Chapter 8

Global Routing

In the placement phase, the exact locations of circuit blocks and pins are determined. A netlist is also generated which specifies the required interconnections. Space not occupied by the blocks can be viewed as a collection of regions. These regions are used for routing and are called as *routing regions.* The process of finding the geometric layouts of all the nets is called *routing.* Nets must be routed within the routing regions. In addition, nets must not short-circuit, that is, nets must not intersect each other.

The input to the general routing problem is:

1. Netlist,

2. Timing budget for nets, typically for critical nets only,

3. Placement information including location of blocks, locations of pins on the block boundary as well as on top due to ATM model (sea-of-pins model), location of I/O pins on the chip boundary as well as on top due to C4 solder bumps,

4. RC delay per unit length on each metal layer, as well as RC delay for each type of via.

The objective of the routing problem is dependent on the nature of the chip. For general purpose chips, it is sufficient to minimize the total wire length, while completing all the connections. For high performance chips, it is important to route each net such that it meets its timing budget. Usually routing involves special treatment of such nets as clock nets, power and ground nets. In fact, these nets are routed separately by special routers.

A VLSI chip may contain several million transistors. As a result, tens of thousands of nets have to be routed to complete the layout. In addition, there may be several hundreds of possible routes for each net. This makes the routing problem computationally hard.

One approach to the general routing problem is called *Area Routing,* which is a single phase routing technique. This technique routes one net at a time

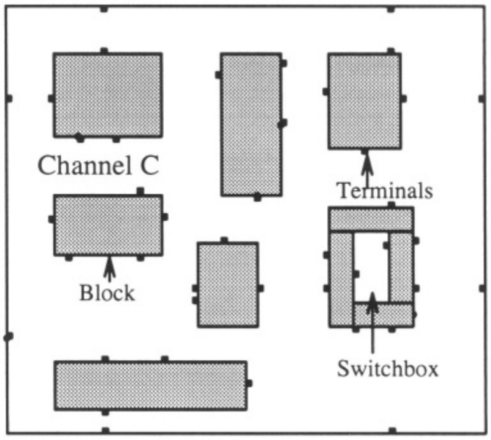

Figure 8.1: Layout of circuit blocks and pins after placement.

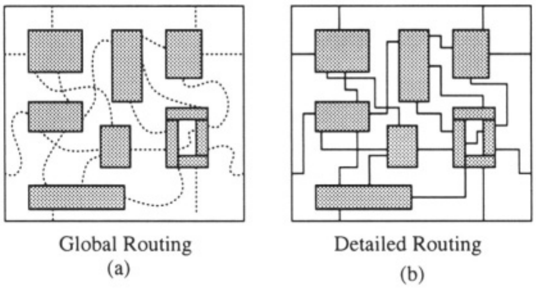

Figure 8.2: Two phases of Routing.

considering all the routing regions. However, this technique is computationally infeasible for an entire VLSI chip and is typically used for specialized problems, and smaller routing regions.

The traditional approach to routing, however, divides the routing into two phases. The first phase is called *global routing* and generates a 'loose' route for each net. In fact it assigns a list of routing regions to each net without specifying the actual geometric layout of wires (see Figure 8.2(a)). The second phase, which is called *detailed routing,* finds the actual geometric layout of each net within the assigned routing regions (see Figure 8.2(b)). Unlike global routing, which considers the entire layout, a detailed router considers just one region at a time. The exact layout is produced for each wire segment assigned to a region, and vias are inserted to complete the layout. In fact, even when the routing problem is restricted to a routing region, such as channels (see definition below), it cannot be solved in polynomial time, i.e., the channel routing problem is NP-complete [Szy85].

In this book, we will briefly describe area routing techniques. Basically, we will follow the two phase approach to routing. In the following, we will discuss global and detailed routing in more detail. Figure8.3 shows a typical two phase routing approach.

The global routing consists of three distinct phases; Region definition, Region Assignment, and Pin assignment. The first phase of global routing is to partition the entire routing space into routing regions. This includes spaces between blocks and above blocks, that is, OTC areas. Between blocks there are two types of routing regions: channels and 2D-switchboxes. Above blocks, the entire routing space is available, however, we partition it into smaller regions called 3D-switchboxes. Each routing region has a capacity, which is the maximum number of nets that can pass through that region. The capacity of a region is a function of the design rules and dimensions of the routing regions and wires. A *channel* is a rectangular area bounded by two opposite sides by the blocks. Capacity of a channel is a function of the number of layers (l), height (h) of the channel, wire width(w) and wire separation(s), i.e., $Capacity = \frac{l \times h}{w+s}$. For example, if for channel C shown in Figure 8.1, $l = 2$, $h = 18\lambda$, $w = 3\lambda$, $s = 3\lambda$, then the capacity is $\frac{2\times 18}{3+3} = 6$. In a five layer process, only M1, M2 and M3 are used for channel routing. Note that channel may also have pins in th middle. The pins in the middle are actually used to make connections to nets routed in 3D-switchboxes. A *2D-switchbox* is a rectangular area bounded on all sides by blocks. It has pins on all four sides as well as pins in the middle. The pins in the middle are actually used to make connections to nets routed in 3D-switchboxes. A *3D-switchbox* is a rectangular area with pins on all six sides. The pins on the bottom are the pins which allow for connections to nets in channels, 2D-switchboxes and nets using ATM (sea-of-pins) on top of blocks. The pins on the top may be required to connect to C4 solder bumps.

Consider the five metal layer process and assume that blocks use upto third metal layer for internal routing. In this case, channel and 2D-switchboxes will be used in Ml, M2 and M3 to route regions between the blocks. Furthermore, the M4 and M5 routing space will be partitioned into several smaller routing regions. The three different routing regions are shown in Figure ref3dswitchbox-6. Another approach to region definition is to partition the M4 and M5 along block boundaries. In this case, channels and 2D-switchboxes will be routed in five metal layers. In addition the regions on top of blocks will be 3D-switchboxes and need to be routed in M4 and M5.

The second phase of global routing can be called region assignment. The purpose of this phase to identify the sequence of regions through which a net will be routed. This phase must take into account the timing budget of each net and routing congestion of each routing region. After the region assignment, each net is assigned a pin on region boundaries. This phase of global routing is called pin assignment. The region boundaries can be between two channels, channel and 3D-switchbox, 2D-switchbox and a 3D-switchbox among others. The pin assignment phase allows the regions to be somewhat independent.

After global routing is complete, the output is pin locations for each net on

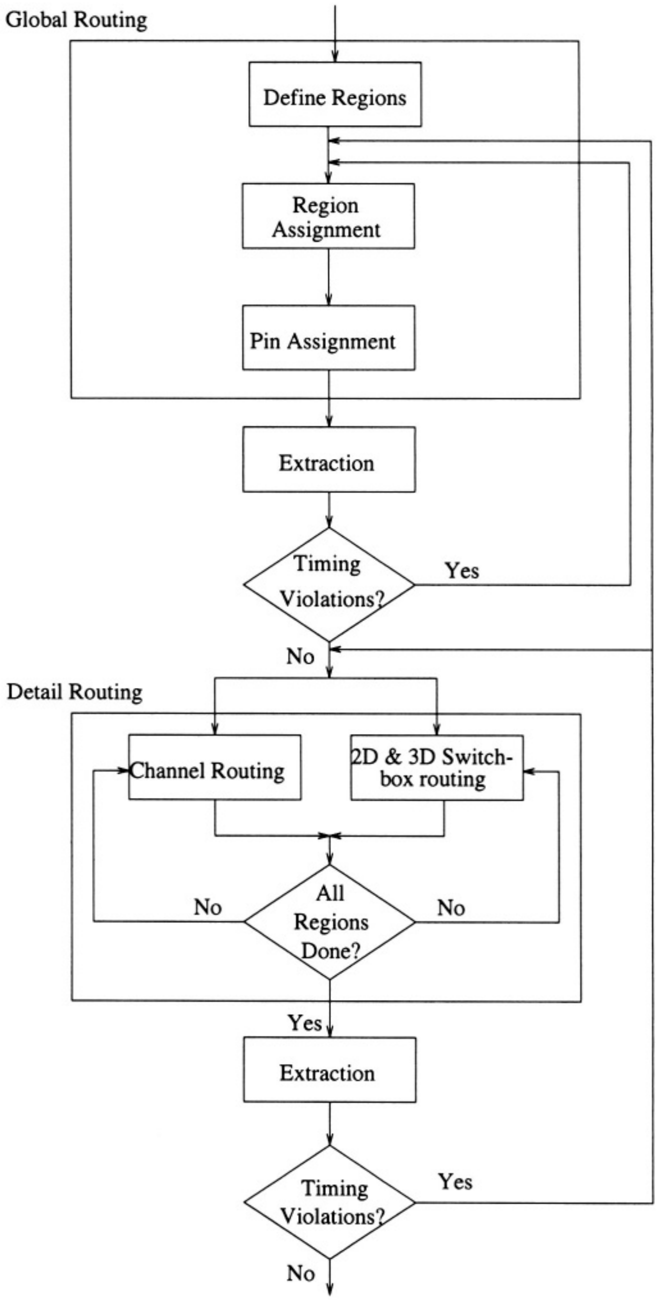

Figure 8.3: The two phase routing flow.

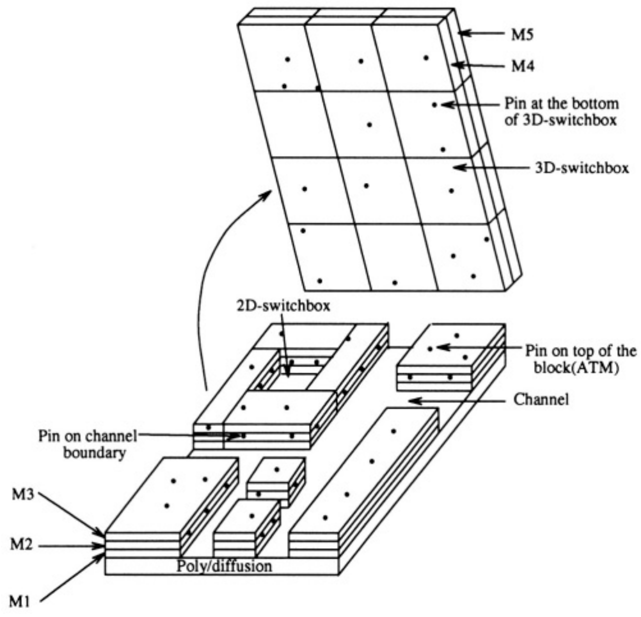

Figure 8.4: Three different routing regions.

the all the region boundaries it crosses. Using this information, we can extract the length of the net and estimate its delay. If some net fails to meet its timing budget, it needs to be ripped-up or global routing phase needs to be repeated.

Detailed routing includes channel routing, 2D-switchbox and 3D-switchbox routing. Typically channels and 2D-switchboxes should be routed first, since channels may expand. After channels and 2D-switchboxes have been routed, the pin locations for 3D-switchboxes are fixed and then their routing can be completed. Channels are routed in a specific order to minimize the impact of channel expansion on the floorplan.

After detailed routing is completed, exact wire geometry can be extracted and used to compute RC delays. The delay model not only considers the geometry (length, width, layer assignments and vias) of a net, but also the relationship of this net with other nets. If some nets fail to meet their timing constraints, they need to be ripped-up or detailed routing of the specific routing region needs to be repeated.

In this chapter, we discuss techniques for global routing. We will also discuss some techniques that can be used for area routing. In Chapter 7 we will discuss the detailed routing techniques. Chapter 8 is dedicated to routing techniques on top of blocks. Chapter 9 discusses the routing of special nets, such as clock and power nets.

Global routing has to deal with two types of nets. Critical nets, which must be routed in high performance layers and other nets. Fir very critical

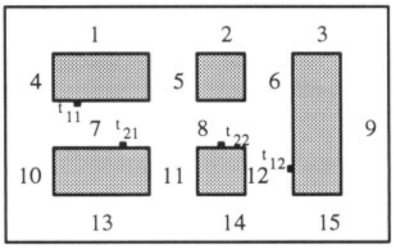

Figure 8.5: The horizontal and vertical channels.

