CHAPTER

Global and detailed routing

12

Huang-Yu Chen
National Taiwan University, Taipei, Taiwan

Yao-Wen Chang
National Taiwan University, Taipei, Taiwan

ABOUT THIS CHAPTER

After placement, the routing process determines the precise paths for nets on the chip layout to interconnect the pins on the circuit blocks or pads at the chip boundary. These precise paths of nets must satisfy the design rules provided by chip foundries to ensure that the designs can be correctly manufactured. The most important objective of routing is to complete all the required connections (*i.e.*, to achieve 100% routability); otherwise, the chip would not function well and may even fail. Other objectives, such as (1) reducing the routing wirelength and (2) ensuring each net to satisfy its required timing budget, have become essential for modern chip design. For modern large-scale circuit design, a chip may contain billions of transistors and millions of nets. To handle the high complexity, a routing algorithm often adopts the two-stage approach of global routing followed by detailed routing. Global routing first partitions the routing region into tiles and decides tile-to-tile paths for all nets, whereas detailed routing determines the exact tracks and vias for nets.

This chapter starts with a discussion of the routing problem. After introducing the problem definition, the techniques of general-purpose routing are described. This is followed by the introduction of popular global-routing algorithms that cover sequential and concurrent approaches. The second half of this chapter discusses detailed routing, for which channel and full-chip routing techniques are discussed, followed by modern routing techniques considering signal integrity and chip manufacture and yield. This chapter concludes with routing trends and future directions of routing. After reading through this chapter, the reader should have a clear picture about popular global and detailed routing algorithms. This background will be valuable in implementing/developing routing algorithms to meet the design needs.

12.1 INTRODUCTION

Routing is an important step in the design of ***integrated circuits*** (ICs). It generates wiring to interconnect pins of the same signal, while obeying the manufacturing **design rules**. As IC process advances to nanometer technology, foundries may fabricate billions of transistors in a single chip, and the number of transistors per die will still grow drastically in the near future. This increasing complexity imposes substantial challenges for physical design, especially for routing.

Research in VLSI routing has received much attention in the literature. Routing is typically a very complex combinatorial problem. To make it manageable, the routing problem is usually solved by use of a two-stage approach of **global routing** followed by **detailed routing**. Global routing first partitions the routing region into tiles and decides tile-to-tile paths for all nets while attempting to optimize some given objective function (*e.g.*, total wirelength and circuit timing). Then, guided by the paths obtained in global routing, detailed routing assigns actual tracks and vias for nets.

Figure 12.1 illustrates the process of global routing and detailed routing. After placement, we have a placed layout shown in Figure 12.1a, which contains the information about the exact locations of blocks, pins of blocks, and I/O pads at chip boundaries. We are also provided with a **netlist** that describes a list of connections by indicating which pins or pads should be electrically connected to form a set of nets. Figure 12.1b illustrates some global-routing paths. It first divides the routing region into **tiles** and then generates a "loose" route for each connection by finding the tile-to-tile paths to connect pins and/or pads. Figure 12.1c shows a result of detailed routing, which determines the exact route for each net by searching within the tile-to-tile path. Here, the exact route means a path specified by the actual geometric layout such as metal wires and vias.

In the following we formally give the problem definition of the routing problem and describe the routing model and constraints.

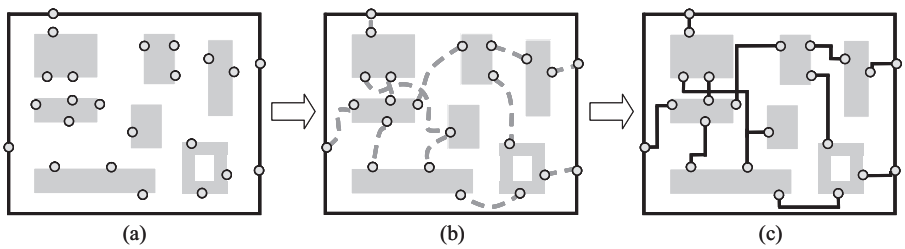

FIGURE 12.1

Routing problem: (a) A given placement result with fixed locations of blocks and pins. (b) Global routing. (c) Detailed routing.

