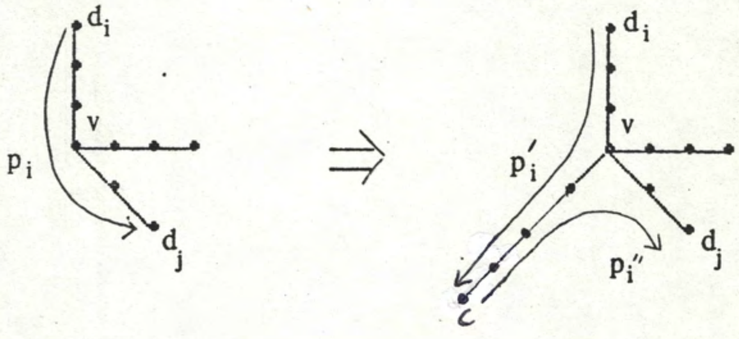


Figure 3.2

ending at b. If we now let q_i denote the path from d_i to d_{i+1} in T_{opt} (for all i, $1 \leq i \leq l\text{-}1$), then clearly:

$$W(p') + \Sigma\{W(q_i) \mid 1 \leq i \leq l+1\} = 2 \cdot W(T_{opt}).$$

However, since p' has weight no less than $W(q_i)$ for every i $(1 \leq i \leq l\text{-}1)$, we have:

$$W(p') \geq (2/l) \cdot W(T_{opt}).$$

We can now apply Lemma 3.1 to T_{opt}, a, and b, to obtain a set P of paths in T_{opt} such that:

$$W(P) = \Sigma\{W(p) \mid p \in P\} \leq 2 \cdot W(T_{opt}) \text{-} (2/l) \cdot W(T_{opt}).$$

Finally, it should be clear from condition (2) of Lemma 3.1 that $W(P)$ is no less than the weight of the tree T_S computed in Step (2) of Algorithm 3.1 for STP. Thus we have:

$$W(T_{alg}) \leq W_D(T_S) \leq W(P) \leq 2(1\text{-}1/l) \cdot W(T_{opt})$$
$$\Rightarrow W(T_{alg})/W(T_{opt}) \leq 2(1\text{-}1/l) \leq 2(1\text{-}1/|D|).$$

□

Therefore, Algorithm 3.1 is guaranteed to compute a near optimal solution to any given instance of the Steiner tree problem. We now note that this algorithm might produce somewhat better results if we could initially augment the distinguished vertex set with several vertices corresponding to Steiner vertices in an optimal solution. An indirect application of this observation yields the following heuristic Steiner tree algorithm.

Algorithm 3.2:

Let $STP=(G,D)$ denote any instance of the Steiner tree problem with $G=(V,E,w)$. Algorithm 3.2 then computes a Steiner tree T_D with respect to D in G as follows:

For each distinct $I \subseteq V\text{-}D$ such that $|I| \leq 2$, compute a Steiner tree $T_{D \cup I}$ with respect to $D \cup I$ in G by simply applying Algorithm 3.1 to the problem $STP_I=(G, D \cup I)$. Then delete vertices and edges from each of these trees (see Step (4) of Algorithm 3.1) until all the remaining leaves are vertices in D (and not $D \cup I$). Finally, let T_D denote the tree which has minimum weight over all those computed.

We now note that T_D will have total weight no more than the weight of the smallest spanning tree computed in Step (2) of Algorithm 3.1. Thus a bound on the weight of this smallest spanning tree will be a bound on T_D.

Lemma 3.2:

Let $G=(V,E,w)$ denote an undirected weighted graph, let $D \subseteq V$ denote any subset of the vertices in G with $|D|>1$, and let T_D denote any Steiner tree with respect to D in G. Further, let $a \in D$ and $b \in D$ denote any two distinct vertices in D. Then there exists a set of paths $P=\{p_1, \ldots, p_{|D|-1}\}$ in T_D such that:

1) Each $p \in P$ denotes the unique path from some vertex $d_i \in D$ to some other vertex $d_j \in D$ in T_D.

2) The graph $G'=(D,E')$, where $E'=\{\{d_i,d_j\} \mid d_i$ and d_j are the two endpoints of some path in $P\}$, forms a tree.

3) Each edge in T_D belongs to at most two distinct paths in P, each edge in a basic path between two distinct vertices $d_i \in D$ and $d_j \in D$ in T_D belongs to at most one path in P, and each edge in a basic path from either $a \in D$ or $b \in D$ to some other vertex in T_D belongs to at most one path in P.

Proof:

The proof of this lemma follows from the same induction argument as that used to prove Lemma 3.1. The only difference occurs in the case where $m=k+1$ and T_D has exactly two leaves. In this case we allow a and b to be arbitrarily assigned to vertices in D.

$\square$

Theorem 3.2:

Let $STP=(G,D)$ denote any instance of the Steiner tree problem with $G=(V,E,w)$. Further, let T_{opt} denote a minimum weight Steiner tree with respect to D in G and let T_{alg} denote a Steiner tree generated by Algorithm 3.2 for STP. If we now let s denote the number of distinct Steiner vertices in T_{opt}, then for $s \leq 2$ we have:

$$W(T_{alg})/W(T_{opt})=1,$$

and for $s>2$ we have:

$$W(T_{alg})/W(T_{opt}) \leq 2(1-1/s).$$

Proof:

Let SV denote the set of all Steiner vertices in T_{opt}. Therefore, $s=|SV|$ and we have two possible cases to consider; $s=|SV|\leq 2$ and $s=|SV|>2$. If $s\leq 2$, then clearly one of the augmented distinguished vertex sets $(D\cup I)$ generated by Algorithm 3.2 will contain exactly those vertices belonging to T_{opt}. For this particular distinguished vertex set Algorithm 3.1 is guaranteed to compute a Steiner tree with weight equal to $W(T_{opt})$. Thus the final Steiner tree, T_{alg}, computed by Algorithm 3.2 must also have weight equal to $W(T_{opt})$. Therefore, for $s\leq 2$ we have:

$$W(T_{alg})/W(T_{opt})=1.$$

If $s=|SV|>2$, then for each Steiner vertex $v\in SV$ let E_v denote the set of all edges belonging to a basic path with endpoint v in T_{opt} and let $W(E_v)$ denote the sum of the weights of all edges in E_v:

$$W(E_v)=\Sigma\{w(e)\mid e\in E_v\}.$$

Furthermore, let $a\in SV$ and $b\in SV$ denote two distinct Steiner vertices in T_{opt} such that:

$$W(E_a\cup E_b)=\Sigma\{w(e)\mid e\in E_a\cup E_b\}$$

is maximum over all possible pairs of distinct Steiner vertices in T_{opt}. (Notice that the weight of any edge in both E_a and E_b is counted only once.) Now consider the edge sets X, Y, and Z such that:

1) X contains exactly those edges belonging to a basic path between two distinguished vertices in T_{opt}.
2) Y contains exactly those edges beloging to a basic path between two Steiner vertices in T_{opt}.
3) Z contains exactly those edges belonging to a basic path between a Steiner vertex and a distinguished vertex in T_{opt}.

Further, let $W(X)$, $W(Y)$, and $W(Z)$ denote the sum of the weights of all edges in X, Y, and Z respectively. We now note that the sets X, Y, and Z partition the edge set of T_{opt} and thus $W(X)+W(Y)+W(Z)=W(T_{opt})$. In addition, we note that $Y\cup Z=\cup\{E_v \mid v\in SV\}$ and thus by our selection of vertices a and b we obtain the relation:

$$\begin{aligned}(s/2)\cdot W(E_a\cup E_b)&\geq W(Y)+W(Z)=W(T_{opt})-W(X)\\ \Rightarrow W(E_a\cup E_b)&\geq 2\cdot[W(T_{opt})-W(X)]/s \qquad (3.1)\end{aligned}$$

Finally, we apply Lemma 3.2 to G, $D\cup\{a,b\}$, T_{opt}, a, and b to obtain a set P of paths in T_{opt} such that:

$$W(P)=\Sigma\{W(p)\mid p\in P\}\leq 2\cdot W(T_{opt})-W(X)-W(E_a\cup E_b).$$

It should now be clear from condition (2) of Lemma 3.2 that $W(P)$ is no less than the weight of the spanning tree computed in Step (2) of Algorithm 3.1 for the augmented vertex set $D\cup\{a,b\}$. Therefore $W(P)$ must be no less than the weight of the smallest spanning tree computed in Step (2) of Algorithm 3.1 and so we have:

$$\begin{aligned}
&W(T_{alg})\leq W(P)\leq 2\cdot W(T_{opt})-W(X)-W(E_a\cup E_b) & \\
&\Rightarrow W(T_{alg})\leq 2\cdot W(T_{opt})-W(X)-2\cdot[W(T_{opt})-W(x)]/s & \text{[from (3.1)]}\\
&\Rightarrow W(T_{alg})\leq 2\cdot W(T_{opt})-2\cdot W(T_{opt})/s & \text{[since } s>2]\\
&\Rightarrow W(T_{alg})/W(T_{opt})\leq 2(1-1/s). &
\end{aligned}$$

□

We have now shown that both Algorithm 3.1 and Algorithm 3.2 are guaranteed to compute a near optimal solution to any given instance of the Steiner tree problem. It should also be clear, however, that Algorithm 3.2 will always generate a better solution than Algorithm 3.1. Let us now consider the time complexity of each of these two Steiner tree algorithms.

Theorem 3.3:

Let $STP=(G,D)$ denote any instance of the Steiner tree problem with $G=(V,E,w)$. Then Algorithm 3.1 has worst case time complexity $O(|V|^3)$ for computing a solution to STP.

Proof:

The proof of this theorem follows directly from a simple analysis of the worst case time complexity of each step in Algorithm 3.1.

□

Theorem 3.4:

Let $STP=(G,D)$ denote any instance of the Steiner tree problem with $G=(V,E,w)$. Then Algorithm 3.2 has worst case time complexity $O(|V|^4\cdot\log|V|)$ for computing a solution to STP.

Proof:

The proof of this theorem follows from the observation that the minimum weight paths between every pair of vertices in the graph G can be precomputed in time $O(|V|^3)$ [Aho74]. The construction of each path graph $G_{D \cup I}$ can then be accomplished by simply selecting the appropriate minimum weight paths from those previously computed. All other steps are then analyzed in the standard way.

□

3.3 A Practical Steiner Tree Algorithm.

In this section we shall describe a practical version of Algorithm 3.1 which is currently being used in the global routing phase of the PI routing system. This new algorithm is essentially a generalization of Dijkstra's single source shortest path algorithm [Aho74] and thus we shall begin by describing Dijkstra's algorithm.

Let $G=(V,E,w)$ denote an undirected weighted graph and let $v_o \in V$ denote any vertex in V. Dijkstra's algorithm then computes a minimum weight path from v_o to any other vertex in V by constructing a set $S \subseteq V$ such that the minimum weight path from v_o to each vertex in S is composed entirely of vertices in S. If we let $C(v)$ denote the weight of the current minimum weight path from v_o to v passing only through vertices of S, then Dijkstra's algorithm proceeds as follows: (taken from [Aho74])

```
1) $S \leftarrow \{v_o\}$;
2) $C(v_o) \leftarrow 0$;
3) for each $v \in V-\{v_o\}$ do $C(v) \leftarrow w(\{v, v_o\})$;
   comment Let $w(\{v_i, v_j\}) = +\infty$ if $\{v_i, v_j\} \notin E$ and let $w(v_i, v_i) = 0$.
4) while $S \neq V$ do
       begin
5)         choose a vertex $v' \in V-S$ such that $C(v')$ is minimum;
6)         add $v'$ to $S$;
7)         for each $v \in V-S$ do
8)             $C(v) \leftarrow \text{Min}(C(v), C(v') + w(\{v', v\}))$
       end
```

We should now note that Dijkstra's algorithm is essentially an efficient formulation of C.Y. Lee's minimum length path algorithm [Lee61]. As Lee noted, the growth of the set S can be viewed as a *wave expansion*; similar to the result of dropping a pebble into a pond. We can now generalize this wave expansion technique to compute the minimum weight path over all paths between any pair of vertices in a prespecified subset of vertices. The basic approach is to execute Dijkstra's algorithm from several sources simultaneously until the wave expansion from one source reaches another source. This constitutes the basic computational step in our Steiner tree algorithm.

Let $STP=(G,D)$ denote any instance of the Steiner tree problem with $G=(V,E,w)$. If $v \in V$ denotes any vertex in V and $V' \subseteq V$ denotes any subset of the vertices in V, then a minimum weight path between v and V' in G is defined to be any path p between v and some vertex $v' \in V'$ in G such that $W(p)$ is minimum over all possible paths in G between v and a vertex in V'. Now let $B=\{B_1, \ldots, B_m\}$ denote any family of disjoint subsets of V. The basic computational step in our Steiner tree algorithm then consists of computing a path p' in G such that p' connects two vertices belonging to distinct subsets in B and $W(p')$ is minimum over all such paths in G. This computation is performed by a simple generalization of Dijkstra's algorithm which constructs a family of sets $\{S_{B_1} \subseteq V, \ldots, S_{B_m} \subseteq V\}$ such that for each i $(1 \leq i \leq m)$, the minimum weight path between B_i and each vertex in S_{B_i} is composed entirely of vertices in S_{B_i}. If we let $C_{B_i}(v)$ denote the weight of the current minimum weight path between v and B_i passing only through vertices of S_{B_i}, then the path-finding algorithm proceeds as follows:

```
1)  for each $B_i \in B$ do
        begin
2)          $S_{B_i} \leftarrow B_i$;
3)          for each $b \in B_i$ do $C_{B_i}(b) \leftarrow 0$;
4)          for each $v \in V - B_i$ do $C_{B_i}(v) \leftarrow \text{Min}\{w(\{v, b\}) \mid b \in B_i\}$
        end;
5)  while true do
        begin
6)          choose a pair $(B_i \in B,\ v' \in V - S_{B_i})$ such that $C_{B_i}(v')$ is minimum over all such
            pairs;
7)          add $v'$ to $S_{B_i}$;
8)          if $v' \in B_j$ for some $B_j \in B$ then halt (The desired path has been found);
9)          for each $v \in V - S_{B_i}$ do
10)             $C_{B_i}(v) \leftarrow \text{Min}(C_{B_i}(v), C_{B_i}(v') + w(\{v, v'\}))$
        end
```

In the complete Steiner tree algorithm, each set S_{B_i} is called a *group* with *basis* B_i and each vertex in the set B_i is called a *basic vertex* in S_{B_i}. Further, each vertex in the set $S_{B_i} - B_i$ is called a *new vertex* in S_{B_i} and each vertex $v \notin S_{B_i}$ with $C_{B_i}(v) < \infty$ is called a *neighbor vertex* of S_{B_i}. Finally, in the current implementation of the algorithm there exists a data structure called a *mark* which records the relationship (basis, new, or neighbor) between every group-vertex pair. The entire Steiner tree algorithm can now be described as follows.

Algorithm 3.3:

1) If $|D| = 2$, the standard single source Dijkstra algorithm is used to compute the desired Steiner tree.

2) If $|D| > 2$, the algorithm begins by generating $|D|$ distinct groups each with a unique basis $\{d\} \subseteq D$. Following this initialization, the generalized Dijkstra algorithm is used to compute a minimum weight path in G between any pair of basic vertices belonging to distinct groups. This path is then output as a fragment of the desired Steiner tree and the two groups containing the endpoints of the path are *merged*. The merging process simply involves replacing the two endpoint groups with a new group containing no neighbor or

new vertices and having basis equal to the union of the basis sets for the two groups being replaced. The path-finding procedure is then continued on this new group structure. Such iteration continues until there is only one remaining group. At this point the set of paths output by the algorithm can be used to form the desired Steiner tree.

It should now be clear that Algorithm 3.3 is essentially an efficient implementation of the first two steps of Algorithm 3.1 and thus the same bounds on solution optimality must hold (i.e., $W(T_{alg})/W(T_{opt}) \leq 2(1-1/|D|)$). Notice, however, that this algorithm computes a Steiner tree with worst case time complexity $O(|D| \cdot |E| \cdot \log|V|)$. (This follows from the fact that the algorithm essentially involves $O(|D|)$ Dijkstra computations, each with time complexity $O(|E| \cdot \log|V|)$.) Finally, notice that this algorithm can also be applied to the more general Steiner tree problem in which each distinguished vertex corresponds to a set of vertices. The problem is then to compute a minimum weight tree containing at least one vertex from each set. The solution of this problem simply requires the initialization of each group with a set of basis vertices rather than a single basic vertex. This technique was also noted by D. Hightower [Hig74] for the case in which there are only two distinguished vertices (or vertex sets).

Chapter 4 - Region Routing.

4.1 Introduction.

In this chapter we consider the problem of wire routing within a rectangular region. We begin by developing a new realistic multiple layer grid-based wiring model. We then use this model to construct polynomial-time algorithms for the well known river routing and channel routing problems. More specifically, we present efficient polynomial-time algorithms for river routing with arbitrarily many conducting layers and "provably good" polynomial-time heuristic algorithms for two-layer channel routing.

Assume we are given a rectangular region with fixed dimensions on a routing plane and a set of connection points constrained to lie on the perimeter of the region. In addition, assume we are given a partition of the connection points into nets which specify interconnect required within the region. We now consider the problem of producing wiring which has the properties that all points belonging to a given net are interconnected and no two points belonging to different nets are *shorted* (i.e., interconnected). Furthermore, all wiring must be located entirely within the given region and must satisfy some prespecified set of wiring rules.

In 1971, Hashimoto and Stevens [Has71] presented a simple heuristic algorithm for a restricted region routing problem in which each net has cardinality two and there are constraints on the locations at which connection points can lie and the type of wiring that can be produced. In particular, all wiring is required to lie on a rectilinear grid with vertically oriented wires assigned to one conducting layer and horizontally oriented wires assigned to another. In 1976, Deutsch [Deu76] generalized this work to include nets containing arbitrarily many connection points. (This version of the region routing problem is called the *channel routing problem*.) Empirically, Deutsch's algorithm performs very well in practice. However, A. LaPaugh and E. Lloyd [LaP80] have shown that there exist pathological problem instances for which it produces very poor results. Moreover, LaPaugh [LaP80] and Szymanski [Szy81] have proven NP-Completeness for various versions of the channel routing problem.

A somewhat simpler restricted region routing problem, called the *river routing problem*, has also received a great deal of attention in the recent literature. An instance of the river routing problem has the properties that each net contains exactly two connection points, one constrained to lie at a fixed location along the top of the rectangular region and the other constrained to lie at a fixed location along the bottom, such that the i^{th} net contains both the i^{th} connection point (numbered from left-to-right) along the top of the region and the i^{th} connection point along the bottom. In 1980, M. Tompa [Tom80] presented a polynomial-time algorithm for river routing under a general single-layer grid-free wiring model. Tompa showed that an optimal solution need only contain wires composed of straight line segments and circular arcs. One year later, D. Dolev and others [Dol81] developed a polynomial-time algorithm for river routing under a single-layer rectilinear grid-based wiring model. This result was later extended by Dolev and Siegel [DS81] to include single-layer rectilinear plus diagonal grid-based wiring models.

In Section 4.2, we shall formalize the notion of a region routing problem and thus develop a general routing model. This model will serve to unify many of the individual routing problems and wiring models that have appeared in the recent literature (including those discussed above). In the process, we will introduce a new class of realistic multiple layer grid-based wiring models which will be used in Sections 4.4, 4.5, and 4.6 to construct provably good algorithms for the river routing and channel routing problems.

4.2 A General Routing Model.

In this section we shall formalize the notions of a region routing problem and an associated wiring model. We will begin by introducing the fundamental concept of a routing region which essentially defines the boundary of a routing area and the topology of wiring allowed within that area (e.g., the slope of legal interconnect lines or the portions of the wiring area through which wires may pass). A routing region will be defined to have fixed dimensions and thus a region routing problem will be a *decision problem* rather than an optimization problem. That is, a region routing problem will be a problem of determining whether or not a routing can be generated within a given region. Notice that this differs from the problem of determining the smallest region for which a routing exists.

Once we have introduced the concept of a routing region, we can formalize the notion of wiring by defining the structure of a wire and a wiring tree within a routing region. This will be used to define the concept of a wiring model. A wiring model essentially consists of a routing region, defining the topology of a legal wire, and a set of wiring rules, defining the properties of permissible wirings. For example, a wiring model might require that all wiring be placed on a rectilinear grid with horizontally oriented segments located on one conducting layer and vertically oriented segments located on another; as in the Thompson model [Tho80]. Another wiring model might allow diagonal wire segments but require that each net be routed on a single layer. Thus a wiring model essentially specifies wire topology, restrictions on layer assignment, and restrictions on wire overlap or crossing. Our goal is to develop a framework for studying and comparing the effect of various wiring models on a particular type of routing problem.