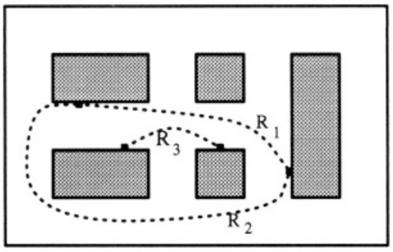

Figure 8.6: Two possible connections between source and target.

nets, global router must use a path which takes the nets from its channel (or 2D-switchbox) through 3D-switchboxes to the termination point in a channel or a 2D-switchbox or C4 bump. Other nets which may not need use of M4 and M5 can be routed through a sequence of channels. Global router must not allocate more nets to a routing region than the region capacity. Let us illustrate this concept of global routing by an example. Suppose that each channel in Figure 8.5 has unit capacity. We consider routing of two nets $N_1 = \{t_{11}, t_{12}\}$ and $N_2 = \{t_{21}, t_{22}\}$. There are several possible routes for net N_1. Two such routes R_1 and R_2 are shown in Figure 8.6. If the objective is to route just N_1, obviously R_1 is a better choice. However, if both N_1 and N_2 are to be routed, it is not possible to use R_1 for N_1 since it would make N_2 unroutable. Thus global routing is computationally hard since it involves trade-offs between routability of all nets and minimization of the objective function. In fact, we will see that global routing of even a single multi-terminal net is NP-complete. In order to simplify presentation, in the rest of the chapter, we will consider global routing with channels and 2D-switchboxes. We will note exceptions for 3D-switchboxes, as and when appropriate. In addition, we will assume that timing constraints are translated into length constraints, hence the objective is to route each net within its length budget.

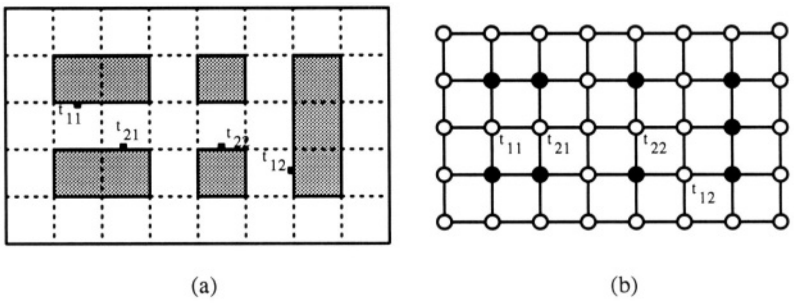

Figure 8.7: Grid Graph Model.

8.1 Problem Formulation

The global routing problem is typically studied as a graph problem. The routing regions and their relationships and capacities are modeled as graphs. However, the design style strongly effects the graph models used and as a result, there are several graph models. Before presenting the problem formulation of global routing, we discuss three different graph models which are commonly used.

The graph models for area routing capture the complete layout information and are used for finding exact route for each net. On the other hand, graph models for global routing capture the adjacencies and routing capacities of routing regions. We discuss three graph models *viz;* grid graph model, checker board model and the channel intersection graph model. Grid graphs are most suitable for area routing while the channel intersection graphs are most suitable for global routing.

1. **Grid Graph Model:** The simplest model for routing is a *grid graph.* The grid graph $G_1 = (V_1, E_1)$ is a representation of a layout. In this model, a layout is considered to be a collection of unit side square cells arranged in a $h \times w$ array. Each cell c_i is represented by a vertex v_i, and there is an edge between two vertices v_i and v_j, if cells c_i and c_j are adjacent. A terminal in cell c_i is assigned to the corresponding vertex v_i. The capacity and length of each edge is set equal to one, i.e., $c(e) = 1$, $l(e) = 1$. It is quite natural to represent blocked cells by setting the capacity of the edges incident on the corresponding vertex to zero. Figure 8.7(b) shows a grid graph model for a layout in Figure 8.7(a).

   Given a grid graph, and a two terminal net, the routing problem is simply to find a path connecting the vertices, corresponding to the terminals, in the grid graph. Whereas, for a multi-terminal net, the problem is to find a Steiner tree in the grid graph.

   The more general routing problems may consider k-dimensional grid graphs, however, the general techniques for routing essentially remain

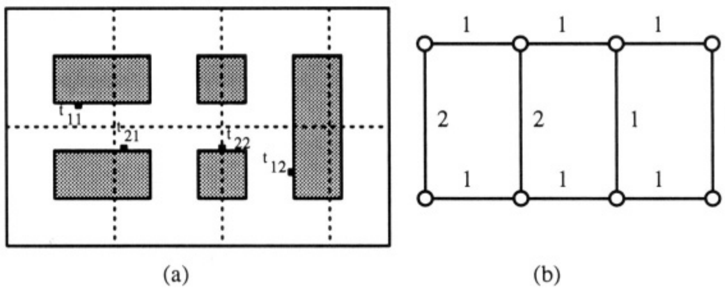

Figure 8.8: Checker Board Graph.

the same in all grids. In fact, routing in grids, should be considered as area routing, since the actual detailed route of the net is determined.

2. **Checker Board Model:** Checker board model is a more general model than the grid model. It approximates the entire layout area as a 'coarse grid' and all terminals located inside a coarse grid cell are assigned that cell number. The checker board graph $G_2 = (V_2, E_2)$ is constructed in a manner analogous to grid graph. The edge capacities are computed based on the actual area available for routing on the cell boundary. Figure 8.8(b) shows a checker board graph model of a layout in Figure 8.8(a). Note that the partially blocked edges have unit capacity, whereas, the unblocked edges have a capacity of 2. Given the cell numbers of all terminals of a net, the global routing routing problem is to find a routing in the coarse grid graph.

   A checker board graph can also be formed from a cut tree of floorplan. A block b_i in a floorplan is represented by a vertex v_i and there is an edge between vertices v_i and v_j if the corresponding blocks b_i and b_j are adjacent to each other. Note that, unlike the cells in a grid, two adjacent modules in a cut tree of a floorplan may not entirely share a boundary with each other. Figure 8.9(b) shows an example of a checker board graph for a cut tree of a floorplan in Figure 8.9(a).

3. **Channel Intersection Graph Model:** The most general and accurate model for global routing is the channel intersection model. Given a layout, we can define a channel intersection graph $G_3 = (V_3, E_3)$, where each vertex $v_i \in V_3$ represents a channel intersection CI_i. Two vertices v_i and v_j are adjacent in G_3 if there exists a channel between CI_i and CI_j. In other words, the channels appear as edges in G_3. Figure 8.10(b) shows a channel intersection graph for a layout in Figure 8.10(a). Let $c(e)$ and $l(e)$ be the capacity and length of a channel associated with edge $e \in E_3$. The channel intersection graph should be extended to include the pins as vertices so that the connections between the pins can be considered in this graph. For example, the extended channel intersection graph in

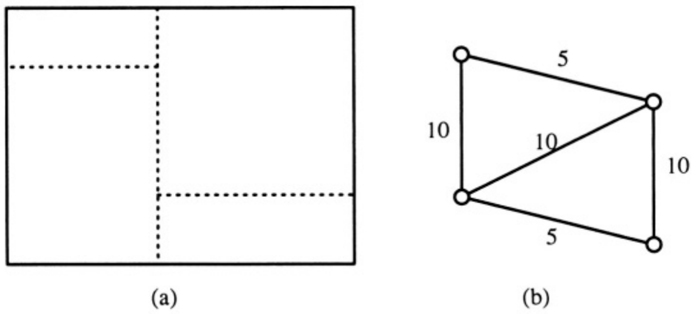

Figure 8.9: Checker Board Graph of a Floorplan.

Figure 8.11(b) is obtained by adding vertices representing terminals to the channel intersection graph in Figure 8.10(b).

In the rest of the chapter, the type of routing graph will be clear from the context and will be denoted as $G = (V, E)$.

The global routing problem of two terminal nets is to find path for each net in the routing graph such that the desired objective function is optimized. In addition, the number of nets using each edge (traffic through the corresponding channel) should not violate the capacity of that edge. For example, the global routes for nets N_1 and N_2 are shown as the paths P_1 and P_2 in Figure 8.11 (b). It is obvious from the example that routing of one net at a time causes ordering problem for nets. It is important to note that the overall optimal solution may consist of suboptimal solutions of individual nets.

For a net with more than two terminals, the path model discussed above is not appropriate. In fact, global routing of multi-terminal nets can be formulated as a Steiner tree problem. As defined in Chapter 3, a Steiner tree is a tree interconnecting a set of specified points called *demand points* and some other points called *Steiner points*. The number of Steiner points is arbitrary. The global routing problem can be viewed as a problem of finding a Steiner tree for each net in the routing graph such that the desired objective function is optimized. In addition, the capacity of the edges must not be violated. As discussed earlier, a typical objective function is to minimize the total length of selected Steiner trees. In high-performance circuits, the objective function is to minimize the maximum wire length of selected Steiner trees. A more precise objective function for high-performance circuits is to minimize the maximum diameter of selected Steiner trees. The *diameter* of a Steiner tree is defined as the maximum length of a path between any two vertices in the Steiner tree. If there is no feasible solution to an instance of a global routing problem, then the netlist is not routable as the capacity constraints of some edges can not be satisfied. In such cases, the placement phase has to be carried out again.

The formal statement of global routing problem is as follows: Given, a netlist $\mathcal{N} = \{N_1, N_2, \ldots, N_n\}$, the routing graph $G = (V, E)$, find a Steiner tree T_i

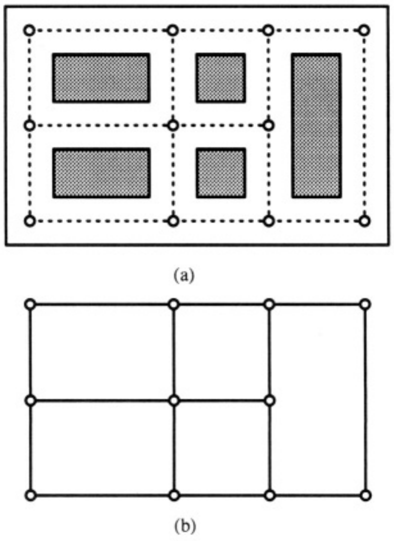

Figure 8.10: Channel intersection graph.

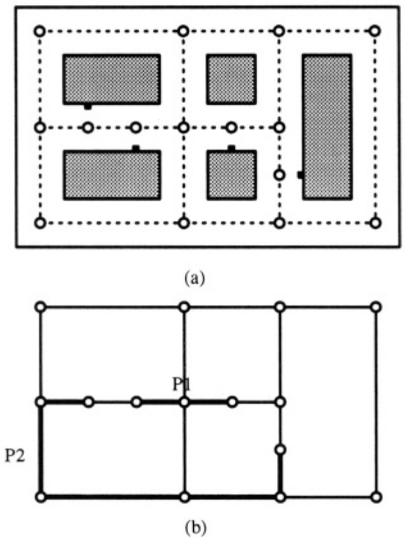

Figure 8.11: Extended channel intersection graph.

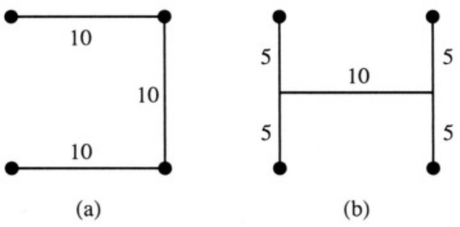

Figure 8.12: Difference between diameter and length of a net.

for each net $N_i, 1 \leq i \leq n$, such that, the capacity constraints are not violated, i.e., $U(e_j) \leq c(e_j)$ for all $e_j \in E$, where $U(e_j) = \sum_{i=1}^{n} x_{ij}$ is the number of wires that pass through the channel corresponding to edge e_j ($x_{ij} = 1$ if e_j is in T_i, it is 0 otherwise). A typical objective function is to minimize the total wire length ($\sum_{i=1}^{n} L(T_i)$), where $L(T_i)$ is the length of Steiner tree T_i.

In the case of high-performance chips the objective function is to minimize the maximum wire length ($\max_{i=1}^{n} L(T_i)$). Note that minimization of maximum wire length may not directly reduce the diameter of the Steiner trees. Consider the example shown in Figure 8.12. The two Steiner trees are both of length 30, but the Steiner tree shown in Figure 8.12(b) has diameter equal to 20, which is much smaller that the diameter of the tree shown in Figure 8.12(a).

8.1.1 Design Style Specific Global Routing Problems

The objective of global routing in each design style is different. We will discuss the global routing problem for full custom, standard cell and gate array. Global routing problem for FPGA and MCM is discussed in Chapters 11 and 12 respectively.

