Chapter 9

Detailed Routing

In a two-phase routing approach, detailed routing follows the global routing phase. During the global routing phase, wire paths are constructed through a subset of the routing regions, connecting the terminals of each net. Global routers do not define the wires, instead, they use the original net information and define a set of restricted routing problems. The detailed router places the actual wire segments within the region indicated by the global router, thus completing the required connections between the terminals.

The detailed routing problem is usually solved incrementally, in other words, the detailed routing problem is solved by routing one region at a time in a predefined order. The ordering of the regions is determined by several factors including the criticality of routing certain nets and the total number of nets passing through a region. A routing region may be channel, 2D-switchbox or a 3D-switchbox. Channels can expand in Y direction and their area can be determined exactly only after the routing is completed. If this area is different than the area estimated by the placement algorithm, the placement has to be adjusted to account for this difference in area. If the floorplan is slicing then a left to right sweep of the channels can be done such that no routed channel has to be ripped up to account for the change of areas. Consider the example shown in Figure 9.1 (a). In this floorplan, if channel 1 is routed first followed by routing of channel 2 and channel 3, no rerouting would be necessary. In fact, complete routing without rip-up of an already routed channel is possible if the channels are routed in the reverse partitioning order. If the floorplan is non-slicing, it may not be possible to order the channels such that no channel has to be ripped up. Consider the example shown in Figure 9.1(b). In order to route channel 2, channel 1 has to be routed so as to define all the terminals for channel 2. Channel 2 has to be routed before channel 3 and channel 3 before channel 4. Channel 4 requires routing of channel 1 giving rise to a cyclic constraint for ordering the channels. This situation is resolved by the use of L-channels or 2D-switchboxes. L-channels are not simple to route and are usually decomposed. Figure 9.1(c) shows decomposition of an L-channel into two 3-sided channels while Figure 9.1(d) shows decomposition of an L-channel into

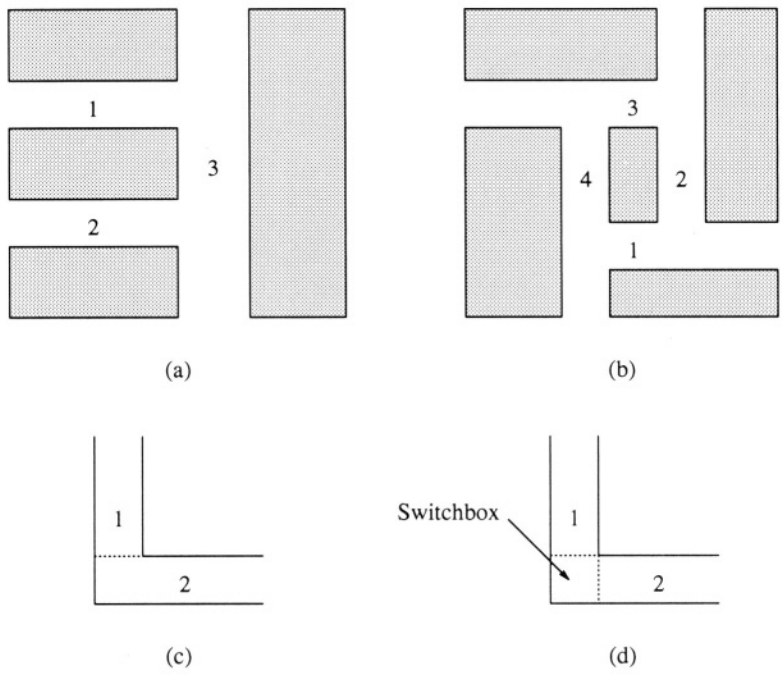


Figure 9.1: Channels and Switchboxes.

two 3-sided channels and a 2D-switchbox. The area of switchboxes (both 2D and 3D) is fixed and the main issue is routability. That is, given a specific area and pin locations for all the nets, is this switchbox routable ?. If a switchbox is unroutable, then the design must be re-global routed. In terms of routing complexity, channels are easy to route, 2D-switchboxes are harder and 3D-switchboxes are hardest to route.

Characteristics of a routing problem largely depend upon the topology of the routing region. Routing regions consist of one or more layers. In the general case, even single-layer routing problems are NP-complete [Ric84]. In multi-layer routing problems, the wires can switch adjacent layers at certain locations using *vias.* A via is an electrical connection (contact) between wire *segments* on adjacent layers. In many multi-layer models, the layers are restricted to contain either horizontal or vertical *segments* (a straight piece of wire placed on a single layer) of a wire. This type of model is known as a *restricted layer model* or *reserved layer model.* Multilayer routing problems are also NP-complete [Szy85], even when the routing region has a simple shape. For this reason many of the algorithms for multi-layer routing problems are heuristic in nature. Different detailed routing strategies have been developed with a variety of objectives, but all the detailed routing problems share some common characteristics. These characteristics deal with routing constraints. For

example, wires must satisfy some geometric restrictions which often concern wire thickness, separation, and path features. One obvious restriction present in all routing problems is intersection; that is, no two wires from different nets are allowed to cross each other on the same layer.

A primary objective function of a router is to meet timing constraints for each net and complete the routing of all the nets. Channel routers attempt to minimize the total routing area. Various secondary objective functions have also been considered, such as, improve manufacturability by minimizing the number of vias and jogs, improve performance by minimizing crosstalk between nets and delay for critical nets, among others. Minimizing vias is important, since vias are difficult to fabricate due to the mask alignment problem. In addition, via's increase delay and are therefore undesirable in high-performance applications. Other objective functions include minimization of the average or total length of a net, and minimization of the number of vias per net.

In this chapter, we discuss the routing problem and various algorithms proposed to solve different versions of the routing problem. In the next section, we first formulate the routing problem and classify different routing problems.

9.1 Problem Formulation

As mentioned earlier, the detailed routing problem is solved by solving one routing region at a time. The routing area is first partitioned into smaller regions. Since, the global router only assigns wires to different regions, the detailed routing problem is to find the actual geometric path for each wire in a region. The complexity of the routing problems varies due to many factors including shape of the routing region, number of layers available, and number of nets. However, the shape of the region is perhaps the most important factor. Before presenting the routing problem formally, we describe important considerations and models used in routing.

9.1.1 Routing Considerations

In general, the routing problem has many parameters. These parameters are usually dictated by the design rules and the routing strategy.

1. **Number of terminals:** Majority of nets are two terminal nets, however, the number of terminals in a net may be very large. This is especially true for global nets such as clock nets. In order to simplify the routing problem, traditionally, routing algorithms assume all nets to be two terminal nets. Each multi-terminal net is decomposed into several two terminal nets. More recently, algorithms which can directly handle multi-terminal nets have also been developed.

2. **Net width:** The width of a net depends on the layer it is assigned and its current carrying capacity. Usually, power and ground nets have different widths and routers must allow for such width variations.

3. **Pin locations:** In channels, pins are located on the top and bottom boundaries. In addition, pin may be located on the sides as well as in the middle of the channel to connect to 3D-switchboxes. The pins on the sides are assigned by the global router. In 2D-switchboxes, the pin are located on all four sides as well as in the middle. The most general form of routing region is a 3D-switchbox, which has pins on all six sides. The pins of the bottom are assigned by the global router so that nets are pass from channels and 2D-switchboxes to 3D-switchboxes and vice versa. The pins on the sides allow nets to pass from one 3D-switchbox to another. The pins on the top allow nets to connect to C4 solder bumps.

4. **Via restrictions:** The final layout of a chip is specified by means of masks. The chip is fabricated one layer at a time, and the masks for the various layers must align perfectly to fabricate the features such as vias which exist in two layers. Perfect alignment of masks is difficult, and thus vias were normally only allowed between adjacent layers. Even between two layers, minimization of vias reduces mask alignment problems. Improvements in the chip manufacturing technology have reduced mask alignment problems, and today stacked vias (vias passing through more than two layers) can be fabricated. However, vias still remain a concern in routing problems and must be minimized to improve yield, performance and area.

5. **Boundary type:** A boundary is the border of the routing region which contains the terminals. Most detailed routers assume that the boundaries are regular (straight). Even simple routing problems which can be solved in polynomial time for regular boundaries become NP-hard for the irregular boundary routing problem. Some recent routers [Che86, CK86, VCW89] have the capability of routing within irregular boundaries.

6. **Number of layers:** Almost all fabrication processes allow three or four layers of metal for routing. Recently, a fifth metal layer has also become available; however, its usage is restricted due to its cost. Six and seven layer processes are expected to be available within two to three years. Most existing detailed routers assume that there are two or three layers available for routing. Recently, several n-layer routers have also been developed. Each layer is sometimes restricted to hold either vertical or horizontal segments of the nets. It is expected that as the fabrication technology improves, more and more layers will be available for routing. In our formulation of five metal process, channel and 2D-switchbox routers must route in M1, M2 and M3. While, 3D-switchbox router must route in M4 and M5.

7. **Net types:** Some nets are considered critical nets. Power, ground, and clock nets fall in this category. Power, and ground wires need special consideration since they are normally wider than signal wires. Clock nets require very careful routing preference, since the delay of the entire chip may depend on clock routing. Due to this type of restriction placed on

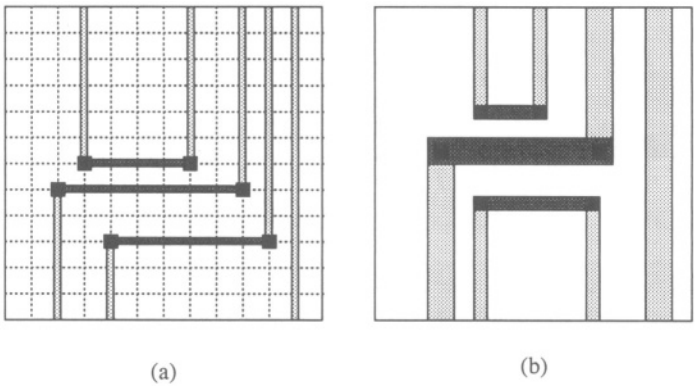


Figure 9.2: (a) Grid-based. (b) Gridless.

critical nets, they need to be routed before signal nets using specialized routers or often routed by hand.

9.1.2 Routing Models

For ease of discussion and implementation of net-routing problems, it is often necessary to work at a more abstract level than the actual layout. In many cases, it is sufficient to use a mathematical wiring model for the nets and the rules that they must obey. For instance, wires are usually represented as paths without any thickness, but the spacing between these wires is increased to allow for the actual wire thickness and spacing in the layout. The most common model used is known as the *grid-based* model. In this model, a rectilinear (or possibly octilinear) grid is super-imposed on the routing region and the wires are restricted to follow paths along the grid lines. A horizontal grid line is called a *track* and a vertical grid line is called a *column.* Any model that does not follow this 'gridded' approach is referred to as a *gridless model*

In the grid-based approach, terminals, wires and vias are required to conform to the grid. The presence of a grid makes computation easy but there are several disadvantages associated with this approach, including the large amount of memory required to maintain the grid and restricted wire width. The gridless approach, on the other hand, allows arbitrary location of terminals, nets, and vias. Moreover, nets are allowed arbitrary wire widths. Due to these advantages, the gridless approach is gaining more popularity than the grid-based approach [Che86, CK86]. Figure 9.2 illustrates some of the differences in grid-based and gridless routing.

Routing problems can also be modeled based on the layer assignments of horizontal and vertical segments of nets. This model is applicable only in multilayer routing problems. If any net segment is allowed to be placed in any layer, then the model is called an *unreserved layer model.* When certain type of seg-

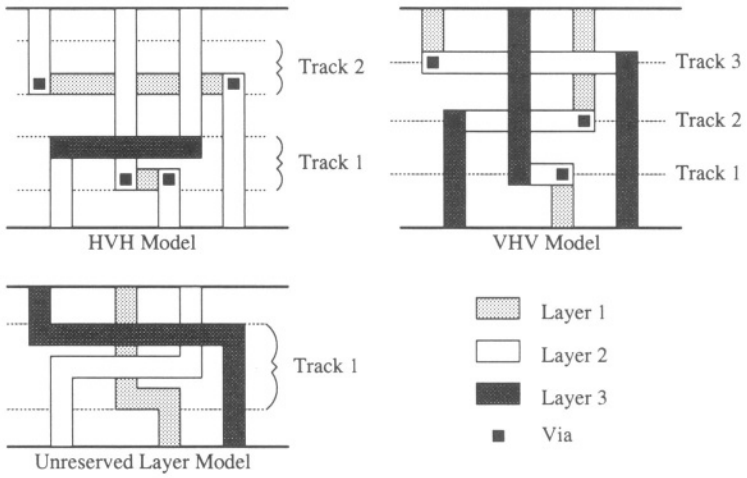


Figure 9.3: A comparison between HVH, VHV, and unreserved layer models.

ments are restricted to particular layer(s), then the model is called a *reserved layer model.* Most of the existing routers use reserved layer models. In a two-layer routing problem, if the layer 1 is reserved for vertical segments and layer 2 is reserved for horizontal segments, then the model is called a VH model. Similarly, a HV model allows horizontal segments in layer 1 and vertical segments in layer 2. Two-layer models can be extended to three-layer routing models: VHV (Vertical-Horizontal-Vertical) or HVH (Horizontal-Vertical-Horizontal). In the VHV model the first and third layers are reserved for routing the vertical segments of nets and the second layer is reserved for routing the horizontal segments. On the other hand, in the HVH model, the first and third layers are reserved for routing the horizontal segments of nets and the second layer is reserved for routing the vertical segments. The HVH model is preferred to the VHV model in channel routing because, in contrast with the VHV model, the HVH model offers a potential 50% reduction in channel height.

Figure 9.3 shows an example of the HVH model using two tracks, the VHV model using three tracks, and the unreserved model using only one. The HVH model and unreserved layer models show more than one trunk per track in Figure 9.3. This is done because the horizontal segments were placed on different layers, the figure offsets them slightly for a clearer perspective of the routing. An unreserved layer model has several other advantages over the reserved layer model. This model uses less number of vias and in fact, in most cases, can lead to an optimal solution, i.e., a solution with minimum channel height. The unreserved routing model also has it disadvantages, such as, routing complexity, blocking of nets, among others. Generally speaking, reserved layer and gridded routers are much faster than gridless and unreserved layer routers.

Another unreserved layer model based on use of *knock-knees* has also been

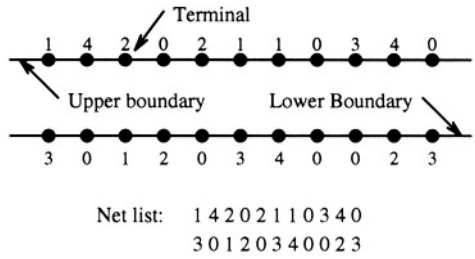


Figure 9.4: A channel and its associated net list.

proposed. The knock-knee model allows two nets to share a grid point if they are in different layers. This model has the advantage of avoiding undesirable electrical properties caused due to overlap of wire segments, such as capacitive coupling.

We now discuss the problem formulation for both channel and switchbox routing problems.

9.1.3 Channel Routing Problems

A *channel* is a routing region bounded by two parallel rows of terminals. Without loss of generality, it is assumed that the two rows are horizontal. The top and the bottom rows are also called *top boundary* and *bottom boundary,* respectively. Each terminal is assigned a number which represents the net to which that terminal belongs to (see Figure 9.4). Terminals numbered zero are called *vacant terminals.* A vacant terminal does not belong to any net and therefore requires no electrical connection. The net list of a channel is the primary input to most of the routing algorithms.

The horizontal dimension of the routed channel is called the *channel length* and the vertical dimension of the routed channel is called the *channel height.* The horizontal segment of a net is called a *trunk* and the vertical segments that connect the trunk to the terminals are called its *branches.* The horizontal line along which a trunk is placed is called a *track.* A *dogleg* is a vertical segment that is used to maintain the connectivity of the two trunks of a net on two different tracks. A pictorial representation of the terms mentioned above is shown in Figure 9.5.

A channel routing problem (CRP) is specified by four parameters: Channel length, Top (Bottom) terminal list, Left (Right) connection list, and the number of layers. The channel length is specified in terms of number of columns in grid based models, while in gridless models it is specified in terms of λ. The Top and the Bottom lists specify the terminals in the channel. The Top list is denoted by $T = (T_1, T_2, \ldots, T_m)$ and the bottom list by $B = (B_1, B_2, \ldots, B_m)$. In grid based models, T_i (B_i) is the net number for the terminal at the top (bottom) of the ith column, or is 0 if the terminal does not belong to any

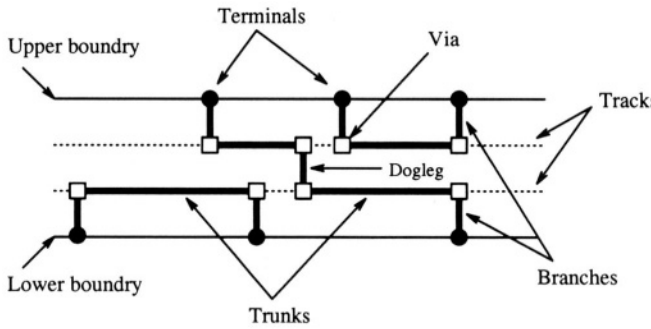


Figure 9.5: Terminology for channel routing problems.

net. In gridless model, each terminal, T_i (B_i), indicates the net number to which the ith terminal. The Left (Right) Connection list, consist of nets that enter the channel from the left (right) end of the channel. It is an ordered list if the channel to the left (right) of the given channel has already been routed.

Given the above specifications, the problem is to find the interconnections of all the nets in the channel including the connection sets so that the channel uses minimum possible area. A solution to a channel routing problem is a set of horizontal and vertical segments for each net. This set of segments must make all terminals of the net electrically equivalent. In the grid based model, the solution specifies the channel height in terms of the total number of tracks required for routing. In gridless models, the channel height is specified in terms of λ.

The main objective of the channel routing is to minimize the channel height. Additional objectives functions, such as, minimizing the total number of vias used in a multilayer routing solution, and minimizing the length of any particular net are also used. In practical designs, each channel is assigned a height by the floorplanner and the channel router's task is to complete the routing within the assigned height. If channel router cannot complete the routing in the assigned height, channel has to expand, which changes the floorplan. This requires routing the channels in a predefined order, so that such expansions can be accommodated, without major impact on the floorplan.

In grid based models, the channel routing problem is essentially assignment of horizontal segments of nets to tracks. Vertical segments are used to connect horizontal segments of the same net in different tracks and to connect the terminals to the horizontal segments. In gridless models, the problem is somewhat similar except the assignment of horizontal segments is to specific locations in the channel rather than tracks. There are two key constraints which must be satisfied while assigning the horizontal and vertical segments.

1. **Horizontal Constraints:** There is a horizontal constraint between two nets if the trunks of these two nets overlap each other when placed on the

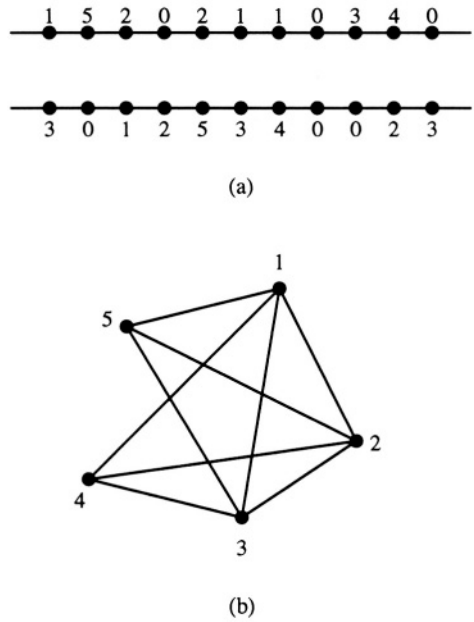


Figure 9.6: A routing problem and its HCG.

same track. For a net N_i, the interval spanned by the net, denoted by I_i is defined by (r_i, l_i), where r_i is the right most terminal of the net and l_i is the leftmost terminal of the net. Given a channel routing problem, a *horizontal constraint graph* (HCG) is a undirected graph $G_h = (V, E_h)$ where

$$V = \{v_i | v_i \text{ represents } I_i \text{ corresponding to } N_i\}$$

$$E_h = \{(v_i, v_j) |\ I_i \text{ and } I_j \text{ have non-empty intersection}\}$$

Note that HCG is in fact an interval graph as defined in chapter 3. Figure 9.6(a) shows a channel routing problem and the associated horizontal constraint graph is shown in Figure 9.6(b).

The HCG plays a major role in determining the channel height. In a grid based two-layer model, no two nets which have a horizontal constraint maybe assigned to the same track. As a result, the maximum clique in HCG forms a lower bound for channel height. In the two-layer gridless model, the summation of widths of nets involved in the maximum clique determine the lower bound.