Finally, the concept of a wiring model will be combined with the notions of a pin and a net to develop a formal definition of the region routing problem. A net will be simply defined as a set of pins and a pin should be considered an abstraction of a physical connection point constrained to lie somewhere along the perimeter of a particular routing region. However, we would like our model to incorporate many different types of routing problems having various restrictions on the locations at which pins can be placed. (For example, the river routing problem is defined such that each pin has a fixed location along either the top or bottom of the routing area. Similarly, the channel routing problem is defined such that each pin is constrained to lie either at some fixed location along the top or bottom of the routing area, at some unspecified location along the left side of the routing area, or at some unspecified location along the right side of the routing region.) Thus the notion of a pin will be defined under a set of restrictions on the points at which it may be located.

Definition 4.1:

A (rectangular) *routing region* R is a 5-tuple, $R=(l,h,\Lambda,V,E)$, such that:

1) $l,h,\Lambda \in Z^{+}$.

2) V denotes a set of *grid points* (x,y) such that the integers x and y satisfy the relations $0 \leq x \leq l+1$ and $0 \leq y \leq h+1$.

3) E denotes a non-empty set of *routing edges* of the form $(\{v_i, v_j\}, k)$, where v_i and v_j are distinct grid points in V and the integer k satisfies the relation $1 \leq k \leq \Lambda$.

The routing region R is said to have *length* l, *height* h, and to contain Λ *conducting layers*. For each routing edge $e = (\{v_i, v_j\}, k)$ in E we say that v_i and v_j are the *endpoints* of e and that e *connects* v_i and v_j while *on conducting layer* k. Furthermore, we define the function *layer*(e) such that:

$$layer(e) = k, \text{ for each edge } e = (\{v_i, v_j\}, k) \text{ in } E.$$

Finally, any routing edge which connects two grid points with the same y-coordinate is called a *horizontal edge* and any routing edge which connects two grid points with the same x-coordinate is called a *vertical edge*.

Definition 4.2:

Let $R = (l, h, \Lambda, V, E)$ denote a routing region and let $i \in Z$ denote an integer which satisfies the relation $0 \leq i \leq h+1$. We then define *track* i of R to be the ordered pair track i $= (V_i, E_i)$ such that:

1) $V_i \subseteq V$ denotes the set of all grid points in V with y-coordinate equal to i.
2) $E_i \subseteq E$ denotes the set of all (horizontal) edges in E which connect pairs of grid points in V_i.

Note that track i is not necessarily connected.

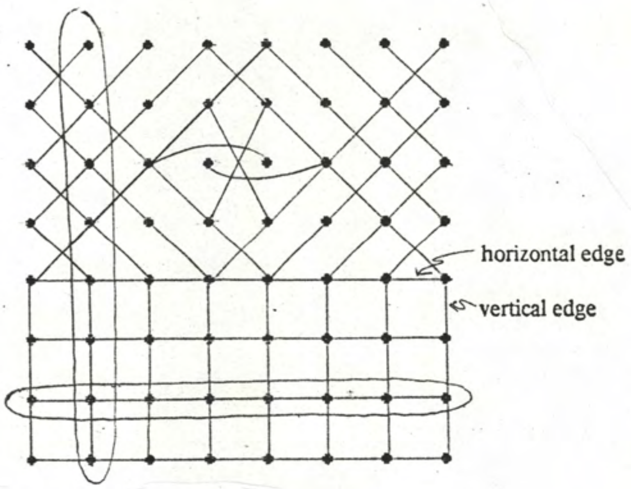
Definition 4.3:

Let $R = (l, h, \Lambda, V, E)$ denote a routing region and let $j \in Z$ denote an integer which satisfies the relation $0 \leq j \leq l+1$. We then define *column* j of R to be the ordered pair column j $= (V'_j, E'_j)$ such that:

1) $V'_j \subseteq V$ denotes the set of all grid points in V with x-coordinate equal to j.
2) $E'_j \subseteq E$ denotes the set of all (vertical) edges in E which connect pairs of grid points in V'_j.

Note that column j is not necessarily connected.

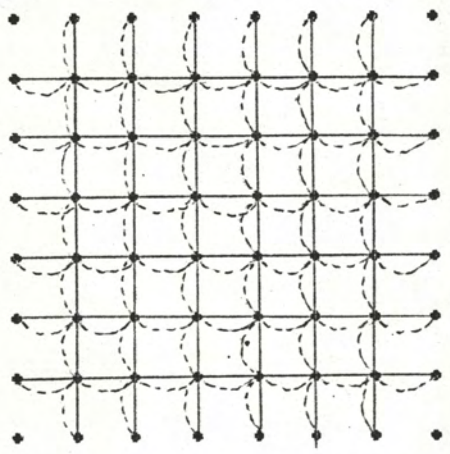
An example of an arbitrary routing region is shown in Figure 4.1. Notice that a routing region can assume any one of many different topologies depending on the structure of the



Routing Region

(1 Conducting Layer)

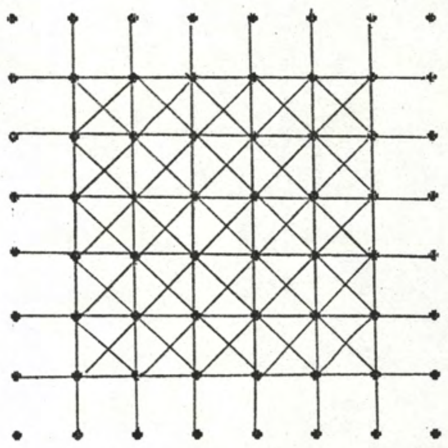
Figure 4.1



Rectilinear Routing Region

(2 Conducting Layers)

Figure 4.2



Aug. Rect. Routing Region

(1 Conducting Layer)

Figure 4.3

associated set of routing edges. In the remainder of this chapter we shall be mainly concerned with two specific types of routing regions; the rectilinear routing region and the augmented rectilinear routing region.

Definition 4.4:

A *rectilinear routing region* $R^1=(l,h,\Lambda,V,E^1)$ is a routing region with the property that the edge set E^1 forms a multi-layer rectilinear grid by containing essentially all routing edges connecting pairs of horizontally or vertically adjacent grid points:

$$\begin{aligned} E^1=\{(\{v_i,v_j\},k)\mid\ & v_i=(x_i,y_i)\in V,\ v_j=(x_j,y_j)\in V,\ |x_i-x_j|+|y_i-y_j|=1,\\ & 1\leq x_i\leq l \text{ or } 1\leq x_j\leq l,\ 1\leq y_i\leq h \text{ or } 1\leq y_j\leq h,\\ & k\in Z^+,\ \text{and } 1\leq k\leq\Lambda\}. \end{aligned}$$

An *augmented rectilinear routing region* $R^2=(l,h,\Lambda,V,E^2)$ is a routing region with the property that the edge set E^2 forms a multi-layer "diagonally augmented" rectilinear grid by containing essentially all routing edges connecting pairs of horizontally, vertically, or diagonally adjacent grid points:

$$\begin{aligned} E^2=E^1\cup\{(\{v_i,v_j\},k)\mid\ & v_i=(x_i,y_i)\in V,\ v_j=(x_j,y_j)\in V,\ 1\leq x_i,x_j\leq l,\\ & 1\leq y_i,y_j\leq h,\ (x_i-x_j)^2+(y_i-y_j)^2=2,\\ & k\in Z^+,\ \text{and } 1\leq k\leq\Lambda\}. \end{aligned}$$

Figure 4.2 shows an example of a rectilinear routing region and Figure 4.3 shows an example of an augmented rectilinear routing region. In each case we note that there are no routing edges in track 0, track $h+1$, column 0, or column $l+1$. This restriction is imposed to model the fact that it is generally undesirable to have wiring located on the border of a routing region. The following definitions will formalize the relationship between routing edges and wiring within a routing region.

Definition 4.5:

Let $R=(l,h,\Lambda,V,E)$ denote a routing region. A *wire* W, from grid point v_1 to grid point v_n in R, is a sequence of distinct grid points and routing edges $W=v_1e_1v_2e_2\ldots v_{n-1}e_{n-1}v_n$ $(n\geq 2)$ such that $v_i\in V$ for each i $(1\leq i\leq n)$, $e_i\in E$ for each i $(1\leq i\leq n-1)$, and e_i connects v_i and

v_{i+1} for each i $(1 \leq i \leq n\text{-}1)$.

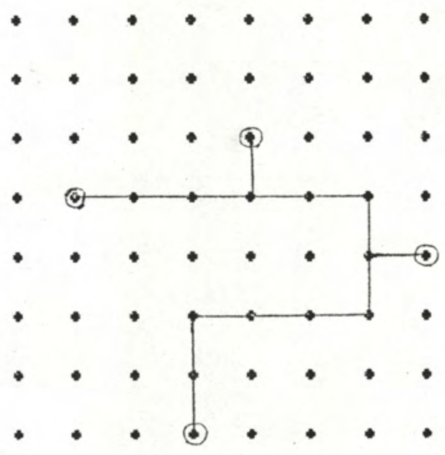
Definition 4.6:

Let $R=(l,h,\Lambda,V,E)$ denote a routing region and let $V' \subseteq V$ denote a subset of the grid points in V with $|V'|>1$. A *wiring tree* T with respect to V' in R is an ordered pair $T=(V_T,E_T)$ such that:

1) $V' \subseteq V_T \subseteq V$.
2) $E_T \subseteq E$ with each edge in E_T connecting a pair of grid points in V_T.
3) For each pair of distinct grid points $(v_i \in V',\ v_j \in V')$, there exists exactly one wire from v_i to v_j in R which is composed entirely of edges in E_T.
4) The sets V_T and E_T are minimal with respect to (1), (2), and (3). That is, there are no subsets of V_T or E_T which also satisfy (1), (2), and (3).

We say that T *interconnects* V' in R.

An example of a wiring tree is depicted in Figure 4.4. It should be noted that both a wire and a wiring tree may be composed of routing edges from many different conducting layers. Thus we require the following definition.



A Wiring Tree in a Rectilinear Routing Region
Figure 4.4

Definition 4.7:

Let $R=(l,h,\Lambda,V,E)$ denote a routing region, let $V'\subseteq V$ denote a subset of the grid points in V with $|V'|>1$, and let $T=(V_T,E_T)$ denote a wiring tree with respect to V' in R. For each integer k ($1\leq k\leq\Lambda$), the set of all edges of T which are on conducting layer k is denoted by:

$$layer_k(T)=\{e\in E_T \mid layer(e)=k\}.$$

Further, for each grid point $v\in V_T$ we define $low_T(v)$ to be the least integer i such that there exists an edge in E_T which is on conducting layer i and has v as an endpoint. Similarly, we define $high_T(v)$ to be the greatest integer i such that there exists an edge in E_T which is on conducting layer i and has v as an endpoint. The set of *internal contact points* of T is then defined by:

$$IC(T)=\{v\in V_T \mid low_T(v)\neq high_T(v)\}.$$

Finally, the set of *contact points* of T is defined by:

$$CON(T)=V'\cup IC(T).$$

The definitions of a routing region and a wiring tree within a routing region can now be combined to formalize the concept of a *wiring model*. A wiring model essentially consists of a set of wiring rules defining the structure of permissible wirings within a given routing region. Thus we begin by introducing three common restrictions on the general structure of permissible wirings. We then present a formal definition of a wiring model.

Definition 4.8:

Let $R=(l,h,\Lambda,V,E)$ denote a routing region and let T_1 and T_2 denote two distinct wiring trees in R. The wiring trees T_1 and T_2 are said to satisfy wiring restriction *NEO* [No Edge Overlap] if there does not exist a pair of routing edges e_1 belonging to T_1 and e_2 belonging to T_2 such that e_1 and e_2 have the same pair of endpoints.

Definition 4.9:

Let $R=(l,h,\Lambda,V,E)$ denote a routing region and let $T=(V_T,E_T)$ denote a wiring tree in R. The wiring tree T is said to satisfy wiring restriction *LPT* [Layer Per Tree] if $E_T=layer_k(T)$ for some $1\leq k\leq\Lambda$.

Definition 4.10:

Let $R=(l,h,\Lambda,V,E)$ denote a rectilinear routing region with $\Lambda=2$ conducting layers and let $T=(V_T,E_T)$ denote a wiring tree in R. The wiring tree T is said to satisfy wiring restriction *LPD* [Layer Per Direction] if $layer_1(T)$ contains only horizontal edges and $layer_2(T)$ contains only vertical edges. Notice that wiring restriction *LPD* is defined only for rectilinear routing regions with two conducting layers.

Definition 4.11:

A *wiring model* M is a 3-tuple, $M=(R,\sigma,\rho)$, such that:

1) $R=(l,h,\Lambda,V,E)$ denotes a routing region.

2) $\sigma\in\mathbb{R}^+$ denotes a positive real number called the *minimum separation distance* for M.

3) $\rho\subseteq\{NEO,LPT,LPD\}$ specifies the set of wiring restrictions that must be satisfied under M.

It now remains to define the structure of permissible wiring under a given wiring model. Before proceeding, however, we require a technical formulation of the concept of a routing edge as a set of points in the Euclidean x-y plane.

Definition 4.12:

Let $R=(l,h,\Lambda,V,E)$ denote a routing region, let $e=(\{v_i,v_j\},k)$ denote any routing edge in E, and let $T=(V_T,E_T)$ denote a wiring tree in R. We define $line(e)$ to be the set of all points in the x-y plane which lie on the line segment connecting points $v_i=(x_i,y_i)$ and $v_j=(x_j,y_j)$:

$$line(e)=\{(x,y)\mid x,y\in\mathbb{R} \text{ and } (x,y)=t\cdot(x_i,y_i)+(1\text{-}t)\cdot(x_j,y_j) \text{ for some } 0\leq t\leq 1\}.$$

Furthermore, for each integer k $(1\leq k\leq\Lambda)$, we define $span_k(T)$ as:

$$span_k(T)=[\cup\{line(e)\mid e\in layer_k(T)\}]\cup\{v\mid v\in V_T \text{ and } low_T(v)\leq k\leq high_T(v)\}.$$

Informally, $span_k(T)$ simply specifies all points in the x-y plane at which conducting layer k contains some portion of the wiring tree T. Notice that this includes any contact point of T requiring a physical contact cut through conducting layer k.

Definition 4.13:

Let $M=(R,\sigma,\rho)$ denote a wiring model in which $R=(l,h,\Lambda,V,E)$. Further, let T_1 and T_2 denote two distinct wiring trees in R. The wiring trees T_1 and T_2 are said to be *compatible* under M if:

1) For each integer k $(1 \leq k \leq \Lambda)$, the Euclidean distance between any point $a \in span_k(T_1)$ and any point $b \in span_k(T_2)$ is at least σ.
2) The wiring trees T_1 and T_2 satisfy those wiring restrictions (*NEO*, *LPT*, or *LPD*) specified by ρ.

We now note that a wiring model has been defined with respect to only three types of wiring restrictions. For our purposes this is sufficient. It should be clear, however, that the definition can be extended to include other wiring restrictions as may be necessary or desired. Figure 4.5 shows several examples of compatible wiring trees under various wiring models. Observe the distinction between "edge overlap" and "point overlap" among wiring trees.

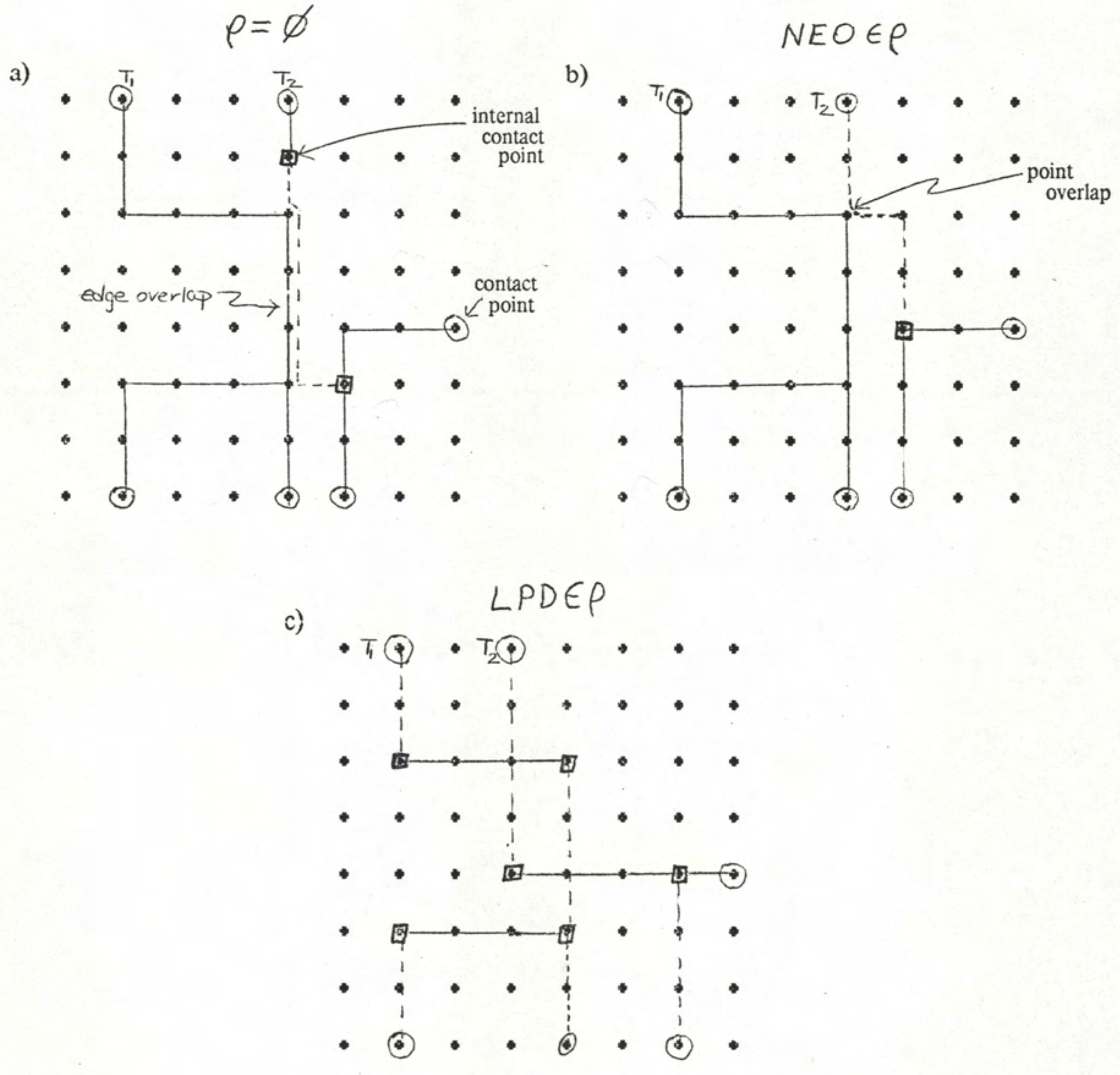
We are now ready to formalize the concept of a region routing problem. We begin by defining a pin structure.

Defintion 4.14:

Let $R=(l,h,\Lambda,V,E)$ denote a routing region and let $Q \subseteq V$ denote the set of all grid points $(x,y) \in V$ with either $x \in \{0,l+1\}$ or $y \in \{0,h+1\}$; but not both $x \in \{0,l+1\}$ and $y \in \{0,h+1\}$. Each grid point in Q is called a *legal pin location point* in R and the set Q is called the *pin location set* for R. Now let $P=\{p_1,\ldots,p_m\}$ denote a set of *pins* such that $m \leq |Q|$ and let $\pi : P \rightarrow Q$ denote a one-to-one function which maps each distinct pin in P into a distinct legal pin location point in Q. The set P is then called a *valid pin set* for R and the mapping π is called a *legal placement* of the pin set P in R.

Definition 4.15:

Let $R=(l,h,\Lambda,V,E)$ denote a routing region, let $P=\{p_1,\ldots,p_m\}$ denote a valid pin set for R, and let π denote a legal placement of the pin set P in R. Further, assume that $\pi(p_i)=(x_i,y_i)$ for each $p_i \in P$. We then define the functions $\pi^x : P \rightarrow Z$ and $\pi^y : P \rightarrow Z$ such that $\pi^x(p_i)=x_i$ and $\pi^y(p_i)=y_i$ for each $p_i \in P$.



Compatible Wiring Trees under Rectilinear Routing Regions
Figure 4.5

Definition 4.16:

Let $R=(l,h,\Lambda,V,E)$ denote a routing region, let $P=\{p_1,\ldots,p_m\}$ denote a valid pin set for R, and let $\Pi=\{\pi_1,\ldots,\pi_r\}$ denote a set of legal placements of the pin set P in R. The ordered pair $PS=(P,\Pi)$ is then called a *pin structure* in R.