1. **Full custom:** The global routing problem formulation for full custom design style is similar to the general formulation described above. The only difference is how capacity constraints guide the global routing solution. In the general formulation the edge capacities cannot be violated. In full custom, since channels can be expanded, some violation of capacity constraints is allowed. However, major violation of capacities which leads to significant changes in placement are not allowed. In such case, it may be necessary to carry out the placement again.

2. **Standard cell:** In the standard cell design style, at the end of the placement phase, the location of each cell in a row is fixed. In addition, the capacity and location of each feedthrough is fixed. However, the channel heights are not fixed. They can be changed by varying the distance between adjacent cell rows to accommodate wires assigned by a global router. As a result, they do not have a predetermined capacity. On the other hand, feedthroughs have predetermined capacity. The area of a

standard cell layout is determined by the total cell row height and the total channel height, where the total cell row height is the summation of all cell row heights and the total channel height is the summation of all channel heights. As the total cell row height is fixed, the layout area could only be minimized by minimizing the total channel height. As a result, standard cell global routers attempt to minimize the total channel height. Other optimization functions include the minimization of the total wire length and the minimization of the maximum wire length.

The edge set of $G = (V, E)$ are partitioned into two disjoint sets E^v and E^h, i.e., $E = E^v \cup E^h$. Edges in E^v represent feedthroughs, whereas, edges in E^h represent channels. Capacity of each edge $e_j \in E^v$ is equal to the number of wires that can pass through the corresponding feedthrough. Whereas, the capacity of an edge $e \in E^h$ is set to infinity. Let E^h_{ij} represent a j^{th} edge in i^{th} channel and let $E^h_i = \cup_{\forall j} E^h_{ij}$ for all $i = 1, 2, \ldots, p$, where p is the total number of channels in the layout.

Thus, the global routing problem is to find a Steiner tree T_i for each net $N_i, 1 \leq i \leq n$, such that, the capacity constraints are not violated, i.e., $U(e_j) \leq c(e_j)$ for all $e_j \in E^v$, where $U(e_j) = \sum_{i=1}^{n} x_{ij}$ is the number of wires that go through the feedthrough corresponding to edge e_j ($x_{ij} = 1$ if e_j is in T_i, it is 0 otherwise). The optimization function is either to minimize the total wire length ($\sum_{i=1}^{n} L(T_i)$) or to minimize the maximum wire length ($\max_{i=1}^{n} L(T_i)$) or to minimize the total channel height $\sum_{i=1}^{p} \max\{U(e) | e \in E^h_i\}$. $L(T_i)$ is the length of Steiner tree T_i.

If there is no feasible solution for a global routing problem, feedthrough capacities are not sufficient, (see Figure 8.13.) Additional feedthroughs should be inserted in order to allow global routing.

Recently, a new approach, called over-the-cell routing, has been presented for standard cell design, in which, in addition to the channels and feedthroughs the over-the-cell areas are available for routing. Availability of over-the-cell areas changes the global routing problem. In Chapter 8, this approach is discussed in detail.

3. **Gate array:** In gate array design style, the size and location of all cells and the routing channels and their capacities are fixed by the architecture. This is the key difference between gate array and other design styles. Unlike the full custom design style and standard cell design style the primary objective of the global routing in gate arrays is to guarantee *routability.* The secondary objective may be to minimize the total wire length or to minimize the maximum wire length. Other than these objectives, the formulation of global routing problem in gate array design style is same as the general global routing formulation. If there is no feasible solution to a given instance of global routing problem, the netlist can not be routed (see Figure 8.14). In this case, the placement phase has to be carried out again as the capacity of routing channels is fixed in gate array design style.

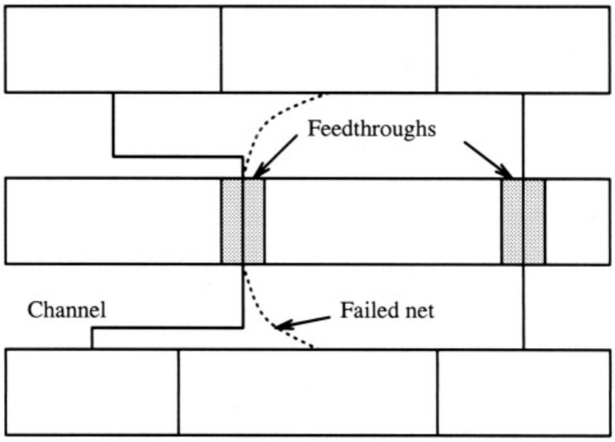

Figure 8.13: Not enough feedthroughs in standard cell design style.

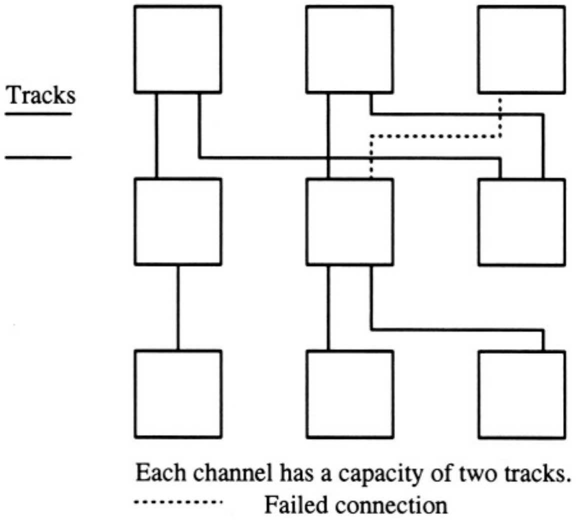

Figure 8.14: Not all nets are routable in gate array design style.

8.2 Classification of Global Routing Algorithms

Basically, there are two kinds of approaches to solve global routing problem; the sequential and the concurrent.

1. **Sequential Approach:** In this approach, as the name suggests, nets are routed one by one. However, once a net has been routed it may block other nets which are yet to be routed. As a result, this approach is very sensitive to the order in which the nets are considered for routing. Usually, the nets are sequenced according to their criticality, half perimeter of the bounding rectangle and number of terminals. The criticality of a net is determined by the importance of the net. For example, clock net may determine the performance of the circuit and therefore it is considered to be a very important net. As a result, it is assigned a high criticality number. The nets on the critical paths are assigned high criticality numbers since they also play a key role in determining the performance of the circuit. The criticality number and other factors can be used to sequence nets. However, sequencing techniques do not solve the net ordering problem satisfactorily. In a practical router, in addition to a net ordering scheme an improvement phase is used to remove blockages when further routing of nets is not possible. However, this also may not overcome the shortcoming of sequential approach. One such improvement phase involves 'rip-up and reroute' [Bol79, DK82] technique, while other involves 'shove-aside' technique. In 'rip-up and reroute', the interfering wires are ripped up, and rerouted to allow routing of the affected nets. Whereas, in 'Shove-Aside' technique, wires that will allow completion of failed connections are moved aside without breaking the existing connections. Another approach [De86] is to first route simple nets consisting of only two or three terminals since there are few choices for routing such nets. Usually such nets comprise a large portion of the nets (up to 75%) in a typical design. After the simple nets have been routed, a Steiner tree algorithm is used to route intermediate nets. Finally, a maze routing algorithm is used to route the remaining multi-terminal nets (such as power, ground, clock etc.) which are not too numerous.

   The sequential approach includes:

   (a) Two-terminal algorithms:

      i. Maze routing algorithms

      ii. Line-probe algorithms

      iii. Shortest path based algorithms

   (b) Multi-terminal algorithms:

      i. Steiner tree based algorithms

2. **Concurrent Approach:** This approach avoids the ordering problem by considering routing of all the nets simultaneously. The concurrent

approach is computationally hard and no efficient polynomial algorithms are known even for two-terminal nets. As a result, integer programming methods have been suggested. The corresponding integer program is usually too large to be employed efficiently. Hence, hierarchical methods that work top down are employed to partition the problem into smaller subproblems, which can be solved by integer programming. The integer programming based concurrent approach will be presented in this chapter.

8.3 Maze Routing Algorithms

Lee [Lee61] introduced an algorithm for routing a two terminal net on a grid in 1961. Since then, the basic algorithm has been improved for both speed and memory requirements. Lee's algorithm and its various improved versions form the class of maze routing algorithms.

Maze routing algorithms are used to find a path between a pair of points, called the source(s) and the target(t) respectively, in a planar rectangular grid graph. The geometric regularity in the standard cell and gate array design style lead us to model the whole plane as a grid. The areas available for routing are represented as unblocked vertices, whereas, the obstacles are represented as blocked vertices. The objective of a maze routing algorithm is to find a path between the source and the target vertex without using any blocked vertex. The process of finding a path begins with the exploration phase, in which several paths start at the source, and are expanded until one of them reaches the target. Once the target is reached, the vertices need to be retraced to the source to identify the path. The retrace phase can be easily implemented as long as the information about the parentage of each vertex is kept during the exploration phase. Several methods of path exploration have been developed.

8.3.1 Lee's Algorithm

This algorithm, which was developed by Lee in 1961 [Lee61], is the most widely used algorithm for finding a path between any two vertices on a planar rectangular grid. The key to the popularity of Lee's maze router is its simplicity and and its guarantee of finding an optimal solution if one exists.

The exploration phase of Lee's algorithm is an improved version of the breadth-first search. The search can be visualized as a wave propagating from the source. The source is labeled '0' and the wavefront propagates to all the unblocked vertices adjacent to the source. Every unblocked vertex adjacent to the source is marked with a label '1'. Then, every unblocked vertex adjacent to vertices with a label '1' is marked with a label '2', and so on. This process continues until the target vertex is reached or no further expansion of the wave can be carried out. An example of the algorithm is shown in Figure 8.15. Due to the breadth-first nature of the search, Lee's maze router is guaranteed to find a path between the source and target, if one exists. In addition, it is guaranteed to be the shortest path between the vertices.

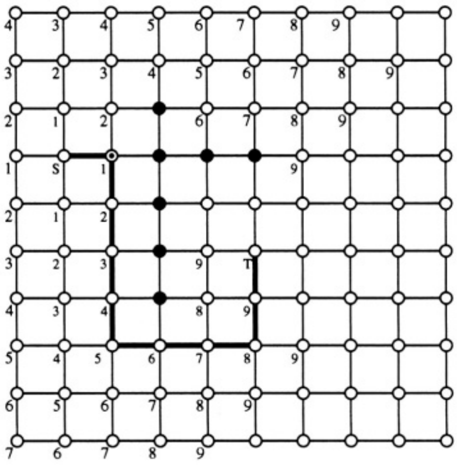

Figure 8.15: A net routed by Lee's algorithm.

The input to the Lee's Algorithm is an array *B*, the source(s) and target(t) vertex. $B[v]$, denotes if a vertex v is blocked or unblocked. The algorithm uses an array L, where $L[v]$ denotes the distance from the source to the vertex v. This array will be used in the procedure *RETRACE* that retraces the vertices to form a path *P*, which is the output of the Lee's Algorithm. Two linked lists *plist* (Propagation list) and *nlist* (Neighbor list) are used to keep track of the vertices on the wavefront and their neighbor vertices respectively. These two lists are always retrieved from tail to head. We also assume that the neighbors of a vertex are visited in counter-clockwise order, that is top, left, bottom and then right.

The formal description of the Lee's Algorithm appears in Figure 8.16. The time and space complexity of Lee's algorithm is $O(h \times w)$ for a grid of dimension $h \times w$.

The Lee's routing algorithm requires a large amount of storage space and its performance degrades rapidly when the size of the grid increases. There have been numerous attempts to modify the algorithm to improve its performance and reduce its memory requirements.