2. **Vertical Constraints:** A net N_i, in a grid based model, has a vertical constraint with net N_j if there exists a column such that the top terminal of the column belongs to N_i and the bottom terminal belongs to N_j and $i \neq j$. In case of the gridless model, the definition of vertical constraint is

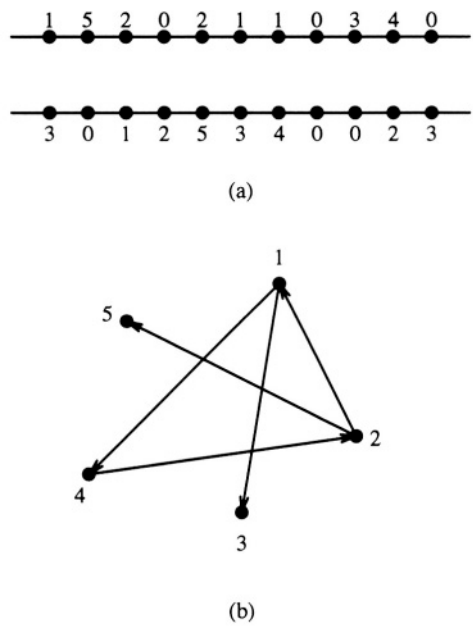


Figure 9.7: A simple routing problem and its VCG.

somewhat similar except that the overlap is between the actual vertical segments rather than terminals in a column. Given a channel routing problem, a *vertical constraint graph* (VCG) is a directed graph $G_v = (V, E_v)$, where,

$$E_v = \{(v_i, v_j) | N_i \text{ has vertical constraint with } N_j\}$$

It is easy to see that a vertical constraint, implies a horizontal constraint, however, the converse is not true. Figure 9.7(b) shows the vertical constraint graph for the channel routing problem in Figure 9.7(a).

Consider the effect of a directed path in the vertical constraint graph on the channel height. If doglegs are not allowed then the length of the longest path in VCG forms a lower bound on the channel height in the grid based model. This is due to the fact that no two nets in a directed path may be routed on the same track. Note that if VCG is not acyclic than some nets must be doglegged. Figure 9.8(a) shows a channel routing problem with a vertical constraint cycle while Figure 9.8(b) shows how a dogleg can be used to break a vertical constraint cycle. Figure 9.8(c) shows vertical constraint cycle involving four nets. In Figure 9.8(d), we show one possible routing for the example in Figure 9.8(c).

The two constraint graphs can be combined to form a mixed graph called the Combined Constraint Graph (CCG) which has the same vertex set as the

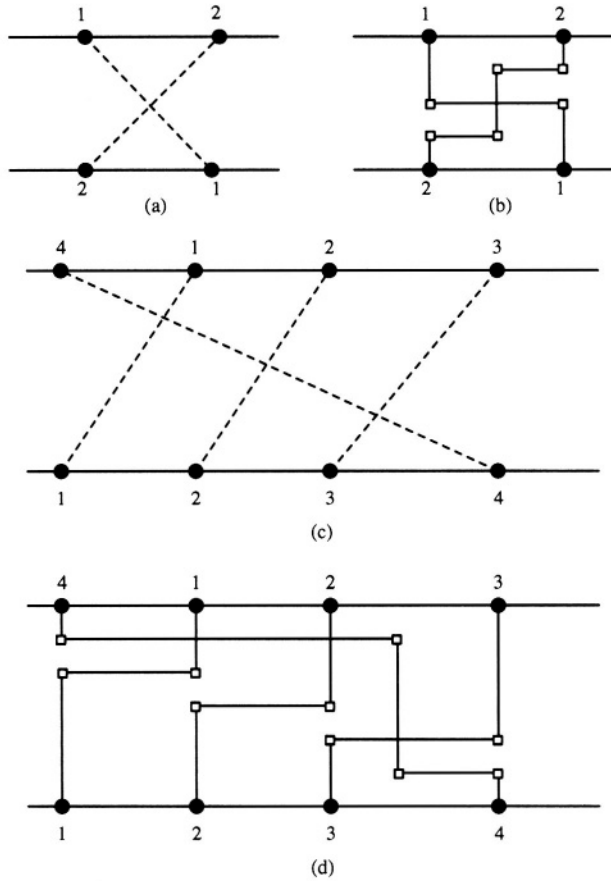


Figure 9.8: A cyclic vertical constraint.

HCG and VCG while the edge set is the union of E_h and E_v. The combined constraint graph for Figure 9.6(a) is shown in Figure 9.9.

Two interesting graphs related to channel routing problem are the permutation graph and the circle graph. The permutation graph can only be defined for channel routing problem for two terminal nets and no net has both of its terminal on one boundary (see Chapter 3). These graphs allow us to consider the channel routing problem as a graph theoretic problem.

Note that, we do not address the channel routing problem with pins in the middle of the channel in this book. This problem is largely a research topic and it is currently solved by using area routers.

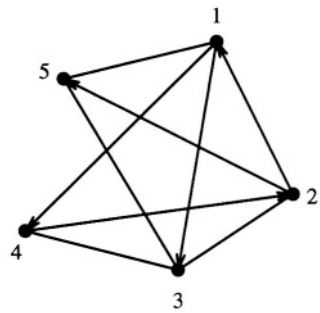


Figure 9.9: A combined constraint graph.

9.1.4 Switchbox Routing Problems

Switchbox routing problem is a generalization of the channel routing problem, where terminals are located on all four sides. Switchboxes are formed in two ways. There maybe be a four sided enclosed region within which the routing must be completed or a four sided region maybe formed due to the intersection of two channels. A switchbox is formally defined as a rectangular region R $(h \times w)$ where h and w are positive integers. Each pair (i, j) in R is a grid point. The ith column and jth row or track are the sets of grid points. The 0th and hth columns are the LEFT and RIGHT boundaries respectively. Similarly, the 0th and wth rows are TOP and BOTTOM boundaries respectively. The connectivity and location of each terminal are represented as $\text{LEFT}(i) = k$, $\text{RIGHT}(i) = k$, $\text{TOP}(i) = k$, or $\text{BOTTOM}(i) = k$ depending upon the side of the switchbox it lies on, where i stands for the coordinate of the terminal along the edge and k is a positive integer specifying the net to which the ith terminal belongs to.

Since it is assumed that the terminals are fixed on the boundaries, the routing area in a switchbox is fixed. Therefore, the objective of switchbox routing is not to minimize the routing area but to complete the routing within the routing area. In other words, the switchbox routing problem is a routability problem, i.e., to decide the existence of a routing solution. Unlike the channel routing problem, switchbox routing problem is typically represented by its circle graph (see Chapter 3).

Note that we do not address the 3D-switchbox routing in this book. This problem is solved by using area routing approaches. Some concepts and algorithms related to 3D-switchbox and OTC routing will be discussed in Chapter 8.

9.1.5 Design Style Specific Detailed Routing Problems

In this section, we discuss the detailed routing problem with respect to different design styles.

1. **Full custom:** The full custom design has both channels and switchboxes. As explained earlier, depending on the design, the order in which the channels and 2D-switchboxes are routed is important. A 3D-switchbox can only be routed after all the channels and 2D-switches under it have been routed. The objective of a detailed routing algorithm is to complete the routing in a manner that each net meets it timing constraint and minimum area is utilized for routing. Other constraints such as manufacturability, reliability and performance constraints are also used.

2. **Standard cells:** The standard cell design style has channels of uniform lengths which are interleaved with cell rows. Hence the detailed routing problem is reduced to routing channels. Unlike in the full-custom design, the order in which the channels are routed is not important. This is possible since global router assigns pins in the feedthroughs. Typically, regions on top of cells can be used for 3D-switchbox routing. This will be explained in more detail in Chapter 8. The objective is to route all the nets in the channel so that the height of the channel is minimized. Additional constraints such as minimizing the length of the longest net and restricting length of critical nets within some prespecified limits are used for high performance standard cell designs.

3. **Gate arrays:** The gate arrays have channels of fixed size and hence the detailed routing algorithms have to route all the nets within the available routing regions. If the detailed router cannot route all the nets, the partitioning process may have to be repeated till the detailed routers can route all the nets. For high performance routing net length constraints must added.

9.2 Classification of Routing Algorithms

There could be many possible ways for classifying the detailed routing algorithms. The algorithms could be classified on the basis of the routing models used. Some routing algorithms use grid based models while some other algorithms use the gridless model. The gridless model is more flexible as all the wires in a design need not have the same widths. Another possible classification scheme could be to classify the algorithms based on the strategy they use. Thus we could have greedy routers, hierarchical routers, etc. to name a few. We classify the algorithms based on the number of layers used for routing. Single layer routing problems frequently appear as sub-problems in other routing problems which deal with more than one layers. Two and three layer routing problems have been thoroughly investigated. Recently, due to improvements in the fabrication process, fourth and fifth metal layers have also been allowed but this process is expensive compared to three-layer metal process. Several multi-layer routing algorithms have also been developed recently, which can be used for routing MCMs which have up to 32 layers.

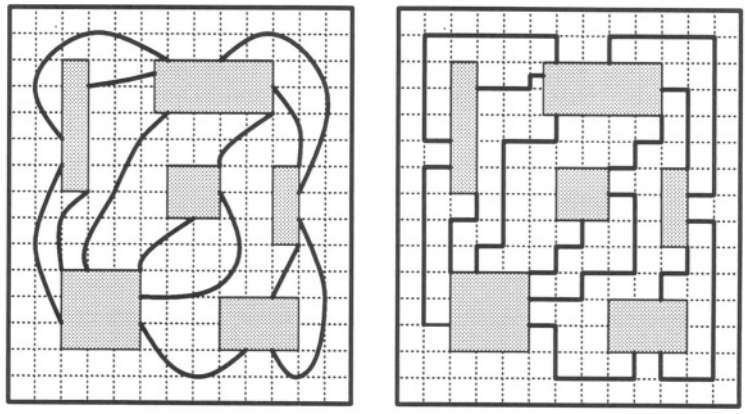


Figure 9.10: Single layer routing problem.

9.3 Single-Layer Routing Algorithms

A general single-layer routing problem can be stated as follows. Given a routing region, a netlist $\mathcal{N} = \{ N_1, N_2, \ldots, N_n \}$, a set of terminals $\{T_{i,j}, i = 1, \ldots, n, j = 1, \ldots, n_i\}$where, $T_{i,j}$ specifies the jth terminal of net N_i, a set of blocks $\mathcal{B} = \{B_1, B_2, \ldots, B_l, B_{l+1}, B_{l+2}, \ldots, B_m\}$, where $B_1, B_2, \ldots, B_l$ are flippable and $B_{l+1}, B_{l+2}, \ldots, M_m$ are not flippable (a block is flippable if orientation of its terminals is not determined). Also given is a set of design rule parameters which specify the necessary widths of wire segments and the minimum spacing between wires. The single-layer routing problem is to find a set of wire segments inside the routing region which complete the connections required by the netlist without violating any design rule. Figure 9.10 shows an instance of a single-layer routing problem. Figure 9.10(a) gives the global routing of the instance of the problem and Figure 9.10(b) gives the detailed routing of wires on a single layer.

Although the general single layer routing problem is conceptually easier than the multi-layer routing problem, it is still computationally hard. In single-layer routing, the fundamental problem is to determine whether all the nets can be routed. This problem is called *single-layer routability problem* and is known to be NP-complete [Ric84]. Figure 9.11 shows an instance of a single-layer routing problem that is unroutable.

There are many practical restricted versions of the single-layer routing problem which are easier to handle than the general single-layer routing problem [MST83]. For example, consider the following:

1. There are no flippable blocks, i.e., $l = 0$.
2. All the blocks are flippable, i.e., $m = l$.
3. All the nets are two-terminal nets with no flippable blocks.

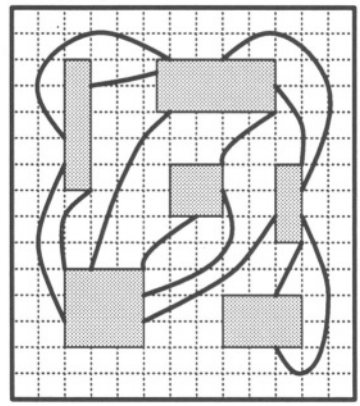


Figure 9.11: A unroutable single layer example.

4. All the nets are two-terminals nets with all flippable blocks.

5. There are no blocks inside the routing region and all nets are two-terminal nets.

6. There are no blocks inside the routing region, the nets are two-terminal and the terminals lie on a single row.

Problems 3, 4, and 5 are commonly known as the variations of *river routing*. Problem 6 is known as *single row routing problem*.

Several special cases of the single layer routing problem can be solved in polynomial time [BP83, DKS+87, LP83, Ma190, SD81, Tom81].

Although, single layer routing problem appears restricted when one considers that fabrication technology allows three layers for routing. However, single layer routing still can be used for power and ground routing, bus routing, over-the-cell routing and some clock routing problems. During floorplanning, the sequence of the input and output busses is determined for each block. Since a bus may have a very large number of nets, it is advisable to pre-route the buses. Buses are routed such that the output bus of a block is in the same sequence as that of the input bus of a receiving block. Since the input and output busses of the blocks have the same sequence, it may be possible to make the interconnections between blocks on a single layer. This also minimizes vias and area required for bus routing. Power and ground nets are sometimes also routed in a single layer due to electrical considerations. The power and ground routing problems will be considered in Chapter 9. In the three layer environment, in certain regions, the two underlying metal layers may be blocked and only the top layer is available for routing. In this case, additional nets may be routed using single layer techniques on the third layer. This is a typical situation in over the cell routing. We will discuss over-the-cell routing in Chapter 8. Finally, for high performance circuits, clock nets may be routed in a single layer,

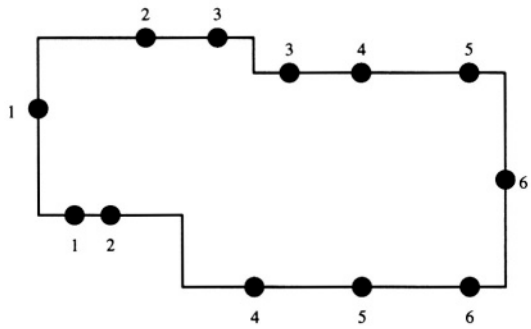


Figure 9.12: An example of the general river routing problem.

as electrical effects of vias are undesirable due to performance considerations. In this section, we discuss two special cases of single layer routing problem, the river routing problem and the single row routing problem.

9.3.1 General River Routing Problem

River routing is a special case of the single layer routing under following assumptions. All the terminals lie on the boundary of the region. Every net consists of exactly two terminals and there are no blocks in the region. Terminals are located in such a way that no crossover between nets is necessary for a solution to exist, that is, nets are planar. Figure 9.12 shows an example of a general river routing problem.

A special case of general river routing problem, which has attracted a lot of attention is simply called the *river routing* problem [Hsu83a, JP89, LP83, MST83, TH90]. It is essentially a single layer channel routing problem for two terminal nets, such that each net has one terminal on each boundary. Figure 9.13 shows an example of a river routing problem. We will concentrate on the general river routing problem and present an algorithm for an arbitrary shaped rectilinear routing region.

9.3.1.1 General River Routing Algorithm

In this section, we discuss the general river routing algorithm presented by Hsu [Hsu83a]. This algorithm is capable of routing in arbitrary shaped rectilinear routing regions and guarantees that a solution will be found if one exists. The algorithm is gridless and allows arbitrary net widths and wire separations. Although, the algorithm is developed for two terminal nets, it can be easily extended for multi-terminal nets. We start by defining some terminology.

Let a path be a alternating sequence of horizontal and vertical segments connecting two terminals of a net. A terminal is called starting terminal if it is connected to the first segment of a path. Similarly, the terminal connected

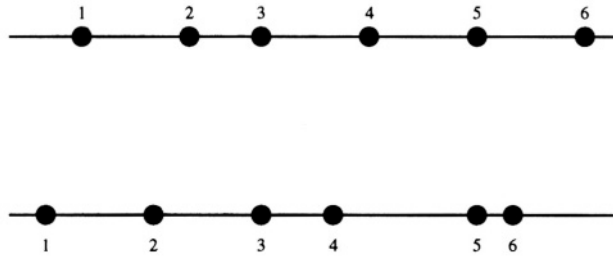


Figure 9.13: A simple river routing problem.

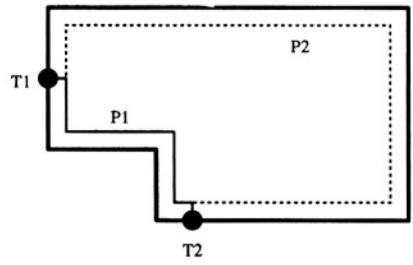


Figure 9.14: Two possible paths of a net along the boundary.

with the last segment is called an ending terminal. Without loss of generality, it will be assumed that every path is counter-clockwise along the boundary. Every net has two possible paths along the boundary and therefore there are the two possible choices of starting terminal for the net. Figure 9.14 shows two possible paths along the boundary for a net $N_i = T_1, T_2$. Path P_1 has T_1 as its starting terminal and path P_2 has T_2 as its starting terminal. The general river routing algorithm routes one net at a time and consists of four phases. In the first phase, the starting terminal of each net is determined. In the next phase, net order is determined by the sequence of terminals on the boundaries. Based on the net order, path searching is done by routing each net, in order, as close to the boundaries as possible. Unnecessary corners are then removed by flipping the corners in the last phase. We will now briefly discuss these phases.

Starting Terminal Assignment: As stated earlier, each net has two possible paths along the boundary. The starting terminal for a net is chosen independent of all other nets, such that the shorter path is selected. In order to select a starting terminal for a net, the length of the path in the counter-clockwise direction is computed and compared to the half of the total length of the boundary of the routing region. In figure 9.14, terminal $T_{i,1}$ is assigned to be the starting terminal, since path P_1 is shorter than path P_2. Figure 9.16

shows an example of the starting terminal assignment for a netlist.

Net Ordering: Every path is counter-clockwise and begins at the starting terminal, as a result, the order in which the nets are routed is very important. A net can only be routed after all the nets 'contained' by the net are already routed. A net N_i is contained by another net N_j, if all the terminals of N_i are on the boundary between the starting and ending terminals of the net N_j. Note that only the counter-clockwise boundary is considered.

To determine the net order, a circular list of all terminals ordered in counter-clockwise direction according to their positions on the boundaries is generated. A planarity check is performed to determine if the given instance is routable. If the given instance is routable, the nets are ordered by NET-ORDERING algorithm as given below. The basic idea is to just push the starting terminal of the nets on the stack as they are encountered. A number is assigned to a net N_i, when the algorithm encounters the ending terminal of net N_i and the top item on the stack is the starting terminal of net N_i. This ensures that all the nets contained in net N_i are assigned a number, before assigning a number to net N_i. Net N_i is then deleted from further consideration and algorithm continues until all nets have been numbered. The formal description of the algorithm appears in Figure 9.15.

Consider the example shown in Figure 9.16. Starting at terminal 1, terminals are considered in counter-clockwise order. The net N_1 is assigned first, as its ending terminal is encountered, while the top of the stack has the starting terminal of N_1. The starting terminal of N_2 is pushed onto the stack, followed by pushing of starting terminal of net N_3. The net N_3 is number second as its ending terminal is encountered next. The final net ordering for the example in Figure 9.16 is {1,3,8,7,6,5,2,4}. The 's' next to a terminal indicates that the terminal is a starting terminal.

Path Searching: Based on the net order, each net is routed as close to the pseudo-boundary as possible. For the first net, the pseudo-boundary is the boundary of the region. For the second net, the segments of the first net and the segments of the boundary not covered by the first net form the pseudo-boundary. In other words, each time a net is routed, the region available for routing is modified and the boundary of this region is referred to as *pseudo-boundary*. The path of the net is checked for design rule violations by checking the distances between the counter-clockwise path of the net and the pseudo-boundary not covered by the net. If a violation occurs, it implies that the given problem is unroutable. Figure 9.17 shows the pseudo-boundary 'abcdihgf' for net N_i and the path is created by routing as close to 'abcd' as possible. The path is then checked against the remaining segments of the pseudo-boundary, i.e., 'ihgf' for design rule violations.

Corner Minimization: Once the path searching for all nets has been completed without design rule violation, a feasible solution has been found. However, the routing technique described above pushes all the paths outward

```
Algorithm NET-ORDERING
begin
    for i = 1 to 2n do
        if END-TERMINAL(T_i) then
            MATCHED(T_i) = 0;
        else
            MARKED(T_i) = 0;
    stack = φ;
    i = 1;
    T = any terminal in the circular list;
    while i ≤ n do
        if START-TERMINAL(T) and MARKED(T) = 0
        then PUSH(T, stack);
            MARKED(T) = 1;
        else if END-TERMINAL(T) and MATCHED(T) = 0
        then T1 = POP(stack);
            if T = T1 then
                MATCHED(T) = 1;
                ASSIGN-NUMBER(i, NET(T));
                i = i + 1;
            else exit;
        T = next terminal in the circular list;
end.
```

Figure 9.15: Algorithm NET-ORDERING

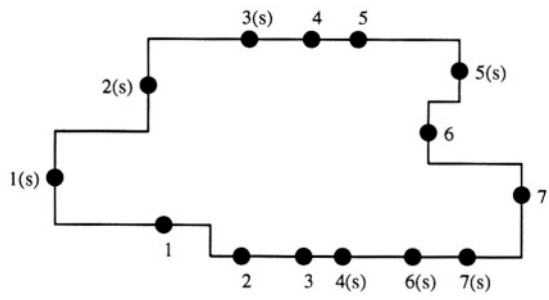


Figure 9.16: The assignment of starting terminals

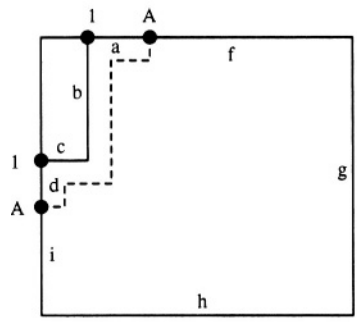


Figure 9.17: Pseudo-boundary and path creation.