Informally, a pin should be considered an abstraction of a physical connection point constrained to lie somewhere along the perimeter of a particular routing region. A pin structure then simply specifies a set of pins and a set of constraints on the locations at which each of these pins can lie. Notice that a pin can be forced to lie at a particular pin location point, or that a set of pins can be forced to lie at pin location points which are consistent with some prespecified partial order, by simply defining the set Π appropriately.

Definition 4.17:

Let $R=(l,h,\Lambda,V,E)$ denote a routing region, let $PS=(P,\Pi)$ denote a pin structure in R, and let $\pi\in\Pi$ denote a legal placement of the pin set P in R. A *net* N under PS is then defined to be a subset of the pins in P such that $|N|>1$. Further, the *placement* of net N under the mapping π is defined by:

$$\pi(N)=\{(x,y)\mid(x,y)=\pi(p_i)\text{ for some }p_i\in N\}.$$

We shall now present a formal definition of an instance of the region routing problem and a solution to an instance of the region routing problem.

Definition 4.18:

An instance of the *region routing problem* is defined to be an ordered triple, $RRP=(M,PS,Nets)$, such that:

1) $M=(R,\sigma,\rho)$ denotes a wiring model.
2) $PS=(P,\Pi)$ denotes a pin structure in R.
3) $Nets=\{N_1,\ldots,N_n\}$ denotes a set of n nets under PS, for some integer n, such that $\cup\{N_i\mid 1\leq i\leq n\}=P$ and $N_i\cap N_j=\emptyset$ for all i,j such that $1\leq i,j\leq n$ and $i\neq j$.

Thus the set of nets defines a partition of the pin set P.

Definition 4.19:

Let $RRP=(M,PS,Nets)$ denote any instance of the region routing problem with $M=(R,\sigma,\rho)$, $PS=(P,\Pi)$, and $Nets=\{N_1,\ldots,N_n\}$. Then a *solution* to RRP consists of

1) A legal placement $\pi\in\Pi$ of the pin set P in R.
2) A set of n pairwise compatible trees $T_1,\ldots,T_n$ under M, such that T_i interconnects $\pi(N_i)$ in R. The wiring tree T_i is said to *implement* net N_i in R.

We now note that a given instance of the region routing problem may in fact have no solution. This would be the case, for example, if the associated routing region had insufficient length or height to accomodate the necessary wiring. Thus a region routing problem is essentially a decision problem [Gar79] for which we must either compute a solution or determine that none exists.

It should be clear that instances of the region routing problem can occur in various forms depending on the type of wiring model employed, the type of pin structure employed, and the character of the associated net set. In the next section we shall describe three classes of region routing problems which lead to formalizations of the well known river routing problem, channel routing problem, and fixed-pin routing problem. The remainder of this chapter will then be devoted to the presentation of new algorithms for solving various versions of each of these problems.

4.3 Restricted Region Routing Problems.

In this section we show how several well known signal routing problems can be formulated as a restricted region routing problem. We then discuss the implications of altering the wiring model associated with a problem instance.

4.3.1 The River Routing Problem.

Assume we are given a rectangular region with fixed dimensions, a set of n physical connection points $\{a_1,\ldots,a_n\}$ with distinct fixed locations ordered from left-to-right along the top of the region, and a set of n physical connection points $\{b_1,\ldots,b_n\}$ with distinct fixed locations ordered from left-to-right along the bottom of the region; as shown in Figure 4.6. We now consider the problem of generating n wires, satisfying some specified set of wiring rules, such that each wire connects a distinct pair of points (a_i, b_i) and lies entirely within the rectangular region. This wiring problem is commonly called the river routing problem and has received a great deal of attention in the recent literature [Dol81, DS81, Lei81, Tom80]. We shall now present a formal definition of an instance of the river routing problem by simply formulating it as an instance of the region routing problem.

Definition 4.20:

An instance of the *river routing problem* is an ordered triple, $RVRP=(M, PS, Nets)$, such that:

1) $M=(R, \sigma, \rho)$ denotes a wiring model in which $R=(l, h, \Lambda, V, E)$.

2) $PS=(P, \Pi)$ denotes a pin structure in R with $P=\{p_1, \ldots, p_{2n}\}$, for some integer n, and $\Pi=\{\pi\}$ consisting of a single legal placement which satisfies the properties:

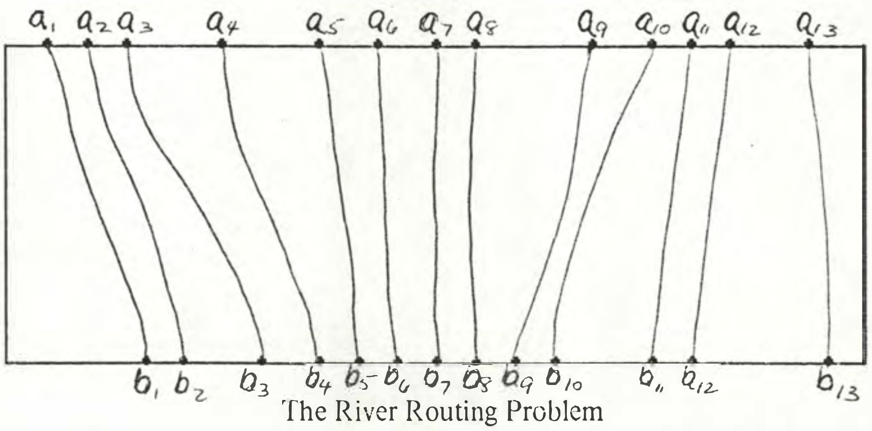
   a) $\pi^y(p_i)=h+1$ for all i $(1 \leq i \leq n)$.

   b) $\pi^y(p_i)=0$ for all i $(n+1 \leq i \leq 2n)$.

   c) $\pi^x(p_i) < \pi^x(p_j)$ for all i, j such that $1 \leq i < j \leq n$ or $n+1 \leq i < j \leq 2n$.

3) $Nets=\{N_1, \ldots, N_n\}$ denotes a set of n nets under PS with $N_i=\{p_i, p_{n+i}\}$ for all i $(1 \leq i \leq n)$.

Notice that $RVRP$ is also an instance of the region routing problem.


a_1 a_2 a_3 a_4 a_5 a_6 a_7 a_8 a_9 a_{10} a_{11} a_{12} a_{13}
b_1 b_2 b_3 b_4 b_5 b_6 b_7 b_8 b_9 b_{10} b_{11} b_{12} b_{13}

The River Routing Problem

Figure 4.6

For the case where R is a rectilinear routing region containing $\Lambda=1$ conducting layer, it has been shown [Dol81] that there exist very efficient polynomial-time algorithms for solving the river routing problem. In Section 4.4 we shall extend this result to include wiring models for which $\Lambda>1$ and $\{NEO, LPT\}\cap\rho=\emptyset$. Furthermore, we will show that this result is independent of whether or not LPT belongs to ρ. Thus we can always achieve a solution in which no wiring tree contains internal contact points. We should point out that there is no known polynomial-time algorithm for solving the river routing problem under the more common, but no more realistic, Thompson [Tho80] wiring model in which $\Lambda=2$ and $LPD\in\rho$.

4.3.2 The Channel Routing Problem.

Let us now consider a somewhat more general routing problem in which the nets are essentially composed of arbitrary subsets of some set of physical connection points with distinct fixed locations along the top and bottom of a rectangular region. In addition, each net may contain at most two physical connection points, one constrained to lie along the left side of a region and the other constrained to lie along the right, whose exact locations are unspecified and may be determined as a function of the required wiring. Such a problem is generally known as a channel routing problem [Has71, Deu76]. We now present a formal definition of an instance of the channel routing problem.

Definition 4.21:

An instance of the *channel routing problem* is an ordered triple, $CRP=(M, PS, Nets)$, such that:

1) $M=(R,\sigma,\rho)$ denotes a wiring model in which $R=(l,h,\Lambda,V,E)$.

2) $PS=(P,\Pi)$ denotes a pin structure in R with $\{P_{top}, P_{bottom}, P_{left}, P_{right}\}$ defining a partition of P such that $|P_{top}|\leq l$, $|P_{bottom}|\leq l$, $|P_{left}|\leq h$, and $|P_{right}|\leq h$. Further, $\Pi=\{\pi_1, \ldots, \pi_r\}$ denotes the set of *all* legal placements of the pin set P in R with:

   a) $\pi(p)=(x_p, h+1)$ for all $\pi\in\Pi$ and $p\in P_{top}$.

   b) $\pi(p)=(x_p, 0)$ for all $\pi\in\Pi$ and $p\in P_{bottom}$.

   c) $\pi^x(p)=0$ for all $\pi\in\Pi$ and $p\in P_{left}$.

d) $\pi^x(p)=l+1$ for all $\pi\in\Pi$ and $p\in P_{right}$.

3) $Nets=\{N_1,\ldots,N_n\}$ denotes a set of n nets under PS, for some integer n, with $\cup\{N_i \mid 1\leq i\leq n\}=P$, $|N_i\cap P_{left}|\leq 1$ for all i $(1\leq i\leq n)$, $|N_i\cap P_{right}|\leq 1$ for all i $(1\leq i\leq n)$, and $N_i\cap N_j=\emptyset$ for all i,j such that $1\leq i,j\leq n$ and $i\neq j$.

Notice that *CRP* is also an instance of the region routing problem.

The general channel routing problem has recently been proven NP-complete for the case in which $M=(R,\sigma,\rho)$ denotes a two-layer rectilinear wiring model with $\sigma=1$ and $LPD\in\rho$ [Szy81]. Furthermore, there are no known polynomial-time algorithms guaranteed to compute a solution to such a problem whenever the associated routing region has height within some constant times the minimum necessary for a solution to exist. Therefore, in Section 4.5 we will consider algorithms for solving various restricted versions of the channel routing problem under somewhat relaxed, but still realistic, two-layer wiring models in which $LPD\notin\rho$. In particular, we will present a new class of algorithms that are guaranteed to correctly solve an instance of such a problem if the associated routing region has height at least four times the minimum necessary for a solution to exist. We note that the complexity of the general channel routing problem under these *relaxed* wiring models is still open (i.e., NP-completeness has not been proven).

4.3.3 The Fixed-Pin Routing Problem.

The last type of region routing problem we shall deal with in this chapter is called the fixed-pin routing problem. This problem is exactly the one encountered in the last phase of the PI routing system where nets are simply composed of subsets of a set of physical connection points with distinct fixed locations along the perimeter of some rectangular region. An instance of the fixed-pin routing problem is easily formulated as an instance of the region routing problem in the following manner.

Definition 4.22:

An instance of the *fixed-pin routing problem* is an ordered triple, $FPRP=(M,PS,Nets)$, such that:

1) $M=(R,\sigma,\rho)$ denotes a wiring model.

2) $PS=(P,\Pi)$ denotes a pin structure in R with $|\Pi|=1$.

3) $Nets=\{N_1, \ldots, N_n\}$ denotes a set of n nets under PS, for some integer n, with $\cup\{N_i \mid 1 \leq i \leq n\}=P$ and $N_i \cap N_j=\emptyset$ for all i,j such that $1 \leq i,j \leq n$ and $i \neq j$.

Notice that *FPRP* is also an instance of the region routing problem.

In Section 4.6 we will turn our attention from theory to practice and consider an unusual, but practical, heuristic algorithm for the fixed-pin routing problem. This algorithm, called the *quick routing* algorithm, is based on the assumption that many of the instances of the fixed-pin routing problem which arise in practice can be solved by a very simple set of heuristics. The philosophy underlying the development of the quick routing algorithm was discussed in Section 2.6. Thus Section 4.6 will consist of a detailed description of the algorithm itself.

4.3.4 Wiring Model Considerations.

The remainder of this chapter will deal with new algorithms for solving various versions of the river routing problem, the channel routing problem, and the fixed-pin routing problem. For the most part we will assume that the wiring model associated with each problem instance is based on a rectilinear routing region and a minimum separation distance of $\sigma=1$. Wiring models of this type are desirable for two important reasons. First of all, they closely reflect the most widely supported design styles; allowing only vertical and horizontal interconnect lines [Mea79]. Thus they are realistic and can be used to develop algorithms that are of practical as well as theoretical interest. The second reason for considering models of this type is that the minimum separation constraint for compatible wiring trees takes on a very clean form as demonstrated in the following lemma.

Lemma 4.1:

Let $M=(R,\sigma,\rho)$ denote a wiring model in which $R=(l,h,\Lambda,V,E)$ is a rectilinear routing region and $\sigma=1$. Further, let T_1 and T_2 denote two wiring trees in R. Then for each integer k $(1 \leq k \leq \Lambda)$, the minimum distance between the points in $span_k(T_1)$ and the points in $span_k(T_2)$ will be at least σ if and only if $span_k(T_1) \cap span_k(T_2)=\emptyset$.

Proof:

The proof of Lemma 4.1 follows directly from the simple observation that two line segments corresponding to routing edges in a rectilinear routing region will have minimum distance strictly less than $\sigma = 1$ if and only if they intersect.

□

Let us now consider wiring models based on augmented rectilinear routing regions. Recall that such models allow diagonal as well as vertical and horizontal interconnect lines. Although the most widely supported design styles are those allowing only vertical and horizontal interconnect lines, there is growing support for the incorporation of diagonal lines. Furthermore, it appears that significant savings in wiring area are possible under such design styles. Thus there is practical justification for employing wiring models based on augmented rectilinear routing regions. Notice that these models will be most useful when $\sigma = \sqrt{2}/2$ and the following analog of Lemma 4.1 can be applied.

Lemma 3.2:

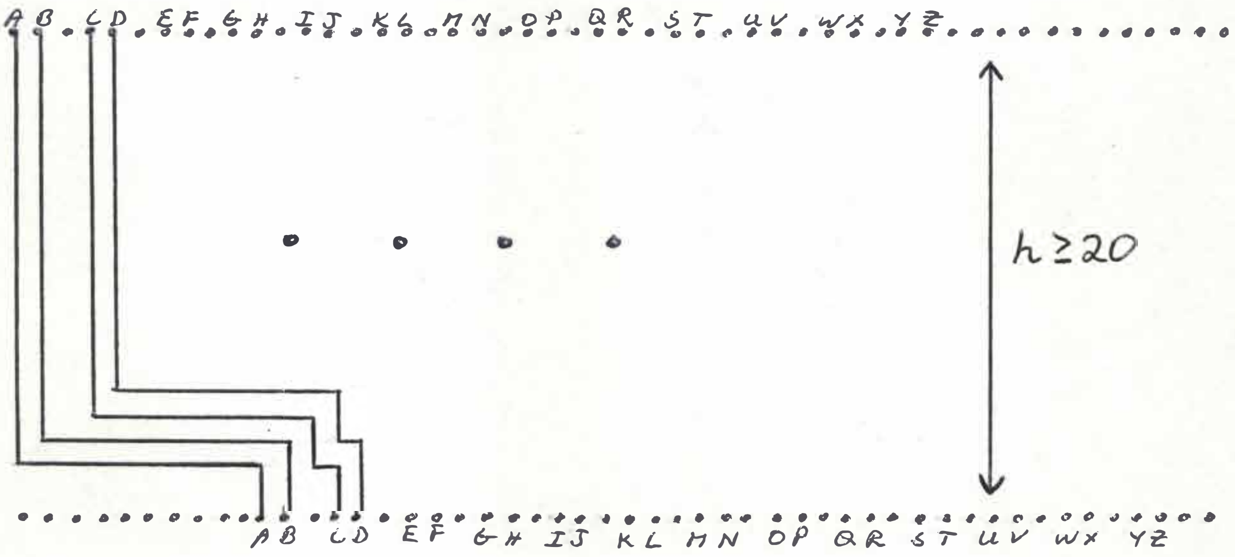
Let $M = (R, \sigma, \rho)$ denote a wiring model in which $R = (l, h, \Lambda, V, E)$ is an augmented rectilinear routing region and $\sigma = \sqrt{2}/2$. Further, let T_1 and T_2 denote two wiring trees in R. Then for each integer k $(1 \leq k \leq \Lambda)$, the minimum distance between the points in $span_k(T_1)$ and the points in $span_k(T_2)$ will be at least σ if and only if $span_k(T_1) \cap span_k(T_2) = \emptyset$.

Proof:

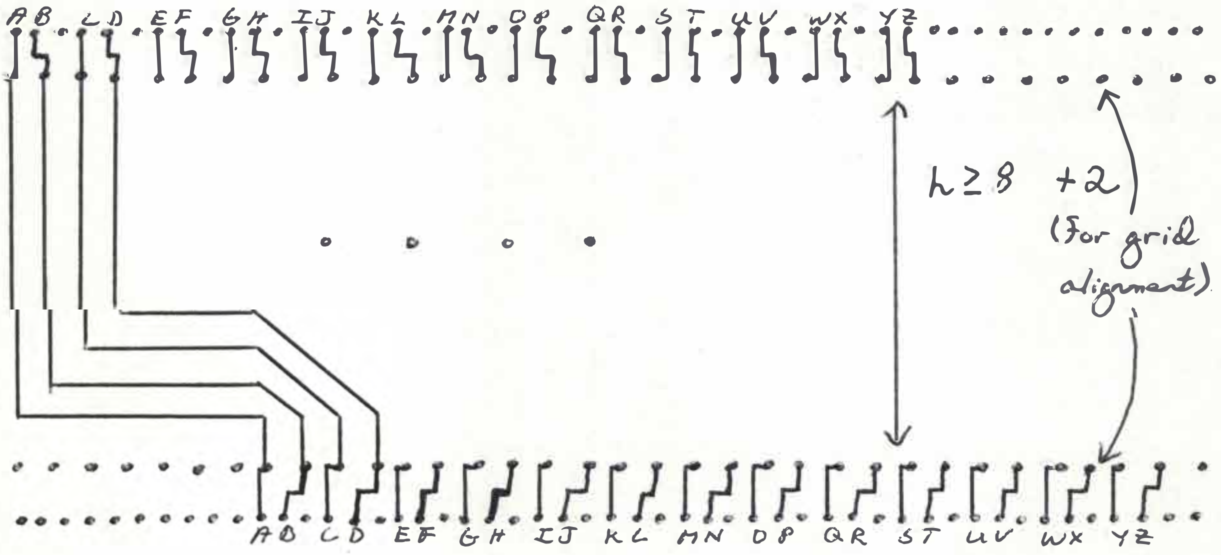
Once again the proof of Lemma 4.2 follows directly from the simple observation that two line segments corresponding to routing edges in an augmented rectilinear routing region will have minimum distance strictly less than $\sigma = \sqrt{2}/2$ if and only if they intersect.

□

We should now point out that wiring models based on augmented rectilinear routing regions and a minimum separation distance of $\sigma = \sqrt{2}/2$ have practical application whenever a grid size of $\sqrt{2}$ times the specified minimum separation distance is acceptable. In practical terms this means that such wiring models will be useful whenever the physical connection points for a given routing problem are located with less than (roughly) 2/3 maximum density along the perimeter of the given rectangular region. More specifically, there should be no more than two physical connection points located within any interval of length $2(\sqrt{2}\sigma)$ along the perimeter of the region. However, for such less dense routing problems we achieve substantial wiring improvements by employing these models and thus such models will also be considered in the following sections. (See Figure 4.7)


AB CD EF GH IJ KL MN OP QR ST UV WX YZ
h ≥ 20
AB CD EF GH IJ KL MN OP QR ST UV WX YZ

Rectilinear Routing Region


AB CD EF GH IJ KL MN OP QR ST UV WX YZ
h ≥ 8 +2
(For grid alignment)
AB CD EF GH IJ KL MN OP QR ST UV WX YZ

Augmented Rectilinear Routing Region

Figure 4.7

4.4 Optimal River Routing Algorithms.

In this section we present efficient polynomial-time algorithms for solving any given instance of the river routing problem under two different multi-layer wiring models. Moreover, we show that the power of these wiring models for river routing is unaffected by enforcement of wiring restriction *LPT* (i.e., whether or not we allow wiring trees to be composed of edges from more than one conducting layer). Thus there is no benefit in generating wiring trees which contain internal contact points. The results presented here extend the work of Tompa [Tom80] and Dolev and others [Dol81] for river routing under single-layer wiring models. Tompa uses a general grid-free wiring model and Dolev and others use a standard grid-based rectilinear wiring model.