Lee's algorithm requires up to $k+1$ bits per vertex, where k bits are used to label the vertex during the exploration phase and an additional bit is needed to indicate whether the vertex is blocked. For an $h \times w$ grid, $k = \log_2(h \times w)$. Acker [Ake67] noticed that, in the retrace phase of Lee's algorithm, only two types of neighbors of a vertex need to be distinguished; vertices toward the target and vertices toward the source. This information can be coded in a single bit for each vertex. The vertices in wavefront L are always adjacent to the vertices in wavefront $L - 1$ and $L + 1$. Thus, during wave propagation, instead of using a sequence 1, 2, 3, ..., the wavefronts are labeled by a sequence

```
Algorithm LEE-ROUTER $(B, s, t, P)$
    input: $B, s, t$
    output: $P$
begin
    $plist = s$;
    $nlist = \phi$;
    $temp = 1$;
    $path\_exists$ = FALSE;
    while $plist \neq \phi$ do
        for each vertex $v_i$ in $plist$ do
            for each vertex $v_j$ neighboring $v_i$ do
            if $B[v_j]$ = UNBLOCKED then
                $L[v_j] = temp$;
                INSERT($v_j$,$nlist$);
                if $v_j = t$ then
                    $path\_exists$ = TRUE;
                    exit while;
        $temp = temp + 1$;
        $plist = nlist$;
        $nlist = \phi$;
    if $path\_exists$ = TRUE then RETRACE $(L, P)$;
    else path does not exist;
end.
```

Figure 8.16: Algorithm LEE-ROUTER.

like 0, 0, 1, 1, 0, 0, ... The predecessor of any wavefront is labeled differently from its successor. Thus, each scanned vertex is either labeled '0' or '1'. Besides these two states, additional states ('block' and 'unblocked') are needed for each vertex. These four states of each vertex can be represented by using exactly two bits, regardless of the problem size. Compared to Acker's scheme, Lee's algorithm requires at least 12 bits per vertex for a grid size of 2000 × 2000.

It is important to note that Acker's coding scheme only reduces the memory requirement per vertex. It inherits the search space of Lee's original routing algorithm, which is $O(h \times w)$ in the worst case.

8.3.2 Soukup's Algorithm

Lee's algorithm explores the grid symmetrically, searching equally in the directions away from target as well as in the directions towards it. Thus, Lee's algorithm requires a large search time. In order to overcome this limitation, Soukup proposed an iterative algorithm in 1978 [Sou78]. During each iteration, the algorithm explores in the direction toward the target without changing the direction until it reaches the target or an obstacle, otherwise it goes away from

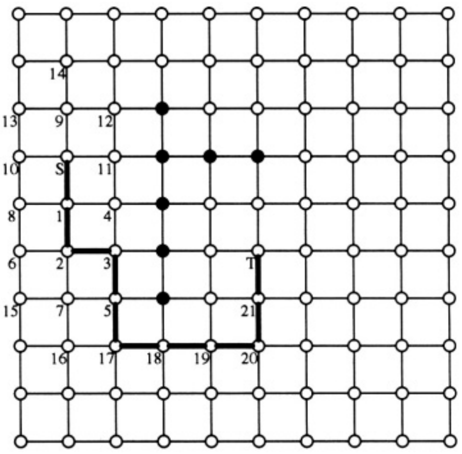

Figure 8.17: A net routed by Soukup's router.

the target. If the target is reached, the exploration phase ends. If the target is not reached, the search is conducted iteratively. If the search goes away from the target, the algorithm simply changes the direction so that it goes towards the target and a new iteration begins. However, if an obstacle is reached, the breadth-first search is employed until a vertex is found which can be used to continue the search in the direction toward the target. Then, a new iteration begins. Figure 8.17 illustrates the Soukup's algorithm with an example. In Figure 8.17, the number near a vertex indicates the order in which that vertex was visited.

Figure 8.18 contains the formal description of Soukup's Algorithm. The notation used in the algorithm is similar to that used in the Lee's algorithm except for the array L. We use $L[v]$ to denote the order in which the vertex v is visited during the exploration phase in this algorithm. Function DIR(v_1, v_2) returns the direction from v_1 to v_2. Function NGHBR-IN-DIR(v_1, v_2) returns the neighbor of v_2 which is in the direction from v_1 to v_2.

The Soukup's Algorithm improves the speed of Lee's algorithm by a factor of 10 to 50. It guarantees finding a path if a path between source and target exits. However, this path may not be the shortest one. The search method for this algorithm is a combined breadth-first and depth-first search. The worst case time and space complexities for this algorithm are both $O(h \times w)$, for a grid of size $h \times w$.

8.3.3 Hadlock's Algorithm

An alternative approach to improve upon the speed was suggested by Hadlock in 1977 [Had75]. The algorithm is called Hadlock's minimum detour algorithm. This algorithm uses $A*$ search method.

```
Algorithm SOUKUP-ROUTER ($B, s, t, P$)
    input: $B, s, t$
    output: $P$
begin
    $plist = s$;
    $nlist = \phi$;
    $temp = 1$;
    $path\_exists$ = FALSE;
    while $plist \neq \phi$ do
        for each vertex $v_i$ in $plist$ do
            for each vertex $v_j$ neighboring $v_i$ do
                if $v_j = t$ then
                    $L[v_j] = temp$;
                    $path\_exists$ = TRUE;
                    exit while;
                if $B[v_j]$ = UNBLOCKED then
                (* If the direction of the search is toward the
                target, the search continues in this direction *)
                    if $DIR(v_i, v_j)$ = TO-TARGET
                    then $L[v_j] = temp$;
                        $temp = temp + 1$;
                        INSERT ($v_j$, $plist$);
                        while $B$[NGHBR-IN-DIR($v_i, v_j$)]=
                            UNBLOCKED do
                            $v_j$ = NGHBR-IN-DIR($v_i, v_j$);
                            $L[v_j] = temp$;
                            $temp = temp + 1$;
                            INSERT ($v_j$, $plist$);
                    else
                        $L[v_j] = temp$;
                        $temp = temp + 1$;
                        INSERT ($v_j, nlist$);
        $plist = nlist$;
        $nlist = \phi$;
    if $path\_exists$ = TRUE then RETRACE ($L, P$);
    else path does not exist;
end.
```

Figure 8.18: Algorithm SOUKUP-ROUTER.

```
Algorithm HADLOCK-ROUTER($B, s, t, P$)
    input: $B, s, t$
    output: $P$
begin
    $plist = s$;
    $nlist = \phi$;
    $detour = 0$;
    $path\_exists$ = FALSE;
    while $plist \neq \phi$ do
        for each vertex $v_i$ in $plist$ do
            for all vertices $v_j$ neighboring $v_i$ do
                if $B[v_j]$ = UNBLOCKED then
                    $D[v_j]$ = DETOUR-NUMBER($v_j$);
                    INSERT ($v_j$,$nlist$);
                    if $v_j = t$ then
                        $path\_exists$ = TRUE;
                        exit while;
        if $nlist = \phi$ then
            $path\_exists$ = FALSE;
            exit while;
        $detour$ = MINIMUM-DETOUR( $nlist$ );
        for each vertex $v_k$ in $nlist$ do
            if $D[v_k] = detour$ then INSERT($v_k, plist$);
        DELETE ($nlist, plist$);
    if $path\_exists$ = TRUE then RETRACE ($L, P$);
    else path does not exist;
end.
```

Figure 8.19: Algorithm HADLOCK-ROUTER.

Hadlock observed that the length of a path (P) connecting source and target can be given by $M(s,t) + 2d(P)$, where $M(s,t)$ is Manhattan distance between source and target and $d(P)$ is the number of vertices on path P that are directed away from the target. The length of P is minimized if and only if d is minimized as $M(s,t)$ is constant for given pair of source and target. This is the essence of Hadlock's algorithm. The exploration phase, instead of labeling the wavefront by a number corresponding to the distance from the source, uses the detour number. The detour number of a path is the number of times that the path has turned away from the target. Figure 8.20 illustrates the Hadlock's algorithm with an example. In Figure 8.20, the number near a vertex indicates the order in which that vertex was visited.

A formal description of Hadlock's Algorithm is given in Figure 8.19. Function DETOUR-NUMBER(v) returns detour number of a vertex v. Procedure DELETE(*nlist, plist*) deletes the vertices which are in *plist* from *nlist*. Func-

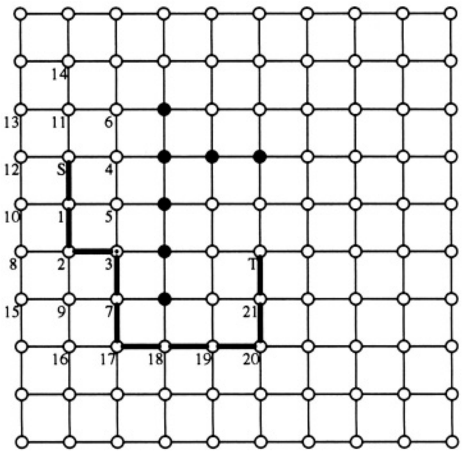

Figure 8.20: A net routed by Hadlock's Algorithm.

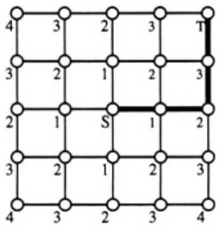

Figure 8.21: Lee's algorithm in the worst case.

tion MINIMUM-DETOUR(*nlist*) returns the minimum detour number among all vertices in the list *nlist.*

The worst case time and space complexity of Hadlock's algorithm is $O(h \times w)$ for a grid of size $h \times w$.

8.3.4 Comparison of Maze Routing Algorithms

Maze routing algorithms are grid based methods. The time and space required by these algorithms depend linearly on their search space.

The search in Lee's algorithm is conducted by using a wave propagating from the source. The algorithm searches symmetrically in every direction, using the breath-first search technique. Thus, it guarantees finding a shortest path between any two vertices if such a path exists. However, the worst case happens when the source is located at the center and the target is located at a corner of routing area, in which all the vertices have to be scanned before the

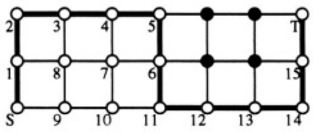

Figure 8.22: Soukup's algorithm does not find the shortest path.

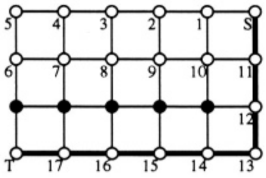

Figure 8.23: Soukup's algorithm in the worst case.

target is reached, (see Figure 8.21.)

The Soukup's algorithm remedies the shortcoming of the breadth-first search method by using a depth-first search until an obstacle is encountered. If an obstacle is encountered, a breadth-first search method is used to get around the obstacle. The search time in Soukup's algorithm is usually smaller than the Lee's algorithm due to the nature of depth-first search method. However, this algorithm may not find a shortest path between the source and target. In Figure 8.22, the Soukup's algorithm explores all the vertices and does not find the shortest path between s and t. The worst case of Soukup's algorithm occurs when the search goes in the direction of the target, which is opposite the direction of the passageway through the obstacle. Figure 8.23 shows an example in which Soukup's algorithm scans all vertices while finding a path between s and t.

The Hadlock's algorithm aims at both reducing the search time and finding an optimal path between given two vertices. Basically, the Hadlock's algorithm

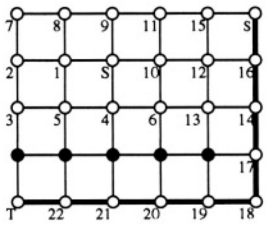

Figure 8.24: Hadlock's algorithm in the worst case.

is a breadth-first search method. As a result, it finds a shortest path if one exists. The difference between the Hadlock's algorithm and Lee's algorithm is the way in which the wavefront is labeled. The Hadlock's algorithm label the wavefront by the detour number instead of the distance from the source used in the Lee's algorithm. In this way, the search can prefer the direction toward the target to the direction away from the target. This search time is shorter than the Lee's algorithm. When the direction of the search goes toward the target and opposite the passageway through the obstacle, the worst case happens (see Figure 8.24).

All the maze routers and many of their variations are grid based methods. Information must be kept for each grid node. Thus, a very large memory space is needed to implement these algorithms for a large grid. To give an approximate estimate, a chip of size $10000\lambda \times 10000\lambda$ requires as much as 350 MBytes of memory and 66 seconds to route one net on a 15MIPS workstation. There may be 5000 to 10000 nets in a typical chip. Such numbers make these maze routing algorithms infeasible for large chips. In order to reduce the large memory requirements and run times, line-probe algorithms were developed.

8.4 Line-Probe Algorithms

The line-probe algorithms were developed independently by Mikami and Tabuchi in 1968 [MT68], and Hightower in 1969 [Hig69]. The basic idea of a line probe algorithm is to reduce the size of memory requirement by using line segments instead of grid nodes in the search. The time and space complexities of these line-probe algorithms is $O(L)$, where L is the number of line segments produced by these algorithms.