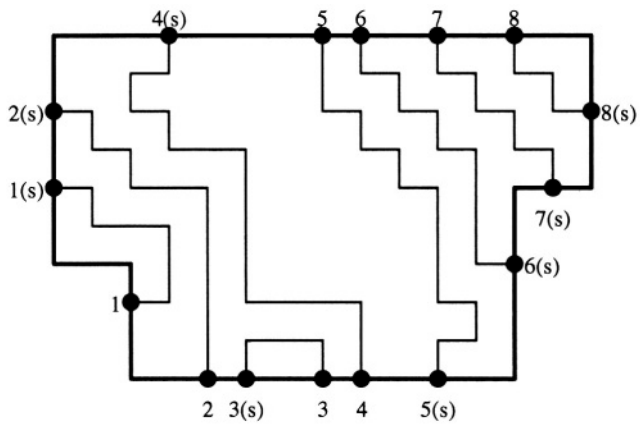


Figure 9.18: A boundary-packed solution after path searching.

against the boundaries and the excess space is left vacant in the center of the routing region. Figure 9.18 shows an example of the routing after path searching.

The corner minimization is a systematic method of flipping corners toward the inside of the routing region. Corners are minimized one net at a time. The order of the nets for this operation is precisely the reverse of the order determined by the previous net ordering step. That is, the corners of the paths are minimized starting with nets from the center of the routing region towards the boundary of the routing region.

Every corner of a path belongs to one of the eight possible cases as shown in Figure 9.19. Since every path is routed in the counter-clockwise direction, in four cases the corners can be flipped towards the inside of the routing region. Figure 9.19 shows the four cases a, b, c, d which can be respectively transformed

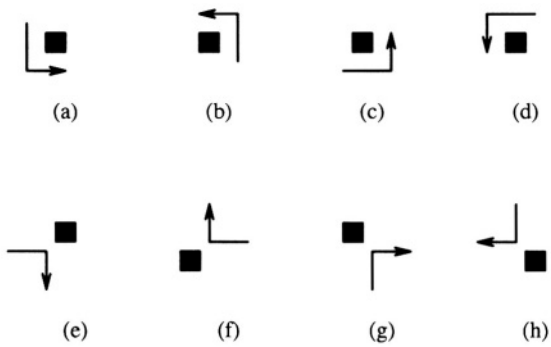


Figure 9.19: Eight possible cases of a corner.

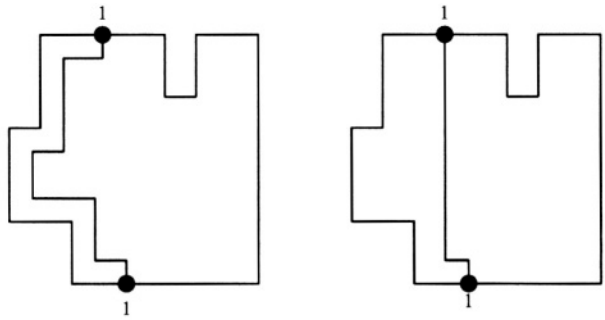


Figure 9.20: Length minimization by flipping corners.

to cases e, f, g, h by flipping towards the inside. The inside of the region is indicated by a filled dot. A pseudo-boundary is generated in the same way as in the path searching step and design violation checks are performed before all corner flips. If the intended corner flip does not create any design rule violation, the corner is flipped and two corners are eliminated from the path. Otherwise, this corner is skipped and the next corner of the path is checked. Figure 9.20 shows an example of a path before and after flipping of corners.

9.3.2 Single Row Routing Problem

Given a set of two-terminal or multi-terminal nets defined on a set of evenly spaced terminals on a real line, called the *node axis,* the single row routing problem (SRRP) is to realize the interconnection of the nets by means of non-crossing paths. Each path consists of horizontal and vertical line segments on a single layer, so that no two paths cross each other. Moreover, no path is allowed to intersect a vertical line more than once, i.e., backward moves of nets

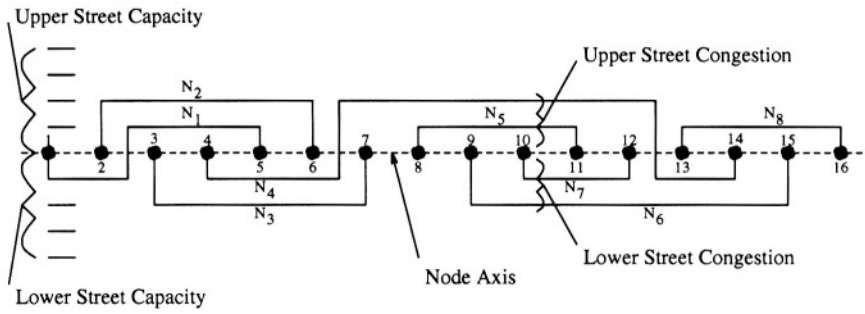


Figure 9.21: Basic terminology and a single row realization of the net list L_1.

are not allowed.

For an example consider the net list $\mathcal{N} = \{N_1, N_2, \ldots, N_8\}$ where $N_1 = \{1, 5\}$, $N_2 = \{2, 6\}$, $N_3 = \{4, 14\}$, $N_4 = \{3, 7\}$, $N_5 = \{8, 11\}$, $N_6 = \{9, 15\}$, $N_7 = \{10, 12\}$, $N_8 = \{13, 16\}$. A single row realization of $\mathcal{N}$ is shown in Figure 9.21.

The area above the node axis is called the *upper street* while the area below the node axis is called the *lower street.* The number of horizontal tracks available for routing in the upper street is called *upper street capacity.* Similarly the number of horizontal tracks available in the lower street is called the *lower street capacity.* Due to symmetry in single row routing, the upper street capacity is usually equal to the lower street capacity. For a given realization, the number of the horizontal tracks needed in the upper street is called the *upper street congestion* (C_{us}) and the number of horizontal tracks needed in the lower street is called the *lower street congestion* (C_{ls}). The term *dogleg* is used to describe a bend in a net, when it makes an interstreet crossing. The *between-nodes congestion* C_B of a realization is the maximum number of interstreet crossings between a pair of adjacent terminals. For the realization shown in Figure 9.21, $C_{us} = 2, C_{ls} = 2, C_B = 1$. The net N_1 is doglegged once, while the net N_3 is doglegged twice.

The objective function considered most often is to minimize the maximum of upper and lower street congestions, i.e., minimize Q_0, where $Q_0 = \max\{C_{us}, C_{ls}\}$. To minimize the separation between the two adjacent terminals it is sometimes necessary to minimize C_B. In practical problems, $Q_0 \leq 3$ and $C_B \leq 2$. Other objective functions include minimizing the total number of doglegs in a realization or to minimize number of doglegs in a wire.

9.3.2.1 Origin of Single Row Routing

The SRRP was introduced by So in the layout design of multilayer circuit boards [So74]. It has received considerable attention [HS84a, KKF79, RS83, RS84, TKS76, TMSK84, TKS82]. So proposed a systematic approach to the routing of large multi-layer printed circuit board problem(MPCBP). This ap-

proach consists of a well defined decomposition of the MPCBP into several independent single layer single row routing problems. The scheme decomposes the MPCBP routing problem into five phases:

1. via assignment,
2. placement of via columns,
3. layering,
4. single row routing, and
5. via elimination,

In the via assignment phase, each multi-terminal net is decomposed into several two terminal nets. A net whose terminals lies on the same row or same column is connected by a wire. A net is not directly decomposable if it contains two terminals not in the same row or same column. In this case vias are introduced to facilitate decomposition of the net. In the second phase of decomposition the via columns are permuted to minimize the wire lengths. Obviously this change is meaningful only if vias appear column-wise. In particular, in this step, the locations of two via columns are exchanged without violating any of the net connections.

In the third phase of decomposition, a single row routing problem is decomposed into several single row routing problems so that each subproblem can be routed to satisfy the upper and lower street constraints in a different layer. Usually, half the available layers are used for realization of the row problems, and the other half of the available layers is used for the column problems.

Sufficient conditions for a realization with minimum congestion along with a routing algorithm were presented by Ting, Kuh, and Shirakawa [TKS76]. It was shown that an arbitrary set of nets can be realized if upper and lower street capacities are unbounded. Kuh, Kashiwabara, and Fujisawa [KKF79], presented an interval diagram representation of the single row routing problem. This representation played an important role in the research and development of several algorithms for the single row routing problem. The interval diagram representation of an example is given in Figure 9.22. The broken line shown in Figure 9.22(b) is called the *reference line.* The layout is obtained by stretching out the reference line and setting it on top of the node axis. The interval lines for each net are mapped topologically onto vertical and horizontal paths. The nets and its segments above the reference line are mapped onto paths in the upper street, while the nets and its segments below the reference line are mapped onto paths in the lower street. This process defines a unique realization as shown in Figure 9.22(c). An important implication of the interval diagram representation is that it reduces the single row routing problem to finding an optimal permutation of nets and thus greatly enhances the understanding of the problem.

Kuh, Kashiwabara, and Fujisawa [KKF79] also proposed an algorithm for minimizing street congestion. It was based on the number of possible orderings

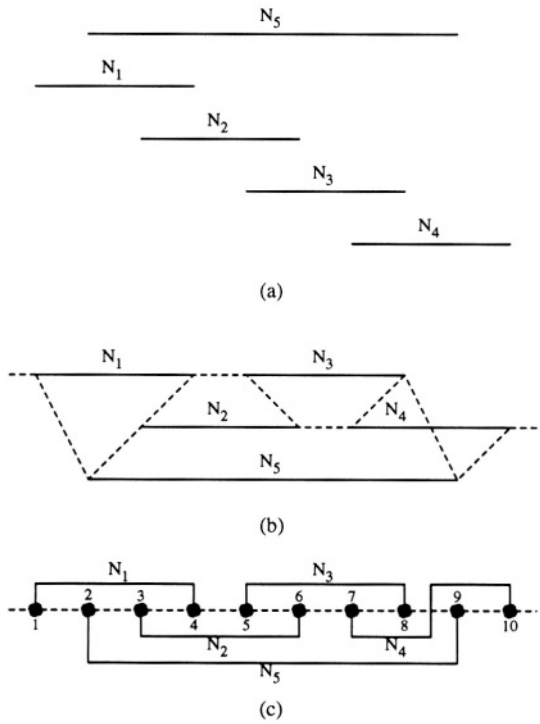


Figure 9.22: Interval diagram representation

(permutations) of nets that can be considered for routing. Another important contribution of Kuh, Kashiwabara, and Fujisawa [KKF79] is the development of necessary and sufficient conditions for the optimal realization of the single row routing problem. These conditions were based on the idea of the *cut number* of a net. The cut number of a terminal is the number of nets passing over that terminal. The cut number of a net is maximum among all cut numbers of its terminals. Example in Figure 9.23 shows the concept of cut number. Let q_i be the cut number of net N_i. Then in the Figure 9.23, $q_1 = 2$, $q_2 = 3$, $q_3 = 4$, $q_4 = 3$, $q_5 = 4$, and $q_6 = 2$. Let $q_{\max}$ and $q_{\min}$ be the maximum and minimum over the cut numbers of all nets, respectively. Then in the Figure 9.23, $q_{\max} = 4$ and $q_{\min} = 2$.

The main idea behind the necessary and sufficient condition is the optimal partitioning of nets at each terminal. A realization is optimal with congestion equal to $q_t = \frac{q_{\max}}{2}$ if at each terminal with cut number c there are at least $k = c - q_t$ nets above and k nets passing below that terminal, not counting the net to which the terminal belongs. In other words, if c nets cover a terminal, then a realization with congestion equal to q_t is optimal only if the nets covering this terminal can be partitioned into two sets, each containing at least k nets.

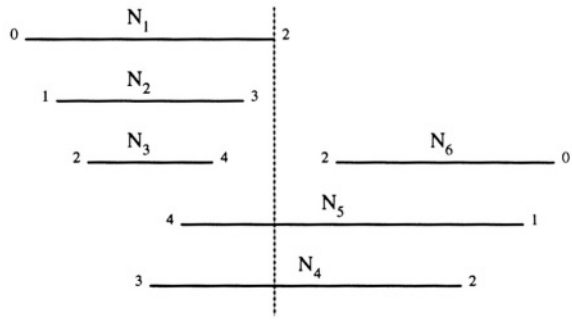


Figure 9.23: The concept of cut numbers.

Although this condition can be used to verify whether a given realization is optimal, it does not lead to any optimal routing algorithm. In fact there is strong evidence that no such algorithm may exist. Arnold [Arn82] proved that the problem of finding a layout with minimum congestion is NP-Hard.

Based on the concept of cut-numbers, a lower bound is also presented in [KKF79].

Theorem 9 *For any feasible realization* $Q_0 \geq \max\{q_{\min}, \lceil \frac{q_{\max}}{2} \rceil\}$.

In [TMSK84] Trang, Marek-Sadowska, and Kuh proposed a heuristic algorithm for minimizing street congestion. The proposed algorithm is based on permuting the nets according to their cut number. It is observed that nets having larger cut numbers should be placed inside, i.e., near the middle of the permutation, while nets having lower cut numbers should be placed outside, i.e., at the ends of the permutation. In [DL87b] Du and Liu showed that the algorithm in [TMSK84] does indeed produce optimal results if all the nets belong to one 'group'. However, if the net list has more than one group than the algorithm given in [TMSK84] may not produce optimal results. The set of nets that covers at least one common node is said to form a *group.* Du and Liu proposed an algorithm that takes the group structure into account. They use the idea of *local cut number* that is, the cut number of a net with respect to a group. The algorithm routes the largest group first and then tries to route the nets in the adjacent group while trying to satisfy the heuristic criterion of placing nets with larger local cut numbers inside and nets with smaller local cut numbers outside. This algorithm produced better results than the one reported in [TMSK84].

In [DIN87] Du et al. investigated the problem of minimizing the between-node congestion which is the congestion between two adjacent nodes. They developed a fast algorithm for the case when the number of horizontal tracks available as well as the number of vertical tracks available between adjacent nodes is fixed. Their algorithm is an extension of Han and Sahni's algorithm.

9.3.2.2 A Graph Theoretic Approach

In [SD89a, SD89b, SDR89, SDR90], Sherwani and Deogun developed a new graph-theoretic approach to single row routing problems. This approach models a single row routing problem with three graphs, an overlap graph, a containment graph and an interval graph. It was found that several relationships exist between the properties of an SRRP and the graph representation. In [SD89a], a new heuristic algorithm has been developed based on this approach. This algorithm achieves substantially better results than the existing algorithms. In [SD89b], new lower bounds for SRRP have been developed and in [SDR89] the problem of single row routing with a restricted number of doglegs is investigated.

We will briefly discuss the principle results obtained by the graph theoretic approach. Let R be a set of evenly spaced terminals on the node axis. Let $\mathcal{N} = \{N_1, N_2, \ldots, N_n\}$ be a set of two-terminal nets defined on R. Each net N_i can be uniquely specified by two distinct terminals l_i and r_i called the left touch point and the right touch point, respectively, of N_i. Abstractly, a net can be considered as an interval bounded by left and right touch points. Thus for a given set of nets, an interval diagram depicting each net as an interval can be easily constructed. Given an interval diagram corresponding to an SRRP, we can define an interval graph G_{I} containment graph G_{C} and overlap graph G_O. The definitions of these graphs may be found in chapter 3.

The approach presented in [SD89a] uses modified cut numbers. The cut number is a very important criterion in determining the position of a net in the final routing. Usually only cliques are considered in computation of cut-numbers. For improving the utility of cut-numbers, they consider not only cliques but also the clique intersections. Two cliques are said to have *high clique intersection* if the number of nets they have in common is at least equal to one-half of the maximum size of the two cliques, otherwise cliques have low clique intersection. If the clique intersection is relatively high between two cliques, these cliques are collapsed to form a bigger *pseudo-clique.* For routing purposes a pseudo-clique is treated as a clique. This operations of clique collapsing continues until all clique intersections are relatively low. The cut-number of a net with respect to its pseudo-clique is called the *modified cut-number.* It is easy to see that this approach behaves like Trang et al.'s algorithm if all clique intersections are high while behaving like Du et al.'s algorithm if all clique intersections are low. In addition, it even produces a good solution for problem sets for which some clique intersections are high and some clique intersections are low.

9.3.2.3 Algorithm for Street Congestion Minimization

In [SD89a], an algorithm to minimize the street congestion was developed. The basic function of the algorithm, denoted as SRRP_ROUTE, is to find maximal pseudo-cliques in the interval graph G_{I} representing the sub-problem under consideration. This goal is accomplished by finding and collapsing maximal cliques. If the clique intersection of the two neighboring (pseudo-)cliques is

```
Algorithm SRRP-ROUTE ( )
begin
    FIND-CLIQUES (L, C)
    COMBINE-CLIQUES (L, C, D)
        (* D contains super-cliques SC_j, j = 1, ..., r *)
    MAX-PSEUDO-CLIQUE (D, SC_k)
    SOLVE (SC_k, M)
    for j = 1 to k - 1
        INSERT (SC_j)
    for j = k + 1 to r
        INSERT (SC_j)
end.
```

Figure 9.24: Algorithm SRRP-ROUTE

high then these are combined and a pseudo-clique is formed. All modified cut-numbers are computed according to the pseudo-cliques. First, the maximum pseudo-clique SC_{ik} is routed using a greedy approach similar to the approach used in [TMSK84]. Other pseudo-cliques to the left and right of this maximum pseudo-clique are then routed. An outline of the algorithm is in Figure 9.24.

In the following, we give a brief description of the main procedures of SRRP-ROUTE:

Procedure FIND-CLIQUES: This procedure decomposes the given problem into several smaller single-row-routing problems by identifying the linear ordering of cliques $C_i, 1 \leq i < n$ of the interval graph G_{I}.

Procedure COMBINE-CLIQUES: This procedure finds the clique intersections between adjacent cliques, and forms a pseudo-clique if the clique intersection is high. This process is carried out until all clique intersections between all pseudo cliques are low. The clique collapsing parameter can be changed.

Procedure SOLVE: This procedure returns a permutation of the nets of a sub-problem obtained by placing them according to the greedy heuristic based on the modified cut numbers. This procedure is used to route the maximum pseudo-clique.

Procedure INSERT: This procedure combines solutions of two adjacent sub-problems to produce a solution for the larger problem defined by the combination of the two sub-problems. It inserts the new nets belonging to the new clique into the existing solution so that nets with higher modified cut numbers are assigned to inner tracks, while nets lower modified cut numbers are assigned to outer tracks.

The number of nets in any sub-problem cannot be greater than n. Moreover, procedure SOLVE has a time complexity of $O(n \log n)$. Procedure INSERT has a time complexity of $O(n)$. The first loop thus takes $O(n^2 \log n)$ time. Similarly, the second loop also takes $O(n^2 \log n)$. Therefore, the worst case time complexity of the algorithm is $O(n^2 \log n)$.

9.3.2.4 Algorithm for Minimizing Doglegs

The problem of finding a layout without doglegs is of interest because of the limited amount of inter-pin distance available in IC's. This problem has been considered before by Raghavan et al., [RS84] when an algorithm for checking feasibility of routing without doglegs was developed. The authors however did not present a characterization. Using the graph model, in [SD89a] a characterization of single row routing problems which can be solved without doglegs is presented.

Theorem 10 *An SRRP can be routed without any doglegs if and only if the corresponding overlap graph is bipartite.*

Similarly, a sufficient condition for routing with at most one dogleg per net was also established.

Theorem 11 *An SRRP can be realized with at most one dogleg per net if the corresponding containment graph G_C is null.*

Using this graph representation, three algorithms for minimum-bend single row routing problem have recently been reported [SWS92]. It was shown that the proposed algorithms have very tight performance bounds. In particular, it is proved that the maximum number of doglegs per net is bounded by *O(k),* where k is the size of the maximum clique in certain graph representing the problem. Expected value of k is $\Theta(\sqrt{n})$ and in practical examples k = 0(1), where n is the number of nets.

We will briefly describe one of the algorithms, which is based on the decomposition of the given SRRP into several smaller SRRPs so that interval graph for each subproblem is null. This operation is called *independent set decomposition* of G_I. The motivation for this algorithm is derived from the fact that using the interval graph representing a SRRP, the problem can be decomposed into k subproblems and each one of these subproblems can be routed without any doglegs. The key therefore, is to combine the routing of these subproblems such that maximum number of doglegs per net is minimized. The independent set decomposition of G_I can be achieved by using the algorithm to find maximum clique in an interval graph described in Chapter 3. Using the k independent sets, an algorithm, denoted K-DOGLEG-I is presented, which combines the routing of these sets into a routing for the given SRRP. The formal description of the algorithm is in Figure 9.25.