4.4.1 The Rectilinear Wiring Model.

The main result to be presented in this section is a necessary and sufficient condition for any given instance of the river routing problem to have a solution under a general multi-layer rectilinear wiring model. In addition, the sufficiency part of this result will yield an efficient polynomial-time algorithm for actually computing a solution if one exists. Our basic approach is similar to that taken in [Dol81] for single layer wiring models. We begin by defining some notation.

Let $RVRP=(M,PS,Nets)$ denote any instance of the river routing problem with $M=(R,\sigma,\rho)$, $PS=(P,\Pi)$, and $Nets=\{N_1,\ldots,N_n\}$. Within this section we will assume that $R=(l,h,\Lambda,V,E)$ denotes a rectilinear routing region, that $\sigma=1$, and that $\rho=\emptyset$. Thus we are dealing with a general multi-layer rectilinear wiring model.

We now recall that each net $N_i \in Nets$ is defined by exactly two pins; $p_i \in P$ and $p_{n+i} \in P$. Further, the pin p_i is constrained to lie at a particular pin location point along the top of the routing region R and the pin p_{n+i} is constrained to lie at a particular pin location point along the bottom of R. We now let a_i denote the x-coordinate of the point at which p_i is constrained to lie and we let b_i denote the x-coordinate of the point at which p_{n+i} is constrained to lie. Thus the ordered pair (a_i,b_i) completely specifies the net N_i and so these two notations can be used interchangeably. In addition, the integer a_i will be called the *entry coordinate* of the net N_i and integer b_i will be called the *exit coordinate* of the net N_i. Finally, the net N_i will be called a *falling* net if $a_i < b_i$, a *trivial* net if $a_i = b_i$, and a *rising* net if $a_i > b_i$.

Now for each integer i $(1 \leq i \leq n)$ and each integer τ $(\tau \geq 0)$, let $G_{\tau,i}$ denote the number of distinct nets whose implementation in R must intersect a line drawn either from point $(b_i, 1)$ to point $(b_i+\tau-1, h)$ if N_i is a falling net, or from point (a_i, h) to point $(a_i+\tau-1, 1)$ if N_i is not a falling net. Thus the value of $G_{\tau,i}$ is defined as follows: (cf., [Dol81])

$$G_{\tau,i} = \begin{cases} |\{(a_j, b_j) \mid a_i \leq a_j \leq b_i+\tau-1\}| & \text{if } a_i < b_i \\ |\{(a_j, b_j) \mid b_i \leq b_j \leq a_i+\tau-1\}| & \text{if } a_i \geq b_i. \end{cases} \tag{4.1}$$

Next we define a sequence of integers $(\tau_1, \ldots, \tau_n)$ such that τ_i $(1 \leq i \leq n)$ denotes the smallest non-negative integer which satisfies the relation:

$$\Lambda \cdot \tau_i \geq G_{\tau_i, i}. \tag{4.2}$$

Notice that such a sequence of integers must always exist since $0 \leq G_{\tau,i} \leq n$ for all i $(1 \leq i \leq n)$ and all τ $(\tau \geq 0)$. Further, we have the following lemma.

Lemma 4.3:

Let i denote any integer such that $1 \leq i \leq n$ and let τ' denote any integer such that $\tau' \geq \tau_i$. Then the integer τ' must satisfy the relation $\Lambda \cdot \tau' \geq G_{\tau', i}$.

Proof:

We prove Lemma 4.3 inductively by simply showing that $\Lambda(\tau+1) \geq G_{\tau+1,i}$ for any non-negative integer τ such that $\Lambda \cdot \tau \geq G_{\tau,i}$:

$$\begin{aligned} \Lambda \cdot \tau \geq G_{\tau,i} &\Rightarrow \Lambda \cdot \tau + \Lambda \geq G_{\tau,i} + \Lambda \\ &\Rightarrow \Lambda(\tau+1) \geq G_{\tau,i} + 1 \\ &\Rightarrow \Lambda(\tau+1) \geq G_{\tau+1,i}. \end{aligned}$$

□

We are now ready to state and prove the two main theorems of this section. The first theorem yields a necessary condition for $RVRP$ to have a solution. The second theorem then shows that this condition is also sufficient.

Theorem 4.1:

RVRP has *no* solution if $h<\tau_{max}$, where $\tau_{max}=\text{Max}\{\tau_i \mid 1\leq i\leq n\}$ and h denotes the height of the associated routing region R.

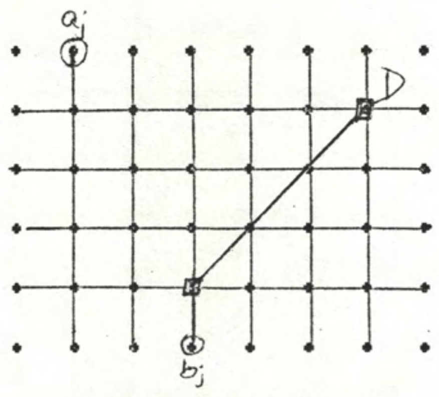
Proof:

Let j denote any integer such that $1\leq j\leq n$ and $\tau_{max}=\tau_j$. Further, assume that $h<\tau_{max}=\tau_j$ and that RVRP has a solution. We then have two possible cases to consider; $a_j<b_j$ or $a_j\geq b_j$. If $a_j<b_j$ then consider the diagonal line D drawn from grid point $(b_j,1)$ to grid point (b_j+h-1,h) as shown in Figure 4.8. Clearly no more than $\Lambda\cdot h$ distinct wiring trees can intersect D in any solution to *RVRP*. However, $G_{h,j}$ denotes the minimum number of distinct wiring trees which must intersect D in any solution to *RVRP*. Thus we must have the relation:

$$\Lambda\cdot h\geq G_{h,j}.$$

Combining this with the definition of τ_j we obtain the result $h\geq\tau_j$ and thus we have a contradiction. A similar contradiction argument holds for the case when $a_j\geq b_j$.

□



Diagonal Constraint Line D
Figure 4.8

Theorem 4.2:

RVRP has a solution if $h \geq \tau_{max}$, where $\tau_{max} = \text{Max}\{\tau_i \mid 1 \leq i \leq n\}$ and h denotes the height of the associated routing region R.

Proof:

We shall prove Theorem 4.2 constructively by presenting an algorithm which is guaranteed to compute a solution to *RVRP* if $h \geq \tau_{max}$. Since *RVRP* is an instance of the river routing problem, we recall that there exists exactly one legal placement $\pi \in \Pi$ of the pin set P. Thus our algorithm need only compute the required set of compatible wiring trees.

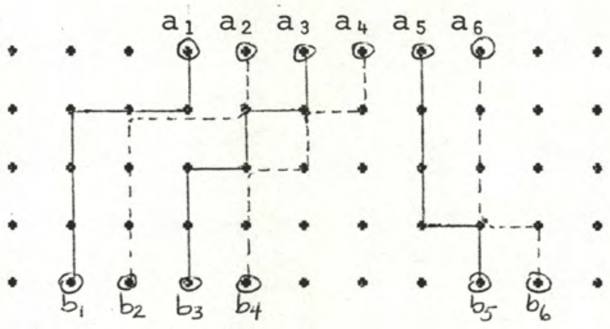
Algorithm 4.1:

This algorithm proceeds net by net, in order of increasing net entry coordinate, generating a complete single layer wiring tree for each net in turn. Further, the wiring tree for any given net (a_i, b_i) is generated by a simple "greedy" algorithm as follows:

1) If $a_i < b_i$ then the algorithm selects a conducting layer k which is not being used by any wiring tree containing grid point (a_i, h). The algorithm then generates a single layer wiring tree for net (a_i, b_i), utilizing conducting layer k, by beginning at grid point $(a_i, h+1)$ and extending to grid point $(b_i, 0)$. This extension is performed by alternating between vertical extension downward as far as possible and then horizontal extension to the right only until vertical extension downward is once again possible.

2) If $a_i \geq b_i$ then a similar procedure is employed. In this case the algorithm selects a conducting layer which is not being used by any wiring tree containing grid point $(b_i, 1)$ and then the tree for (a_i, b_i) is generated from grid point $(b_i, 0)$ to grid point $(a_i, h+1)$. The extension alternates between vertical upward (as far as possible) and horizontal to the right.

An example of a set of wiring trees that might be generated by Algorithm 4.1 is shown in Figure 4.9. It should now be clear that Algorithm 4.1 will generate only wiring trees which are mutually compatible in R. Furthermore, any pair of these trees which intersect a common



Wiring Trees Generated by Algorithm 4.1
Figure 4.9

column in R must either both be implementations of falling nets or both be implementations of rising nets. Thus there is no *interaction* between trees implementing nets of different type (i.e., falling, trivial, or rising). Finally, we note that the only way in which this algorithm can terminate unsuccessfully is by reaching a net for which there is no available conducting layer. This gives rise to the following lemma.

Lemma 4.4:

If Algorithm 4.1 fails to compute a solution to *RVRP* then $h < \tau_{max}$.

Proof:

If Algorithm 4.1 fails to compute a solution to *RVRP* then there must exist some net (a_j, b_j) such that when Algorithm 4.1 begins processing (a_j, b_j) there is no available conducting layer. We now have three possible cases to consider; $a_j < b_j$, $a_j = b_j$, or $a_j > b_j$. If $a_j = b_j$, then clearly there can be no wiring tree containing grid point $(b_j, 1)$ when Algorithm 4.1 begins processing (a_j, b_j). Thus we have a contradiction.

If $a_j < b_j$, then it must be the case that there are exactly Λ distinct wiring trees containing grid point (a_j, h) when Algorithm 4.1 begins processing (a_j, b_j). Further, each of these wiring trees must lie on a distinct conducting layer and must be the implementation of a falling net. Let us now consider a diagonal line D drawn from grid point $(a_j - h + 1, 1)$ to grid point (a_j, h). Due to the fact that Algorithm 4.1 extends the wiring trees for falling nets vertically downward whenever possible, it follows that every grid point which intersects D must also be contained in exactly Λ

distinct wiring trees; each corresponding to a falling net. Furthermore, these $\Lambda \cdot h$ wiring trees which intersect D must all be distinct and must all be implementations of (falling) nets (a_i, b_i) with:

$$a_i < a_j \text{ and } b_i \geq a_j - h + 1. \tag{4.3}$$

Now let (a_z, b_z) denote the (falling) net with the smallest exit coordinate not less than $a_j - h + 1$. It then follows that each of the $\Lambda \cdot h$ nets which were shown to satisfy (4.3) must also satisfy:

$$a_z \leq a_i \leq b_z + h - 1. \tag{4.4}$$

In addition, net (a_j, b_j), which is distinct from each of the $\Lambda \cdot h$ nets shown to satisfy (4.4), must also satisfy (4.4). Thus it follows that:

$$G_{h,z} \geq \Lambda \cdot h + 1 \Rightarrow \Lambda \cdot h < G_{h,z}.$$

Combining this with (4.2) and Lemma 4.3 we obtain the result $h < \tau_z \leq \tau_{max} \Rightarrow h < \tau_{max}$.

Finally, if $a_j > b_j$ then we obtain the same result by a similar argument on rising nets. This completes the proof of both Lemma 4.4 and Theorem 4.2.

□

We have now shown that the relation $h \geq \tau_{max}$, where $\tau_{max} = \text{Max}\{\tau_i \mid 1 \leq i \leq n\}$, is both a necessary and sufficient condition for *RVRP* to have a solution. Furthermore, we have presented an algorithm, Algorithm 4.1, which is guaranteed to compute a solution to *RVRP* if one exists. Let us now consider the following interesting corollary to Theorem 4.2.

Corollary 4.1:

If *RVRP* has a solution, then it has a solution in which no wiring tree contains internal contact points.

Proof:

The proof of this corollary follows immediately from the fact that Algorithm 4.1 generates only single layer wiring trees.

□

Corollary 4.1 tells us that the power of the general multi-layer rectilinear wiring model for river routing is unaffected by enforcement of wiring restriction *LPT* (i.e., whether or not $LPT \in \rho$). Thus there is no benefit in generating wiring trees which have internal contact points.

4.4.2 The Augmented Rectilinear Wiring Model.

In this section we develop a necessary and sufficient condition for an instance of the river routing problem to have a solution under a general multi-layer diagonally augmented wiring model. In addition, the sufficiency part of this result will once again yield an efficient polynomial-time algorithm for actually computing a solution if one exists. The approach closely parallels that followed in Section 4.4.1.

Let $RVRP=(M,PS,Nets)$ denote any instance of the river routing problem with $M=(R,\sigma,\rho)$, $PS=(P,\Pi)$, and $Nets=\{N_1,\ldots,N_n\}$. Within this section we will assume that $R=(l,h,\Lambda,V,E)$ denotes an augmented rectilinear routing region, that $\sigma=\sqrt{2}/2$, and that $\rho=\emptyset$. Thus we are dealing with a general multi-layer diagonally augmented wiring model. Further, each net $N_i \in Nets$ may once again be denoted by an ordered pair (a_i,b_i), where a_i specifies the entry coordinate of the net N_i and b_i specifies the exit coordinate of the net N_i. The notions of a falling net, a trivial net, and a rising net are then defined as in the previous section.

Now for each integer i $(1\leq i\leq n)$, let H_i denote the number of distinct nets whose implementation in R must intersect a line drawn either from point $(b_i,1)$ to point (b_i,h) if N_i is a falling net, or from point (a_i,h) to point $(a_i,1)$ if N_i is not a falling net. Thus the value H_i is defined as follows:

$$H_i=\begin{cases}|\{(a_j,b_j)\ \mid\ a_i\leq a_j\leq b_i\}| & \text{if } a_i<b_i\\ \\ |\{(a_j,b_j)\ \mid\ b_i\leq b_j\leq a_i\}| & \text{if } a_i\geq b_i.\end{cases} \tag{4.5}$$

Next we define a sequence of integers $(\gamma_1,\ldots,\gamma_n)$ such that γ_i $(1\leq i\leq n)$ denotes the smallest non-negative integer which satisfies the relation:

$$\Lambda\cdot\gamma_i\geq H_i. \tag{4.6}$$

Notice that such a sequence of integers must always exist since $0\leq H_i\leq n$ for all i $(1\leq i\leq n)$. Further, we have the following lemma.

Lemma 4.5:

Let i denote any integer such that $1\leq i\leq n$ and let γ' denote any integer such that $\gamma'\geq\gamma_i$. Then the integer γ' must satisfy the relation $\Lambda\cdot\gamma'\geq H_i$.

Proof:

The proof of Lemma 4.5 follows directly from (4.6) and the fact that H_i denotes a fixed integer which is independent of γ'.

□

We now have the following two theorems which extend the results of Section 4.4.1 to include diagonally augmented wiring models.

Theorem 4.4:

RVRP has no solution if $h<\gamma_{max}$, where $\gamma_{max}=\text{Max}\{\gamma_i \mid 1\leq i\leq n\}$ and h denotes the height of the associated augmented rectilinear routing region R.

Proof:

Let j denote any integer such that $1\leq j\leq n$ and $\gamma_{max}=\gamma_j$. Further, assume that $h<\gamma_{max}=\gamma_j$ and that *RVRP* has a solution. We then have two possible cases to consider; $a_j<b_j$ or $a_j\geq b_j$. If $a_j<b_j$ then consider the vertical line D drawn from grid point $(b_j,1)$ to grid point (b_j,h) as shown in Figure 4.10. Clearly no more than $\Lambda\cdot h$ distinct wiring trees can intersect D in any solution to *RVRP*. However, H_j denotes the minimum number of distinct wiring trees which must intersect D in any solution to *RVRP*. Thus we must have the relation:

$$\Lambda\cdot h\geq H_j.$$

Combining this with the definition of γ_j we obtain the result $h\geq\gamma_j$ and thus we have a contradiction. A similar contradiction argument holds for the case when $a_j>b_j$.

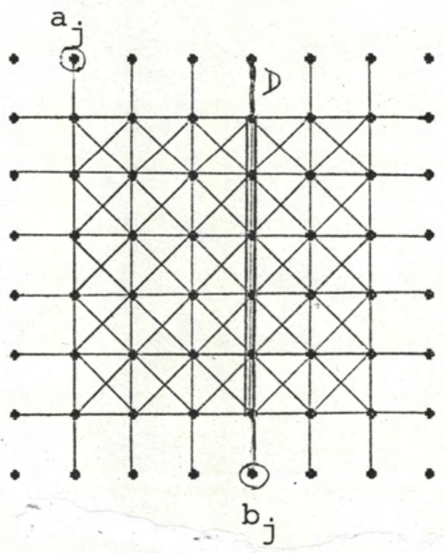
□

Theorem 4.5:

RVRP has a solution if $h\geq\gamma_{max}$, where $\gamma_{max}=\text{Max}\{\gamma_i \mid 1\leq i\leq n\}$ and h denotes the height of the associated augmented rectilinear routing region R.

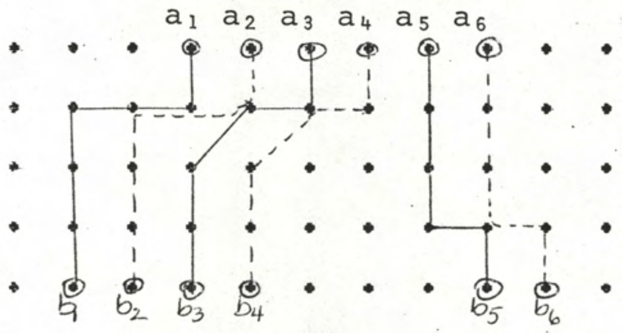
Proof:

We shall prove Theorem 4.5 constructively by presenting an algorithm which is guaranteed to compute a solution to *RVRP* if $h\geq\gamma_{max}$. Since *RVRP* is an instance of the river routing problem, we once again recall that there exists exactly one legal placement $\pi\in\Pi$ of the pin set P. Thus our algorithm need only compute the required set of compatible wiring trees.



Vertical Constraint Line D

Figure 4.10



a_1 a_2 a_3 a_4 a_5 a_6
b_1 b_2 b_3 b_4 b_5 b_6

Wiring Trees Generated by Algorithm 4.2

Figure 4.11

Algorithm 4.2:

This algorithm proceeds net by net, in order of increasing net entry coordinate, generating a complete single layer wiring tree for each net in turn. Further, the wiring tree for any given net (a_i, b_i) is generated by a simple "greedy" algorithm as follows:

1) If $a_i < b_i$ then the algorithm selects a conducting layer k which is not being used by any wiring tree containing grid point (a_i, h). The algorithm then generates a single layer wiring tree for net (a_i, b_i), utilizing conducting layer k, by beginning at the grid point $(a_i, h+1)$ and extending to grid point $(b_i, 0)$. This extension is performed by alternating between downward extension (vertical preferred over diagonal) as far as possible and then horizontal extension to the right only until downward extension is once again possible.

2) If $a_i \geq b_i$ then a similar procedure is employed. In this case the algorithm selects a conducting layer which is not being used by any wiring tree containing grid point $(b_i, 1)$ and then the tree for (a_i, b_i) is generated from grid point $(b_i, 0)$ to grid point $(a_i, h+1)$. The extension alternates between upward (as far as possible) and horizontal to the right.

An example of a set of wiring trees that might be generated by Algorithm 4.2 is shown in Figure 4.11. In light of Lemma 4.2 it should now be clear that Algorithm 4.2, like Algorithm 4.1, will generate only wiring trees which are mutually compatible in R. Furthermore, any pair of these trees which intersect a common column in R must either both be implementations of falling nets or both be implementations of rising nets. Thus there is no interaction between trees implementing nets of different type (i.e., falling, trivial, or rising). Finally, we note that the only way in which this algorithm can terminate unsuccessfully is by reaching a net for which there is no available conducting layer. This gives rise to the following lemma.

Lemma 4.6:

If Algorithm 4.2 fails to compute a solution to *RVRP* then $h < \gamma_{max}$.

Proof:

If Algorithm 4.2 fails to compute a solution to *RVRP* then there must exist some net (a_j, b_j) such that when Algorithm 4.2 begins processing (a_j, b_j) there is no available conducting layer. We now have three possible cases to consider; $a_j < b_j$, $a_j = b_j$, or $a_j > b_j$. If $a_j = b_j$, then clearly there can be no wiring tree containing grid point $(b_j, 1)$ when Algorithm 4.2 begins processing (a_j, b_j). Thus we have a contradiction.