The basic operations of these algorithms are as follows. Initially, lists *slist* and *tlist* contain the line segments generated from the source and target respectively. The generated line segments do not pass through any obstacle. If a line segment from *slist* intersects with a line segment in *tlist*, the exploration phase ends; otherwise, the exploration phase proceeds iteratively. During each iteration, new line segments are generated. These segments originate from 'escape' points on existing line segments in *slist* and *tlist*. The new line segments generated from *slist* are appended to *slist*. Similarly, segments generated from a segment in *tlist* are appended to *tlist*. If a line segment from *slist* intersects with a line segment from *tlist*, then the exploration phase ends. The path can be formed by retracing the line segments in set *tlist*, starting from the target, and then going through the intersection, and finally retracing the line segments in set *slist* until the source is reached.

The data structures used to implement these algorithms play an important role in the efficiency considerations of the search for obstructions to probes. Typically two lists, one for the horizontal lines and one for the vertical lines are used. The use of two separate lists allows lines parallel to the direction of the probe to be ignored, thus expediting the search.

The Mikami and the Hightower algorithms differ only in the process of

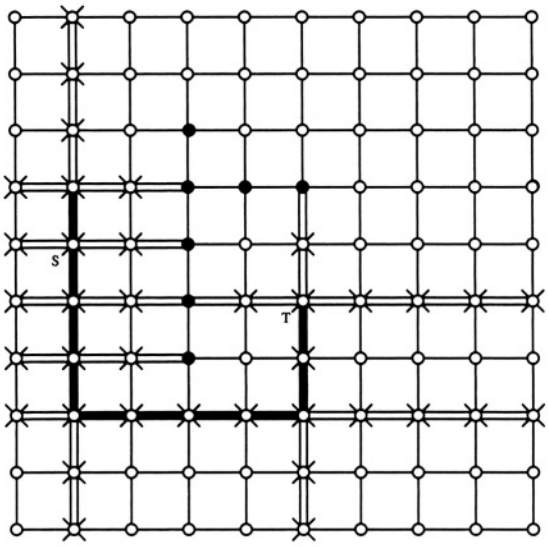

Figure 8.25: A net routed by Mikami-Tabuchi's Algorithm.

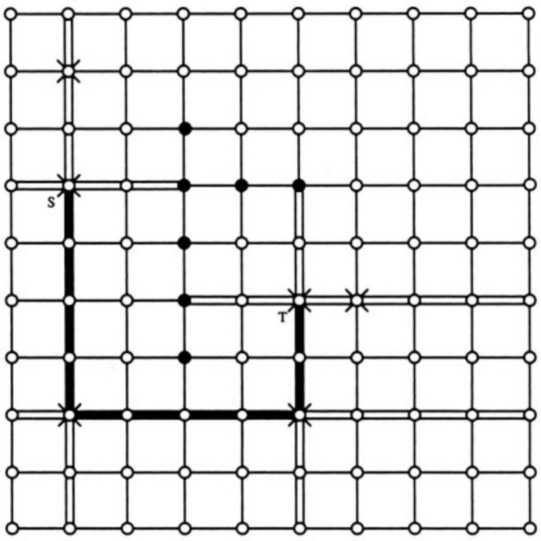

Figure 8.26: A net routed by Hightower's Algorithm.

```
Algorithm LINE-PROBE-ROUTER($s, t, P$)
    input: $s, t$
    output: $P$
begin
    $new\_slist$ = line segments generated from $s$;
    $new\_tlist$ = line segments generated from $t$;
    while $new\_slist \neq \phi$ and $tlist \neq \phi$ do
        $slist = new\_slist$;
        $tlist = new\_tlist$;
        for each line segment $l_i$ in $slist$ do
            for each line segment $l_j$ in $tlist$ do
                if INTERSECT($l_i, l_j$)=TRUE then
                    $path\_exists$ = TRUE;
                    exit while;
        $new\_slist = \phi$;
        for each line segment $l_i$ in $slist$ do
            for each escape point $e$ on $l_i$ do
                GENERATE($l_k, e$);
                INSERT($l_k, new\_slist$);
        $new\_tlist = \phi$;
        for each line segment $l_i$ in $tlist$ do
            for each escape point $e$ on $l_i$ do
                GENERATE($l_k, e$);
                INSERT($l_k, new\_tlist$);
    if $path\_exists$=TRUE then RETRACE;
    else a path can not be found;
end.
```

Figure 8.27: Algorithm LINE-PROBE-ROUTER.

choosing escape points. In Mikami's algorithm, every grid node on the line segment is an 'escape' point, which generates new perpendicular line segments. This search is similar to the breadth-first search, and is guaranteed to find a path if one exists. However, the path may not be the shortest one. Figure 8.25 shows a path generated by Mikami's algorithm. On the other hand, Hightower's algorithm makes use of only a single 'escape' point on each line segment. In the simple case of a probe parallel to the blocked vertices, the escape point is placed just past the endpoint of the segment. Figure 8.26 shows a path generated by Hightower's algorithm. Hightower has described three such processes, designed to help the router find a path around different types of obstacles. The disadvantage in generating fewer escape points in Hightower's algorithm essentially means that it may not be able to find a path joining two points even when such a path exists.

A formal description of these two algorithm is given in Figure 8.27. (As

	Algorithms				
	Maze Routing			Line-Probe	
	Lee	Soukup	Hadlock	Mikami	Hightower
Time complexity	$h \times w$	$h \times w$	$h \times w$	L	L
Space complexity	$h \times w$	$h \times w$	$h \times w$	L	L
Finds path if one exists?	yes	yes	yes	yes	no
Is the path shortest?	yes	no	yes	yes	no

Table 8.1: Comparison of different algorithms.

these two algorithms basically are the same, we just use one description for both of them.) Procedure GENERATE(l, e) generates a line-probe l from an escape point e, whereas, *INSERT(l , list)* adds a line-probe l to the *list*. Function INTERSECT(l_i, l_j) returns TRUE if line-probes l_i and l_j intersect, it returns FALSE otherwise.

Maze routers and many of their variations are grid based methods. Information must be kept for each grid node. Thus, a very large memory space is needed to implement these algorithms for a large grid. The line-probe algorithms, however, require the information to be kept for each line segment. Since the number of line segments is very small compared to the nodes in a grid, the required memory is greatly reduced. The key difference between the two line probe algorithms is that, the Mikami's algorithm can find a path between any two vertices if one exists. This path may not be the shortest path. Hightower's algorithm may not be able to find a path joining two vertices even if such a path exists. A comparison of the maze routing algorithms and line-probe algorithms in their worst cases is given in Table 8.1. ($h \times w$ denotes the size of grid and L denotes the number of line segments generated in line-probe algorithms).

8.5 Shortest Path Based Algorithms

A simple approach to route a two-terminal net uses Dikjstra's shortest algorithm [Dij59]. Given, a routing graph $G = (V, E)$, a source vertex $s \in V$ and a target vertex $t \in V$ a shortest path in G joining s and t can be found in $O(|V|^2)$ time. The algorithm in Figure 8.28 gives formal description of an algorithm based on Dijkstra's shortest path algorithm for global routing a set $\mathcal{N}$ of two-terminal nets in a routing graph G. The output of the algorithm is a set $\mathcal{P}$ of paths for the nets in $\mathcal{N}$. A path $P_i \in \mathcal{P}$ gives a path for net $N_i \in \mathcal{N}$. The time complexity of the algorithm SHORT-PATH-GLOBAL-ROUTER is $O(\mathcal{N}|V|^2)$.

Note that the length of an edge is increased by a factor $\alpha > 1$ whenever a congested edge is utilized in the path of a net. This algorithm is suitable

```
Algorithm SHORT-PATH-GLOBAL-ROUTER($G, \mathcal{N}, \mathcal{P}$)
    input: $G, \mathcal{N}$
    output: $\mathcal{P}$
begin
    for each $N_i$ in $\mathcal{N}$ do
        $P_i$ = DIJKSTRA-SHORT-PATH($G, N_i$);
        for each $e_j$ in $P_i$ do
            $c(e_j) = c(e_j) - 1$;
            if $c(e_j) < 0$ then $l(e_j) = \alpha \times l(e_j)$;
end.
```

Figure 8.28: Algorithm SHORT-PATH-GLOBAL-ROUTER.

for channel intersection graph, since it assumes that congested channels can be expanded. If the edge congestions are strict, the algorithm can be modified to use 'rip-up and reroute' or 'shove aside' techniques [Bol79, DK82].

8.6 Steiner Tree based Algorithms

Global routing algorithms presented so far are not suitable for global routing of multi-terminal nets. Several approaches have been proposed to extend maze routing and line-probe algorithms for routing multi-terminal nets. In one approach, the multi-terminal nets are decomposed into several two-terminal nets and the resulting two-terminal nets are routed by using a maze routing or line-probe algorithm. The quality of routing, in this approach, is dependent on how the nets are decomposed. This approach produces suboptimal results as there is hardly any interaction between the decomposition and the actual routing phase. In another approach, the exploration can be carried out from several terminals at a time. It allows the expansion process to determine which pairs of pins to connect, rather than forcing a predetermined net decomposition. However, the maze routing and line-probe algorithms cannot optimally connect the pins. In addition, these approaches inherit the large time and space complexities of maze routing and line-probe algorithms.

The natural approach for routing multi-terminal nets is Steiner tree approach. Usually *Rectilinear Steiner Trees* (RST) are used. A rectilinear Steiner tree is a Steiner tree with only rectilinear edges. The length of a tree is the sum of lengths of all the edges in the tree. It is also called the cost of the tree. The problem of finding a minimum cost RST is NP-hard [GJ77]. In view of NP-hardness of the problem, several heuristic algorithms have been developed. Most of the heuristic algorithms depend on minimum cost spanning tree. This is due to a special relationship between Steiner trees and minimum cost spanning trees. Hwang [Hwa76a, Hwa79] has shown that the ratio of the cost of a minimum spanning tree (MST) to that of an optimal RST is no greater than

$\frac{3}{2}$. Let S be a net to be routed. We define an underlying grid $G(S)$ of S (on an oriented plane) as the grid obtained by drawing horizontal and vertical lines through each point of S (see Figure 8.29). Let $G_c(S)$ be a complete graph for S. An MST for net S is a minimum spanning tree of $G_c(S)$ (see Figure 8.29.) Note that, there may be several MST's for a given net and they can be found easily. Using Hwang's result, an approximation of the optimal RST can be obtained by rectilinearizing each edge of an MST. Different ways of rectilinearizing the edges of T give different approximations. If an edge (i, j) of T is rectilinearized as a shortest path between i and j on the underlying grid $G(S)$, then it is called as a staircase edge layout. For example, all the edge layouts in Figure 8.29 are staircase layouts. A staircase layout with exactly one turn on the grid G(S) is called as an L-shaped layout. A staircase layout having exactly two turns on the grid $G(S)$ is called as a Z-shaped layout. For example, the edge layout of P_3 and P_1 in Figure 8.29 are L-shaped and Z-shaped layouts respectively. An RST obtained from an MST T of a net S, by rectilinearizing each edge of T using staircase layouts on G(S) is called S-RST. An S-RST of T, in which the layout of each MST edge is a L-shaped layout is called an L-RST of T. An S-RST of T, in which the layout of each MST edge is a Z-shaped layout is called a Z-RST of T. An optimal S-RST (Z-RST, L-RST) is an S-RST (Z-RST, L-RST) of the least cost among all S-RST's (Z-RST's, L-RST's). It is easy to see that an optimal L-RST may have a cost larger than an optimal S-RST (see Figure 8.30), which in turn may have a cost larger than the optimal RST. Obviously, least restriction on the edge layout gives best approximation. However, as the number of steps allowed per edge is increased it becomes more difficult to design an efficient algorithm to find the optimal solution.

The organization of the rest of this section is as follows: First, we discuss a separability based algorithm to find an optimal S-RST from a separable MST. This is followed by a discussion on non-rectilinear Steiner trees. We also discuss MIN-MAX Steiner tree that are used for minimizing the traffic in the densest channels. These three approaches do not consider the presence of obstacles while finding approximate rectilinear Steiner tree for a net. At the end of this section, we discuss a weighted Steiner tree approach that works in presence of obstacles and simultaneously minimizes wire lengths and density of the routing regions.