Theorem 12 *The Algorithm K-DOGLEG-I routes a given net list L with at most $O(k)$ doglegs per net in $O(n \log n)$ time, where $k = C_I$ and n is the total number of nets.*

```
Algorithm K-DOGLEG-I()
begin
    Phase 1:
        (* Use Left_edge algorithm to decompose 𝒩 into
        k independent net lists *)
        (* lists (𝒩1, 𝒩2,..., 𝒩k) of 𝒩. *)
        𝒩i=LEDGE(𝒩); (i = 1, .., k).
        (* Assign 𝒩1 to the upper street. *)
        for ( Ni ∈ 𝒩1  i = 1, .., m1 ) do
            T1^U = T1^U ⋃ Ni;
        (* Assign 𝒩2 to the lower street. *)
        for ( Ni ∈ 𝒩2  i = 1, .., m2 ) do
            T1^B = T1^B ⋃ Ni;
    Phase 2:
        (* Insert the remaining independent net lists.*)
        t = U; u = 1; l = 1;
        for (G_I^i  i = 3, .., k) do
            for ( Nj ∈ 𝒩i(j = 1, .., mi) ) do
                (* Find the smallest track which contains Nj. *)
                k = min{q|1 ≤ q ≤ p, Nj ∈ Tq^t};
                if (Nj contained by previously routed net at Tk^t)
                                then
                    (* Insert Nj under Tk^t. *)
                    INSERT(Nj, Tk^t);
                else
                    (* Assign the new net to the outer track. *)
                    Tp^t = Tp^t ⋃ Nj;
                (* Switch street. *)
                if ( t = U ) then
                    t = B; l = l + 1; p = l;
                else
                    t = U; u = u + 1; p = u;
end.
```

Figure 9.25: Algorithm K-DOGLEG-I

Proof: Given a net list L, the algorithm decomposes it into k independent net lists $L_i, i = 1, \ldots, k$, and routes the first independent net list L_1 on the upper street and the second independent set L_2 on the lower street. This operation can be completed without any doglegs. Then the algorithm inserts all the nets in the remaining $k - 2$ independent net lists into the existing layout. Since inserting one independent net list causes at most $O(1)$ more doglegs to each net in the layout, hence inserting all the remaining $k - 2$ independent net lists causes at most $O(k)$ more doglegs to each net in the layout. So the total dogleg number per net is $O(k)$. On the other hand, in an interval graph, k is equal to C_I. Therefore, the algorithm K-DOGLEG-I can route a given net list with at most $O(C_I)$ doglegs per net.

All operations of the K-DOGLEG-I algorithm, except the track finding operation can be carried out in constant time. Each find operation can be accomplished by a binary search in $O(\log n)$ time. Therefore the total time complexity is $O(n \log n)$. □

9.4 Two-Layer Channel Routing Algorithms

Two-layer channel routing differs from single-layer routing in that two planar set of nets can be routed if vias are not allowed, and a non-planar set of nets can be routed if vias are allowed. For this reason, checking for routability is unnecessary, as all channel routing problems can be completed in two layers of routing if vias are allowed. Therefore, the key objective function is to minimize the height of the channel.

For a given grid-based channel routing problem, any solution to the problem requires at least a minimum number of tracks. This requirement is called the *lower bound* for that problem. Since the lower bound is the minimum number of tracks that is required, it is unnecessary to reduce the number of tracks beyond the lower bound and therefore, it is important to calculate the lower bound of the number of tracks before solving a particular routing instance. Following theorem presents the lower bounds for channel routing problems assuming two-layer reserved layer routing models with no doglegs allowed. Let $h_{\max}$ and $v_{\max}$ represent the maximum clique in the HCG and the longest path in VCG, respectively for a routing instance.

Theorem 13 *The lower bound on the number of tracks of a two-layer dogleg free routing problem is* $\max\{h_{\max}, v_{\max}\}$.

For grid-less channel routing problems, the width of nets must be taken into account while computing v_{max} and h_{max}.

9.4.1 Classification of Two-Layer Algorithms

One method of classifying two-layer channel routing algorithms would be to classify them based on the approach the algorithms use. Based on this classification scheme we have:

1. LEA based algorithms: LEA based algorithms start with sorting the trunks from left to right and assign the segments to a track so that no two segments overlap.

2. Constraint Graph based routing algorithms: The constraint based routing algorithms use the graph theoretic approach to solve the channel routing problem. The horizontal and vertical constraints are represented by graphs. The algorithms then apply different techniques on these graphs to generate the routing in the channel.

3. Greedy routing algorithm: The greedy routing algorithm uses a greedy strategy to route the nets in the channel. It starts with the leftmost column and works towards the right end of the channel by routing the nets one column at a time.

4. Hierarchical routing algorithm: The hierarchical router generates the routing in the channel by repeatedly bisecting the routing region and then routing each net within the smaller routing regions to generate the complete routing.

In the following subsection, we present a few routers from each category.

9.4.2 LEA based Algorithms

The Left-Edge algorithm (LEA), proposed by Hashimoto and Stevens [HS71], was the first algorithm developed for channel routing. The algorithm was initially designed to route array-based two-layer PCBs. The chips are placed in rows and the areas between the rows and underneath the boards are divided into rectangular channels. The basic LEA has been extended in many different directions. In this section, we present the basic LEA and some of its important variants.

9.4.2.1 Basic Left-Edge Algorithm

The basic LEA uses a reserved layer model and is applicable to channel routing problems which do not allow doglegs and any vertical constraints. Consequently, it does not allow cyclic vertical constraints.

The left-edge algorithm sorts the intervals, formed by the trunks of the nets, in ascending order, relative to the x coordinate of the left end points of intervals. It then allocates a track to each of the intervals, considering them one at a time (following their sorted order) using a greedy method. To allocate an interval to a track, LEA scans through the tracks from the top to the bottom and assigns the net to the first track that can accommodate the net. The allocation process is restricted to one layer since the other layer is used for the vertical segments (branches) of the nets. The detailed description of LEA is in Figure 9.26. Figure 9.27 shows a routing produced by LEA. Net N_1 is assigned to track 1. Net N_2 is assigned to track 2 since it intersects with N_1 and cannot be assigned to track 1. Net N_3 is similarly assigned to track 3. Net N_4 is

```
Algorithm LEFT-EDGE (N, I)
begin
    FORM-INTERVAL(N, I);
    FORM-HCG(I,HCG);
    d = DENSITY(HCG);
    let T = {T1T2, ..., Td} denote the set of routing
        tracks from top to bottom;
    SORT-INTERVAL(I);
    for i = 1 to n do
        for j = 1 to d do
            if DOES-NOT-OVERLAP(Ii, Tj) then
                assign interval Ii to Tj;
    for i = 1 to n do
        (* connect the vertical segments of net Ni to its *)
        (* horizontal segment *)
        VERTICAL-SEGMENT(left(Ii), left(Ni));
        VERTICAL-SEGMENT(right(Ii), right(Ni));
end.
```

Figure 9.26: Algorithm LEFT-EDGE

assigned to track 1 since it does not intersect with N_1. The following theorem which establishes the optimality of LEA is easy to prove.

Theorem 14 *Given a two-layer channel routing problem with no vertical constraints, LEA produces a routing solution with minimum number of tracks.*

The input to the algorithm is a set of two-terminal nets $\mathcal{N} = \{N_1, N_2, \ldots, N_n\}$. Procedure FORM-INTERVAL forms interval set $\mathcal{I} = \{I_1, I_2, \ldots, I_n\}$ from N. Once the intervals are formed, FORM-HCG forms the horizontal constraint graph HCG from $\mathcal{I}$. Note that the HCG is a interval graph corresponding to interval set $\mathcal{I}$. Procedure DENSITY computes the maximum clique size in HCG. This maximum clique size is a lower bound on the given channel routing problem instance. SORT-INTERVAL sorts the intervals in $\mathcal{I}$ in the ascending order of their x-coordinate on their left edge. Procedure VERTICAL-SEGMENT connects the vertical segments with the corresponding horizontal segment. The time complexity of this algorithm is $O(n \log n)$, which is the time needed for sorting n intervals.

The assumption that no two nets share a common end point is too restrictive, and as a result LEA is not a practical router for most channel routing problems. The restrictions placed on the router in order to achieve optimal results are not practical for most channel routing problems. However, LEA can be used to route PCB routing problems with vertical constraints since there is sufficient space between the adjacent pins to create a *jog*. LEA is also useful as a initial router for routing of channels with vertical constraints. The basic

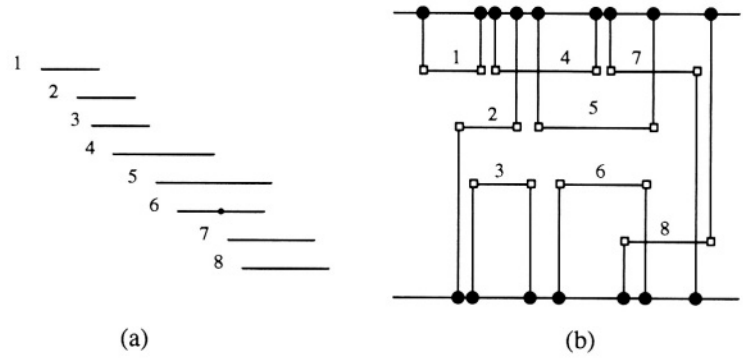


Figure 9.27: Left-edge channel routing.

idea is to create a layout with design rule violations and then use clean up procedures to remove the violations.

9.4.2.2 Dogleg Router

One of the drawbacks of LEA is that it places an entire net on a single track. It has been observed that this leads to routings with more tracks than necessary. Consider Figure 9.28(a), which shows a simple channel routing problem that has been routed using LEA and uses three tracks. On the other hand, if a dogleg is introduced in net N_2, the same problem can be routed using only two tracks. We recall that a dogleg is a vertical segment that is used to maintain the connectivity of two trunks (subnets) that are on two different tracks. The insertion of doglegs, may not necessarily reduce the channel density. A badly placed dogleg can lead to an increase in channel density. Finding the smallest number and locations of doglegs to minimize the channel density is shown to be NP-complete [Szy85].

Deutsch [Deu76] proposed an algorithm known as *dogleg router* by observing that the use of doglegs can reduce channel density. The dogleg router is that it allows multi-terminal nets and vertical constraints. Multi-terminal nets may have terminals on both sides of the channel and often form long horizontal constraint chains. In addition, there are several critical nets, such as clock nets, which pose problems because of their length and number of terminals. These type of nets can be broken into a series of two-terminal subnets using doglegs and each subnet can be routed on a different track. Like LEA, the Dogleg router uses a reserved layer model. Restricting the doglegs to the terminal positions reduces the number of unnecessary doglegs and consequently reduces the number of vias and the capacitance of the nets. The dogleg router cannot handle cyclic vertical constraints.

The dogleg router introduces two new parameters: *range* and *routing sequence.* Range is used to determine the number of consecutive two-terminal

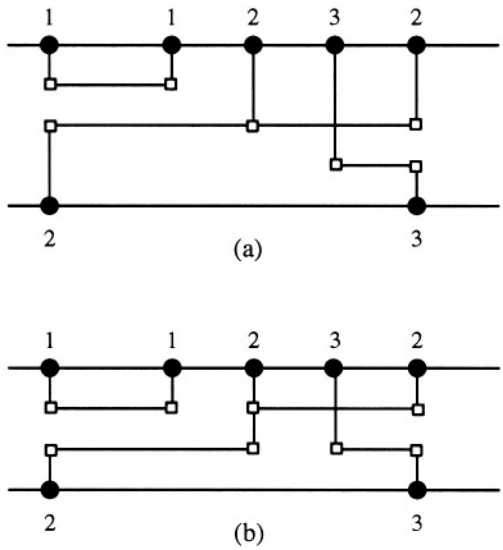


Figure 9.28: Using a dogleg to reduce channel density.

subnets of the same net which can be placed on the same track. Increasing the range parameter will result in fewer doglegs. The routing sequence specifies the starting position and the direction of routing along the channel. Unlike LEA, the routing can start from any end and work towards the opposite end. Different results can be obtained by starting at different corners: top-left, top-right, bottom-left, bottom-right. Furthermore, instead of starting from the top to the bottom or from the bottom to the top, the algorithm can alternate between topmost and bottommost tracks. This scheme results in eight different routing sequences: top-left $\rightarrow$ bottom-left, top-left $\rightarrow$ bottom-right, top-right $\rightarrow$ bottom-left, top-right $\rightarrow$ bottom-right, bottom-left $\rightarrow$ top-left, bottom-left $\rightarrow$ top-right, bottom-right $\rightarrow$ top-left, and bottom-right $\rightarrow$ top-right. (The left side of the arrow indicates the starting corner and the right side of the arrow indicates the alternate corner). Consider the example shown in Figure 9.29(a). If the range is set to 1 and we set the routing sequence to top-left $\rightarrow$ bottom-right, then Figure 9.29(b) shows routing steps in dogleg router. Notice that nets N_2 and N_3 use doglegs.

The complexity of the algorithm is dominated by the complexity of LEA. As a result, the complexity of the algorithm is $O(n \log n + nd)$, where n is the total number of two-terminal-nets after decomposition and d is the total number of tracks used. Note that the parameter's range and routing sequence can be changed to get different solutions of the same routing problem. A large value of range keeps the number of doglegs smaller. If the number of two-terminal subnets of a net is less than the value of a range, then that net is routed without any dogleg. Varying the routing sequence can also lead to a reduced

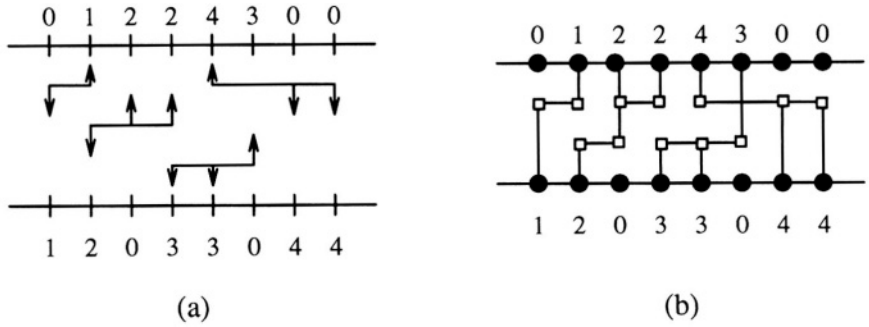


Figure 9.29: Example routed by dogleg router.

channel height. Dogleg router can easily be extended to gridless routing model. Experimentally dogleg routers achieve far superior results as compared to LEA, often requiring very few tracks beyond the channel density.

9.4.2.3 Symbolic Channel Router: YACR2

LEA does not allow vertical constraints thereby making it impractical for most of the channel routing problems. If a given channel is routed using LEA, then vertical constraint violations may be introduced by the router which need to be removed to get a legal routing solution. Note that a vertical constraint violation is a localized problem and may be resolved by anyone of the two methods:

1. local rip-up and reroute
2. localized maze routing.

In the second approach, vacant space surrounding the column in which vertical constraint violation occurred can be used to resolve the violation. Usually several horizontal segments of tracks as well as several vertical columns are not used for routing of any nets. Since the general maze routing technique is very time consuming and vertical constraint violations are local in nature, special maze routing techniques can used to remove vertical constraint violations. In case any vertical constraint violations cannot be resolved, new tracks can be added to resolve the constraints.

Based on these observations, Reed, Sangiovanni-Vincentalli, and Santamauro [RSVS85] proposed YACR2 (Yet Another Channel Router). In order to explain how vertical constraint violations are handled in YACR2 we define the concept of vertical overlap factor, which indicates the total number of tracks that a vertical constraint violation spans. Precisely stated, let us assume that column c_i has a vertical constraint violation between net N_t that has to be connected to the top boundary and net N_b that has to be connected

to the bottom boundary. Also assume that N_t is assigned to track t_p and N_b is assigned to track t_q. Track t_p is above track t_q and tracks are numbered in increasing order from top to bottom boundary. Then vertical constraint $vof(c_i) = (p - q + 1)$. For any column c_j, if there is no vertical constraint violation in c_j, then $vof(c_j) = 0$. The basic idea of YACR2 is to select nets in an order and assign nets to tracks in a way such that $vof(c_i)$ is as minimum as possible for each column c_i. After assigning nets to tracks, specialized maze routing techniques are used to resolve the violations. If a vertical constraint violation cannot be resolved using maze routing technique, additional tracks are used to complete the routing.

The algorithm works in four different phases. First three phases are essentially for assigning nets to tracks with the objective of minimizing $vof(c_i)$ for each column c_i. In attempt to minimize $vof(c_i)$, the algorithm starts with the nets belonging to the maximum density column. After assigning tracks to the nets belonging to the maximum density column, it uses LEA to assign tracks to nets that are to the right of the maximum density column and then assigns to the nets that are to the left of the maximum density column. A modified LEA is used to assign tracks that are to the left of the maximum density column. It can be thought of as a right-edge algorithm, since it works from right to left.

As mentioned earlier that the goal of selecting and assigning nets to tracks is to minimize the total number of vertical constraint violations so that it is easy for the simplified maze routers to complete the routing. However, it is impossible to determine all the vertical constraint violations caused by the placement of a certain net. In fact some of the vertical constraint violations may occur between the net under consideration and nets yet to be routed. Since the nets are routed without doglegs, the vertical constraint graph can be used to estimate the possibility of an assignment giving rise to a violation, and the difficulty involved in removing the violation if it occurs. The techniques of selecting and assigning nets used in [RSVS85] are rather complicated and readers are referred to [RSVS85] for the details. It should be noted that any technique will work; however, vof may be very high for some column making vertical constraint violation resolution steps rather complicated.

After track assignments of horizontal segments, at the end of phase III, has been achieved, appropriate vertical segments are placed in the columns with $vof = 0$. In phase IV, the columns with vertical constraint violations are examined one at a time to search for legal connection between the nets and their terminals. Instead of applying the general purpose maze routing technique, three different maze routing techniques are applied to resolve the vertical constraint violations in this phase. These three techniques (strategies) are called maze1, maze2, and maze3. At each column with a vertical constraint violation, mazel strategy is used first. If mazel fails to resolve the violation, maze2 is applied. If maze2 fails, then maze3 is applied to resolve the violation. In case all three strategies fail, the channel is enlarged by adding one track and the process is repeated.

To explain how the maze routing techniques work, let us assume that column c_i has a vertical constraint violation between net N_t that has to be connected to

the top boundary and net N_b that has to be connected to the bottom boundary. Also assume that N_b is assigned to track t_p and N_t is assigned to track t_q. Tack t_p is above track t_q.

The Mazel technique checks for either one of the following:

1. No vertical segments exist between t_{p-1} and t_q on column c_{i-1} or c_{i+1}.

   In this case a jog is used in net N_t in track t_{p-1} to resolve the violation (see Figure 9.30(a) and (b));

2. No vertical segment exist between t_p and t_{q+1} on column c_{i-1} or c_{i+1}.

   In this case a jog is used in net N_b in track t_{q+1} to resolve the violation (see Figure 9.30(c) and (d));

3. No vertical segment exist between t_{p-1} and some t_s, between t_p and t_q, on column c_{i-1} and between t_{s-1} and t_{q+1} on column c_{i+1}, or vice versa.

   In this case net N_t uses jogs in tracks t_{p-1} and t_s and net N_b uses jogs in tracks t_{s-1} and t_{q+1} to resolve vertical constraint violation (see Figure 9.30(e) and (f)).

In case mazel technique cannot resolve the vertical constraint violation, maze2 technique is used in attempt to resolve the violation. Maze2 checks for one of the following:

1. A track, column pair (t_r, c_j) such that: (a) there are no horizontal segments in track t_r between columns c_i and c_j; (b) there are no vertical segments in c_j between t_r and t_q; (c) the horizontal segment of net N_t in track t_q either crosses column c_j or can be extended to c_j without causing a horizontal constraint violation; and (d) t_r is above t_p.

   In this case, net N_t uses a dogleg in track t_r to resolve the violation as shown in Figure 9.31 (a) and (b).

2. A track, column pair (t_r, c_j) such that: (a) there are no horizontal segments in track t_r between columns c_i and c_j; (b) there are no vertical segments in c_j between t_r and t_p; (c) the horizontal segment of net N_b in track t_p either crosses column c_j or can be extended to c_j without causing a horizontal constraint violation; and (d) t_r is below t_q.

   In this case, net N_b uses a dogleg in track t_r to resolve the violation as shown in Figure 9.31(c) and (d).

If none of the conditions in maze2 techniques are satisfied, maze3 technique is applied. As opposed to the local maze routing, the pattern based approach of YACR2 is efficient and avoids long routes; at the same time, it is limited in scope as opposed to local maze routing techniques. If none of the maze routing techniques can resolve vertical constraint violations, new tracks are added to complete the routing.

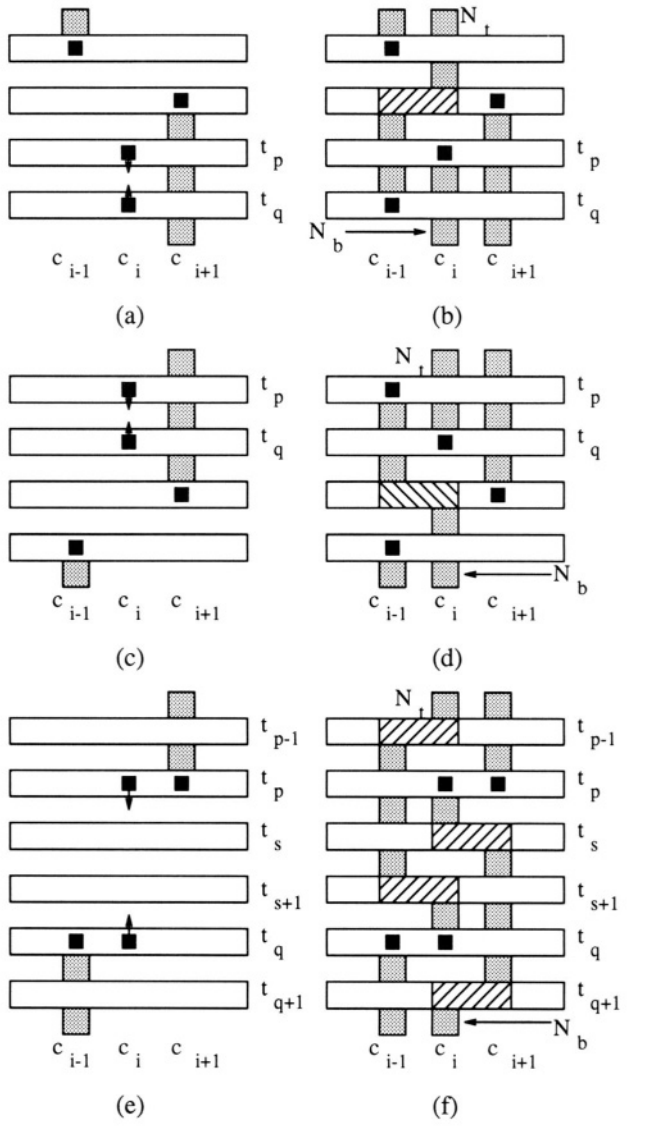


Figure 9.30: Maze1 routing.