If $a_j < b_j$, then it must be the case that there are exactly Λ distinct wiring trees containing grid point (a_j, h) when Algorithm 4.2 begins processing (a_j, b_j). Further, each of these wiring trees must lie on a distinct conducting layer and must be the implementation of a falling net. Let us now consider a vertical line D drawn from grid point (a_j, h) to grid point $(a_j, 1)$. Due to the fact that Algorithm 4.2 extends the wiring trees for falling nets downward whenever possible, it follows that every grid point which intersects D must also be contained in exactly Λ distinct wiring trees; each corresponding to a falling net. Furthermore, these $\Lambda \cdot h$ wiring trees which intersect D must all be distinct and must all be implementations of (falling) nets (a_i, b_i) with:

$$a_i < a_j \text{ and } b_i \geq a_j. \tag{4.7}$$

Now let (a_z, b_z) denote the (falling) net with the smallest exit coordinate not less than a_j. It then follows that each of the $\Lambda \cdot h$ nets which were shown to satisfy (4.7) must also satisfy:

$$a_z \leq a_i \leq b_z. \tag{4.8}$$

In addition, net (a_j, b_j), which is distinct from each of the $\Lambda \cdot h$ nets shown to satisfy (4.8), must also satisfy (4.8). Thus it follows that:

$$H_z \geq \Lambda \cdot h + 1 \Rightarrow \Lambda \cdot h < H_z.$$

Combining this with (4.6) and Lemma 4.5 we obtain the result $h < \gamma_z \leq \gamma_{max} \Rightarrow h < \gamma_{max}$.

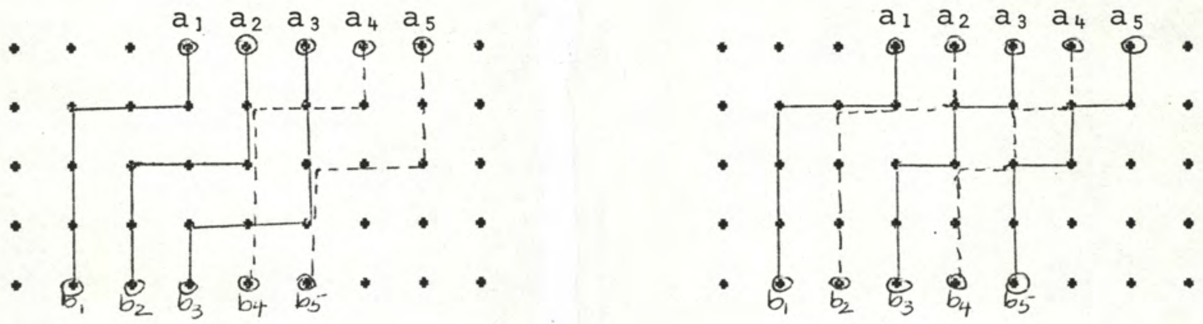
Finally, if $a_j > b_j$ then we obtain the same result by a similar argument on rising nets. This completes the proof of both Lemma 4.6 and Theorem 4.5.

□

We have now shown that the relation $h \geq \gamma_{max}$, where $\gamma_{max} = \text{Max}\{\gamma_i \mid 1 \leq i \leq n\}$, is both a necessary and sufficient condition for *RVRP* to have a solution when a general multi-layer diagonally augmented wiring model is employed. Furthermore, we have presented as algorithm, Algorithm 4.2, which is guaranteed to compute a solution to *RVRP* if one exists. In addition, it should now be clear that an analog of Corollary 4.1 can be proven under the diagonally augmented rectilinear wiring model and thus once again there is no benefit in generating wiring trees which have internal contact points.

4.4.3 Systematic Layer Selection.

The river routing algorithms presented in the previous two sections both allow conducting layers to be assigned to wiring trees in a more or less arbitrary manner as long as certain constraints are satisfied. Thus each of these algorithms can in fact be used to compute a wide variety of solutions to a given instance of the river routing problem (see Figure 4.12). Notice, however, that the constraints on layer selection are determined dynamically as wiring trees are generated and thus there is no way of knowing a priori how conducting layers will be assigned to wiring trees. Although these procedures allow flexibility, they do so at the cost of reduced efficiency. (e.g., How do we efficiently determine which conducting layers are available for assignment to a given wiring tree?) Thus there is justification for developing a more deterministic method of layer assignment. In this section we present a simple class of systematic layer selection techniques which can always be employed. We begin by reformulating the results of Sections 4.4.1 and 4.4.2 as suggested by C. Leiserson and R. Pinter [Lei81].



A Variety of River Routing Solutions
Figure 4.12

Lemma 4.7:

Let $RVRP=(M, PS, Nets)$ denote any instance of the river routing problem with a general multi-layer rectilinear wiring model as defined in Section 4.4.1. Further, let $G_{\tau,i}$ be as defined in (4.1) and let τ_i be as defined in (4.2). Then $RVRP$ has a solution if and only if $a_{i+\Lambda\cdot h}\geq b_i+h$ for each falling net $N_i\in Nets$ and $b_{i+\Lambda\cdot h}\geq a_i+h$ for each rising or trivial net $N_i\in Nets$.

Proof:

From Theorems 4.1 and 4.2 we know that $RVRP$ has a solution if and only if $h\geq\tau_i$ for each i ($1\leq i\leq n$). Further, by (4.2) and Lemma 4.3 we have that $h\geq\tau_i$ for each i ($1\leq i\leq n$) if and only if $\Lambda\cdot h\geq G_{h,i}$ for each i ($1\leq i\leq n$). Finally, applying (3.1) we obtain the result:

$$\Lambda\cdot h\geq G_{h,i} \text{ (for each } 1\leq i\leq n) \Leftrightarrow \begin{cases} a_{i+\Lambda\cdot h}\geq b_i+h \text{ for all } (a_i,b_i) \text{ such that } a_i<b_i \\ \\ b_{i+\Lambda\cdot h}\geq a_i+h \text{ for all } (a_i,b_i) \text{ such that } a_i\geq b_i. \end{cases}$$

□

Lemma 4.8:

Let $RVRP=(M, PS, Nets)$ denote any instance of the river routing problem with a general multi-layer diagonally augmented wiring model as defined in Section 4.4.2. Further, let H_i be as defined in (4.5) and let γ_i be as defined in (4.6). Then $RVRP$ has a solution if and only if $a_{i+\Lambda\cdot h}\geq b_i+1$ for each falling net $N_i\in Nets$ and $b_{i+\Lambda\cdot h}\geq a_i+1$ for each rising or trivial net $N_i\in Nets$.

Proof:

From Theorems 4.4 and 4.5 we know that $RVRP$ has a solution if and only if $h\geq\gamma_i$ for each i ($1\leq i\leq n$). Further, by (4.6) and Lemma 4.5 we have that $h\geq\gamma_i$ for each i ($1\leq i\leq n$) if and only if $\Lambda\cdot h\geq H_i$ for each i ($1\leq i\leq n$). Finally, applying (4.5) we obtain the result:

$$\Lambda\cdot h\geq H_i \text{ (for each } 1\leq i\leq n) \Leftrightarrow \begin{cases} a_{i+\Lambda\cdot h}\geq b_i+1 \text{ for all } (a_i,b_i) \text{ such that } a_i<b_i \\ \\ b_{i+\Lambda\cdot h}\geq a_i+1 \text{ for all } (a_i,b_i) \text{ such that } a_i\geq b_i. \end{cases}$$

□

Let us now return to the layer selection problem and show how these two lemmas can be utilized to prove the correctness of a very simple systematic layer assignment technique. Consider redefining Algorithms 4.1 and 4.2 in such a manner that the wiring tree implementing net N_i (for each $1 \leq i \leq n$) is always assigned conducting layer:

$$k = \lfloor [(i\text{-}1) \bmod \Lambda \cdot m] / m \rfloor + 1, \tag{4.9}$$

where m denotes any positive integer. Thus the first m nets are assigned conducting layer 1, the next m nets are assigned conducting layer 2, and so on until nets $N_{(\Lambda\text{-}1)m+1}$ through $N_{\Lambda m}$ are assigned conducting layer Λ; at which point the process repeats. In particular, for $\Lambda = 2$ and $m = 1$ all odd numbered nets are assigned conducting layer 1 and all even numbered nets are assigned conducting layer 2. Notice that such algorithms perform layer assignment in a systematic manner which is completely independent of net structure. Therefore, if we can prove that these new versions of Algorithms 4.1 and 4.2 are correct, we will have obtained our desired result. In the remainder of this section we will show that in fact the modified versions of Algorithms 4.1 and 4.2 will correctly compute a solution to any given instance of the river routing problem, under the appropriate wiring model, whenever m evenly divides the height of the associated routing region. Thus $m = 1$ will always yield correct algorithms.

Theorem 4.6:

Let $RVRP = (M, PS, Nets)$ denote any instance of the river routing problem under a general multi-layer rectilinear wiring model as defined in Section 4.4.1. Further, let Algorithm 4.1A be a modification of Algorithm 4.1 in which layer selection is performed as defined in (4.9); with m denoting any positive integer such that $m|h$. Then Algorithm 4.1A is guaranteed to compute a solution to $RVRP$ whenever one exists.

Proof:

Let m be any positive integer such that $m|h$. Further, for each integer k $(1 \leq k \leq \Lambda)$, let $RVRP_k = (M', PS_k, Nets_k)$ denote that single layer instance of the river routing problem in which M' is identical to M except for $\Lambda = 1$, $Nets_k \subseteq Nets$ includes exactly those nets in $RVRP$ which are associated with conducting layer k under (4.9), and PS_k includes exactly those pins in $RVRP$ which are associated with nets in $Nets_k$. Notice that Algorithm 4.1A is equivalent to Algorithm

4.1 whenever $\Lambda=1$ and thus by Theorems 4.1 and 4.2 we are guaranteed that Algorithm 4.1A will correctly solve $RVRP_k$ for each k $(1\leq k\leq\Lambda)$. Now since Algorithm 4.1A generates only single-layer wiring trees and associates conducting layers with nets in a prespecified manner as defined by (4.9), it follows that Algorithm 4.1A will compute a solution to $RVRP$ if and only if each $RVRP_k$ $(1\leq k\leq\Lambda)$ has a solution. Thus it only remains to show that every $RVRP_k$ $(1\leq k\leq\Lambda)$ has a solution if and only if $RVRP$ has a solution.

$(\Rightarrow)$ If every $RVRP_k$ $(1\leq k\leq\Lambda)$ has a solution then clearly a superimposition of any set of solutions to these Λ problems will be a solution to $RVRP$. Thus it follows that $RVRP$ in fact has a solution.

$(\Leftarrow)$ If $RVRP$ has a solution then by Lemma 4.7 we must have that:

$$\begin{gathered} a_{i+\Lambda\cdot h}\geq b_i+h \text{ for each falling net } N_i\in Nets \\ \text{and} \\ b_{i+\Lambda\cdot h}\geq a_i+h \text{ for each rising or trivial net } N_i\in Nets. \end{gathered} \tag{4.10}$$

Now for each integer k $(1\leq k\leq\Lambda)$, let N_{k_i} denote the net with i^{th} largest entry coordinate in $RVRP_k$. Under (4.9) we then have that:

$$N_{k_i}=N_{i+\lfloor(i-1)/m\rfloor\cdot(\Lambda-1)\cdot m+(k-1)\cdot m}\in Nets. \tag{4.11}$$

Thus if N_{k_i} is a falling net we must have:

$$\begin{aligned} a_{k_{i+h}} &= a_{i+h+\lfloor(i+h-1)/m\rfloor\cdot(\Lambda-1)\cdot m+(k-1)\cdot m} && \text{[by (4.11)]} \\ &= a_{i+\lfloor(i-1)/m\rfloor\cdot(\Lambda-1)\cdot m+(k-1)\cdot m+\Lambda\cdot h} && \text{[since } m|h\text{]} \\ &\geq b_{i+\lfloor(i-1)/m\rfloor\cdot(\Lambda-1)\cdot m+(k-1)\cdot m}+h && \text{[by (4.10) and (4.11)]} \\ &= b_{k_i}+h && \text{[by (4.11)]} \end{aligned}$$

Furthermore, by a similar argument we obtain the result $b_{k_{i+h}}\geq a_{k_i}+h$ if N_{k_i} is a rising or trivial net. Combining this with Lemma 4.7 we have that each $RVRP_k$ $(1\leq k\leq\Lambda)$ in fact has a solution. This completes the proof of Theorem 4.6.

□

We now note that an analogous proof holds for the diagonally augmented rectilinear wiring model and Algorithm 4.2. In this proof, however, we use Lemma 4.8 rather than Lemma 4.7.

4.4.4 Algorithmic Efficiency.

Let us now consider the time complexity of the algorithms presented in the previous three sections. Each algorithm's time complexity is (upper-)bounded by the sum over all nets of the time required to select a layer and generate a wiring tree for each net. Further, for any given net we observe that the actual wiring tree computation has time linear in the size of the tree generated. (Notice that this can be made to hold even if the tree is represented solely by the endpoints of its maximal length line segments.) Finally, the layer selection process in each algorithm requires at most constant time per net if systematic selection is employed. Combining these observations we see that each algorithm employing systematic layer assignment has time complexity which is linear in the size of the output generated. If systematic layer assignment is not utilized, then additional time will be required to search for an available conducting layer for each net. In this case, the algorithms will be $O(\Lambda \cdot n)$ for a problem instance containing n nets.

4.5 Provably Good Channel Routing Algorithms.

Recall that an instance of the channel routing problem is an ordered triple $CRP=(M,PS,Nets)$ with $M=(R,\sigma,\rho)$, $PS=(P_{top}\cup P_{bottom}\cup P_{left}\cup P_{right},\Pi)$, and $Nets=\{N_1,\ldots,N_n\}$. Let us now consider a restricted version of the channel routing problem in which no connection points are allowed on the left and right sides of a routing region (i.e., $P_{left}=P_{right}=\emptyset$) and in which each net contains exactly two distinct connection points; one constrained to lie along the top of the routing region and the other constrained to lie along the bottom of the routing region (i.e., $|N_i|=2$, $|N_i\cap P_{top}|=1$, and $|N_i\cap P_{bottom}|=1$ for each i, $1\leq i\leq n$). This restricted version of the channel routing problem is now called the *restricted two-pin channel routing problem*. We now present several provably good heuristic algorithms for solving the restricted two-pin channel routing problem. These algorithms are provably good in the sense that they are guaranteed to compute a solution to a given routing problem if the associated routing region has height greater than four times the minimum necessary for a solution to exist. In order to obtain these results we assume a wiring model which allows a limited amount of edge overlap. In particular, we assume that R is either a two-layer rectilinear routing region or a two-layer augmented rectilinear routing region and that $\rho=\emptyset$.

4.5.1 Notation and Basic Observations.

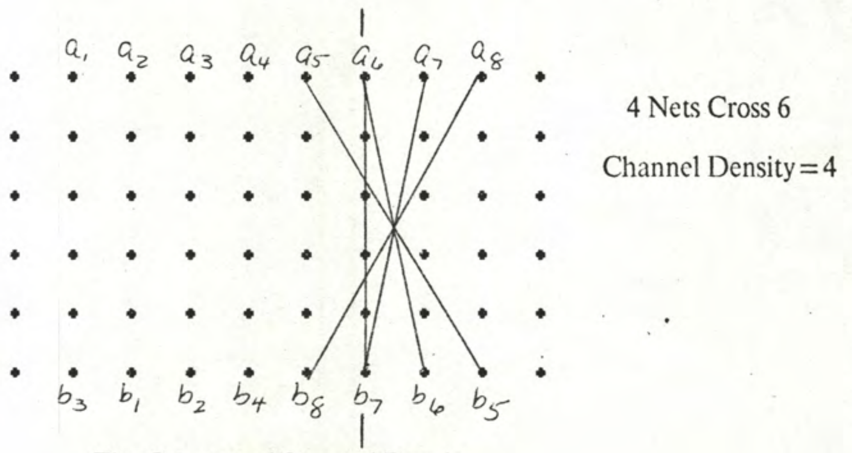
In this section we develop a more convenient notation for a net (cf. Section 4.4.1) and then introduce the concept of channel density [Deu76]. This concept is used to derive lower bounds on the routing region height required for an instance of the restricted two-pin channel routing problem to have a solution.

Let $RCRP=(M,PS,Nets)$ denote any instance of the restricted two-pin channel routing problem. If $M=(R,\sigma,\rho)$, $PS=(P,\Pi)$, and $Nets=\{N_1,\ldots,N_n\}$, then by Definition 4.21 and the previous discussion, every pin in P must be constrained under Π to lie at some fixed location along either the top or the bottom of the associated routing region R. Furthermore, each net $N_i\in Nets$ must be defined by exactly two distinct pins, $p_i\in P$ and $p_i'\in P$, such that p_i is constrained to lie at a particular pin location point along the top of R and p_i' is constrained to lie at a particular pin location point along the bottom of R. Now let a_i denote the x-coordinate of the point at which p_i is constrained to lie and let b_i denote the x-coordinate of the point at which p_i' is constrained to lie. Thus the ordered pair (a_i, b_i) completely specifies the net N_i and so these two notations can be used interchangeably. In addition, the integer a_i will be called the entry coordinate of the net N_i and the integer b_i will be called the exit coordinate of the net N_i. Finally, the net N_i will be called a falling net if $a_i<b_i$, a trivial net if $a_i=b_i$, and a rising net if $a_i>b_i$. We are now ready to define the notion of the *density* of an instance of the restricted two-pin channel routing problem [Deu76].

Definition 4.23:

Let $RCRP=(M, PS, Nets)$ denote any instance of the restricted two-pin channel routing problem and let x denote any integer. A net $N_i \in Nets$ is said to *cross* x if either $a_i \leq x < b_i$ or $b_i \leq x < a_i$. The *channel density* of $RCRP$ is then defined to be the maximum over all $x \in Z$ of the number of nets which cross x.

Figure 4.13 illustrates the concept of channel density for a restricted two-pin channel routing problem. This concept can now be employed to derive a lower bound on the routing region height required for an instance of the restricted two-pin channel routing problem to have a solution; assuming a rectilinear or augmented rectilinear routing region.



a_1 a_2 a_3 a_4 a_5 a_6 a_7 a_8
b_3 b_1 b_2 b_4 b_8 b_7 b_6 b_5
4 Nets Cross 6
Channel Density=4

The Concept of Channel Density
Figure 4.13

Lemma 4.9:

Let $RCRP=(M, PS, Nets)$ denote any instance of the restricted two-pin channel routing problem such that $M=(R, \sigma, \rho)$ denotes a wiring model in which $R=(l, h, \Lambda, V, E)$ is a rectilinear routing region. Further, let d denote the channel density of $RCRP$ and assume that $\{NEO, LPD\} \cap \rho = \emptyset$. Then $RCRP$ has no solution if $h < d/\Lambda$. However, if $\{NEO, LPD\} \cap \rho \neq \emptyset$ then $RCRP$ has no solution if $h < d$.

Proof:

Let $x' \in Z$ denote any integer such that the number of nets in $RCRP$ which cross x' is equal to d. Then clearly in any solution to $RCRP$ at least d distinct wiring trees must contain horizontal edges of R connecting a grid point in column x' and a grid point in column $x'+1$. Now if $\{NEO, LPD\} \cap \rho = \emptyset$, then at most $\Lambda \cdot h$ distinct wiring trees can contain such routing edges in any solution to $RCRP$. Furthermore, if $\{NEO, LPD\} \cap \rho \neq \emptyset$, then at most h distinct wiring trees can contain such routing edges in any solution to $RCRP$. Therefore, if $RCRP$ has a solution and $\{NEO, LPD\} \cap \rho = \emptyset$ then we must have the relation:

$$\Lambda \cdot h \geq d \Rightarrow h \geq d/\Lambda .$$

Similarly, if $RCRP$ has a solution and $\{NEO, LPD\} \cap \rho \neq \emptyset$ then we must have the relation:

$$h \geq d.$$

□

Lemma 4.10:

Let $RCRP=(M, PS, Nets)$ denote any instance of the restricted two-pin channel routing problem such that $M=(R, \sigma, \rho)$ denotes a wiring model in which $R=(l, h, \Lambda, V, E)$ is an augmented rectilinear routing region. Further, let d denote the channel density of $RCRP$ and assume that $\{NEO, LPD\} \cap \rho = \emptyset$. Then $RCRP$ has no solution if $h < d/\Lambda$.

Proof:

Let $x' \in Z$ denote any integer such that the number of nets in $RCRP$ which cross x' is equal to d. Then clearly at least d distinct wiring trees must contain grid points from column x' of R in any solution to $RCRP$. However, at most $\Lambda \cdot h$ distinct wiring trees can contain grid points from column x' of R in any solution to $RCRP$. Therefore, if $RCRP$ has a solution then we must have the relation:

$$\Lambda \cdot h \geq d \Rightarrow h \geq d/\Lambda .$$

□

In the remainder of this section we will always assume that $R=(l,h,\Lambda,V,E)$ is a two-layer rectilinear or augmented rectilinear routing region (i.e., $\Lambda=2$). Therefore, $d/2$ will always be a lower bound on the region height necessary for a problem instance to have a solution.