8.6.1 Separability Based Algorithm

In [HVW85], Ho, Vijayan, and Wong presented an approach to obtain an optimal S-RST from an MST, if the MST satisfies a special property called *separability.* A pair of nonadjacent edges is called *separable* if staircase layouts of the two edges does not intersect or overlap. An MST is called as a *separable MST* (SMST) if all pairs of non-adjacent edges satisfy this property. In other words, such an MST is called to have *separability* property. If an edge is deleted from an SMST, the staircase layouts of the two resulting subtrees do not intersect or overlap each other. Overlaps can occur only between edges that are incident on a common vertex. This property enables the use of dynamic

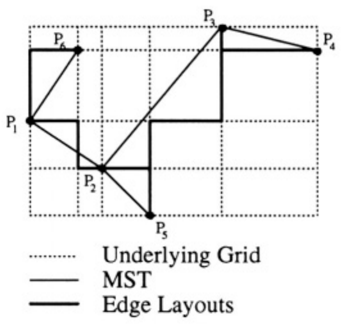

Figure 8.29: Grid, MST, and edge layouts.

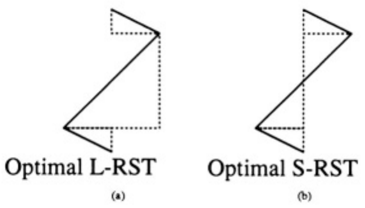

Figure 8.30: Different edge layouts for an MST.

programming techniques to obtain an optimal S-RST. (i, j)

The algorithm works in two steps. In the first step, an SMST T is constructed for a given net N_p by using a modified Prim's algorithm [Pri57] in $O(|N_p|^2)$ time. In the second step, an optimal Z-RST is obtained by using the SMST obtained in the first step in $O(|N_p| \times t_{\max}^6)$ time, where $t_{\max}$ is the maximum of $t(e)$ over all edges e and $t(e)$ denote the number of edges on the underlying grid $G(N_p)$ traversed by any staircase layout of an edge e of the MST T of net N_p. According to the Z-sufficiency Theorem in [HVW85], an optimal Z-RST is an optimal S-RST. The optimal S-RST is used as an approximation of the minimum cost RST. In the rest of this section, we discuss these two steps in detail.

1. **Algorithm SMST:** Let $G_c(N_p)$ denote the complete graph for net N_p. For any vertex i and j, $x(i)$ and $y(i)$ denote the x, y-coordinates of vertex i on a Cartesian plane, and $dist(i, j)$ denotes the total length of shortest path between i and j. Function PRIM(G_c, *W,T*), takes the complete graph(G_c) and an array(W) containing weights of edges in G_c as input. It generates a separable MST T using a modified Prim's algorithm for MST, which has a time complexity of $O(|N_p|^2)$. The formal description of algorithm SMST appears in Figure 8.31. The time complexity of algorithm SMST is $O(|N_p|^2)$.

```
Algorithm SMST($N_p, T$)
    input:$N_p$
    output:$T$
begin
    Construct the complete graph $G_c(N_p)$ of the net $N_p$.
    for each edge $(i, j)$ of $G_c(N_p)$ do
        $3tuple(i, j) = (dist(i, j), -|y(i) - y(j)|,$
            $-\max(x(i), x(j)))$;
    PRIM($G_c(N_p), 3tuple(i, j), T$);
end.
```

Figure 8.31: Algorithm SMST.

2. **Algorithm Z-RST:** The input to the algorithm is an SMST T of a net N_p. By hanging the input separable MST T by any leaf edge r, a rooted tree T_r can be obtained. For each edge e in T, let T_e denote the subtree of T_r that hangs by the edge e. Given a Z-shaped layout z of an edge e, we let $M_z[e]$ denote the Z-RST of the subtree T_e, which has the minimum cost among all Z-RST's of T_e, in which the layout of the edge e is constrained to be the Z-shape z. $M_z[e]$ can be computed recursively as follows: Let $e_i, i = 1, 2, \ldots, d$ be the d child edges of e in the rooted tree T_r. For each child edge e_i and for each possible Z-shaped layout z_{ij} of the edge e_i, recursively compute the constrained optimal Z-RST's $M_{z_{ij}}[e_i]$ of the subtrees T_{e_i}. Let the number of such constrained Z-RST's for a subtree T_{e_i} be denoted as $t(e_i)$. Taking one such Z-RST for each subtree T_{e_i}, and merging these subtree Z-RST's with the layout z of the edge e, results in a Z-RST of T_e. Since the tree T has the separability property, the only new overlaps that can occur during this merging are among the edges $e, e_1, \ldots, e_d$, which are all incident on a common point. Therefore, the total amount of overlap in the resulting Z-RST of T_e is the sum of the overlaps among the layouts of the edges $e, e_1, \ldots, e_d$, added to the sum of overlaps in the selected Z-RST's of the subtrees T_{e_i}. Enumerate all combinations of selecting one of the Z-RST's $M_{z_{ij}}[e_i]$ for each subtree T_{e_i}, and for each such combination compute the resulting Z-RST of T_e. The constrained optimal Z-RST $M_z[e]$ of the subtree T_e is simply the one with the least cost. To compute the optimal Z-RST of the entire rooted tree T_r, recursively compute (as explained above) the constrained optimal Z-RST's $M_z[r]$ for each Z-shaped layout z of the root edge r, and select that Z-RST of the smallest cost, (see Figure 8.32)

   A recursive definition of Function LEAST-COST is given Figure 8.33. Function LEAST-COST($z, T_e, cost, M_z[e]$) takes a Z-shaped layout z of an edge e, and a subtree T_e as input. The output of function LEAST-COST is the optimal Z-RST(denoted as $M_z[e]$) of T_e for the z layout of edge e and the cost(denoted as $CostM_z[e]$) of $M_z[e]$. Function CHILD-

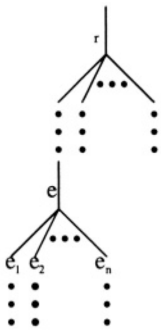

Figure 8.32: Structure used in the algorithm Z-RST.

EDGES-NUM(e) returns the number of child edges of an edge *e*.

Let r be the leaf edge that is used to hang the SMST *T*, and T_r be the tree obtained, then the output of the algorithm in Figure 8.34 is the optimal Z-RST (*M*) of *T* and its cost *CostM.*

The fact that the algorithm Z-RST constructs the optimal S-RST follows from the separability of the input MST and from the Z-sufficiency theorem stated below.

Theorem 6 *(Z-Sufficiency Theorem): Given an SMST T of a point set S of cardinality n, there exists a Z-RST of T whose cost is equal to the cost of an optimal S-RST of T.*

The worst case time complexity of algorithm Z-RST is $O(|N_p| \times t_{\max}^6)$, where $t_{\max}$ is the maximum of $t(e)$ over all edges *e*.

8.6.2 Non-Rectilinear Steiner Tree Based Algorithm

Burman, Chen, and Sherwani [BCS91] studied the problem of global routing of multi-terminal nets in a generalized geometry called δ-geometry in order to improve the layout and consequently enhance the performance. The restriction of layout to rectilinear geometry, and thus only rectilinear Steiner trees, in the previous Steiner tree based global routing algorithms was necessary to account for restricted computing capabilities. Recently, because of enhanced computing capabilities and the need for design of high performance circuit, non-rectilinear geometry has gained ground. In order to obtain smaller length Steiner trees, the concept of separable MST's in δ-geometry was introduced. In δ-geometry, edges with angles $i\pi/\delta$, for all i, are allowed, where δ (≥ 2) is a positive integer. $\delta = 2$, 4 and ∞ correspond to rectilinear, 45° and Euclidean geometries respectively. Obviously, we can see that δ-geometry always includes rectilinear edges and is a useful with respect to the fabrication technologies. It has been proved [BCS91]

```
**Function** LEAST-COST($z, T_e, CostM_z[e], M_z[e]$)
    **input:** $z, T_e$
    **output:** $CostM_z[e], M_z[e]$
**begin**
    **if** CHILD-EDGES-NUM($e$) $\neq 0$ **then**
        **for** each child edge $e_i$ of $e$ **do**
            **for** each layout $z_{ij}$ of $e_i$ **do**
                LEAST-COST($z_{ij}, T_{e_i}, CostM_{z_{ij}[e_i]}, M_{z_{ij}}[e_i]$);
        **for** (each combination of the layouts containing one
            optimal Z-RST layout for each $T_{e_i}$) **do**
            merge layouts in the combination with the layout $z$
            of edge $e$;
            calculate the resulting cost of merged layout;
        $CostM_z[e]$ = minimum cost among all merged layouts;
        $M_z[e]$ = the layout corresponding to the minimum cost;
    **else** (* The bottom edge is reached *)
        $M_z[e] = z$;
        $CostM_z[e]$ = cost of $z$;
**end.**
```

Figure 8.33: Function LEAST-COST.

```
**Algorithm** Z-RST ($r, T_r, CostM, M$)
    **input:** $r, T_r$
    **output:** $CostM, M$
**begin**
    $CostM = \infty$;
    **for** each z-shaped layout $z$ of $r$ **do**
        LEAST-COST ($z, T_r, CostTempM, TempM$);
        **if** $CostTempM < CostM$ **then**
            $M = TempM$;
            $CostM = CostTempM$;
**end.**
```

Figure 8.34: Algorithm Z-RST.

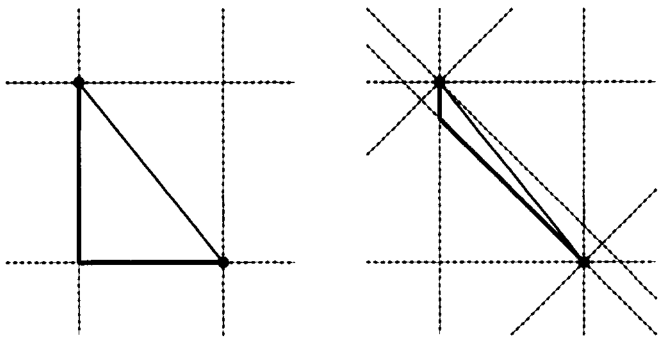

Figure 8.35: Comparison of the rectilinearization in 2-geometry and in 4-geometry.

that for an even $\delta \geq 4$, all minimum cost spanning trees in δ-geometry satisfy the separability property.

Theorem 7 *Any minimum spanning tree for a given point set in the plane is δ-separable for any even $\delta \geq 4$.*

Therefore, there exists a polynomial time algorithm to find an optimal Steiner tree in δ-geometry, which is derivable from the separable minimum spanning tree. The experiments have shown that tree length can be reduced up to 10-12% by using 4-geometry as compared to rectilinear geometry (2-geometry). Moreover, length reduction is quite marginal for higher geometries. As a consequence, it is sufficient and effective to consider layouts in 4-geometry in the consideration of global routing problem. An example of the derivation of a Steiner tree for a simple two-terminal net in 4-geometry is shown in Figure 8.35(b) while Figure 8.35(a) shows the derivation of a Steiner tree for the same net in rectilinear geometry. Clearly, the tree length in 4-geometry is shorter than the one in the rectilinear geometry.

8.6.3 Steiner Min-Max Tree based Algorithm

The approach in [CSW89] uses a restricted case of Steiner tree in global routing problem, called *Steiner Min-Max Tree* (SMMT) in which the maximum weight edge is minimized (real vertices represent channels containing terminals of a net, Steiner vertices represent intermediate channels, weights correspond to densities). They give an $O(\min\{|E| \log\log|E|, |V|^2\})$ time algorithm for obtaining a Steiner min-max tree in a weighted coarse grid graph $G = (V, E)$. The weight of an edge in E is a function of current density, capacity, and measures crowdedness of a border. Each vertex in V is labeled with demand or

```
Algorithm SMMT($G, d, T$)
    input:$G, d$
    output:$T$
begin
    $T$ = An MST of $G$;
    while EXIST-ODSV($T, d$)=TRUE do
        $v$ = GET-ODSV($T, d$); (* a Steiner leaf*)
        REMOVE($v, T$);
    output $T$;
end.
```

Figure 8.36: Algorithm SMMT.

(potential) Steiner depending on whether it is respectively a terminal of net N_i or not. A Steiner min-max tree of G dictates a global routing that minimizes traffic in the densest channel. While the Steiner min-max tree method tends to route nets through less crowded channels, it is also desirable to have nets with short length. Therefore, among all Steiner min-max trees of the given net, we are interested in those with minimum length. The problem of finding a Steiner min-max tree whose total length is minimized is NP-hard.