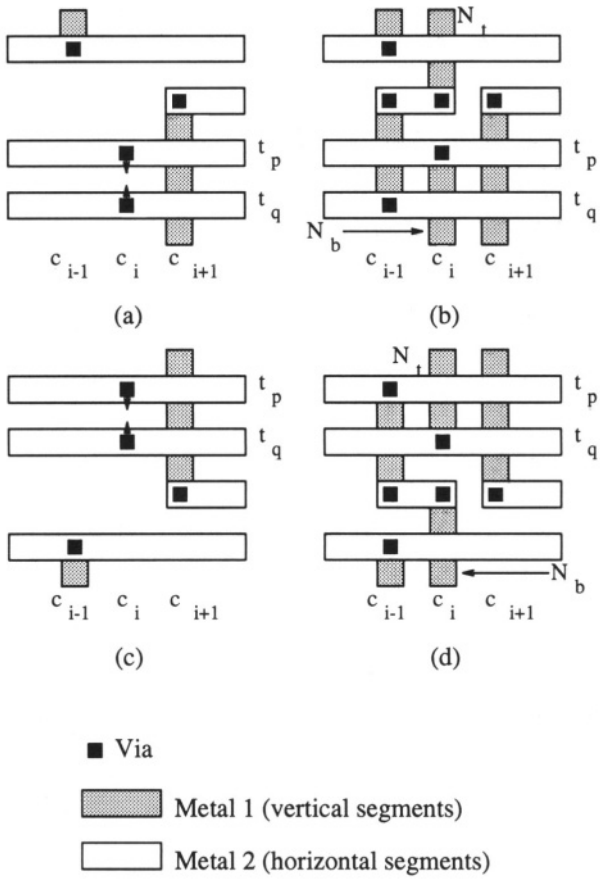


Figure 9.31: Maze2 routing.

9.4.3 Constraint Graph based Routing Algorithms

Consider a channel routing problem with no vertical constraints. Obviously, the number of tracks needed is determined by the maximum clique h_{max} in the horizontal constraint graph(HCG). In this case, LEA produces optimal results, if no doglegs are allowed. In presence of vertical constraints, the length of the longest path v_{max} in vertical constraint graph(VCG) also plays a key role in determining the channel height. In particular, the nets which lie on long paths in the vertical constraint graph, must be carefully assigned to tracks. In order to explain the effect of long vertical chains, let us define length of ancestor and descendent chains of a net N_i. Let v_i represent N_i in VCG. Let A_i denote the length of the longest path from a vertex of zero in-degree to v_i in VCG. Similarly, let D_i denote the length of longest path from v_i to a vertex of zero

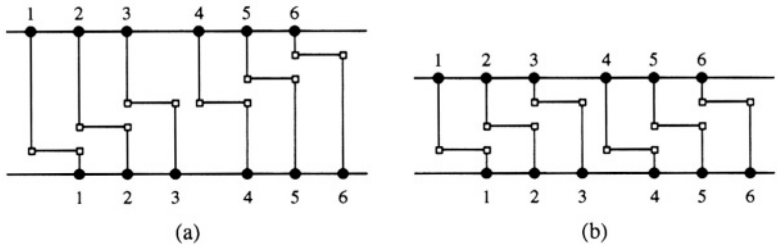


Figure 9.32: Effect of net merging on channel height.

out-degree in VCG. It is easy to see that

$$v_{\max} = \max_{i=1,n}(A_i + D_i) - 1$$

Consider the nets N_3 and N_4 as shown in Figure 9.32. If N_3 and N_4 are assigned to the same track then the channel height is given by

$$ch\_ht \geq \max\{A_3 + A_4, A_3 + D_4, A_4 + D_3, A_4 + D_4\}$$

In other words, if we consider N_3 and N_4 as a new net, then a new vertical constraint chain is created which consists of longer of two ancestor chains and longer of two descendent chains.

$$ch\_ht \geq \max\{\max\{A_3, A_4\} + \max\{D_3, D_4\}\}$$

Figure 9.32(a) shows the effect of assigning two nets to the same track without considering the constraint chains. The channel height for this solution is equal to 6. A better assignment resulting in a channel height of 4 is shown in Figure 9.32(b). Based on the above equation, we make the following observation. In order to minimize the effect of vertical constraint chains on channel height, two nets may be assigned to the same track only if both of them have small ancestor chains, or both of them have small descendent chains. Several algorithms have been developed which are based on this observation. In this section, we discuss the first constraint graph based algorithm and its grid-less variant.

9.4.3.1 Net Merge Channel Router

In 1982, Yoshimura and Kuh [YK82] presented a new channel routing algorithm (YK algorithm) for two-layer channel routing problems based on net merging. This work was the first attempt to analyze the graph theoretic structure of the channel routing problem. YK algorithm considers both the horizontal and vertical constraint graphs and assigns tracks to nets so as to minimize the effect of vertical constraint chains in the vertical constraint graph. It does

not allow doglegs and cannot handle vertical constraint cycles. The YK algorithm partitions the routing channel into a number of regions called *zones* based on the horizontal segments of different nets and their constraints. The basic observation is that a column by column scan of the channel is not necessary as nets within a zone cannot be merged together and must be routed in a separate track. This observation improves the efficiency of the algorithm. The algorithm proceeds from left to right of the channel merges nets from adjacent zones. The nets that are merged are considered as one composite net and are routed on a single track. In each zone, new nets are combined with the nets in the previous zone. After all zones have been considered, the algorithm assigns each composite net to a track. The key steps in the algorithm are zone representation, net merging to minimize the vertical constraint chains, and track assignment. Throughout our discussion, we will use the example given in [YK82], since that example serves as a benchmark.

1. **Zone Representation of Horizontal Segments:** Zones are in fact maximal clique in the interval graph defined by the horizontal segments of the nets. The interval graph of the net list in Figure 9.33(a) is shown in Figure 9.33(e). In terms of an interval graph the clique number is the density of the channel routing problem.

   In order to determine zones, let us define $S(i)$ to be the set of nets whose horizontal segments intersects column *i.* Assign *zones* the sequential number to the columns at which S(i) are maximal. These columns define zone 1, zone 2, etc., as shown in the table 9.33(c), for the example in Figure 9.33. The cardinality of $S(i)$ is called local density and the maximum among all local densities is called maximum density which is the lower bound on the channel density. In should be noted that a channel routing problem is completely characterized by the vertical constraint graph and its zone representation.

2. **Merging of Nets:** Let N_i and N_j be the nets for which the following two conditions are satisfied:

   1. There is no edge between v_i and v_j in HCG.
   2. There is no directed path between v_i and v_j in VCG.

   If these conditions are satisfied, net N_i and net N_j can be merged to form a new composite net.

   The operation of merging net N_i and net N_j modifies the VCG by shrinking node v_i and and node v_j into node $v_{i.j}$, and updates the zone representation by replacing net N_i and net N_j by net $N_{i.j}$ which occupies the consecutive zones including those of net N_i and net N_j.

   Let us consider the example shown in Figure 9.33(a). Net N_6 and net N_9 are merged and the modified VCG along with the zone representation is shown in Figure 9.34. The updated vertical constraint graph and the zone representation correspond to the net list in Figure 9.34, where N_6

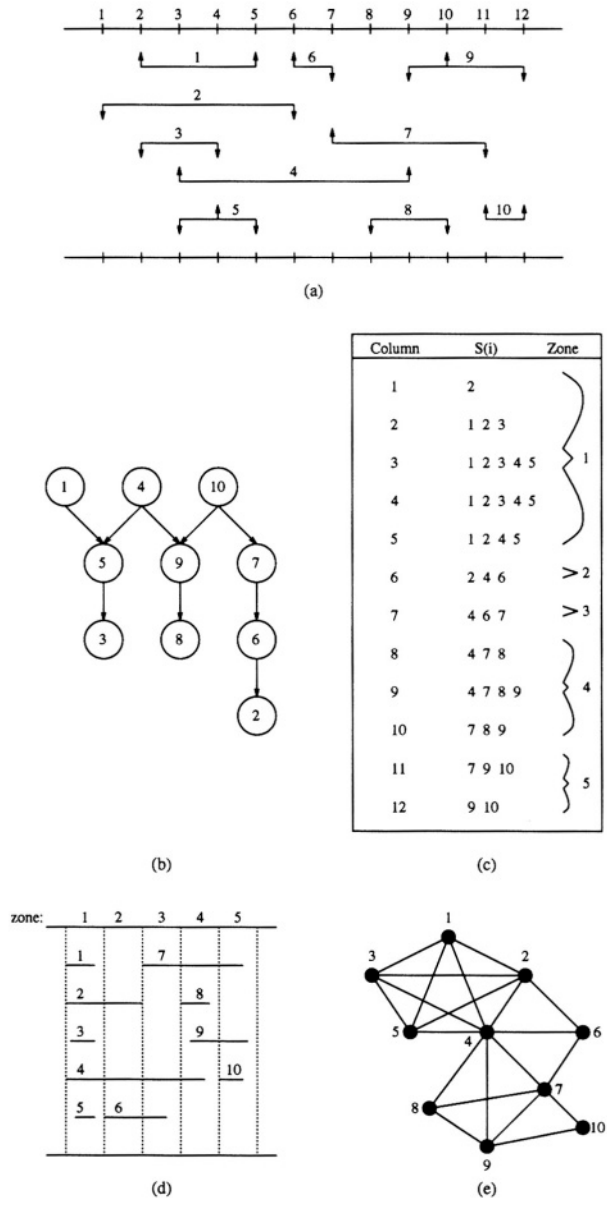


Figure 9.33: Example of zone representations.

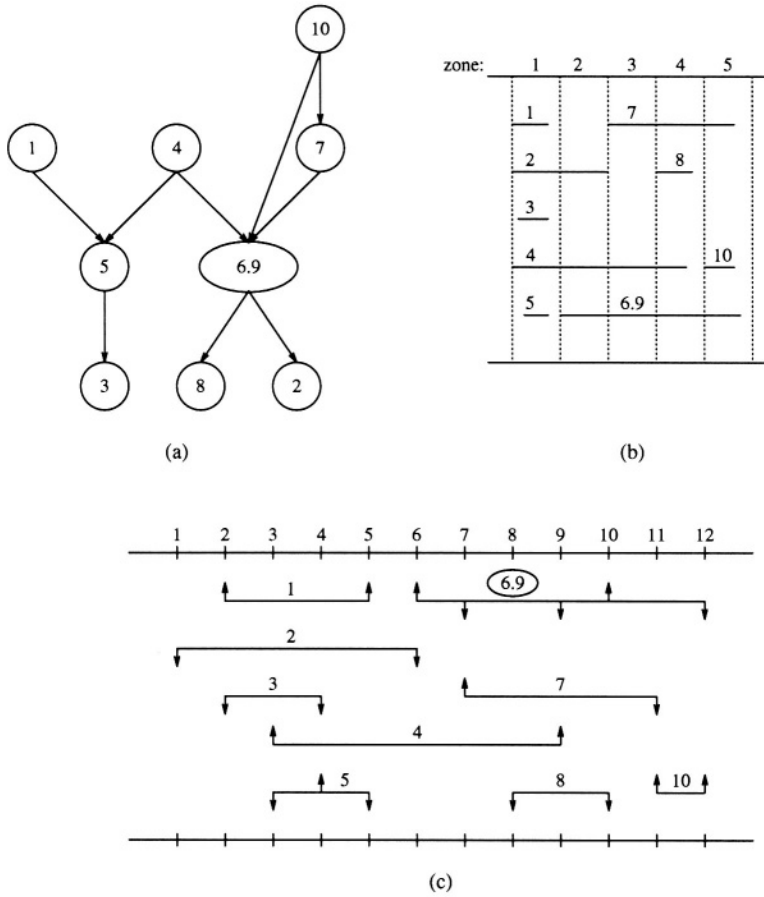


Figure 9.34: Example of merging of nets.

and N_9 are replaced by net $N_{6.9}$. The algorithm, given in Figure 9.35 merges nets as long as two nets from different zones can be merged.

In each iteration, the nets ending in zone z_i are added to the list L. While the nets starting in z_{i+1} are kept in list R. Function MERGE then merges two list L and R so as to minimize the increase in the longest path length in VCG. The list L' returned by function MERGE consists of all the nets merged by the function. These nets are not considered further.

In Figure 9.36 we illustrate how the vertical constraint graph is updated by the algorithm NET-MERGE. The length of the longest path in VCG is 4 and size of the maximum clique is 5, therefore any optimal solution takes at least 5 tracks. In first iteration, $L = \{N_1, N_3, N_5\}$ and $R = \{N_6\}$. There are three possible net mergings, $N_{1.6}$, $N_{3.6}$, and $N_{5.6}$. Merging N_1

```
Algorithm NET-MERGE
begin
    L = φ;
    for z = (z_1 to z_{t-1}) do
        L = L + {z_i − (z_i ∩ z_{i+1})} ;
        R = {z_{i+1} − (z_i ∩ z_{i+1})};
        L' =MERGE(L, R);
        L = L − L';
end.
```

Figure 9.35: Algorithm NET-MERGE

and N_6 creates a path of length 5, merging N_3 and N_6 creates a path of length 4, while merging N_5 and N_6 creates a path of length 4. Therefore either $N_{3.6}$ or $N_{5.6}$ may be formed. Let us merge N_5 and N_6. Similarly, in second iteration net N_1 and net N_7 are merged. In the fourth iteration N_{10} and N_4 are merged. The final graph is shown in Figure 9.36(e). The track assignment is straight forward. Each node in the final graph is assigned a separate track. For example, track 1 can be assigned to net $N_{10.4}$. Similarly, tracks 2 and 3 can be assigned to nets $N_{1.7}$ and net $N_{5.6.9}$, respectively. For net N_2 and net $N_{3.8}$, either track 4 or 5 can be assigned.

It should be noted that finding optimal net pairs for merging is a hard problem. This is due to the fact that the future effects of a net merge cannot be determined.

It is possible to improve the YK algorithm by allowing some look ahead or doing rip-up and re-merge operations. In 1982, YK algorithm represented a major step forward in channel routing algorithms. It formulated the problem and provided a basis for future development of three layer and multi-layer algorithms. It has been extended to three layer and gridless environment. These extended routers will be discussed later in the chapter.

9.4.3.2 Glitter: A Gridless Channel Router

All the algorithms presented thus far in the channel routing are grid-based. The main drawback of grid-based algorithms is that it is difficult to route nets with varying wire widths.

Chen and Kuh [CK86] first proposed a gridless variable-width channel router called *Glitter*. Glitter can utilize multiple layer technology and design rules. Terminals can be located at arbitrary positions and can be located on off-grid points. No columns or tracks are used in routing. Only the wire width, spacing, and via size are under consideration are used. Nets are allowed to have different wire widths to satisfy special design needs and improve the performance of the circuits. Glitter is a reserved-layer model routing algorithm.

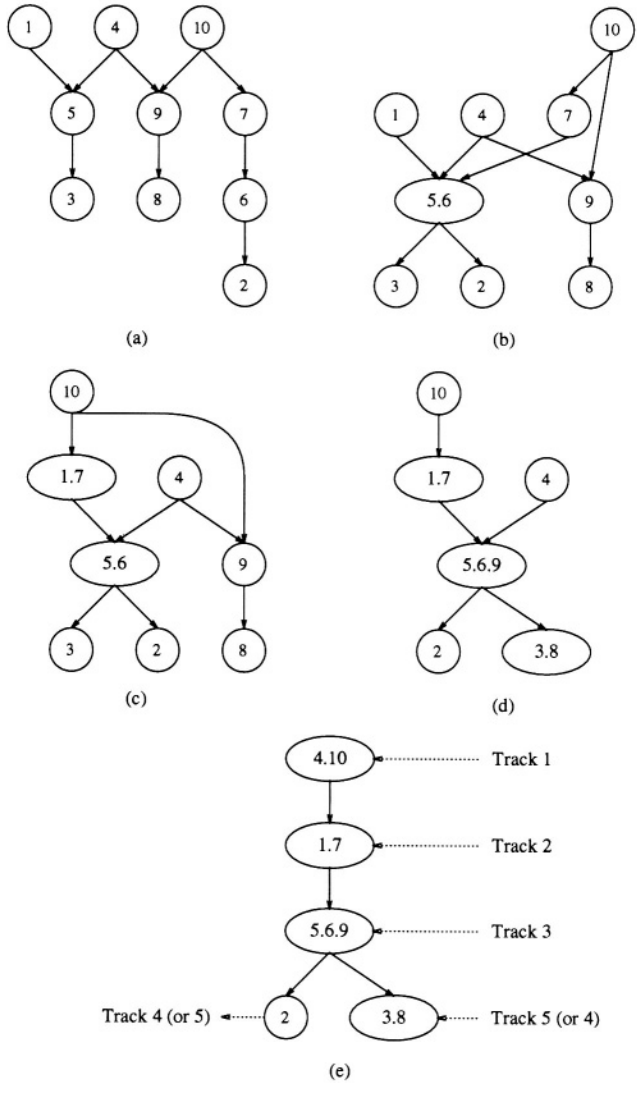


Figure 9.36: Illustration of algorithm NetMerge.

The basic idea of Glitter is somewhat similar to net merge algorithm. Instead of computing the longest vertical constraint chains in terms of tracks, the actual height of the vertical constraint chains is computed and used for assigning nets to locations in the channel.

Another key observation is that if each edge in the combined constraint graph is directed, then they specify a routing. Thus the routing problem is reduced to a problem of assigning directions to edges in the combined constraint graph.

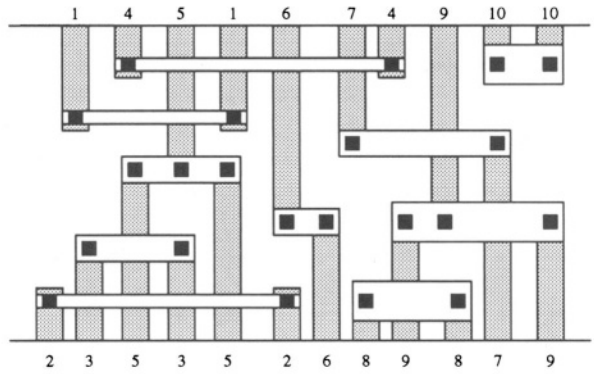
Glitter uses vertical and horizontal constraint graphs to form a graph called *weighted constraint graph.* The weighted constraint graph combines all the vertical and horizontal constraints into the same graph, where each node represents a horizontal net or subnet, each directed edge represents a vertical constraint, and each undirected edge represents a horizontal constraint. The weight of the edge between node A and node B is the minimum vertical distance required between net A and net B. If net A needs to be placed above net B, then the edge should be directed from node A to node B.

To build the weighted constraint graph, vertical and horizontal constraints for each pair of nets (subnets) are checked. If there is more than one constraint, the larger weight will overrule the smaller, and the directed edge will overrule the undirected edge. If there are two contradictory directed edges (there is cycle in VCG), a dogleg must be introduced to break the cycle.

The upper boundary and the lower boundary are also represented by nodes in the weighted constraint graph. Since every net must be placed below the upper boundary, a directed edge will be generated from the upper boundary to each net. Similarly, there is a directed edge from each net to the lower boundary. The weight for a boundary constraint edge is the minimum distance required between the boundary and each net.

Figure 9.37(a) shows a simple example of the variable-width channel-routing problem. In the following analysis, we have assumed the following design rules for the example. The minimum wire spacing in layer 1 and 2 is assumed to be 3 and 2, respectively, the via size is assumed to be 2 x 2, and the minimum overlap width that each layer must extend beyond the outer boundary of the via is assumed to be 1. If the vertical wire width is 4 for every net and the horizontal wire width for each net is specified in Figure 9.37, then we can check the vertical and horizontal constraints for each pair of nets (subnets), and calculate the minimum distance required between them. For example, net 6 must be placed above net 2 by a minimum distance of 7, so there is a directed edge from node 6 to node 2 and the weight of this edge is 7. On the other hand, net 1 should be placed either above or below net 4 because their horizontal spans overlap each other. So there is an undirected edge between node 1 and node 4, and the minimum distance (edge weight) required is 6. The complete weighted constraint graph is shown in Figure 9.37(c).