4.5.2 The Power of Overlap.

Let $RCRP=(M,PS,Nets)$ denote any instance of the restricted two-pin channel routing problem such that $M=(R,\sigma,\rho)$ denotes a wiring model in which $R=(l,h,\Lambda,V,E)$ is a $\Lambda=2$ layer rectilinear routing region, $\sigma=1$, and $\rho=\emptyset$. We will now present an efficient polynomial-time algorithm that is guaranteed to compute a solution to $RCRP$ as long as $h \geq |Nets| = n$. Further, each wiring tree generated by this algorithm will contain exactly one internal contact point and at most one horizontal extension; as shown in Figure 4.14. Before actually describing the algorithm, notice that the location of each pin in $RCRP$ is fixed (i.e., $|\Pi|=1$) and thus the algorithm need only generate the required set of wiring trees.

Algorithm 4.3:

Assume that $RCRP$ contains exactly r rising nets, f falling nets, and t trivial nets (so that $r+f+t=n$). This algorithm then proceeds net by net, in order of increasing net entry coordinate, generating a complete wiring tree for each net in turn. The wiring tree for any given net (a_i,b_i) is generated as follows:

1) If $a_i < b_i$ then let $m \in Z$ denote the number of distinct falling nets with exit coordinate strictly greater than b_i and let $c=h-m$. The wiring tree for net (a_i,b_i) is then composed of a vertical extension from grid point $(a_i,h+1)$ to grid point (a_i,c) on conducting layer 1, a horizontal extension from grid point (a_i,c) to grid point (b_i,c) on layer 2, and a vertical extension from grid point (b_i,c) to grid point $(b_i,0)$ on layer 2 (see Figure 4.14a).

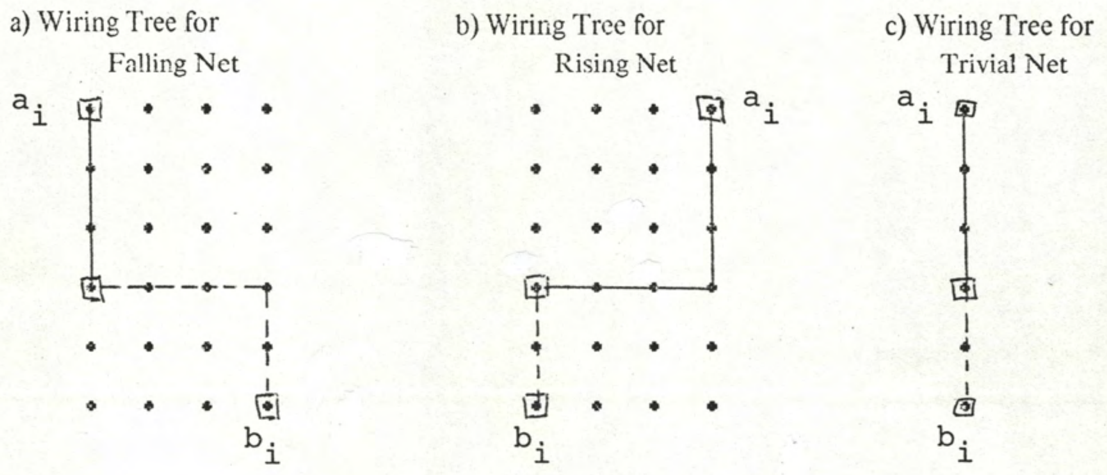
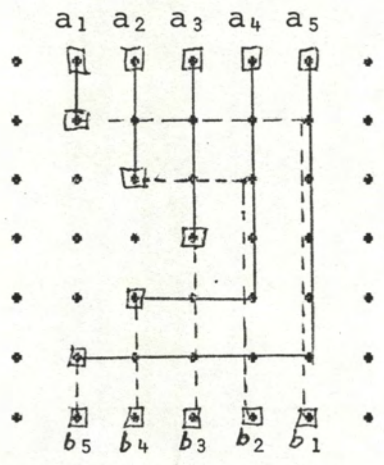

a) Wiring Tree for Falling Net
b) Wiring Tree for Rising Net
c) Wiring Tree for Trivial Net
a_i
b_i
a_i
b_i
a_i
b_i

Figure 4.14



Example of Wiring Trees Generated by Algorithm 4.3

Figure 4.15

2) If $a_i > b_i$ then let $m \in Z$ denote the number of distinct rising nets with entry coordinate strictly greater than a_i and let $c=m+1$. The wiring tree for net (a_i, b_i) is then composed of a vertical extension from grid point $(a_i, h+1)$ to grid point (a_i, c) on conducting layer 1, a horizontal extension from grid point (a_i, c) to grid point (b_i, c) on layer 1, and a vertical extension from grid point (b_i, c) to grid point $(b_i, 0)$ on layer 2 (see Figure 4.14b).

3) If $a_i = b_i$ then the wiring tree for net (a_i, b_i) is composed of a vertical extension from grid point $(a_i, h+1)$ to grid point $(a_i, r+1)$ on conducting layer 1 and a vertical extension from grid point $(a_i, r+1)$ to grid point $(a_i, 0)$ on layer 2 (see Figure 4.14c).

An example of a set of wiring trees that might be generated by Algorithm 4.3 is shown in Figure 4.15. The correctness of this algorithm for computing a solution to *RCRP* is based on the following lemma.

Lemma 4.11:

If T_i and T_j are two distinct wiring trees generated by Algorithm 4.3 for *RCRP* and $h \geq n$, then T_i and T_j are compatible under M.

Proof:

Assume without loss of generality that T_i is the implementation of net $N_i \in Nets$ and T_j is the implementation of net $N_j \in Nets$. We then have four possible cases to consider; the set $\{N_i, N_j\}$ contains at least one trivial net, the set $\{N_i, N_j\}$ contains exactly two rising nets, the set $\{N_i, N_j\}$ contains exactly two falling nets, or the set $\{N_i, N_j\}$ contains one rising net and one falling net.

Case (1): $\{N_i, N_j\}$ contains at least one trivial net.

Assume without loss of generality that N_i is a trivial net. If N_j is also a trivial net then clearly the wiring trees T_i and T_j do not contain a common grid point and so by Lemma 4.1 they must be compatible under M. If N_j is a rising net then T_i and T_j can contain a common grid point only if the horizontal extension of T_j crosses a vertical extension of T_i. However, since $h \geq n$ and N_j is a rising net it must be the case that the horizontal extension of T_j is on conducting layer 1 and the vertical extension of T_i crossed by T_j (if any) is on conducting layer 2 as shown in Figure 4.16a. Thus T_i and T_j must be compatible under M. A similar compatibility argument holds for the case in which N_j is a falling net.

Case (2): $\{N_i, N_j\}$ contains exactly two rising nets.

Assume without loss of generality that the entry coordinate for net N_j is strictly greater than the entry coordinate for net N_i (i.e., $a_j > a_i$). Then it should be clear from the description of Algorithm 4.3 that if $h \geq n$, the horizontal extension for T_j must lie on a lower numbered track than the horizontal extension for T_i. Therefore, T_i and T_j can contain a common grid point only if the horizontal extension of T_j on conducting layer 1 crosses the vertical extension of T_i on conducting layer 2 as shown in Figure 4.16b. Thus T_i and T_j must be compatible under M.

Case (3): $\{N_i, N_j\}$ contains exactly two falling nets.

Assume without loss of generality that the exit coordinate for net N_j is strictly greater than the exit coordinate for net N_i (i.e., $b_j > b_i$). Then it should be clear from the description of Algorithm 4.3 that if $h \geq n$, the horizontal extension for T_j must lie on a higher numbered track than the horizontal extension for T_i. Therefore, T_i and T_j can contain a common grid point only if the horizontal extension of T_j on conducting layer 2 crosses the vertical extension of T_i on conducting layer 1 as shown in Figure 4.16c. Thus T_i and T_j must be compatible under M.

Case (4): $\{N_i, N_j\}$ contains one rising net and one falling net.

Assume without loss of generality that N_i is a rising net and N_j is a falling net. Then it should be clear from the description of Algorithm 4.3 that if $h \geq n$, the horizontal extension for T_j must lie on a higher numbered track than the horizontal extension for T_i. Therefore, T_i and T_j can contain a common grid point only if either the horizontal extension of T_j on conducting layer 2 crosses the vertical extension of T_i on conducting layer 1, the horizontal extension of T_i on conducting layer 1 crosses the vertical extension of T_j on conducting layer 2, or the vertical extension of T_i on conducting layer 1 and the vertical extension of T_j on conducting layer 2 overlap; as shown in Figure 4.17. Thus T_i and T_j must be compatible under M.

□

It now follows directly from Lemma 4.11 that any set of wiring trees generated by Algorithm 4.3 for *RCRP* must be pairwise compatible under *M* if $h \geq n$. Therefore, Algorithm 4.3 is guaranteed to compute a solution to *RCRP* as long as $h \geq n$. Furthermore, this solution will clearly be composed entirely of wiring trees containing exactly one internal contact point and at most one horizontal extension. We should now mention that no such result is possible under wiring models with $LPD \in \rho$. In particular, there exist examples under such models in which some wiring tree must utilize more than one track (see Figure 4.18). Thus the removal of the wiring restriction *LPD* appears to result in substantially more powerful wiring models. Finally, for completeness we note that Algorithm 4.3 has worst case time complexity $O(n)$ for computing a solution to *RCRP* if each wiring tree is represented by the endpoints of its maximal length line segments.

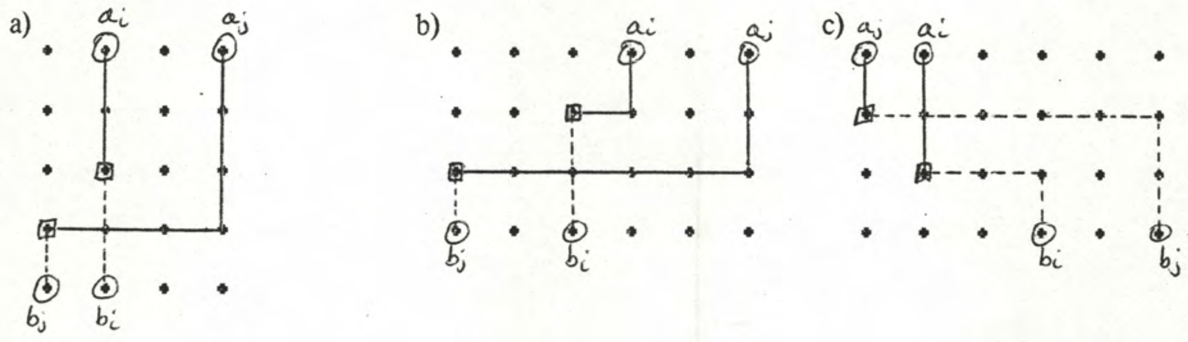


Figure 4.16

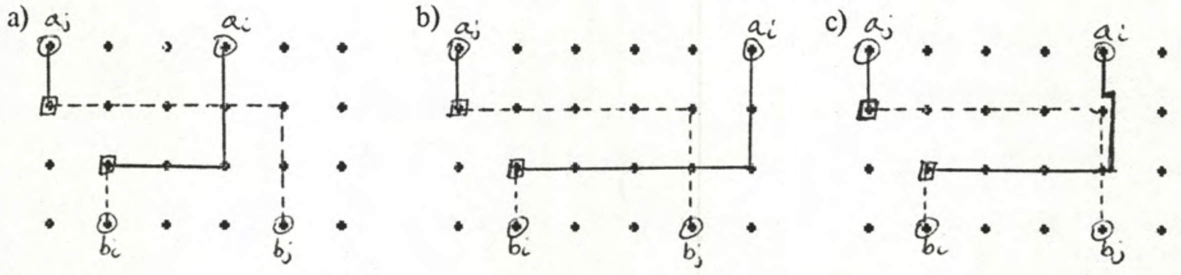
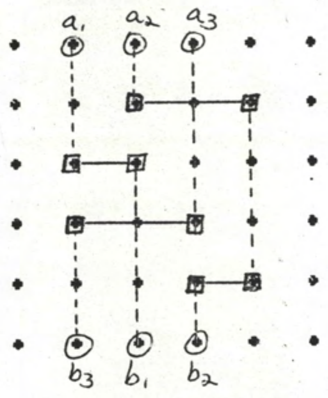


Figure 4.17



Wiring Tree Requiring Two Tracks ($LPD \in P$)

Figure 4.18

4.5.3 Reducing Track Usage.

We will now improve the result of Section 4.5.2 by presenting an algorithm which is guaranteed to compute a solution to any given instance of the restricted two-pin channel routing problem providing the height of the associated routing region is at least $2d+1$, where d denotes the corresponding channel density. This algorithm is an improvement of an earlier algorithm (with the same $2d+1$ bound) due to R. Rivest [RBM81]. The algorithm presented here is guaranteed to compute a solution requiring at most $4n$ internal contact points. The previous algorithm, in contrast, might compute a solution with as many as $d \cdot n$ internal contact points (where n denotes the number of nets). Recall that $d/2$ is a lower bound on the routing region height necessary for such a problem instance to have a solution.

Once again, let $RCRP=(M, PS, Nets)$ denote any instance of the restricted two-pin channel routing problem such that $M=(R, \sigma, \rho)$ denotes a wiring model in which $R=(l, h, \Lambda, V, E)$ is a $\Lambda=2$ layer rectilinear routing region, $\sigma=1$, and $\rho=\emptyset$. We will now present an efficient polynomial-time algorithm that is guaranteed to compute a solution to $RCRP$ as long as $h \geq 2d+1$, where d denotes the channel density of $RCRP$. The basic approach taken by this algorithm is to generate a set of pairwise compatible wiring trees which satisfies the following properties.

P1) If there are exactly k nets which cross j, then exactly k of the horizontal routing edges connecting a grid point in column j with a grid point in column $j+1$ belong to a wiring tree. Further, each of these routing edges belongs to a distinct wiring tree and lies on a distinct track.

P2) If exactly r of the k nets which cross j are rising and f are falling (so that $r+f=k$), then between columns j and $j+1$:

a) The top-most $2r$ tracks (excluding track $h+1$) are devoted to wiring trees implementing rising nets.

b) The bottom-most $2f$ tracks (excluding track 0) are devoted to wiring trees implementing falling nets.

c) The middle $h-2k$ tracks are unused.

P3) For each wiring tree T_i which implements a falling net (a_i, b_i): If T_i *enters* column b_i on track τ (i.e., T_i contains a horizontal edge connecting grid points $(b_i\text{-}1, \tau)$ and (b_i, τ)), then no horizontal edge connecting grid points $(b_i\text{-}1, \tau+1)$ and $(b_i, \tau+1)$ belongs to a wiring tree. Similarly, for each wiring tree T_i' which implements a rising net (a_i', b_i'): If T_i' *enters* column a_i' on track τ', then no horizontal edge connecting grid points $(a_i'\text{-}1, \tau'\text{-}1)$ and $(a_i', \tau'\text{-}1)$ belongs to a wiring tree.

In addition to these three properties, the algorithm attempts to generate wiring trees such that all horizontal edges belonging to wiring trees implementing falling nets are on conducting layer 1, and all horizontal edges belonging to wiring trees implementing rising nets are on conducting layer 2. Notice that for $h \geq 2d+1$ there is guaranteed (by property P2) to be at least one unused track through the middle of R. We are now ready to describe the algorithm.

Algorithm 4.4:

This algorithm proceeds column by column extending the wiring trees for all nets which cross j in step j. The wiring tree extensions across column j are generated as follows:

1) If column j contains a trivial net $(a_i = j,\ b_i = j)$, the net is wired straight across the column utilizing any free middle track to change layers as shown in Figure 4.19a.

2) If column j contains a rising net $(a_i, b_i = j)$ and exactly r rising nets cross j, the wiring tree for net (a_i, b_i) is extended vertically upward from grid point $(j, 0)$ to the highest available track below track $h\text{-}2r+3$, as shown in Figure 4.19b. Additional extension upward may then be added as required by step (3) below. Falling nets with entry coordinate j are handled similarly as shown in Figure 4.19c. Notice the use of edge overlap when the wiring trees for a rising net and a falling net cross each other over the otherwise empty middle tracks of the routing region.

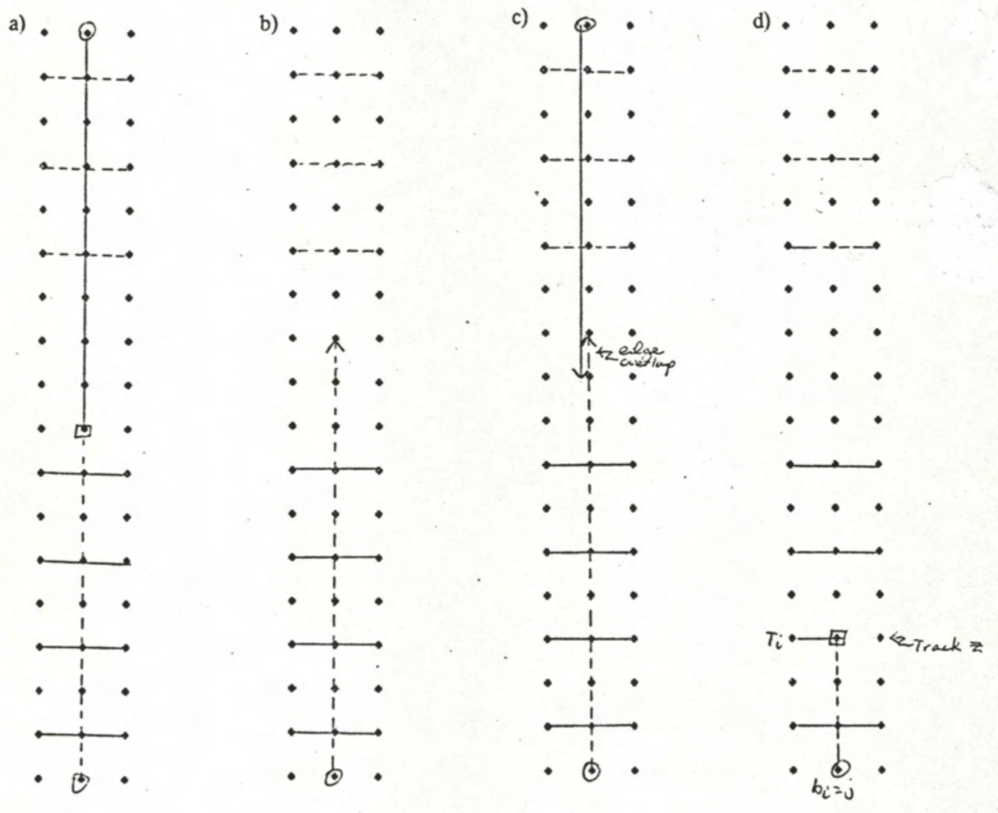


Figure 4.19

3) If a wiring tree T_i for a falling net $(a_i, b_i{=}j)$ enters column j on track z, the tree T_i is extended vertically downward from grid point (j, z) to grid point $(j, 0)$ as shown in Figure 4.19d. In order to insure the preservation of property P2, however, some other wiring tree for a falling net (if any) must replace T_i on track z. Furthermore, this replacement tree T' should be the one which enters column j on the highest track (including track $h+1$). The necessary wiring tree extensions are then generated as shown in Figure 4.20. The only problem arises when T' corresponds to a falling net with exit coordinate $j+1$. In this case, the preservation of property P3 requires that the next lower wiring tree (i.e., below T' in column j) be used for replacement instead of T' (see Figure 4.21). Finally, rising nets with entry coordinate j are handled similarly.

An example of a set of wiring trees that might be generated by Algorithm 4.4 is shown in Figure 4.22. We now have the following theorem.

Theorem 4.7:

Algorithm 4.4 will correctly compute a solution to *RCRP* providing the associated routing region has height at least $2d+1$, where d denotes the channel density of *RCRP*.