Given, a weighted coarse grid graph $G = (V, E)$ and a boolean array d such that $d[v]$ is true if the vertex $v \in V$ corresponds to a terminals of N_i. An SMMT T of N_i can be obtained using algorithm in Figure 8.36. Function EXIST-ODSV(T, d) returns TRUE if there exists a one-degree Steiner vertex in T. Function GET-ODSV(T, d) returns a one-degree Steiner vertex from T. REMOVE(v, T) removes vertex v and edges incident on it from T.

Theorem 8 *Algorithm SMMT correctly computes a Steiner min-max tree of net N_i in weighted grid graph $G = (V, E)$ in $O(\min\{|E| \log\log|E|, |V|^2\})$ time.*

A number of heuristics have been incorporated in the global router based on the min-max Steiner trees. The nets are ordered first according to their priority, length and multiplicity numbers. The global routing is then performed in two phases: the SMMT-phase and the SP-phase. The SP-phase is essentially a minimum-spanning tree algorithm. The SMMT-phase consists of J_1 steps and the SP-phase consists of J_2 steps, where J_1 and J_2 are heuristic parameters based on the importance of density and length minimization in a problem, respectively.

In the SMMT-phase, the nets are routed one by one, using the algorithm SMMT. At the j-th step of the SMMT-phase, if the length of routing of N_i is within a constant factor, c_j of its minimum length then it is accepted, otherwise, the routing is rejected. Once a net is routed during SMMT-phase, it will not be routed again.

```
Algorithm LAYOUT-WRST ($\mathcal{R}, N_i$)
begin
    $T$ = an MST of $N_i$;
    $L_0 = \phi$;
    for $j = 1$ to $n - 1$ do
        $min\_cost = \infty$;
        for $i = 1$ to $k$ do
            FIND-P($i, e_j, \mathcal{R}$);
            $Q_{i,j}$ = MERGE($L_{j-1}, P_i(e_j)$);
            CLEANUP($Q_{i,j}$);
            if WT($Q_{i,j}$) $<$ $min\_cost$ then $L_j = Q_{i,j}$;
end.
```

Figure 8.37: Algorithm LAYOUT-WRST.

In the SP-phase, the nets are routed one by one by employing a shortest-path heuristic and utilizing the results from the SMMT-phase. At the j-th step we accept a routing only if it is better than the best routing obtained so far.

8.6.4 Weighted Steiner Tree based Algorithm

Several global routing algorithms have been developed that consider minimizing the length of Steiner tree as the primary objective and minimizing the traffic through the routing areas as the secondary objective and vice versa. In [CSW92], Chiang, Sarrafzadeh, and Wong proposed a global router that simultaneously minimizes length and density by using a weighted Steiner tree. Consider a set $\mathcal{R} = \{R_1, R_2, \ldots, R_m\}$ of weighted regions in an arbitrary-style layout, where weight of a region is proportional to its density and area. The regions with blockages are assigned infinite weights. A weighted Steiner tree is a Steiner tree with weighted lengths, i.e., an edge with length / in a region with weight w has weighted length lw. A *weighted rectilinear Steiner tree* (WRST) is a weighted Steiner tree with rectilinear edges. A minimum-weight WRST is a WRST with minimum total weight.

The 2-approximate algorithm to find an approximation of minimum-weight WRST is as discussed below: First step of this algorithm is to find an MST T for a given net N_i using Prim's algorithm. Let $e_1, e_2, \ldots, e_{n-1}$ be the edges of T. In the second step, the edges of T are rectilinearized one by one. In general, there are more than one possible staircase layouts for an edge e_j of T. Let $\{P_1(e_j), P_2(e_j), \ldots, P_k(e_j)\}$ be a subset of all possible staircase layouts for edge e_j. Let L_{j-1} denotes the staircase layout of edges $e_1, e_2, \ldots, e_{j-1}$. Let $Q_{i,j}$ be the layout obtained by merging L_{j-1} and $P_i(e_j)$. L_j is selected to be the minimum cost layout among all $Q_{i,j}$.

The formal description of the algorithm is given in Figure 8.37. Function FIND-P($i, e_j, \mathcal{R}$) finds $P_i(e_j)$ and function CLEANUP ($Q_{i,j}$) removes overlapped layouts. Function WT($(Q_{i,j})$ gives the total weighted length of $Q_{i,j}$. The time complexity of algorithm LAYOUT-WRST is $O(|N_i|^2)$.

8.7 Integer Programming Based Approach

The problem of concurrently routing all the nets is computationally hard. The only known technique uses integer programming. In fact, the general global routing problem formulation can be easily modified to a 0/1 integer programming formulation. Given a set of Steiner trees for each net and a routing graph, the objective of such an integer programming formulation is to select a Steiner tree for each net from its set of Steiner trees without violating the channel capacities while minimizing the total wire length. This approach is well suited when there is a preferred set of Steiner trees for each net. However, as the size of input increases the time required to solve corresponding integer program increases exponentially. Thus it is necessary to break down the problem into several small subproblems, solve them independently and combine their solutions in order to solve the original problem.

8.7.1 Hierarchical Approach

In this section, we discuss the hierarchical based integer program for global routing, presented by Heisterman and Lengaur [HL91]. Let $S = \{S_i | 1 \leq i \leq n\}$ denote a set of sets of vertices in the routing graph $G = (V,E)$. Let $T = \{T_{ij}\}, j = 1, 2, \ldots, l_i$, denote a set of Steiner trees for $S_i, i = 1, 2, \ldots, n$. Then, the global routing problem can be formulated as an integer program by taking an integer variable x_{ij} to denote the number of nets which are routed using T_{ij}. S_i is called a net type and T_{ij} a route for S_i. Let n_i denote the number of nets corresponding to the net type S_i for $i = 1, 2, \ldots, n$. The following constraints have to be met:

$$\sum_{j=1}^{l_i} x_{ij} = n_i, i = 1, \ldots, n \text{ (completeness constraints)}$$
$$\sum_{(ij), e \in T_{ij}} x_{ij} + x_e = c(e), e \in E \text{(capacity constraints)}$$

The variable x_e is a slack variable for edge e which denotes the free capacity of e. Technology constraints may have to be added to this system. The cost function to be minimized is

$$\sum_{i=1}^{n} \sum_{j=1}^{l_i} l(T_{ij}) \times x_{ij}$$

where $l(T_{ij})$ is the length of the Steiner tree T_{ij}.

The resulting integer program is denoted by R. It cannot be solved efficiently because of its size and NP-hardness of integer programming. Hierarchical global routing methods break down the integer program into pieces small enough to be solved exactly. The solutions of these pieces are then combined by a variety

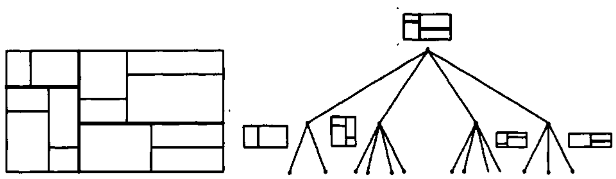
Figure 8.38: A floorplan and its preprocessed cut-tree.

of methods. This results in an approximate solution of the global routing problem.

Hierarchical methods that work top down are especially effective because they can take into account global knowledge about the circuit. Top-down methods start with a cut-tree for the circuit emerging from a floorplanning phase that uses circuit partitioning methods. The cut-tree is preprocessed so that each interior node of the tree corresponds to a simple routing graph as shown in Figure 8.38.

The cut-tree is then traversed top down. At each node a global routing problem is solved on the corresponding routing graph. The solutions for all nodes in a level of the cut-tree are combined. The resulting routing influences the definition of the routing problems for the nodes in the next lower level.

The small integer programs corresponding to the routing problems at each interior node of the cut-tree can be solved by general integer programming techniques. However, this solution may be computationally infeasible. In the best case the linear relaxation of the integer program, i.e., the linear program obtained by eliminating the integrality constraint has to be solved. Since a large number of integer programs have to be solved during the course of global routing, speeding up the computations is necessary. One possibility is to round off the solution of the linear relaxation deterministically or by random methods. This may not lead to an optimal solution. So, it is desirable to exactly solve the integer program corresponding to the routing problem at interior nodes of the cut-tree. Because integer programs corresponding to small routing graphs are quite structured, appropriate preprocessing can substantially reduce the size of the integer programs, and sometimes eliminate them altogether. For example, there exists a greedy algorithm [HL91] to solve the corresponding integer programming problem for a small routing graph H_4 in Figure 8.39. This algorithm will be described in the remainder of the section.

We assume that the length of each edge is the distance between the centers of vertices. In Figure 8.39, a specific floorplan pattern is depicted that is dual to H_4. Figure 8.40 depicts all possible net types and routes for H_4. The size of the integer program R_4 that corresponds to H_4 is reduced by combinatorial arguments on the patterns.

A simple greedy preprocessing strategy can be used for reducing the size of the integer program R_4. This strategy is the first phase of the greedy routing

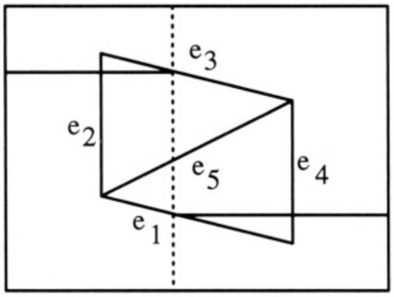

Figure 8.39: The routing graph H_4 and its floorplan.

algorithm. It constructs a smaller integer program R'_4 and is followed by two more phases. The second phase further reduces the size of R'_4 and constructs a small mixed integer program R''_4. The third phase solves the integer program R''_4.

During the first phase, for $i = 1, \ldots, 5$, the algorithm routes $\min\{c(e_i), n_i\}$ nets by using T_{i1}. It delete the routed nets from the problem instance and reduces the capacity of each edge e_i by the number of routed nets crossing e_i. Now if there are still nets left in type S_i, then e_i is saturated. This fact eliminates all routing patterns in Figure 8.40 that include edge e_i. After the deletion, the pattern set for $i = 1, 4$ is identical with the pattern set for $i = 9$, and the pattern set for $i = 2, 3$ is identical with the pattern set for $i = 10$. Thus the net types S_1 and S_4 are merged with net type S_9 and the net types S_2 and S_3 are merged with net type S_{10}. Net type S_5 cannot be eliminated. As a result, the original problem R_4 has been reduced into a problem R'_4 with fewer variables and constraints.

The second phase further reduces R'_4. The result is a very small mixed integer program R''_4 that can be solved with traditional integer programming techniques. Two cases have to be distinguished.