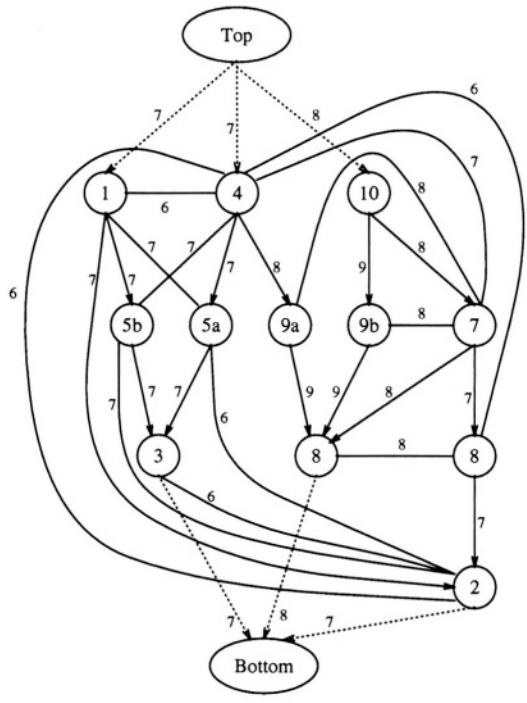
After the weighted constraint graph is generated, the channel-routing problem can be formulated as follows. Given a weighted constraint graph, assign a direction to each undirected edge such that 1) no cycles are generated and 2) the total weight of the maximum weighted directed path (longest path) from



(a)

net	1	2	3	4	5	6	7	8	9	10
width	2	2	4	2	4	4	4	6	6	6

(b)



(c)

Figure 9.37: Variable-width channel routing problem.

the upper boundary to the lower boundary is minimized. In a graph which has m undirected edges, there are 2^m possible solutions.

Since the routing solution can always be obtained by assigning directions to undirected edges, the ordering of edge selection becomes very important. For each undirected edge in the weighted constraint graph, suppose the assignment of one direction will result in cycles. The only choice then is to assign the other direction. Those edges are called critical edges, and they should be assigned directions first.

The ancestor weight $ancw(i)$ of node i is the total weight of the maximum weighted ancestor chain of node i. Similarly, the descendant weight $desw(i)$ of node i is the total weight of the maximum weighted descendant chain of node i.

The label of each undirected edge(i,j) is defined as the maximum of $ancw(i) + weight(i,j) + desw(j)$ and $ancw(j) + weight(i,j) + desw(i)$. The label is a measure of the total weight of the maximum weighted directed path which passes through edge(i,j) if an improper direction is assigned to edge(i,j). The label is defined as ∞ if the undirected edge is a critical edge. If a label is larger than the maximum density of the channel, the meaning of this label becomes significant because it may be the new lower bound for minimum channel height. Undirected edge with a large label should therefore be assigned a proper direction as early as possible, so that the increase of the lower bound is less likely to occur.

Selection of nodes in the graph is based on the ancestor and descendent weights. If an unprocessed node has no ancestors (or all its ancestors have been processed), it can be placed close to the upper boundary (i.e., closer than other unprocessed nodes), and all the undirected edges connected to this node can be assigned outgoing directions. Similarly, if an unprocessed node does not have descendants (or all its descendants have been processed), it should be placed close to the lower boundary, and all the undirected edges connected to it can be assigned incoming directions. The unprocessed nodes with minimum $ancw(i)$ or $desw(i)$ are the candidates to be selected. A node is said to be *processed* if it has been placed close to the upper or lower boundary. An edge is said to be processed if it has been assigned a direction. The routing is complete when all the nodes (or edges) are processed.

The algorithm can be easily extended to accommodate irregularly-shaped channels by adding the boundary information to the weighted constraint graph. The Top node will represent the uppermost boundary, and the Bottom node will represent the lowermost boundary. The weight of boundary constraint edges should be modified to include the amount of indentation.

9.4.4 Greedy Channel Router

Assigning the complete trunk of a net or a two-terminal net segment of a multiterminal net severely restricts LEA and dogleg routers. Optimal channel routing can be obtained if for each column, it can be guaranteed that there is only one horizontal track per net. Based on this observation, one approach to

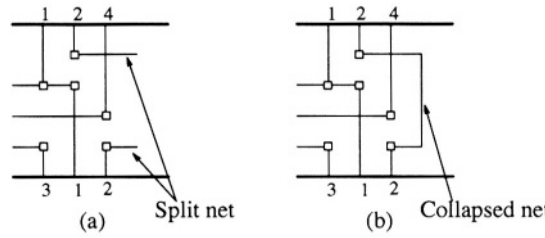


Figure 9.38: (a) A split net. (b) The collapsed split net.

reduce channel height could be to route nets column by column trying to join split horizontal tracks (if any) that belong to the same net as much as possible.

Based on the above observation and approach, Rivest and Fiduccia [RF82] developed *greedy channel router.* This makes fewer assumptions than LEA and dogleg router. The algorithm starts from the leftmost column and places all the net segments of a column before proceeding to the next right column. In each column, the router assigns net segments to tracks in a greedy manner. However, unlike the dogleg router, the greedy router allows the doglegs in any column of the channel, not necessarily where the terminal of the doglegged net occurs.

Given a channel routing problem with m columns, the algorithm uses several different steps while routing a column. In the first step, the algorithm connects any terminal to the trunk segment of the corresponding net. This connection is completed by using the first empty track, or the track that already contains the net. In other words, minimum vertical segment is used to connect a trunk to terminal. For example, net 1 in Figure 9.38(a) in column 3 is connected to the same net. The second step attempts to collapse any *split nets* (horizontal segments of the same net present on two different tracks) using a vertical segment as shown in Figure 9.38(b) A split net will occur when two terminals of the same net are located on different sides of the channel and cannot be immediately connected because of existing vertical constraints. This step also brings a terminal connection to the correct track if it has stopped on an earlier track. If there are two overlapping sets of split nets, the second step will only be able to collapse one of them.

In the third step, the algorithm tries to reduce the range or the distance between two tracks of the same net. This reduction is accomplished by using a dogleg as shown in Figure 9.39(a) and (b). The fourth step attempts to move the nets closer to the boundary which contains the next terminal of that net. If the next terminal of a net being considered is on the top(bottom) boundary of the channel then the algorithm tries to move the net to the upper(lower) track. In case there is no track available, the algorithm adds extra tracks and the terminal is connected to this new track. After all five steps have been completed, the trunks of each net are extended to the next column and the steps are repeated. The detailed description of the greedy channel routing

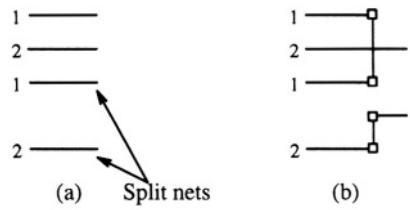


Figure 9.39: (a) Reducing the distance between split nets.

algorithm is in Figure 9.40.

The greedy router sometimes gives solutions which contain an excessive number of vias and doglegs. It has, however, the capability of providing solution even in presence of cyclic vertical constraints. The greedy router is more flexible in the placement of doglegs due to fewer assumptions about the topology of the connections. An example routed by the greedy channel router is shown in Figure 9.41.

9.4.5 Hierarchical Channel Router

In [BP83], Burstein and Pelavin presented a two layer channel router based on the reduction of the routing problem in $(m \times n)$ grid to the routing in $(2 \times n)$ grid and consistent utilization of 'divide and conquer' approach.

Let, $C(i,j)$ denote the cell in ith row and jth column in an $(m \times n)$ grid G. The terminals are assumed to be in the top and the bottom rows of the grid. Let, $h(i,j)$ denote the horizontal boundary shared by the cell $C(i,j)$ and $C(i+1,j)$. Let, $v(i,j)$ denote the vertical boundary shared by the cell $C(i,j)$ and $C(i,j+1)$. Each boundary in G has a capacity, which indicates the number of wires that can pass through that boundary.

In this approach, a large routing area is divided into two rows of routing tiles. The nets are routed globally in these rows using special types of steiner trees. The routing in each row is then further refined by recursively dividing and routing each of the rows. More specifically, the the $(m \times n)$ routing grid is partitioned into two subgrids; the top $(\lceil \frac{m}{2} \rceil \times n)$ and the bottom $(\lfloor \frac{m}{2} \rfloor \times n)$ subgrid. Each column in these subgrids is considered as a supercell. As a result, two rows of supercells are obtained, i.e., a $(2 \times n)$ grid is obtained, (see Figure 9.42.) Capacity of each vertical boundary in this grid will be the sum of corresponding boundary capacities in the original grid. The nets are routed one at a time in this $(2 \times n)$ grid, (see Figure 9.43.) Each row of the $(2 \times n)$ is then partitioned into a $(2 \times n)$ grid. The terminal positions for the routing in the new $(2 \times n)$ subproblems is defined by the routing in the previous level of hierarchy (see Figure 9.44). This divide and conquer approach is continued until single cell resolution is reached.

In the following, we discuss the algorithm for routing a net in the $(2 \times n)$ grid.

```
Algorithm GREEDY-CHANNEL-ROUTER (N)
begin
    d = DENSITY(N);
(* calculate the lower bound of channel density *)
    insert d tracks to channel;
    for i = 1 to m do
        T1 = GET-EMPTY-TRACK;
        if T1 = 0 then
            ADD-TRACK(T1);
            ADD-TRACK(T2);
        else
            T2 = GET-EMPTY-TRACK;
            if T2 = 0 then
                ADD-TRACK(T2);
        CONNECT(T_i, T1);
        CONNECT(B_i, T2);
        join split nets as much as possible;
        bring split nets closer by jogging;
        bring nets closer to either top or bottom boundary;
    while split nets exists do
        increase number of column by 1;
        join split nets as much as possible;
end.
```

Figure 9.40: Algorithm GREEDY-CHANNEL-ROUTER

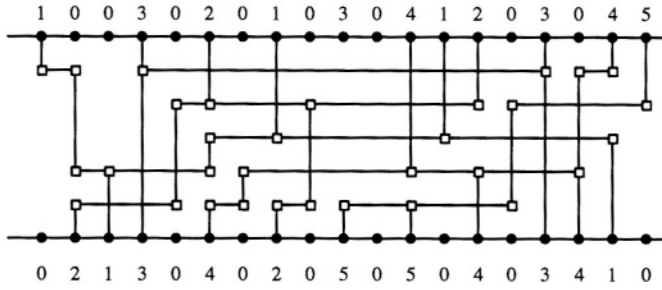


Figure 9.41: Channel routed using a greedy router.

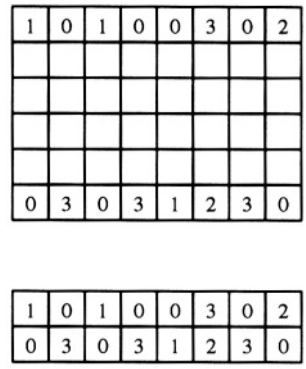


Figure 9.42: Reducing $(m \times n)$ grid to $(2 \times n)$ grid

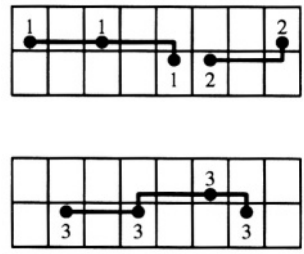


Figure 9.43: First level of hierarchy

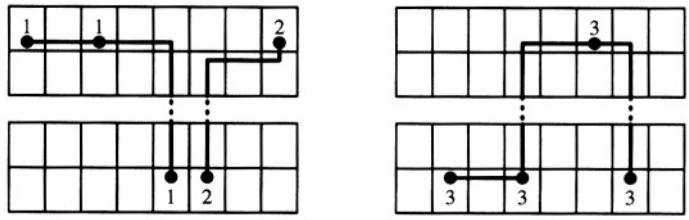


Figure 9.44: Second level of hierarchy

Wiring Within $(2 \times n)$ Grid: Let, p be a $(2 \times n)$ boolean matrix, such that, $p(i,j)$ is *true* if net N_p has a terminal in $C(i,j)$. Let, c_h an array of size n, such that, $c_h(j)$ indicates the cost that must be added to the cost of a net if it crosses the boundary $h(1,j)$. Let, c_v be a $(2 \times (n-1))$ matrix such that $c_v(i,j)$ indicates the cost which must be added to the cost of a net if it crosses the boundary $v(i,j)$. Each element of c_h and c_v is a function of the capacity and utilization of the corresponding boundary; higher the ratio of utilization to the capacity, higher the cost of crossing of that boundary. Such a cost function reduces the probability of utilization of a congested boundary in the routes of the remaining nets. The algorithm to find a minimum cost tree for a net in $(2 \times n)$ grid is a recursive algorithm. This algorithms requires definitions of following terms:

1. $T^1(k)$: It is the minimum cost tree which interconnects the following set of cells:
$$\{C(i,j) : (j \leq k) \& (p(i,j) = true)\} \cup \{C(1,k)\}$$
i.e., it is a minimum cost tree connecting the cells in first k columns that have terminals of N_p and cell $C(1,k)$.

2. $T^2(k)$: It is the minimum cost tree which interconnects the following set of cells:
$$\{C(i,j) : (j \leq k) \& (p(i,j) = true)\} \cup \{C(2,k)\}$$
i.e., it is a minimum cost tree connecting the cells in first k columns that have terminals of N_p and cell $C(1,k)$.

3. $T^3(k)$: It is the minimum cost tree which interconnects the following set of cells:
$$\{C(i,j) : (j \leq k) \& (p(i,j) = true)\} \cup \{C(1,k), C(2,k)\}$$
i.e., it is a minimum cost tree connecting the cells in first k columns that have terminals of N_p, cell $(7(1,k)$ and cell $(7(2,k)$.

4. $T^4(k)$: It denotes a minimum cost forest, containing two different trees $T_1^4(k)$ and $T_2^4(k)$: $T_1^4(k)$ uses cell $(7(1,k)$, $T_2^4(k)$ uses $(7(2,k)$ and the set
$$\{C(i,j) : (j \leq k) \& (p(i,j) = true)\}$$

Let, f and l denote the column number of leftmost and rightmost cells, i.e.,

$$f = \min\{k | p(1,k) \vee p(2,k) = true\},$$

$$l = \max\{k | p(1,k) \vee p(2,k) = true\}.$$

$T^i(k+1), i = 1,2,3,4$ is computed recursively from $T^i(k)$, as discussed below:

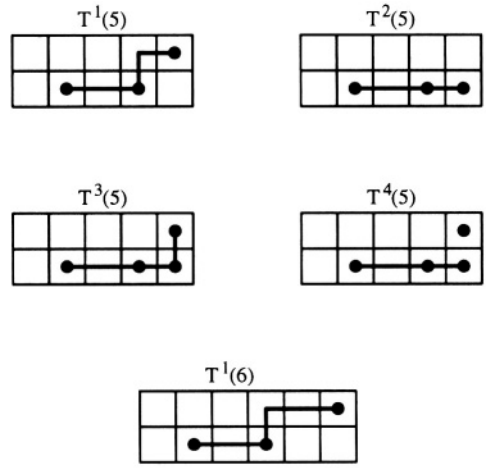


Figure 9.45: Recursively computing $T^1(6)$ from $T^i(5), i = 1, 2, 3, 4$

1. **Basis:** Trees $T^i(k)$ for $k = f$ can be computed trivially and they serve as basis for recursion. For $k \leq f$, $T^1(k)$ and $T^2(k)$ consists of a single vertex $C(1,k)$ and $C(2,k)$ respectively. $T^4(k)$ consists of the disjoint pair of vertices $C(1,k)$ and $C(2,k)$. Whereas, $T^3(1,k)$ consists of a path $C(1,k), \ldots, C(1,s), C(2,s), \ldots, C(2,k)$, where, $1 \leq s \leq k$ and

$$c_h(s) + \sum_{i=s}^{k-1} c_v(1,i) + c_v(2,i) \text{ is minimum.}$$

   Note that if the costs of all horizontal edges are same then $T^3(k)$ is $C(1,k), C(2,k)$.

2. **Recursive Step:** Suppose for $f \leq k \leq l$, $T^i(k), i = 1, 2, 3, 4$ are constructed. In order to construct $T^j(k+1)$, we simply enumerate all possible extensions from $T^i(k), i = 1, 2, 3, 4$ and select the cheapest one. An example in Figure 9.45 shows $T^i(5), i = 1, 2, 3, 4$ and computation of $T^1(6)$ by extending $T^1(5)$ for net 3.

The above algorithm routes a net N_p in *O(nlogm)* time.

In the higher levels of refinement the cost of crossing a boundary is higher as the boundaries are partially utilized in the lower level of hierarchy. As a result, the cost of later routes will be large. If the cost of a route is larger than a user specified value *LRG,* the routes are allowed to detour outside the channel, i.e., vertical tracks are added at the left or right ends of the channel.

Features	Algorithm						
	LEA	Dogleg	Y-K	Greedy	YACR2	Hierar-chical	Glitter
Model	grid-based	grid-based	grid-based	grid-based	grid-based	grid-based	grid-less
Dogleg	not allowed	allowed	allowed	allowed	allowed	allowed	not allowed
Layer assign-ment	reserved	reserved	reserved	reserved	reserved*	reserved	reserved
vertical constra-ints	not allowed	allowed	allowed	allowed	allowed	allowed	allowed
cyclic constra-ints	not allowed	not allowed	not allowed	allowed	allowed	not allowed	allowed

*with some exceptions

Table 9.1: Comparison of different features of two-layer routers.

9.4.6 Comparison of Two-Layer Channel Routers

Extensive research has been done on two layer channel routing. No algorithm is suitable for all problems and applications. Table 9.1 summarizes different features of two-layer routers discussed in this section.

As we have noted earlier, all algorithms are not equally good for all channel routing problems. Therefore, many benchmark channel routing examples have been proposed. The most famous benchmark is the *Deutsch difficult example.* Table 9.2 summarizes the routing results by different algorithms presented in this section on Deutsch difficult example. The table is reproduced from [PL88]. As it is clear from the table that YACR2 produces best routing for Deutsch difficult example both in terms of tracks and vias. However, YACR2 is considerably more complicated to implement as compared to Greedy router, which produces close to optimal results on most practical examples. In fact, it produces a solution within one or two tracks of the optimal.

9.5 Three-Layer Channel Routing Algorithms

The two-layer channel routing problem has been studied extensively in the past decade. In fact there are several two-layer channel routers that can produce solutions very close to the optimal solution (one or two tracks more than the minimum number of tracks required). About five years ago, a third metal layer became feasible. Most of the current gate-array technologies use three layers for routing. For example, the Motorola 2900ETL macrocell array is a bipolar gate array which uses three metal layers for routing. DEC's Alpha chip also

ROUTER	Tracks	Vias	Wire length
LEA	31	290	6526
Dogleg router	21	346	5331
Y-K router	20	403	5381
Greedy router	20	329	5078
Hierarchical router	19	336	5023
YACR2	19	287	5020

Table 9.2: Results on Deutsch difficult example.

uses three metal layer for routing. Intel's 486 chip used a three metal layer process and original Intel pentium was also fabricated on a similar process. As a consequence, a considerable amount of research has also been done on three-layer channel routing problem.

9.5.1 Classification of Three-Layer Algorithms

The three-layer routing algorithm can be classified into two main categories: The reserved layer and the unreserved layer model. The reserved layer model can further be classified into the VHV model and the HVH model.

Following theorems show the lower bounds of channel routing problem in three-layer reserved layer routing model, in terms of the maximum clique size in HCG and the longest path in VCG of the corresponding routing problem.

Theorem 15 *In the three layer VHV model, the lower bound on the number of tracks for a routing problem is* $h_{\max}$.

Theorem 16 *In the three layer HVH model, the lower bound on the number of tracks for a routing problem is* $\max\{v_{\max}, \frac{h_{\max}}{2}\}$.

Note that in VHV routing, the vertical constraints between nets no longer exist. Therefore, the channel height which is equal to the maximum density can always be realized using LEA. Almost all three-layer routers are extensions of two-layer routers. The net-merge algorithm by Yoshimura and Kuh [YK82] has been extended by Chen and Liu [CL84]. The gridless router Glitter [CK86] has been extended to Trigger by Chen [Che86].

Cong, Wong and Liu [CWL87] take an even general approach and obtain a three-layer solution from a two-layer solution. Finally, Pitchumani and Zhang [PZ87] partition the given problem into two subproblems and route them in VHV and HVH models. In this section, we discuss several three-layer channel routing algorithms.

9.5.2 Extended Net Merge Channel Router

In [CL84], Chen and Liu presented a three-layer channel router based on the net merging method and the left edge algorithm used in a two-layer channel

routing algorithm by Yoshimura and Kuh [YK82].

As there are no vertical constraints in VHV and therefore the left edge algorithm is sufficient. In the HVH routing, vertical constraint still exist. As a result, there should not be a directed path in the VCG between nets that are placed in both first and third layers on the same track. The merging algorithm presented in [YK82] can be extended to the HVH problem. In three-layer routing, in addition to merging nets in the same layer between different zones the nets in the same zone between different layers can also be merged. Two types of merging are defined as follows:

1. **Serial merging:** If there is no horizontal and vertical constraints between N_i and N_j then they can be placed on the same layer and the same track. This operation is called as *serial merging.*

2. **Parallel merging:** If nets N_i and net N_j have horizontal constraints and if they do not have vertical constraints then they can be placed on the same track but in different layers. In case of HVH model, one of them is placed in the first layer, whereas the other is placed in the third layer. This operation is referred to as *parallel merging*

As in two-layer routing, the merging procedure is the essential element of the whole algorithm where two sets of nets are merged. Let $\mathcal{N}_P = \{N_1, N_2, \ldots, N_{n_p}\}$ and $\mathcal{N}_Q = \{M_1, M_2, \ldots, M_{n_q}\}$ be two sets of nets to be merged. Two nets $N_i \in \mathcal{N}_P$ and $M_j \in \mathcal{N}_Q$ are merged such that N_1 lies in the longest path in VCG before merging and farthest away from either the source node or the sink node, and M_j is neither ancestor nor descendent of N_i, and after merging N_i and M_j the increase of longest path in VCG is minimum.