Proof:

Let r_j denote the number of rising nets which cross j and let f_j denote the number of falling nets which cross j. It should be clear that Algorithm 4.4 will maintain properties P1, P2, and P3 as the wiring trees for *RCRP* are extended from column to column, providing the height h of the associated routing region is at least $\mathrm{Max}\{2r_j+2f_j+1 \mid 1\leq j\leq l\}$. Furthermore, it follows directly from the description of Algorithm 4.4 that the compatibility of wiring trees generated by this algorithm will be maintained from column to column providing properties P1, P2, and P3 are maintained from column to column. Therefore, Algorithm 4.4 is guaranteed to compute a set of compatible wiring trees for *RCRP* if $h\geq \mathrm{Max}\{2r_j+2f_j+1 \mid 1\leq j\leq l\}$. Notice, however, that $\mathrm{Max}\{r_j+f_j \mid 1\leq j\leq l\}=d$ and thus $\mathrm{Max}\{2r_j+2f_j+1 \mid 1\leq j\leq l\}=2d+1$. The theorem then follows.
□

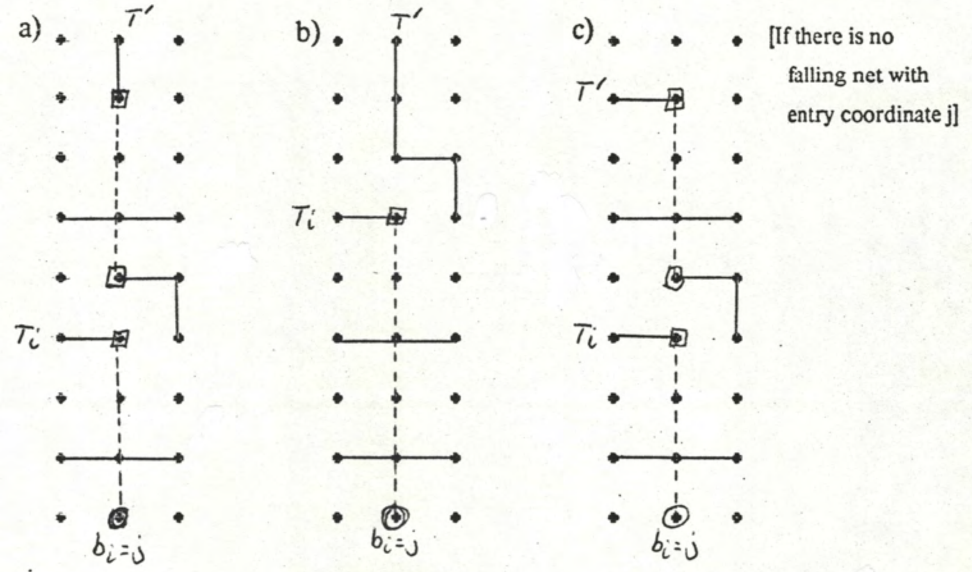


Figure 4.20

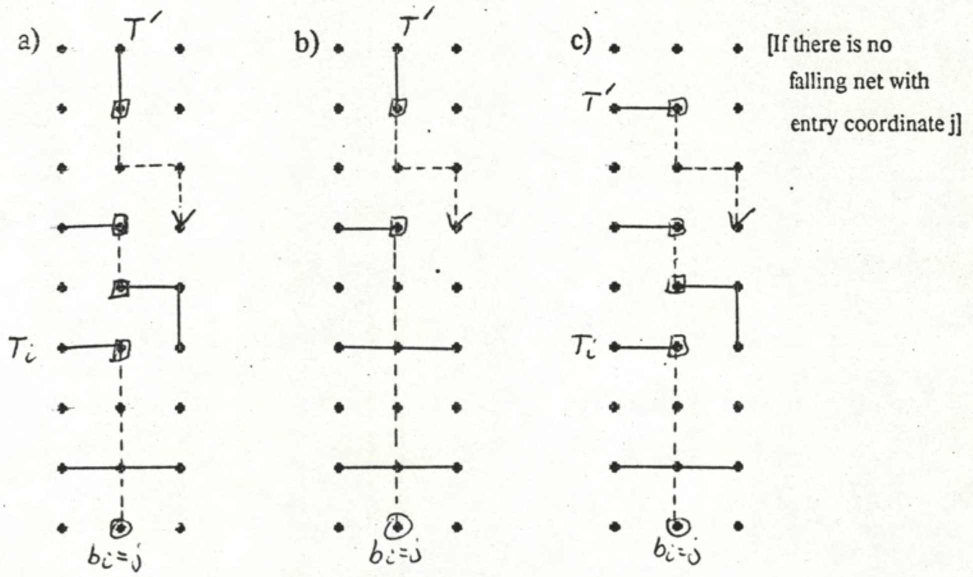
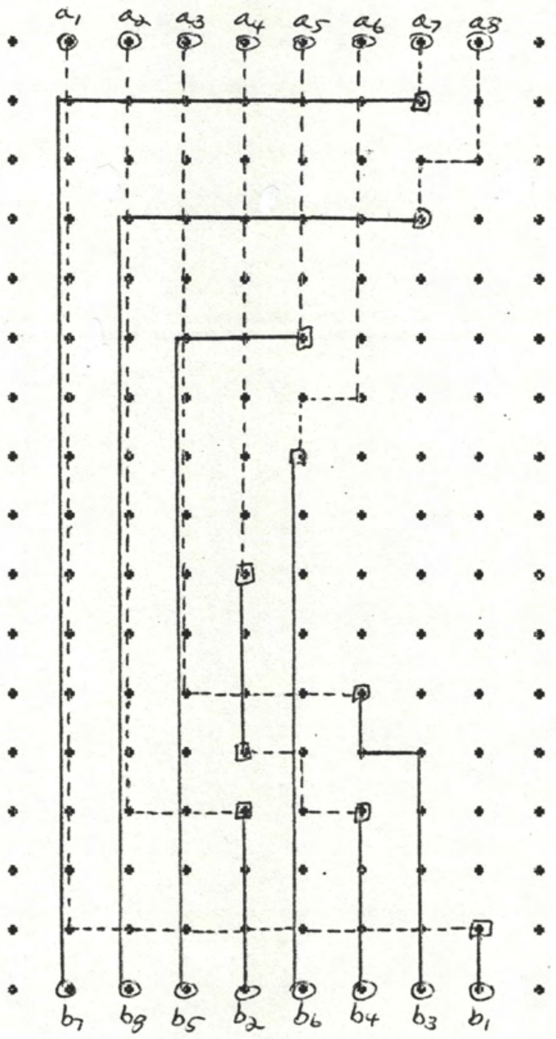


Figure 4.21



A Set of Wiring Trees Generated by Algorithm 4.4
Figure 4.22

It now follows from Lemma 4.9 and Theorem 4.7 that Algorithm 4.4 is guaranteed to compute a solution to *RCRP* if the associated routing region has height greater than four times the minimum necessary for a solution to exist. Further, notice that any such solution will contain at most $4 \cdot n$ total internal contact points and at most d internal contact points in any one wiring tree, where n denotes the number of nets in *RCRP* and d denotes the channel density of *RCRP*. Finally, it should be clear that the worst case time complexity of Algorithm 4.4 is bounded by the product of the number of columns which must be processed and the complexity of processing any one column. In practice, if we represent each wiring tree by the endpoints of its maximal length line segments then we only need process columns having index equal to some net entry or exit coordinate. Furthermore, the complexity of processing any one column under this representation will be $O(d)$. Therefore, Algorithm 4.4 has worst case time complexity $O(d \cdot n)$ under a practical implementation.

4.5.4 An Algorithm for Augmented Rectilinear Wiring Models.

In this section we show how the result of Section 4.5.3 can be substantially improved under a diagonally augmented rectilinear wiring model. In particular, we present a very simple polynomial-time algorithm which is guaranteed to compute a solution to any given instance of the restricted two-pin channel routing problem providing the associated routing region is an augmented rectilinear routing region and has height greater than the problem channel density (i.e., $h \geq d+1$). Further, each wiring tree generated by this algorithm will contain at most one internal contact point.

Let $RCRP=(M, PS, Nets)$ denote any instance of the restricted two-pin channel routing problem such that $M=(R, \sigma, \rho)$ denotes a wiring model in which $R=(l, h, \Lambda, V, E)$ is a $\Lambda=2$ layer augmented rectilinear routing region, $\sigma=\sqrt{2}/2$, and $\rho=\emptyset$. We will now present an efficient polynomial-time algorithm that is guaranteed to compute a solution to *RCRP* as long as $h \geq d+1$, where d denotes the channel density of *RCRP*. Similar to Algorithm 4.4, the basic approach taken by this algorithm is to generate a set of pairwise compatible wiring trees which satisfies the following properties.

P1) If there are exactly k nets which cross j, then exactly k of the routing edges connecting a grid point in column j with a grid point in column $j+1$ belong to a wiring tree. Further, each of these routing edges belongs to a distinct wiring tree.

P2) If exactly r of the k nets which cross j are rising and f are falling (so that $r+f=k$), then between columns j and $j+1$:

a) The top-most r tracks (excluding track $h+1$) are devoted to wiring trees implementing rising nets.

b) The bottom-most f tracks (excluding track 0) are devoted to wiring trees implementing falling nets.

c) The middle h-k tracks are unused.

P3) All horizontal and diagonal routing edges belonging to wiring trees for falling nets are on conducting layer 1 and all horizontal and diagonal routing edges belonging to wiring trees for rising nets are on conducting layer 2.

We are now ready to describe the algorithm.

Algorithm 4.5:

This algorithm proceeds column by column extending the wiring trees for all nets which cross j in step j. The wiring tree extensions across column j are generated as follows:

1) If column j contains a trivial net $(a_i=j, b_i=j)$, the net is wired straight across the column utilizing any free middle track to change layers as shown in Figure 4.23a.

2) If column j contains a rising net $(a_i, b_i=j)$, the wiring tree for net (a_i, b_i) is extended vertically upward from grid point $(j,0)$ as far as possible on conducting layer 2 (see Figure 4.23b). Falling nets with entry coordinate j are handled similarly on conducting layer 1 as shown in Figure 4.23c.

3) If a wiring tree T_i for a falling net $(a_i, b_i=j)$ enters column j on track z, the tree T_i is extended vertically downward on layer 2 from grid point (j,z) to grid

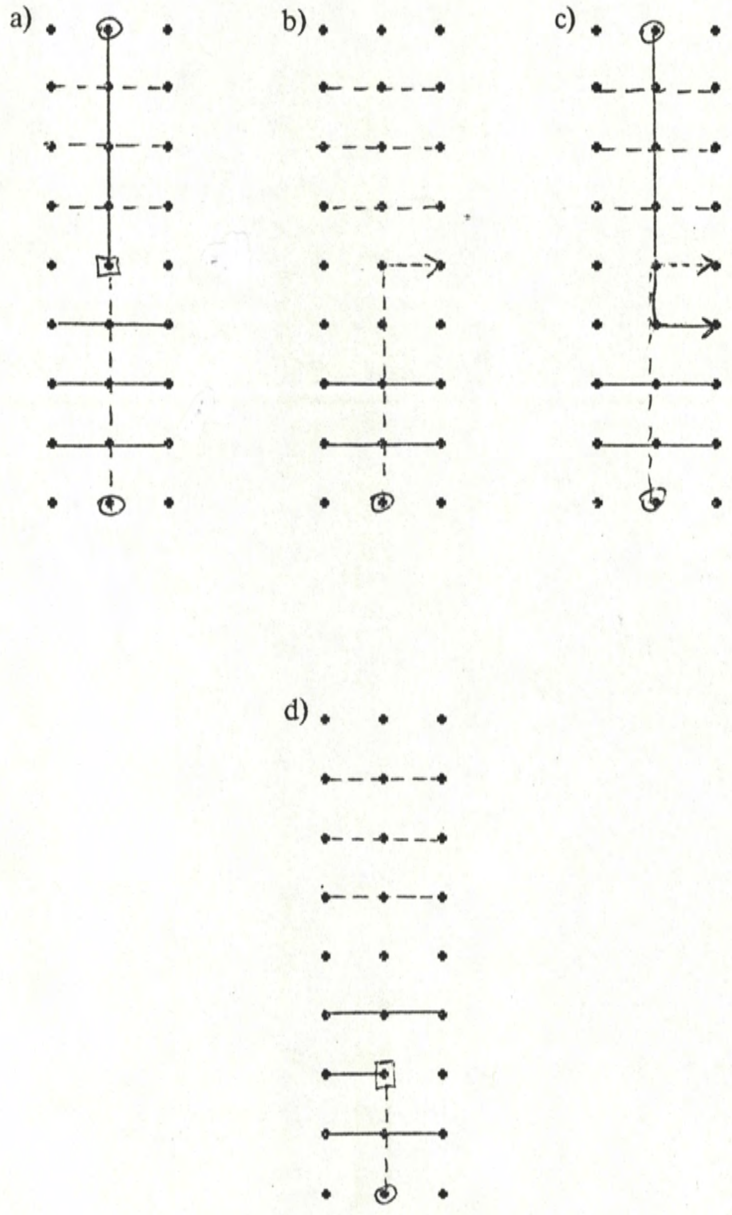


Figure 4.23

point $(j,0)$ as shown in Figure 4.23d. In order to maintain property P2, however, the algorithm must then "close up ranks" between columns j and $j+1$ so that all empty tracks remain in the middle of the routing region. Figure 4.24 illustrates how such a wiring can be generated. Finally, rising nets with entry coordinate j are handled similarly.

An example of a set of wiring trees that might be generated by Algorithm 4.5 is shown in Figure 4.25. We now have the following theorem.

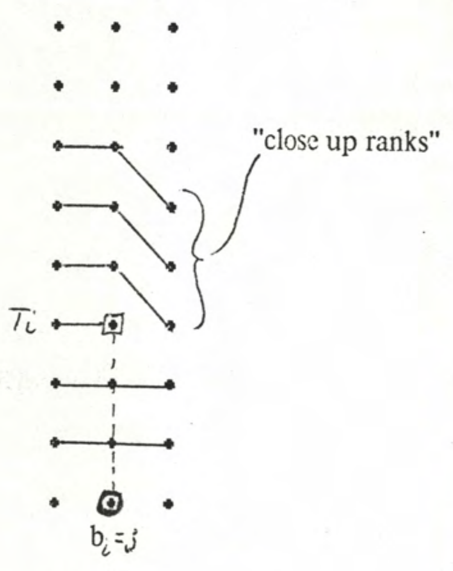
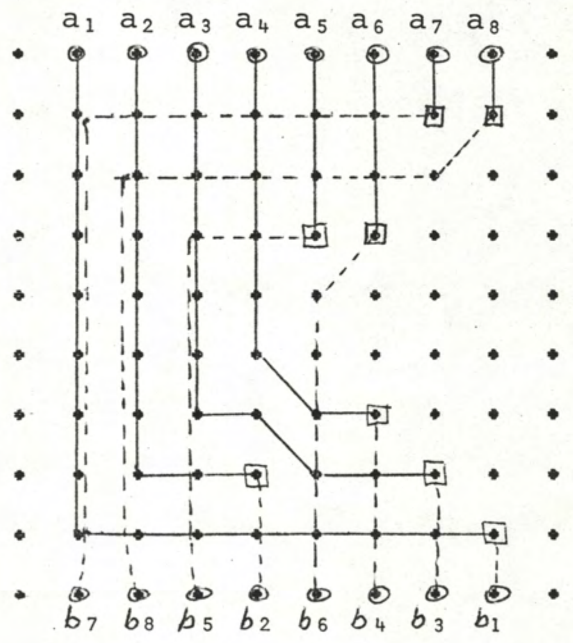


Figure 4.24


a_1 a_2 a_3 a_4 a_5 a_6 a_7 a_8
b_7 b_8 b_5 b_2 b_6 b_4 b_3 b_1

Example of Wiring Trees Generated by Algorithm 4.5

Figure 4.25

Theorem 4.8:

Algorithm 4.5 will correctly compute a solution to *RCRP* providing the associated routing region has height at least $d+1$, where d denotes the channel density of *RCRP*.

Proof:

The proof of this theorem follows trivially from the description of Algorithm 4.5. □

It now follows from Lemma 4.10 and Theorem 4.8 that Algorithm 4.5 is guaranteed to compute a solution to *RCRP* if the associated routing region has height greater than twice the minimum necessary for a solution to exist. Further, notice that any such solution will contain at most one internal contact point for each wiring tree. Thus there will be at most n total internal contact points, where n denotes the number of nets in *RCRP*. Finally we note that Algorithm 4.5, like Algorithm 4.4, has worst case time complexity $O(d \cdot n)$ if each wiring tree is represented by the endpoints of its maximal length line segments.

4.5.5 Further Channel Routing Results.

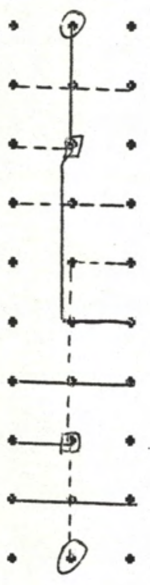
We shall now briefly discuss two extensions of the results presented in Sections 4.5.3 and 4.5.4. We shall first consider problem instances composed of nets containing more than two pins (i.e., composed of multi-pin nets). We then consider problem instances for which the associated wiring model prohibits edge overlap (i.e., $NEO \in \rho$). In each case we shall describe how similar provably good algorithms can be developed.

Let us first consider any instance of the channel routing problem which contains no pins constrained to lie on the left or right sides of the routing region but which may contain nets with more than two pins. If the associated wiring model contains an augmented rectilinear routing region and has $\sigma = \sqrt{2}/2$ and $\rho = \emptyset$, then a straightforward generalization of Algorithm 4.5 can be applied. This generalized algorithm once again proceeds column by column, extending the wiring trees for all nets which cross j in step j. However, at any step j a net is considered to be rising or falling depending on the location (top or bottom of the routing region) of its pin placed with lowest x-coordinate greater than j. In addition, step (3) of the algorithm is modified so that a wiring tree may both *contact a pin* and *continue on* in any one column (see Figure 4.26a). Finally, we note that this generalized algorithm requires a routing region with height at least $d+2$ and will then generate a solution containing at most $|P|$ internal contact points (see Figure 4.26b).

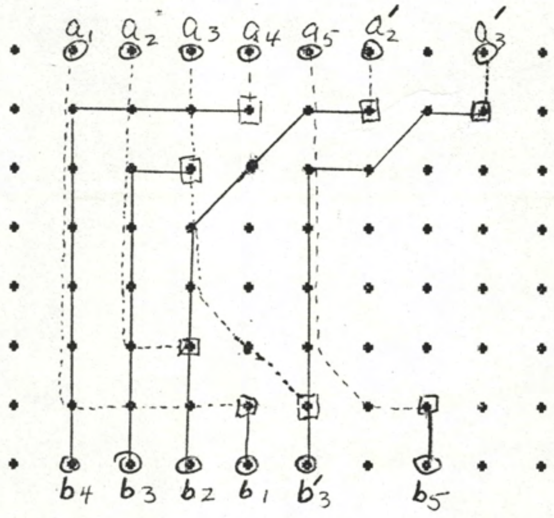
Now consider the case in which the problem instance is composed of a wiring model which contains a rectilinear routing region and has $\sigma=1$ and $\rho=\emptyset$. In this case it is not so easy to generalize Algorithm 4.4. The problem occurs in attempting to modify step (3) of the algorithm so that a wiring tree can both *contact a pin* and *continue on* in any one column. This problem can be avoided, however, at the price of requiring the associated routing region to have height at least $4d+1$. The algorithm can then simply run two distinct horizontal extensions (one within the top-most $4r$ tracks and the other within the bottom-most $4f$ tracks) for each net containing pins placed along both the top and bottom sides of the routing region. This process is illustrated in Figure 4.27.