1. **There are no more nets of Type S_5 to be routed:** In this case, the integer program only contains nets of types $S_6 - S_{11}$. An inspection of Figure 8.40 shows that for $i = 6, \ldots, 10$, the long routing patterns for net type S_i also occur as routing patterns for net type S_{11}. This suggests elimination of the variables corresponding to the long routing patterns for net types S_6 to S_{10} from R'_4. The variables $x_{ij}, i = 6, \ldots, 10$ then count the nets of these types that are routed with the short routes. All other nets of these types should be counted by the variables for net type S_{11}. This can be achieved by introducing slack variables x_i to denote the number of nets of type S_i that can not be routed by the short routing patterns of type S_i for $i = 6, 7, \ldots, 10$. Thus, the completeness constraints can now be given as:

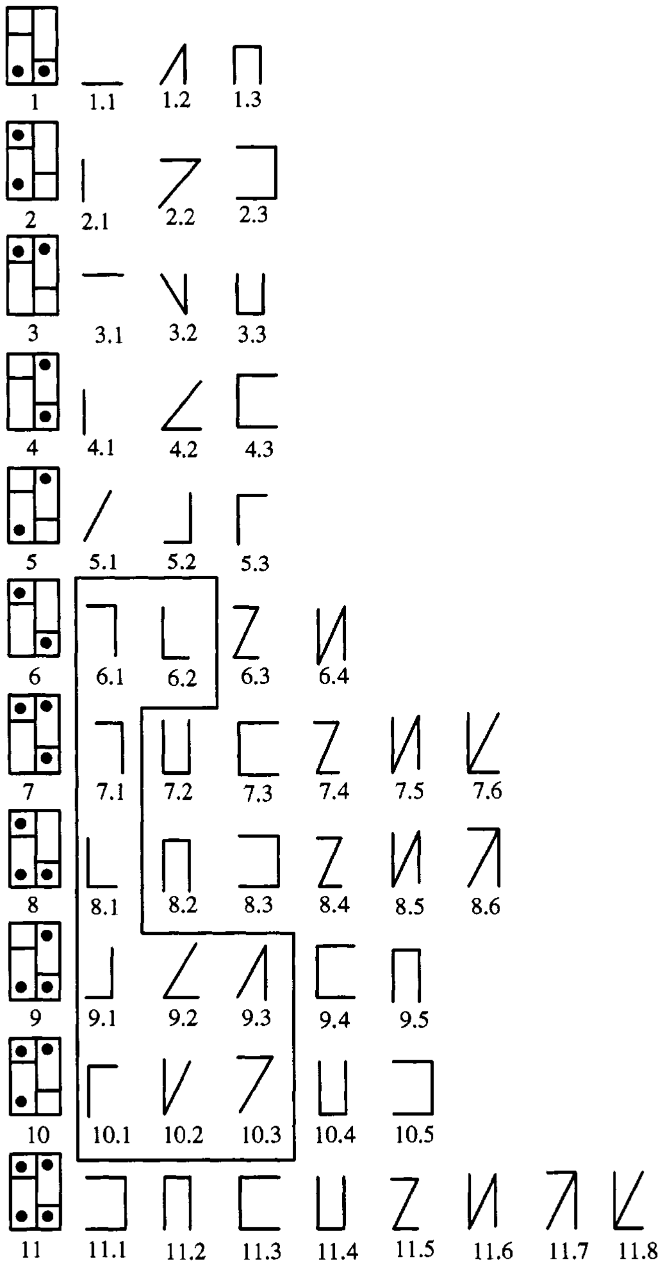

Figure 8.40: All net types and routing patterns for routing graph H_4.

$$x_i + \sum_{j=1}^{l_i} x_{ij} = n_i, i = 6, \ldots, 10$$
$$\sum_{j=1}^{l_{11}} x_{11,j} - \sum_{i=6}^{10} x_i = n_{11}$$

All the other equations remain the same. This yields the integer program R_4''.

The integrality constraints in the resulting integer program can be eliminated for some variables. Specifically, since all coefficients are integers, the integrality constraint can be omitted for one variable per constraint. The number of integer variables is thus reduced and the solution of the integer program by such techniques as branch-and-bound becomes more efficient.

2. **There are more nets of Type S_5 to be routed:** In this case, edgee_5 is saturated after the first phase. This saturation eliminates all routing patterns containing this edge. The resulting integer program is then subjected to an analogous reduction procedure as in the first case. Nets of types $S_7 - S_{10}$ that cannot be routed with short routes are subsumed in type S_{11}.

The third phase of the algorithm solves the small mixed integer program R_4'' obtained in Phase 2 and interprets the solution.

In addition to the formulation of the global routing problem as finding a set of Steiner trees described above, the global routing problem can also be formulated as finding the optimal spanning forest (a generalization of optimal spanning trees) on a graph that contains all of the interconnection information. Cong and Preas presented a concurrent approach based on this formulation [CP88].

8.8 Performance Driven Routing

With the advent of deep submicron technology, interconnect delay has become an important concern in high performance circuit design. Interconnect delay is now a significant part of the total net delay. The reduction in feature sizes has resulted in increased wire resistance and net delay. The increased proximity between the devices and interconnection wires resulting in increased cross-talk noise.

The routers should now model the cross-talk noise between adjacent nets during topology generation. Buffer Insertion, wire sizing, and high performance topology constructions are some of the techniques adopted to reduce generate routing for high performance circuits. Zhou and Wong [Won98] considered crosstalk avoidance during global routing.

Lillis, Cheng, Lin and Ho [CH96] presented techniques for performance driven routing techniques with explicit area-delay trade-off and simultaneous wire sizing. In [Buc98] Lillis and Buch present table-lookup methods for improved performance driven routing.

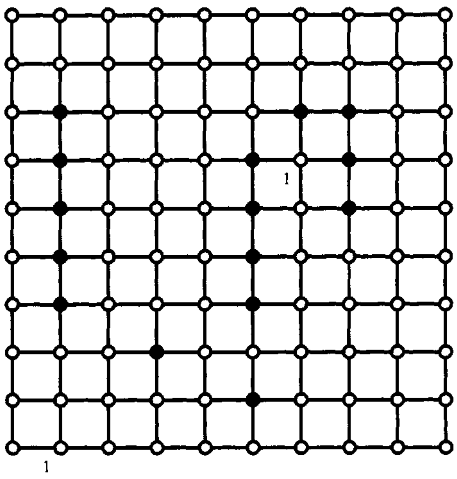

Figure 8.41: A grid, blocked vertices and a two-terminal net.

8.9 Summary

Global routing assigns a sequence of routing channels to each net without violating the capacity of channels. In addition, it typically optimizes the total wire length. In high performance circuits, the optimization function is to minimize the critical RC delay of the nets. Different design style have different objective functions. In standard cell design style, the optimization function is to minimize the total channel height. Whereas, in gate array design style the objective is to guarantee routability.

The global routing algorithms fall roughly into two categories: One is the sequential approach and the other is the concurrent approach. In sequential approach, the nets are routed one by one. However, the nets which have been already routed may block the nets to be routed later. Thus, the order in which the nets are routed is very important. Maze routing algorithms, line-probe algorithms and Steiner tree based algorithms are important classes of algorithms in this approach. The first two class of algorithms are used for two-terminal nets, whereas, the Steiner tree algorithms are used for the multi-terminal nets. The general rectilinear Steiner tree problem is NP-hard, however, approximate algorithms have been developed for this problem. The concurrent approach takes a global view of all nets to be routed at the same time. This approach requires use of computationally expensive methods. One such method uses integer programming. Integer program for an overall problem is normally too large to be handled efficiently. Thus, hierarchical top down methods are used to break the problem into smaller sub-problems. These smaller sub-problems can be solved efficiently. The solutions are then combined to obtain the solution of original global routing problem.

8.10 Exercises

1. Design and implement an algorithm to find the extended channel intersection graph if the size and location of all cells are known.

2. Assume that several nets have been assigned feed-throughs in a standard cell layout with K cell rows. A two-terminal net N that starts at a terminal on cell row i and ends at a terminal on cell row j has to be added to this layout, where $1 \leq i \leq j \leq K$. Design an optimal algorithm to assign feed-throughs to N such that increase in the overall channel height of the layout is minimized.

3. Figure 8.41 shows a grid graph with several blocked vertices. It also shows terminals of a two-terminal net N_1 marked by '1'. Use the Lee's algorithm to find:

   (a) the path for N_1.

   (b) the number of nodes explored in (a).

   Use the Soukup's algorithm to find (a) and (b). Use the Hadlock's algorithm to find (a) and (b).

†4. Extend Lee's maze router so that it generates a shortest path from source to target with the least number of bends.

†5. Design an efficient heuristic algorithm based on maze routing to simultaneously route two 2-terminal nets on a grid graph. Compare the routing produced by this algorithm with that produced by Lee's maze router by routing one net at a time.

6. Give an example for which the Hightower line-probe algorithm does not find a path even when a path exists between the source and the target.

†7. In Mikami's line-probe router, every grid node on the line segment is an escape point on each line segment. Whereas, Hightower's algorithm makes use of only single escape point on each line segment. As a result, Hightower's algorithm runs faster than Mikami's algorithm. Also, Hightower's algorithm may not be able to find a path even when one exists. On the other hand, Mikami's algorithm always finds a path if one exists. The number and location of escape points plays very important role in the performance of the router.

   Implement a line-probe router which can use k number of escape points, where k is a user specified parameter. Use an efficient heuristic for the location of the escape points.

8. In Figure 8.42, terminals of two nets N_1 and N_2 are shown on a grid graph. Terminals of net N_1 are marked by '1' and that of N_2 are marked by '2'. Find an MRST for N_1.

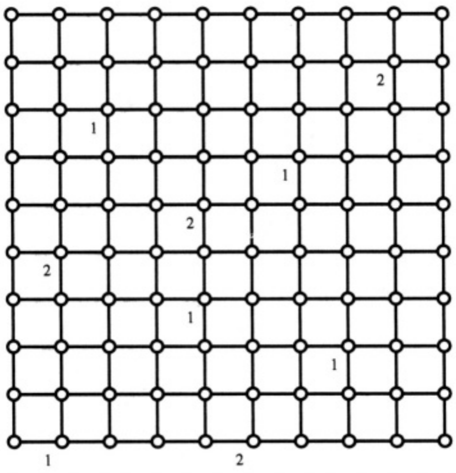

Figure 8.42: A grid and net terminals.

9. For the example in Figure 8.42, find an RST for each net N_1 and N_2 such that they do not intersect with each other and
   (a) the summation of the cost of these two RST's is minimum,
   (b) the maximum of the costs of these two RST's is minimum.

10. Design an algorithm to determine an MRST of a multi-terminal net in a $2 \times n$ grid graph.

11. Compute the number of intersection points in an underlying grid of a set of points in δ-geometry. Is it sufficient to consider just the edges of underlying grid graph to construct a Steiner tree in δ-geometry?

†12. Why does the algorithm Z-RST gives an optimal S-RST for a separable MST? In other words, prove Theorem 6.

†13. Implement the algorithm to find an optimal S-RST for any given net.

†14. Prove Theorem 7 and modify the algorithm Z-RST to use δ-geometry.

†15. Prove Theorem 8.

†16. The problem of finding a Steiner tree for a K-terminal net in a grid graph is known to be NP-complete. Design an efficient heuristic algorithm based on maze routing for this problem.

Bibliographic Notes

Besides the classes of global routing algorithms described above, there are other global routing algorithms that use different approaches and have different optimization functions. Shragowitz and Keel proposed a global router based on a multicommodity flow model [SK87]. Vecchi and Kirkpatrick discussed the global wiring by simulated annealing [VK83]. A practical global router for row-based layout such as sea-of-gate, gate array and standard cell was developed by Lee and Sechen in 1988 [LS88]. Karp and Leighton discuss the problem of global routing in two-dimensional array [KLR+87]. An interior point method (Karmarkar's Algorithm) can be applied to solve the linear programming model of global routing problem [HS85, AKRV89, Van91]. A path selection global router is developed by Hsu, Pan, and Kubitz [HPK87]. A novel feature of the algorithm is that the *active vertices* (vertices in the net which are not yet connected) are modeled as magnets during the path search process. Several global routing algorithms, including the one based on wave propagation and diffraction, a heuristic minimum tree algorithm using "common edge" analysis, an overflow control method, and global rerouting treatment are discussed in [Xio86]. A simple but effective global routing technique was proposed by Nair, which iterates to improve the quality of wiring by rerouting around congested areas [Nai87], A global routing algorithm in a cell synthesis system was proposed by Hill and Shugard [HS90], which includes detailed geometric information specific to the cell synthesis problem. The system models diffusion strips, congestion and existing feedthroughs as a cost function associated with regions on the routing plane.

The placement and routing can be combined together so that every placement can be judged on the basis of the routing cost. Researchers have produced some useful results in this direction. Burstein and Hong presented an algorithm to interleave routing with placement in a gate array layout system [BH83]. Dai and Kuh presented an algorithm for simultaneous floorplanning and global routing based on hierarchical decomposition [DK87a]. Suaris and Kedem presented an algorithm for integrated placement and routing based on quadri-section hierarchical refinement [SK89]. An algorithm which combines the pin assignment step and the global routing step in the physical design of VLSI circuits is presented by Cong [Con89]. The sequential algorithms for routing require large execution time. Jonathan Rose [Ros90] developed a parallel global routing algorithm which route multiple nets in parallel by relaxing data dependencies. The speedup is achieved at expense of losing some quality of the routing. The global routing problem is formulated at each level of hierarchy as a series of the minimum cost Steiner tree problem in a special class of partial 3-trees, which can be solved optimally in linear time. In [CH94] Chao and Hsu present a new algorithm for constructing a rectilinear Steiner tree for a given set of points. In [HXK+93] Hong, Xue, Kuh, Cheng, and Huang present two performance-driven Steiner tree algorithms for global routing which consider the minimization of timing delay during the tree construction as the goal. In [HHCK93] Huang, Hong, Cheng, Kuh propose an efficient timing-driven global routing algorithm where interconnection delays are modeled and included during routing and rerouting process in order to minimize the routing area as well as to satisfy timing constraint.