If the merging is done between two adjacent zones i and $i+1$, then $\mathcal{N}_P$ is the set of nets which terminates at zone i, $\mathcal{N}_Q$ is the set of nets that begin at zone $i+1$.

Let us define the following:

1. Let $\mathcal{N}_B$ be the set of nets which begin at zone $i+1$.

2. n_b is the number of nets in $\mathcal{N}_B$.

3. Let $\mathcal{N}_T$ be the set of nets which include: (a) nets which terminate at zone i. (b) nets which are placed on a track with no horizontal segment of nets on the other layer.

4. n_t is the number of nets in $\mathcal{N}_T$.

Before merging nets, all the nets in $\mathcal{N}_T$ have been placed on certain tracks, while nets in $\mathcal{N}_Q$ have not yet been placed on any track. Therefore, if net $M_2 \in \mathcal{N}_Q$ merges with any net in $\mathcal{N}_T$, no new track appears. Otherwise, if net $M_2 \in \mathcal{N}_Q$ merges with another net in $\mathcal{N}_Q$, then a new track appears. Therefore, if $n_q \leq n_t$ it is possible for the old tracks (where nets in T are placed) to contain all the nets in $\mathcal{N}_Q$. In such a case, in order to avoid the increase in the number of tracks, the parallel merging between nets in $\mathcal{N}_Q$ should be

```
Algorithm MERGE
begin
    N_Q = N_B; n_q = n_b;
    while N_Q ≠ φ do
        find M_j ∈ Q;
        N_Q = N_Q − {M_j};
        if n_q ≤ n_t then N_P = N_T
        else N_P = N_T + N_Q;
        find N_i ∈ N_P;
        if N_i ∈ N_T then
            (* serial (or parallel) merging (N_i, N_j) *)
            N_T = N_T − N_i;
            n_t = n_t − 1;
            n_q = n_q − 1;
        if N_i ∈ N_Q then
            (* parallel merging (N_i, N_j)
            N_Q = N_Q − N_i;
            n_q = n_q − 2;
        if N_i cannot be found then
            place N_j on a new track;
            n_q = n_q − 1;
            n_t = n_t + 1;
end.
```

Figure 9.46: Algorithm MERGE

avoided. Conversely, if $n_q > n_t$, the old tracks are not enough to contain all the nets in $\mathcal{N}_Q$, at least one new track appears. Parallel merging between nets in $\mathcal{N}_Q$ is allowed only in this case.

The details of the merging algorithm are given in Figure 9.46

As a special case, when merging starts, a parallel merging is made between all the nets which pass through the starting zone.

9.5.3 HVH Routing from HV Solution

In [CWL87], Cong, Wong, and Liu presented a general technique that systematically transforms, a two-layer routing solution into a three-layer routing solution. We will refer to this algorithm as CWL algorithm.

The focus of the CWL algorithm is very similar to the YK algorithm. In YK algorithm, nets are merged so that all merged nets forming a composite net are assigned to one track. The objective is to minimize the number of composite nets. In CWL algorithm, composite nets are merged together to form super-composite nets. The basic idea is to merge two composite nets such that the number of super-composite nets is minimized. Two composite nets in

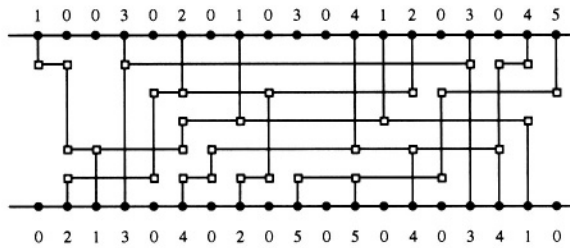
a super-composite net can then be assigned to two different layers on the *same* track. In order to find the optimal pair of composite nets that can be merged to form super-composite nets, a directed acyclic graph called *track ordering graph, TVCG* = (V, E) is defined. The vertices in V represent the composite(tracks) in a given two layer solution. The directed edges in $G(S)$ represent the ordering restrictions on pairs of tracks. Composite interval t_i must be routed above composite interval t_j if there exists a net $N_p \in t_i$ and $N_q \in t_j$, such that N_p and N_q have a vertical constraint. Thus TVCG is in fact a vertical constraint graph between tracks or composite intervals. The objective of CWL algorithm is to find a track pairing which reduces the total number of such pairs. Obviously, we must have at least $\frac{|V|}{2}$ pairs. It is easy to see that the problem of finding an optimal track pairing of a given two layer solution S can be reduced to the problem of two processor scheduling in which tracks of V are tasks and *TCVG* is the task precedence graph. Since the two processor scheduling problem can be optimally solved in linear time [Gab85, JG72], the optimal track permutation can also be found in linear time. Figure 9.47(b) shows the track ordering graph of the two-layer routing solution, shown in Figure 9.47(a), obtained by using a greedy router. Figure 9.47(c) shows an optimal scheduling solution for the corresponding two-processor scheduling problem.

The key problem is the number of tracks which are not paired. This happens due to adjacent vias. If adjacent vias can be removed between two non-paired tracks so that the tracks can be paired together, it would lead to saving of two empty tracks. The basic idea is to move the via aside and then a maze router can be used to connect the portion of the net containing x with the portion of the net containing the horizontal segment h_1 (see Figure 9.48).

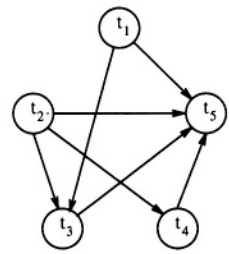
In order to successfully merge non-paired tracks, we must minimize the number of adjacent vias between two tracks. This is accomplished by properly changing the processor (layer) assignment of tasks (tracks). It is easy to see that if tracks t_i and t_j are assigned to be routed in the pth track then the layer on which a particular track gets routed is still to be decided. In other words, for each track, we have two choices. However, the choice that we make for each track can affect the number of adjacent vias. This problem can be solved by creating a graph, which consists of vertices representing both the possible choices for the track. Thus, each track is represented by two vertices. Four vertices of two adjacent tracks are joined by edges. Thus each edge represents a possible configuration of two adjacent tracks. Each edge is assigned a weight which is equal to the number of adjacent vias if this configuration represented by the edge is used. It is easy to see that problem of finding optimal configuration can be reduced to a shortest path problem. Thus, the problem can be optimally solved in $O(n^2)$ time. Figure 9.47(d) shows the graph described above for the problem in Figure 9.47(c).

9.5.4 Hybrid HVH-VHV Router

In [PZ87], Pitchumani and Zhang developed a three-layer channel router that combines both a HVH and a VHV model based on the idea of partitioning



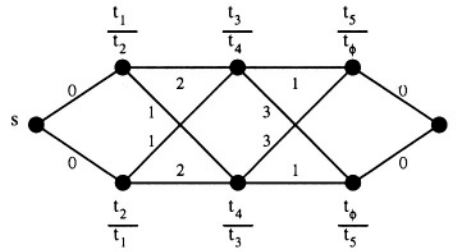
(a)



(b)

Time	P_1	P_2
1	t_1	t_2
2	t_3	t_4
3	t_5	

(c)



(d)

Figure 9.47: (a) A two-layer solution. (b) Track ordering graph. (c) An optimal scheduling solution.

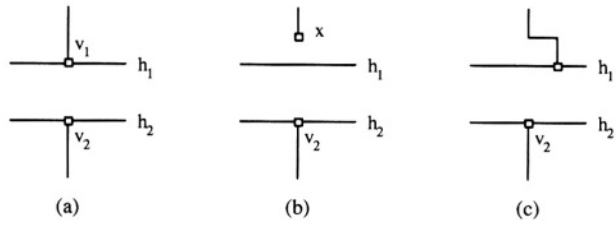


Figure 9.48: Local rerouting.

the channel. In this approach the channel can be thought of as two separate channels, not necessarily of the same size. One portion (upper or lower) is routed using the VHV and the other portion is routed using the HVH model. A transition track is usually needed between the two portions. The algorithm does not allow any dogleg. One of the important feature of this algorithm is that the pure HVH and VHV can be treated as special cases of this hybrid approach. This is due to the fact that, in the extreme case, one of the portions may constitute the whole channel and the other may be nonexistent. Obviously, no transition track is needed in this case. As a consequence, the result of this approach is the best between pure HVH and VHV approaches.

The height of a channel depends on two parameters, $v_{\max}$ and $h_{\max}$. If $v_{\max} >> h_{\max}$, then VHV is best suited for that channel, since it use only $h_{\max}$ number of tracks. On the other, if $h_{\max} >> v_{\max}$, then HVH is best suited for that channel, since it use only $\frac{h_{\max}}{2}$ number of tracks. Figure 9.49(a) and (b) shows the two cases when HVH and VHV routing models generate optimal solution. s and t are two dummy nodes signifying top and bottom of the channel. In practice, many channel routing problems are in fact a combination of both HVH and VHV, as shown in Figure 9.49(c).

The hybrid algorithm partitions the given netlist into two netlist, such that each netlist is best suited for either VHV or HVH style of routing. It then routes them separately thus obtaining two sub-solutions. The algorithm then inserts transition tracks to complete the connections between the two routed sub-solutions. The hybrid algorithm consists of the following steps:

1. choose VHV-HVH or HVH-VHV model;
2. partition the set of nets into two sets; the HVH-set, the set of nets to be routed in the HVH portion and the VHV-set, those to be routed in VHV portion of the channel;
3. route the nets.

The key problem is the partitioning of the channel. It is important to note that not all partitions are routable in the hybrid scheme. It is easy to show that for a net in a partition, all the nets that have vertical constraints with this net must also be in the same partition for a valid routable partition. Based on this

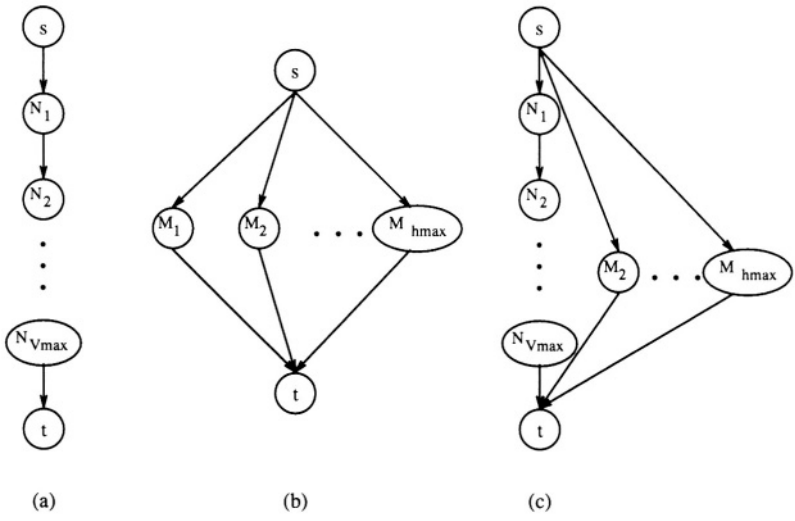


Figure 9.49: (a) CRP suited for VHV. (b) CRP suited for HVH. (c) CRP not suited for either HVH or VHV.

observation, any routable partition can be represented by a cut of the VCG with s and t on opposite sides of the cut, with all the nodes above the cut in upper-set and all the nodes below the cut in lower-set. Figure 9.50 shows an example of a cut inducing a routable partition; upper-set is $\{n_1, \ldots, n_{20}\}$ and lower-set is $\{m_1, \ldots, m_k\}$. The details of the partitioning algorithm based on the weighted cost function may be found in [PZ87].

To illustrate hybrid routing, we use the same example as in [PZ87]. Figure 9.51(a) gives the netlists, while Figure 9.51(b) shows a hybrid routing with VHV (two tracks) in the upper region and HVH (three tracks) for the lower region. The routing uses six tracks (including the transition track), while pure VHV requires eight tracks and pure HVH uses seven. In some cases, the terminal connections of nets may be such that vertical runs that cross the boundary between the regions can change layers on one of the regular tracks; in such cases, the transition track may be removed as shown in Figure 9.51.

9.6 Multi-Layer Channel Routing Algorithms

As the VLSI technology improves, more layers are available for routing. As a result, there is a need for developing multilayer routing algorithms. It should be noted that many standard cell designs can be completed without channel areas by using over-the-cell techniques (see Chapter 8). It may be noted that many over-the-cell routing problems are similar to channel routing problems. In case of full custom, perhaps four layers would be sufficient to obtain layouts without any routing areas on the real estate. However, new technologies such

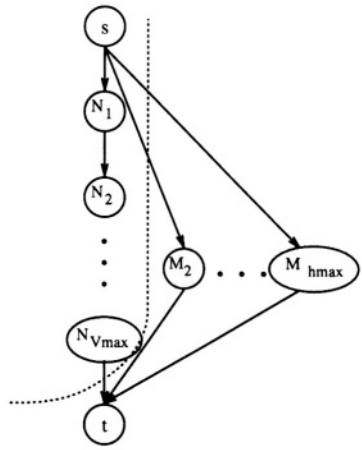


Figure 9.50: Partitioning for hybrid routing.

as MCM requires true multilayer capabilities since as many as 64 layers may be used.

In [Ham85], Hambrusch presented an algorithm for a n-layer channel router. The number of layers, the channel width, the amount of overlap and the number of contact points are four important factors for routing multi-terminal nets in multi-layer channels. An insight into the relationship between these four factors is also presented in [Ham85].

In [BBD+86], Braun developed a multi-layer channel router called *Chameleon.* Chameleon is based on YACR2. The main feature of Chameleon is that it uses a general approach for multilayer channel routing. Stacked vias can be included or excluded, and separate design rules for each layer can be specified. The Chameleon consists of two stages: a partitioner and a detailed router. The partitioner divides the problem into two and three-layer subproblems such that the global channel area is minimized. The detailed router then implements the connections using generalizations of the algorithms employed in YACR2.

In [ED86], Enbody and Du presented two algorithms for n-layer channel routing that guarantee successful routing of the channel for n greater that 3.

9.7 Switchbox Routing Algorithms

A switchbox is a generalization of a channel and it has terminals on all four sides. Switchbox routing problem is more difficult than a channel routing problem, because the main objective of channel routing is to minimize the channel height, whereas the objective of switchbox routing is to ensure that all the net are routed.

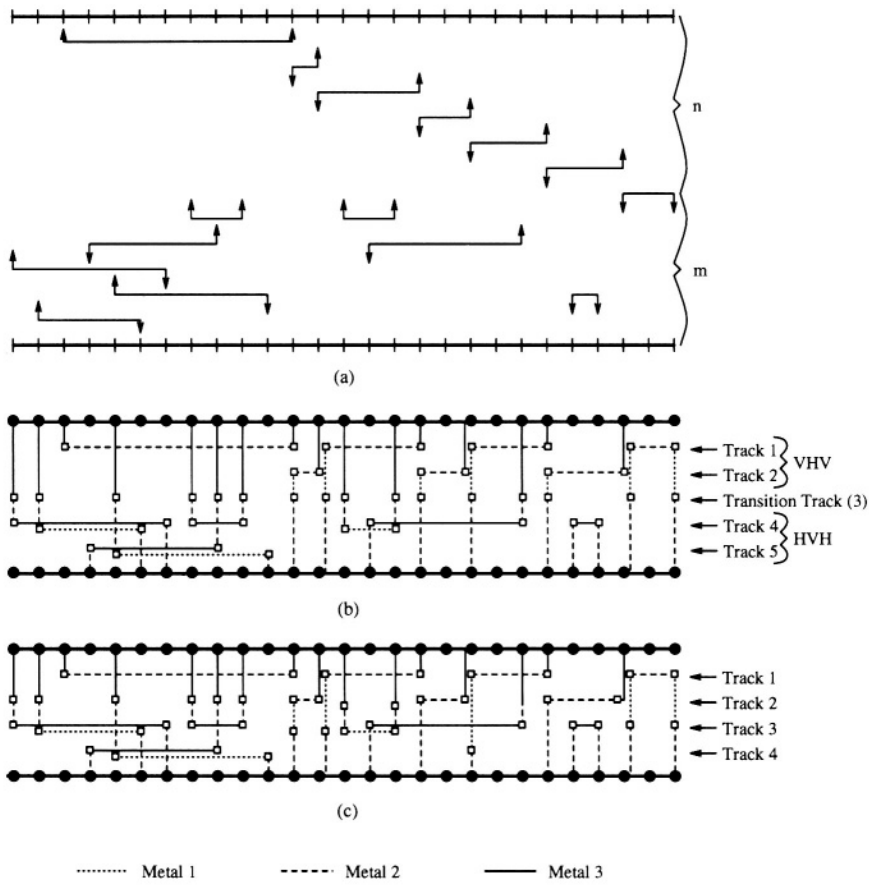


Figure 9.51: Example of hybrid routing.

9.7.1 Classification of switchbox routing algorithms

Switchbox routers can be classified as,

1. Greedy Routers,
2. Rip up and Reroute Routers and
3. Others.

Greedy routers are essentially extension of the Greedy channel router. Rip up and Reroute routers employ some algorithm for finding routes for nets and modifying the routes to accommodate additional nets. Several other techniques, such as computational geometry, simulated evolution have also been applied to switchbox routing. In this section, we review one algorithm from

each category. In this section we describe four different algorithms for switchbox routing. BEAVER [CH88] is an excellent router based on computational geometry.

9.7.2 Greedy Router

In [Luk85], Luk presented a greedy switchbox router, which is an extension of the greedy channel router [RF82]. As opposed to a channel, which is open on the left and the right side, a switchbox is closed from all four sides. Moreover, there are terminals on the left and right boundaries of a switchbox. Thus in addition to the terminals on the upper and lower boundaries, the presence of terminals on the left and right boundaries of the switchbox need to be considered while routing. This is achieved by the following heuristic:

1. **Bring in left terminals:** The terminals on the left boundary are brought to the first column as horizontal tracks.

2. **Jog to right terminals:** The step in greedy channel router in which the nets are jogged to their nearest top or bottom terminal is modified for the nets that have a terminal on the right boundary. Such nets are jogged to their *target rows,* in addition to jogging to the next top or bottom terminal. Jogging a net to the next top or bottom terminal is referred to as $JOG_{t/b}$, whereas, jogging a net to its target row is referred to as JOG_r. A target row of a net is a row of its terminal on the right boundary. The nets are jogged to their target rows according to the following priority.

   (a) First choice is a net whose right side of the target row and the vertical track between the net and target row is empty.

   (b) Second choice is a net whose right side of the target row is empty. In addition, the priority is also based on how close a net can be jogged to its target row.

   (c) Third choice is a net that can be brought closer to its target row.

Ties are resolved by giving higher priority to a net which is further from its target row. The cyclic conditions at this step can be broken by allowing a net to cross its target row. (see Figure 9.52.) Note that the optimal way to reach the target row for a net is to jog once as each jogging wastes a vertical track, too many joggings may result in running out of tracks. Excessive jogging can be avoided by allowing a net to jog to its target row only if it can be brought to or beyond half way between its initial position and the target row.

Several schemes of using $JOG_{t/b}$ and JOG_r have been suggested for a net having terminals on the right and top/bottom boundaries.

1. **Scheme 1:** For a net that has a terminal on the right boundary JOG_r is performed until it reaches its target row. The top and bottom terminals are connected by branching out some net segments (see Figure 9.53(a)).

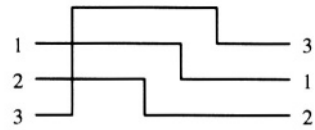


Figure 9.52: Cyclic constraints.

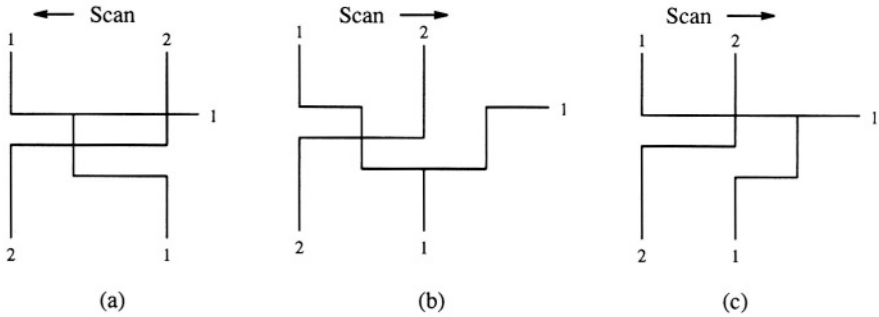


Figure 9.53: Routing schemes: (a) scheme 1,(b) scheme 2,(c) scheme 3

2. **Scheme 2:** $JOG_{t/b}$ is used until all top and bottom terminals are connected, JOG_r is used from the column where the last top/bottom terminal appears (see Figure 9.53(b)).

3. **Scheme 3:** In this scheme $JOG_{t/b}$ and JOG_r are used in parallel. Branching out is avoided by either using $JOG_{t/b}$ or JOG_r at each column (see Figure 9.53(c)).

4. **Scheme 4:** This scheme involves a combination of several schemes. If the rightmost top/bottom terminal of a net is in the rightmost $p\%$ of the switchbox, scheme 1 is used for that net. Otherwise, scheme 2 is used (see Figure 9.54).

Determining Scan Direction: Determining scan direction is equivalent to assigning *left* edge to one of the edges of the switchbox. Let $p_1 = (e_1, e_3)$ and $p_2 = (e_2, e_4)$ be the opposite pairs of edges of the switchbox. The objective of this step is to assign one of these pairs as the left-right pair and the other as the top-bottom pair. This operation is divided into two steps. In the first step, the top-bottom and the left-right pairs are assigned without identifying left or right edges in the left-right pair. In the second step the left and the right edges are identified.