Finally, let us consider any instance of the restricted two-pin channel routing problem under a two-layer rectilinear wiring model with $NEO \in \rho$. For this case, R. Rivest and G. Miller [RBM81] have developed a clever algorithm which generates wiring tree extensions track by track rather than column by column. In addition, this algorithm utilizes at most $2d$-1 tracks. The algorithm simply proceeds by extending wiring trees horizontally within any given track (alternating between left-to-right and right-to-left extension for successive tracks) in a manner which continually reduces the density of the remaining portion of the routing problem. The basic goal is to always extend the wiring tree which requires farthest extension. An example of such a routing is shown in Figure 4.28. We now note that d (and not $d/2$) is a lower bound on the required routing region height under this wiring model (i.e., $NEO \in \rho$). Therefore, this algorithm is guaranteed to compute a solution providing the associated routing region has height at least twice the minimum necessary for a solution to exist. A major drawback of this algorithm is that it may generate solutions containing a large number of internal contact points; as many as $d \cdot n$.

a)



b)



(Density = 4)

Figure 4.26

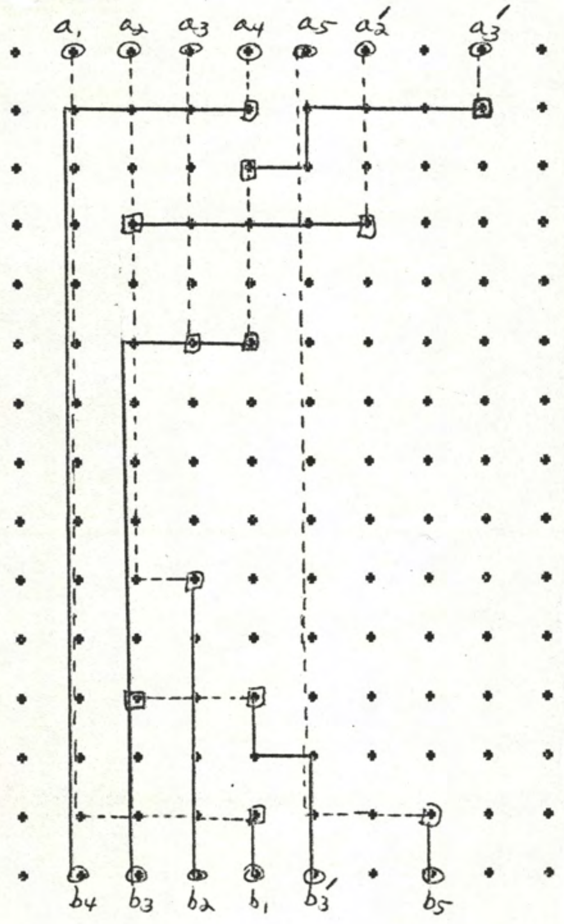


Figure 4.27

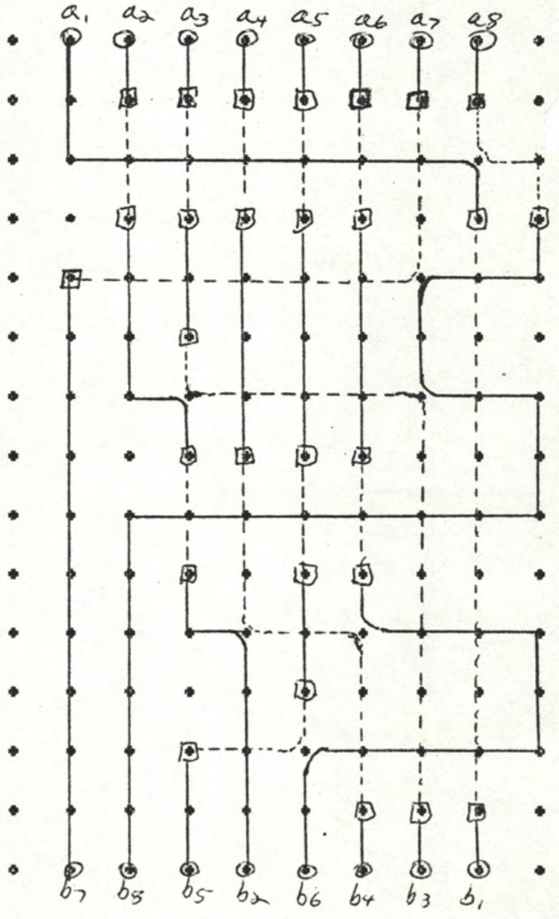


Figure 4.28

4.6 The Quick Router: An Algorithm for the Fixed-Pin Routing Problem.

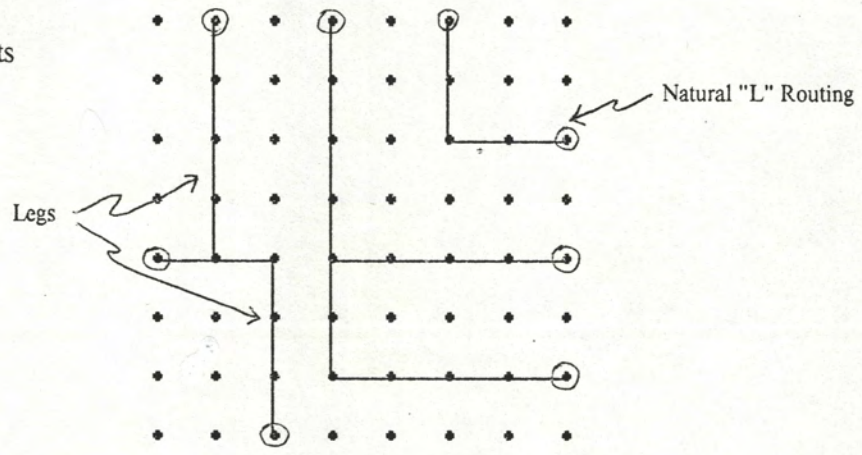
In this section we present a very efficient heuristic algorithm for computing a solution to any given instance of the fixed-pin routing problem with a two-layer rectilinear wiring model. The algorithm we shall describe, known as the quick routing algorithm, has been implemented for use in the PI routing system and was briefly discussed in Section 2.6. It is based on the assumption that many of the instances of the fixed-pin routing problem which arise in practice can be solved by applying a very simple set of heuristics.

Let $FPRP=(M,PS,Nets)$ denote any instance of the fixed-pin routing problem such that $M=(R,\sigma,\rho)$ denotes a wiring model in which $R=(l,h,\Lambda,V,E)$ is a $\Lambda=2$ layer rectilinear routing region and $\sigma=1$. Since *FPRP* is an instance of the fixed-pin routing problem, the pin structure *PS* must contain exactly one legal placement, π, and thus any algorithm for computing a solution to *FPRP* need only generate the required set of compatible wiring trees. The quick router computes such a solution in three steps. The first step consists of determining the topology of each of the required wiring trees. Essentially no consideration is given to layer selection or compatibility at this point. The second step then involves the assignment of conducting layers to each of the segments composing the previously defined tree structures. This layer assignment is performed in a manner which maximizes usage of one of the two available conducting layers. (In practice the two layers are assumed to be metal and polysilicon and thus the use of metal is preferred.) The third and final step consists of a compatibility (or design rule) check to verify the legality of the solution generated. We shall now present the details of each of these three steps.

The first phase of the quick routing algorithm consists of determining the topology of each of the required wiring trees. The algorithm begins by partitioning the set of nets into three disjoint subsets (the *easy nets*, the *hard nets*, and the *impossible nets*) as follows:

1) The set of *easy nets* consists of all nets containing three or less pins with at least one pin constrained to lie on the top or bottom side of the routing region and at least one pin constrained to lie on the left or right side of the routing region (see Figure 4.29a).

a) Examples of
Easy Nets



b) Examples of
Hard Nets

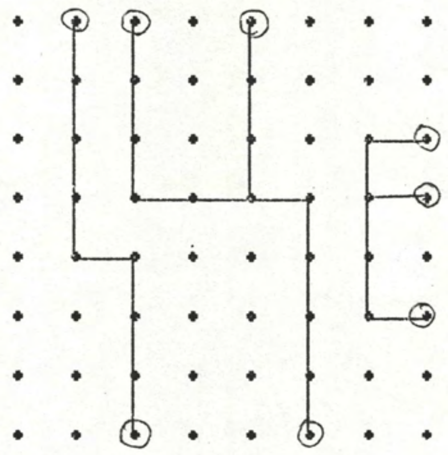


Figure 4.29

2) The set of *hard nets* consists of all other nets containing three or less pins (see Figure 4.29b).

3) The set of *impossible nets* consists of all nets containing more than three pins.

Impossible nets are ignored in the current implementation of the quick router. In this sense the algorithm is guaranteed to fail if *FPRP* contains a net with more than three pins. However, the heuristics that will be described for nets with less than three pins can be generalized to handle nets with more than three pins if desired.

Once the set of nets has been partitioned, a tree structure is generated for each easy net, independently, as follows:

1) For each easy net containing exactly two pins, the quick router generates the natural "L" structure as shown in Figure 4.29a.

2) For each easy net containing exactly three pins, the quick router produces a topology containing a straight line extension from the pin lying on the side of the routing region having unique orientation (horizontal vs. vertical), with legs extending out to the other two pins as shown in Figure 4.29a.

It will then remain only to generate topologies for the wiring trees implementing hard nets. Notice that for each hard net there is need to find a *jog-track* where a portion of the corresponding tree can be placed.

The quick router simply processes the hard nets in order according to the reverse of their sequence of visitation in a counter-clockwise traversal, beginning at the lower-left corner, around the perimeter of the routing region (see Figure 4.30). A jog track is selected for each net in turn as follows:

1) If a net contains exactly two pins, the algorithm searches for a clear jog track by proceeding bottom-up if the pin placed with lower x-coordinate is on the bottom of the routing region, top-down if the pin placed with lower x-coordinate is on the top of the routing region, left-to-right if the pin placed

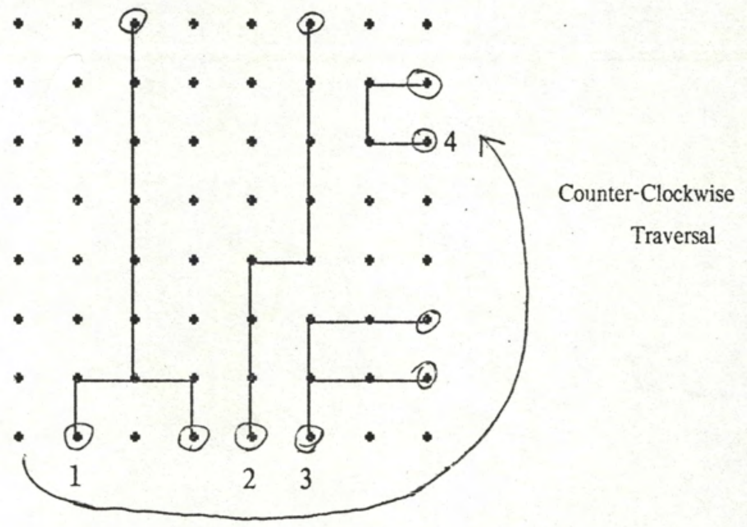


Figure 4.30

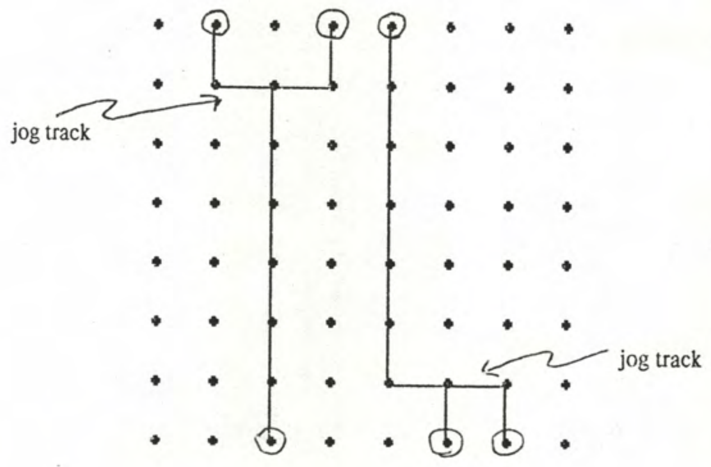
with lower y-coordinate is on the left of the routing region, and right-to-left otherwise.

2) If a net contains exactly three pins, then the algorithm searches for a clear jog track by proceeding from the side of the routing region containing the majority of the pin placements.

Once a jog track has been selected for a net, the topology of the associated wiring tree is generated in the straightforward manner as shown in Figure 4.31.

The second phase of the quick routing algorithm then consists of assigning a conducting layer to each of the line segments composing the previously defined tree structures. This layer assignment is performed by first assigning the preferred layer (call it layer 1) to all line segments having orientation (horizontal vs. vertical) in the direction containing the maximum total length of line segment. The algorithm then assigns layers to all remaining line segments such that layer 2 is used whenever a segment on layer 1 is crossed and layer 1 is used otherwise.

The third and final phase of the quick routing algorithm then consists of a simple compatibility check to verify that all the wiring trees generated are compatible. If so, the algorithm halts and outputs the generated set of wiring trees. If not, the algorithm fails and thus sends an appropriate message.



Wiring Hard Nets
Figure 4.31

It should now be clear that the quick router is extremely efficient. If it succeeds in computing a solution to *FPRP*, then it will have done so very quickly. If it fails, however, then little time will have been wasted.

4.7 Summary.

This chapter has dealt with a general signal routing problem called the region routing problem. An instance of the region routing problem is typically characterized by the requirements that all connection points lie on the perimeter of a given rectangular region and that all wiring be located entirely within this region. We have developed a formal mathematical model for the general region routing problem and then used this model to derive new algorithms for several classes of restricted region routing problems. Most importantly, we have presented optimal polynomial-time algorithms for river routing with arbitrarily many conducting layers and near optimal polynomial-time algorithms for channel routing under a two-layer wiring model which allows limited wire overlap. We should note, however, that these new algorithms are likely to be considered to be of mainly theoretical interest for the following reasons:

1) They all tend to generate wirings in which each conducting layer is used about equally. (This can be highly undesirable in practice if each of the available conducting layers has substantially different electrical properties.)

2) They all generate wirings which contain parallel runs of overlapping wires and are thus susceptible to "cross-talk" via the capacitive coupling of long overlapping wires.

Nonetheless, these results demonstrate that it is possible to develop provably good routing algorithms and so perhaps some day we can have algorithms that are both good in theory and good in practice.

Bibliography

[Aho77] Aho, A.V.; Garey, M.R. and Hwang, F.K., "Rectilinear Steiner Trees: Efficient Special Case Algorithms," *Networks*, Vol. 7, 1977, 37-58.

[Aho74] Aho, A.V.; Hopcroft, J.E. and Ullman, J.D., *The Design and Analysis of Computer Algorithms*, Addison-Wesley Publishing Co., Reading, Mass., 1974.

[Bre77] Breuer, M.A., "Min-Cut Placement," *Journal of Design Automation and Fault-Tolerant Computing*, Vol. 1, No. 4, 1977, 343-362.

[Des72] Breuer, M., ed. *Design Automation of Digital Systems, Volumn 1: Theory and Techniques*, Prentice-Hall, Inc. Englewood Cliffs, N.J., 1972.

[Deu76] Deutsch, D.N., "A 'Dogleg' Channel Router," *Proceedings of the Thirteenth Design Automation Conference*, IEEE, 1976, 425-433.

[Dol81] Dolev, D.; Karplus, K; Siegel, A.; Strong, A. and Ullman, J. D., "Optimal Wiring Between Rectangles," *Proceedings of the Thirteenth Annual ACM Symposium on Theory of Computing*, 1981, 312-317.

[DS81] Dolev, D. and Siegel, A., "The Separation Required for Arbitrary Wiring Barriers," Stanford University (Preprint), 1981.

[EA78] El-Arbi, C., "Une Heuristique Pour Le Probleme De L'Arbre De Steiner," *R.A.I.R.O.*, Vol. 12, No. 2, 1978, 207-212.

[Gar77] Garey, M.R.; Graham, R.L. and Johnson, D.S., "The Complexity of Computing Steiner Minimal Trees," *SIAM Journal of Applied Mathematics*, Vol. 32, 1977, 835-859.

[GJ77] Garey, M.R. and Johnson, D.S., "The Rectilinear Steiner Tree Problem is NP-Complete," *SIAM Journal of Applied Mathematics*, Vol. 32, 1977, 826-834.

[Gar79] Garey, M.R. and Johnson, D.S., *Computers and Intractability*, W.H. Freeman and Company, San Francisco, CA, 1979.

[Gup79] Gupta, U.; Lee, D. and Leung, J., "An Optimal Solution for the Channel-Assignment Problem," *IEEE Transactions on Computers*, Vol. C-28, No. 11, 1979, 807.

[Han72] Hanan, M. and Kurtzberg, J., "A Review of the Placement and Quadratic Assignment Problems," *SIAM Review*, Vol. 14, No. 2, 1972, 324-342.

[Han76] Hanan, M.; Wolff, P.K. and Agule, B.J., "A Study of Placement Techniques," *Journal of Design Automation and Fault-Tolerant Computing*, Vol. 1, No. 1, 1976.

[Has71] Hashimoto, A. and Stevens, J., "Wire Routing by Optimizing Channel Assignment within Large Aperatures," *Proceedings of the Eighth Design Automation Workshop*, IEEE, 1971, 155-169.

[Hig69] Hightower, D., "A Solution to Line-Routing Problems on the Continuous Plane," *Proceedings of the Sixth Design Automation Workshop*, IEEE, 1969, 1-24.

[Hig74] Hightower, D., "The Interconnection Problem: A Tutorial," *Computer*, Vol. 7, No. 4, 1974, 18-32.

[Hig80] Hightower, D.W. and Boyd, R.L., "A Generalized Channel Router," *Proceedings of the Seventeenth Design Automation Conference*, 1980, 12-21.

[Hit69] Hitchcock, R.B., "Cellular Wiring and the Cellular Modeling Technique," *Proceedings of the Sixth Design Automation Workshop*, IEEE, 1969, 25-41.

[Hwa78] Hwang, F.K., "The Rectilinear Steiner Problem," *Journal of Design Automation and Fault-Tolerant Computing*, Vol. 2, No. 4, 1978, 303-310.

[Kaw79] Kawamoto, T. and Kajitani, Y., "The Minimum Width Routing of a 2-Row 2-Layer Polycell-Layout," *Proceedings of the Sixteenth Design Automation Conference*, IEEE, 1979, 290-296.

[Kos81] Koschella, J.J., "A Placement/Interconnect Channel Router: Cutting your PI into Slices," S.B. thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, 1981.

[Kou81] Kou, L.; Markowsky, G. and Berman, L., "A Fast Algorithm for Steiner Trees," *Acta Informatica*, Vol. 15, 1981, 141-145.

[Kuh79] Kuh, E.S.; Kashiwabara, T. and Fujisawa, T., "On Optimal Single-Row Routing," *IEEE Transactions on Circuits and Systems*, Vol. CAS-26, No. 6, 1979, 361-368.

[LaP80] LaPaugh, A.S., "Algorithms for Integrated Circuit Layout: An Analytic Approach," Ph.D. dissertation, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, 1980.

[Lau79] Lauther, U., "A Min-Cut Placement Algorithm for General Cell Assemblies Based on a Graph Representation," *Proceedings of the Sixteenth Design Automation Conference, IEEE*, 1979, 1-10.

[Lee61] Lee, C.Y., "An Algorithm for Path Connections and Its Applications," *IRE Transactions on Electronic Computers*, VEC-10, 1961, 346-365.

[Lei80] Leiserson, C.E., "Area-Efficient Graph Layouts (for VLSI)," *Proceedings of the Twenty-First Annual Symposium on Foundations of Computer Science*, IEEE, 1980, 270-281.

[Lei81] Leiserson, C.E., and Pinter, R.Y., "Optimal Placement for River Routing" to appear in *Proceedings of the CMU Conference on VLSI Systems and Computations*, Oct. 19-21, 1981.

[Mea79] Mead, C.A. and Conway, L.A., *Introduction to VLSI Systems*, Addison-Wesley Publishing Co., Reading, Mass., 1980.

[Per77] Persky, G.; Deutsch, D.N. and Schweikert, D.G., "LTX - A Minicomputer-Based System For Automated LSI Layout," *Journal of Design Automation and Fault-Tolerant Computing*, Vol. 1, No. 3, 1977, 217-255.

[Pre78] Preas, B.T. and Gwyn, C.W., "Methods for Hierarchical Automatic Layout of Custom LSI Circuit Masks," *Proceedings of the Fifteenth Design Automation Conference*, 1978, 206-212.

[Riv81] Rivest, R.L., "The 'PI' (Placement and Interconnect) System," Massachusetts Institute of Technology (Preprint), 1981.

[RBM81] Rivest, R.L.; Baratz, A.E. and Miller, G., "Provably Good Channel Routing Algorithms" to appear in *Proceedings of the CMU Conference on VLSI Systems and Computations*, Oct. 19-21, 1981.

[Szy81] Szymanski, T., Personal Communication through R. Rivest, 1981.

[Tho80] Thompson, C.D., "A Complexity Theory for VLSI," Ph.D. dissertation, Department of Computer Science, Carnegie-Mellon University, 1980.

[Tin76] Ting, B.S.; Kuh, E.S. and 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, 1976., 768-778.

[Tom80] Tompa, M., "An Optimal Solution to a Wire-Routing Problem," *Proceedings of the Twelfth Annual ACM Symposium on Theory of Computing*, 1980, 161-176.

Biographical Note

Alan Edward Baratz was born in Philadelphia, Pennsylvania on November 10, 1954. At the age of 8, he moved to Las Vegas, Nevada where he lived until graduating from Valley High School in June 1972. Following his high school graduation, Mr. Baratz attended the University of California at Los Angeles where he received his B.S. degree cum laude in Mathematics/Computer Science in June 1976. The following September, Mr. Baratz began his graduate work in the Department of Electrical Engineering and Computer Science at the Massachusetts Institute of Technology. He soon joined the Theory of Computation research group in the Laboratory for Computer Science, where he worked under the supervision of Dr. Ronald Rivest. Mr. Baratz received his S.M. degree from the Department of Electrical Engineering and Computer Science in June 1979, and his Ph.D. degree in September 1981. During this time he married the former Raquel Sari Schlamm.

Mr. Baratz will be joining the Communication Network Group at the I.B.M. Thomas J. Watson Research Center in September 1981.