The following measures are defined to achieve the first step. The *augmented density* of p_1 is defined as the overall minimum number of tracks required to maintain the connectivity of the terminals on opposite pair of edges in p_1 as in

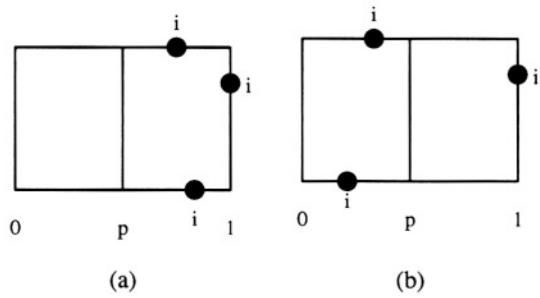


Figure 9.54: Routing scheme 4 (a) JOG_r for net N_i, (b) $JOG_{t/b}$ for net N_i

```
Algorithm GREEDY-SB-ROUTER
begin
    Determine the scan direction;
    Bring left terminals into column 1;
    for i = 1 to M do
        if no empty track exists then
            increase number of tracks;
        bring T(i) and B(i) to empty tracks;
        join split nets as much as possible;
        for each net with no right terminals do
            bring split nets closer by jogging;
        for each net with a right terminal do
            use scheme 4;
        if close to right edge then
            fanout to all target rows;
    while split net exist do
        join split nets as much as possible;
end.
```

Figure 9.55: Algorithm GREEDY-SB-ROUTER

the channel routing problem, plus the tracks required to connect the nets on the edges in p_2 with the nets on p_1. The *augmented channel density ratio* for $p_1 = (e_1, e_3)$ is defined as the ratio of the number of tracks available between e_1 and e_3 to the augmented density of p_1. In a similar manner the augmented channel density of p_2 can be defined. As the property of the greedy heuristic is to minimize the number of tracks perpendicular to the scan direction, it is obvious to assign the left-right pair to the pair that has smaller augmented channel density ratio.

Let $p_k = (e_i, e_j)$ be the left-right edge pair assigned in the last step. The edges in p_k are assigned the left and the right edges based on the following rules:

1. An edge in p_k which has more terminals and especially multiple terminals is selected as left edge. This reduces the burden on JOG_r and fanout operations.

2. An edge in p_k which is close to less congested region is selected as right edge. As a result, more free vertical tracks will be available to join split nets and fanout to target terminals.

The formal description of the GREEDY-SB-ROUTER appears in Figure 9.55.

9.7.3 Rip-up and Re-route Based Router

Shin and Sangiovanni-Vincentelli [SSV86] proposed a switchbox router based on an incremental routing strategy. It employs maze-running but has an additional feature of modifying already-routed nets. Some nets are even ripped-up and re-routed. It is this feature of MIGHTY that makes it suitable for channel and switch box routing. The cost function used for maze routing penalizes long paths and those requiring excessive vias. MIGHTY consists of two entities: a *path-finder* and a *path-conformer.* It is possible for the router to go into a loop in the modification phase. This can be avoided by using some sort of time-out mechanism. The overview of Algorithm MIGHTY is in Figure 9.56.

The worst case time complexity of the algorithm is more than $O(k^3pnL)$, where p and k are the number of terminals and nets, respectively and L is the complexity of the maze routing algorithm.

9.7.4 Computational Geometry Based Router

In [CH88], Cohoon and Heck presented a switchbox routing algorithm called BEAVER, based on a delayed layering scheme with computational geometry techniques. The main objectives of BEAVER are the via and wire length minimization. BEAVER is an unreserved layer model routing algorithm. While routing a net, BEAVER delays the layer assignment as long as possible. One of the important features of BEAVER is that it uses priority queue to determine the order in which nets are interconnected. An overview of BEAVER is given in Figure 9.57.

```
Algorithm MIGHTY
begin
    1. Extend all pins on the boundaries of the region inside
           by one unit;
    L → φ; (* Initialize list *)
    2. (* path finder *)
    for each net do
        MAZE-ROUTE( net, L );
    3. sort L in increasing value of costs;
    4. while L ≠ φ do
        Get next path p from L;
        if no grid cell in p is occupied then
            Implement p; goto step 5;
            else invoke the path-finder to find a new feasible
                minimum path connecting two unconnected
                    subnets of the net;
                Let δ be the increase in cost for the new path p';
            if δ < MAXINCREASE then
                Implement p'; goto step 5;
                (* weak modification *)
                Push implemented nets around to
                obtain a 'good' connection for the given net;
                if weak modification fails then
                    (* strong modification *)
                    Remove an existing connection and
                    try to obtain 'good' connection;
    5. Remove p from L;
end.
```

Figure 9.56: Algorithm MIGHTY

It can be seen from the algorithm that BEAVER uses up to three methods to find interconnections for nets: 1) *corner router,* 2) *line sweep router,* and 3) *thread router.*

Corner router: The corner router tries to connect terminals that form a *corner connection.* Such a connection is formed by two terminals if (1) they belong to the same net, (2) they lie on the adjacent sides of the switchbox, and (3) there are no terminals belonging to the net that lie between them on the adjacent sides. The corner router is also a preferred router since it is the fastest and because its connections tend to be part of the minimum rectilinear steiner tree for the nets.

During the initialization of the corner priority queue, each net is checked to see if it has a corner connection. A corner connection can be simply made by

```
Algorithm BEAVER
begin
    Initialize control information;
    Initialize corner-priority-queue;
    corner route;
    if there are nets to be routed then
        Initialize line-sweep-priority-queue;
        Line sweep route;
        if there are nets to be routed then
            Relax control constraints;
            Reinitialize line-sweep-priority-queue;
            Line sweep route;
            if there are nets to be routed then
                Initialize thread-priority-queue;
                Thread route;
    Perform layer assignment;
end.
```

Figure 9.57: Algorithm BEAVER

examining the control of the terminals that comprise the corner. If the section of the control overlap, then the corner can be realized. Nets with one or two corners need no further checks. However, straightforward connection of four corner nets can introduce cycles. There are two types of cycles as shown in Figure 9.58.

An overlap cycle is removed by routing only three of the corners. A four-terminal cycle can be removed by routing only three of the corners. When the corner router has to decide upon one of two such corners to route, the one with least impact on the routability of other nets if preferred.

Linesweep router: The line sweep router is invoked when no more corner connections can be made. However, if after current net's linesweep realization, some other corner connection become realizable, then the linesweep router is temporarily suspended until the corner priority queue gets emptied. For each net five possibilities are considered: a wire with single bend, a single straight wire, a dogleg connection with a unit-length cross piece, three wires arranged as a horseshoe, and three wires in a stair arrangement. These are shown in Figure 9.59.

To reduce the number of vias, straight line wires are preferred to dogleg connections, dogleg to single-bend connections and single bend to two-bend connections. In looking for its connections, the linesweep router uses the computational geometry technique of plane sweeping. One approach is to use scan-lines to find straight line connections between disjoint subnets of the net in question. It works in $O(n \log n + k)$ time. BEAVER uses three scan lines that

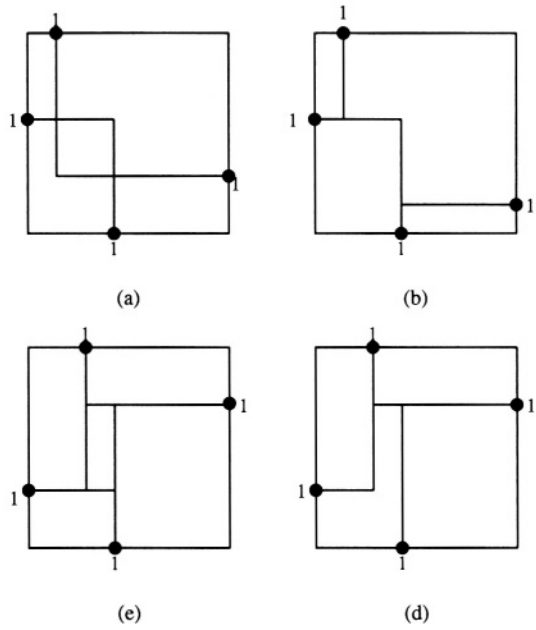


Figure 9.58: (a)Example of an Overlap Cycle and (b) its Solution (c) Example of a Four-Terminal Cycle and (d) its Solution

sweep the plane in tandem: one across the column, one across the row and a third to detect doglegs. Also, it employs bounding function to reduce the computational complexity. If some nets still remain unrealized, the control of existing nets is reduced and the process is repeated a second time. All remaining nets are then routed by a thread router.

Thread router: This router is invoked very sparingly. It is a maze-type router that seeks to find minimal length connections to realize the remaining nets. Since, this router does not restrict its connection to any preferred form, it will find a connection if one exists. Whenever a net N_i consists of more than one routable subnets, a maze expansion, on the lines of Soukup router is initiated. The expansion starts from a small subnet s that has not been used in an endeavor to minimize the wire-length and number of vias.

Layer Assignment: This phase primarily aims at minimizing the number of additional vias introduced. Since, at this stage, all grid points that any wire passes through are known, it is possible to optimally assign unlayered wires to a particular layer. BEAVER can also very easily extend this to achieve metal-maximization. A simple set of heuristics, based on coloring the grid points as red or black is presented in [CH88].

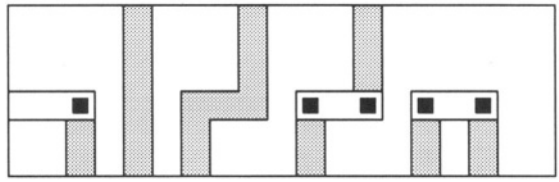


Figure 9.59: Prototype Linesweep Connections

Router	Vias	Wire length
GRS	36	529
MIGHTY	32	530
BEAVER	28	529

Table 9.3: Results on augmented dense switchbox.

9.7.5 Comparison of Switchbox Routers

Table 9.3 compares performance of the switchbox routers discussed in this chapter. It is apparent from the Table 9.3 that BEAVER routes the augmented dense switchbox with minimum number of vias, whereas, there is no significant difference in the wire lengths.

9.8 Summary

Detailed routing is one of the most fundamental steps in VLSI physical design cycle. The detailed routing problem is solved by routing the channels and switchboxes. Channel routing and switchbox routing problems have been studied extensively and several routing algorithms have been proposed by many researchers.

Routing results may differ based on the selection of routing models. A routing model can be grid-based where wires follow paths along the grid lines or gridless, where wires can be placed at any place as long as the design rules are not violated. Another model is be based on the layer assignments of different net segments. In reserved layer model, segments are allowed only to a particular layer(s). Most of the existing routers use reserved layer model. A model is unreserved if any segment is allowed to be placed in any layer.

The most widely considered objective function for routing a channel is the minimization of channel density. Other objectives are to minimize the length of routing nets and to minimize the number of vias. A routing algorithm must take into consideration the following: net types, net width, via restrictions, boundary types, number of layer available, degree of criticality of nets.

The main objective of (channel) routing is to minimize the total routing area. Most successful routers are simple in the approach. Greedy is one of the most efficient and easiest to implement channel routing algorithm. As the number of layers increase, the routing area used decreases. However, it should be noted that adding a layer is usually very expensive. In most of the cases, the routing area can virtually be eliminated using four layers by using advanced over-the-cell and over-the-block routing techniques discussed in [SBP95].

The objective of switchbox routing is to determine the routability. Several switchbox routing algorithms have been developed. Greedy and rip-up and reroute strategies have lead to successful routers.

9.9 Exercises

†1. Develop a river routing algorithm for a simple channel when the channel height is fixed. Note that a simple channel is the one with straight line boundaries. Given a channel routing problem, the router should first check if the given problem can be routed for a given channel height. In case it cannot be routed, the router should stop, otherwise the router should find a solution.

2. Prove that every single row routing problem is routable if no restrictions are placed on the number of doglegs and street congestions.

3. Given the net list in part (c) of Figure 9.61, find a single row routing such that the street congestion on both the streets is less than or equal to 3.

4. For the net list in Figure 9.61, does there exist a solution for which the between-node congestion is no more than 1 ?

†5. Prove Theorem 9.

†6. Prove Theorem 10. Extend this theorem to multi-terminal single row routing problems.

‡7. Prove Theorem 11. Does there exist a necessary and sufficient condition for SRRP with at most k $(k \geq 1)$ doglegs per net ?

‡8. Does there exist a necessary and sufficient condition for SRRP for $C_B = 1$?

‡9. Consider the two row routing problem given in Figure 9.60. Given two rows of terminals separated by w tracks, the objective is to find a single layer routing with minimum congestions in the upper and lower streets. Note that the number of tracks in the middle street is fixed.

10. Give an instance of a channel routing problem in a two layer restricted (HV) model in which,

    (a) The channel height is greater than $v_{\max}$.

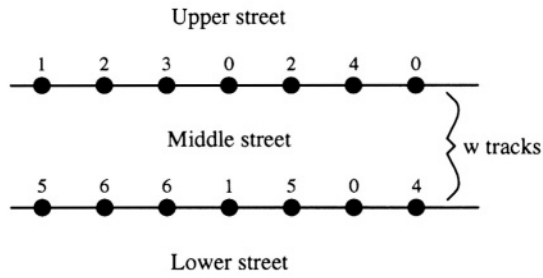
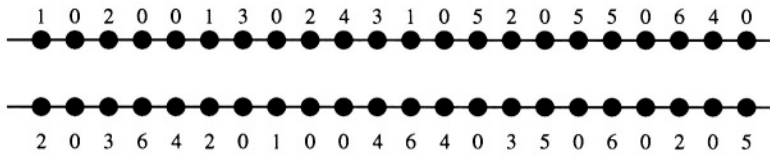
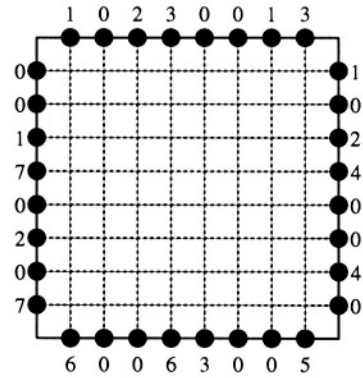


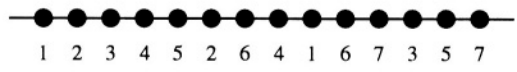
Figure 9.60: A two-row routing problem.



(a)



(b)



(c)

Figure 9.61: Routing problems.

(b) The channel height is greater than $d_{\max}$.

(c) The channel height is greater than both $v_{\max}$ and $d_{\max}$.

11. Give an instance of channel routing problem in two layer Manhattan model in which there are cyclic constraints.

12. Prove with an example that it is possible to get better results in channel routing by using an unrestricted model than the restricted model for a two layer channel routing problem using the Manhattan model.

13. Give an example of a channel routing problem for which the greedy router performs better than the hierarchical router.

14. Give an example of a channel routing problem for which the hierarchical router performs better than the greedy router.

15. Route the channel given in Figure 9.61 (a) using the following routers and compare their results:

(a) LEA.

(b) Dogleg router.

(c) Y-K algorithm.

(d) Greedy channel router.

(e) Hierarchical channel router.

(f) YACR2.

16. In the greedy router, while joining split nets in a column, more than one nets can be joined. Formulate the problem of finding maximum number of nets that can be joined in a single column. Develop an $O(n \log n)$ time complexity algorithm for this problem, where n is the total number of tracks in the channel.

17. In the greedy channel router, while a column is being routed, segments of split nets are brought closer by using doglegs in case they cannot be joined. Develop an efficient strategy that will maximize the number of nets that can be brought closer.

18. Give an instance of channel routing problem in three-layer HVH model in which,

(a) The channel height is greater than $v_{\max}$.

(b) The channel height is greater than $\frac{d_{\max}}{2}$.

(c) The channel height is greater than both $\frac{d_{\max}}{2}$ and $v_{\max}$.

19. Route the channel routing problem given in Figure 9.61 (a) using extended net merge algorithm by Chen and Liu.

20. Show that in the hybrid HVH-VHV routing, the lower bound for the channel density is $d_{\max}/2$, where $d_{\max}$ is the size of the maximum clique in the corresponding HCG.

21. Show that given a CRP, if $v_{\max} = 1$ in the corresponding VCG, then every partition is routable in hybrid routing.

22. Develop a unreserved layer switchbox router when the terminals are located in any metal layer.

23. Develop a algorithms for routing in a three dimensional routing grid with planer upper surface and non-planer lower surface. Assume that the terminals are located at the lower surface.

24. Solve exercise 23 when the grid has non-planer upper surface.

Bibliographic Notes

The general single-layer routing problem was shown to NP-complete in [Ric84]. In [BP83, DKS+87, LP83, SD81, Tom81], several restricted single-layer routing problems have been solved optimally. River routing was defined in [DKS+87] and refined by Leiserson and Pinter [LP83]. Several extensions of the general river routing algorithm have been proposed [JP89, TH90]. In [JP89], river routing algorithm is extended to handle multiple, parallel channels. The river routing is called *feed-through river routing,* because wires must pass through gaps that are to be created between the components in each row. In [TH90], Tuan and Hakimi presented a variation of river routing that minimizes the number of jogs.

Tsukiyama et. al [TKS82], considered the restricted version of the via assignment problem where no net has more than one point in any given column. In [TK78], another restricted version in which each net is forced to use vias from the same via column is considered. Both [TKS82] and [TK78] have shown that, with their respective restrictions, deciding whether k via columns are sufficient to realize the netlist is an NP-Hard problem.

The layering problem was shown to be NP-Hard by Sahni, Bhatt, and Raghavan [SBR80]. As a consequence, heuristic algorithms have been studied for this problem. In practice $\max\{C_{ls}, C_{us}\} \leq 2$ and therefore heuristic algorithms for the restricted layering problem, in which $C_{ls} = C_{us} = 2$ has been considered by several researchers [GKG84, HS84a, TKS82]. The algorithm given in [TKS82] generates solutions with number of layers l, where, $l \leq 1.33 \times l^*$, where l^* is the optimal number of layers. In [HS84a], Han and Sahni presented two fast algorithms for layering. These algorithms consider one net at a time starting from the leftmost net. A net is assigned to the first layer in which the channel capacity allows its insertion. If this net cannot be assigned to any layer, a new layer is started. It was reported that these algorithms perform better than the algorithm proposed in [TKS82], that is, they use less layers. It was also reported that these algorithms are much faster than the one proposed in [TKS82]. Recently, Gonzalez et. al [GKG84] reported some results on layering; it is not, however, clear how their algorithm

compares with earlier algorithms. It appears to be similar to the algorithm given in [HS84a].

Tsui and Smith [TI81] gave another formulation of single row routing problem. They considered only two terminal nets and obtained some necessary and sufficient conditions for the routability of a net list if upper and lower street capacities are known. Their idea is based on the number of blockages that a net would encounter in case all nets are laid out in the same street.

Han and Sahni [HS84a] proposed linear time algorithms for the special case of SRRP when the number of tracks available for routing is restricted to one, two or three. In 1983 they extended their work and presented simpler algorithms than that of [HS84b]. They introduced the notion of *incoming permutation* being the relative ordering of nets with respect to a certain terminal. The algorithm makes a left to right scan on the terminals. If a new net is starting at the terminal under consideration this net is inserted in all the incoming permutations to get a set of new permutations. The permutations which do not meet the street congestion requirement are deleted from further consideration. Thus, the main idea of the algorithm in [HS84a, HS84b] is to keep track of all legal permutations. Obviously, this algorithm is only practical for small values of upper and lower street capacities because of its exponential nature.

Raghavan and Sahni [RS84] investigated the complexity issues of single row routing problem and the decomposition process for the multi-layer circuit board problem. In [RS84] it is shown that the via assignment problem considered by [TK78] remains NP-Hard even for $k = 2$. They also prove that the problem of via column permutation is NP-Hard and remains so even if 2 via columns are allowed per net for decomposing. It is also shown that the problem of minimizing the total number of vias used is NP-Hard. The problem of finding a layout with minimum bends (doglegs) is also proved to be NP-Hard [RS84].

In [HS71], Hashimoto and Stevens first introduced the channel routing problem. Another column-by-column router has been proposed by Kawamoto and Kajitani [KK79] that guarantees routing with upper bound on the number of tracks equal to the density plus one, but additional columns are needed to complete the routing. Ho, lyenger and Zhenq developed a simple but efficient heuristic channel routing algorithm [HIZ91]. The algorithm is greedy in nature and can be generalized to switchboxes and multi-layer problems.

Some other notable effort to solve switchbox problem have been reported by Hamachi and Ousterhout (Detour) [HO84]. This approach is an extension of the greedy approach for channel routing. In [Joo86], Joobbani proposed a knowledge based expert system called WEAVER for channel and switchbox routing. In [LHT89], Lin, Hsu, and Tsai presented a switchbox router based on the principle of evolution. In [GH89], Gerez and Herrmann presented a switchbox router called PACKER. This router is based on a stepwise reshaping technique. WEAVER [Joo86] is an elaborate rule-based expert system router that often produces excellent quality routing at the cost of excessive computation time. SILK [LHT89] a switchbox router based on the simulated evolution technique. A survey and comparison of switchbox routers has been presented by Marek-Sadowska [Sad92]. In [CPH94] Cho, Pyo, and Heath presented a

parallel algorithm, using a conflict resolving method has been developed for the switchbox routing problem in a parallel processing environment.