12.2 PROBLEM DEFINITION

The problem definition for the general routing problem is as follows:
Inputs:

1. A placed layout with fixed locations of chip blocks, pins, and pads
2. A netlist
3. A timing budget for each critical net
4. A set of design rules for manufacturing process, such as resistance, capacitance, and the wire/via width and spacing of each layer

Output:

Wire connection for each net presented by actual geometric layout objects that meet the design rules and optimize the given objective, if specified.

12.2.1 Routing model

Routing in a modern chip is typically a very complex process, and it is thus usually hard to obtain solutions directly. Most routing algorithms are based on a graph-search technique guided by the congestion and timing information associated with routing regions and topologies [Saxena 2007]. A router assigns higher costs to route nets through congested areas to balance the net distribution among routing regions.

Applying the graph-search technique for routing requires modeling the routing resource as a graph where the graph topology can represent the chip structure. Figure 12.2 illustrates the graph modeling. For the modeling, a chip (routing region) is first partitioned into an array of rectangular tiles (or called **global-routing tiles**), each of which may accommodate tens of routing tracks in each dimension, as illustrated in Figure 12.2a. A node in the routing graph represents a tile in the chip, whereas an edge denotes the boundary between two adjacent tiles (see Figures 12.2b–c). Each edge is assigned a capacity according to the physical routing area or the number of tracks in a tile. This graph is called a **global-routing graph**.

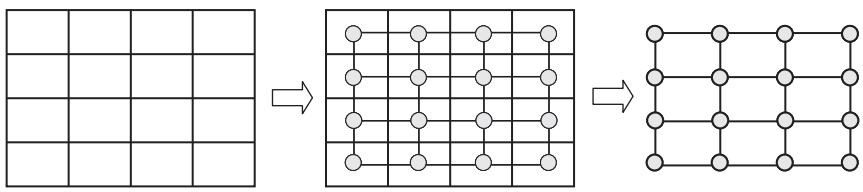

FIGURE 12.2

The global-routing graph: (a) The chip (routing region) is partitioned into an array of rectangular tiles. (b) A node in the routing graph represents a tile in the chip, whereas an edge denotes the boundary between two adjacent tiles. (c) The final global-routing graph.

A global router finds tile-to-tile paths for all nets on the global-routing graph to guide the detailed router. The goal of global routing is to route as many nets as possible while meeting the capacity constraint of each edge and any other constraint, if specified. For example, for timing-driven routing, additional costs can be added to the routing topologies with longer critical path delays. For detailed routing, the router decides the actual physical interconnections of nets by allocating *wires* on each metal layer and *vias* for switching between metal layers.

Generally, there are two different layer models, the **reserved** and **unreserved layer models**. In the reserved layer model, each layer is allowed only one specific routing direction (*i.e.,* **preferred direction**). For example, the technology file may specify that the wires in the first metal layer are allowed to run only in the horizontal direction, the second metal layer contains only vertical wires, etc. A layer model is unreserved if it allows the placement of wires with any directions (*i.e.,* **non-preferred direction**). Most of the existing routers and design methodologies apply the reserved layer model, because it has lower complexity than the unreserved layer model and is much easier for implementation.

There are two kinds of detailed-routing models: the **grid-based** and **gridless** models. For grid-based routing, a routing **grid** is superimposed on the routing region, and then the detailed router finds routing paths in the grid, as shown in Figure 12.3a. The space between adjacent grid lines is called **wire pitch**, which is defined in the technology file and is larger than or equal to the sum of the minimum width and spacing of wires. Note that the router has to control the searching space such that the path in the horizontal/vertical layers can only

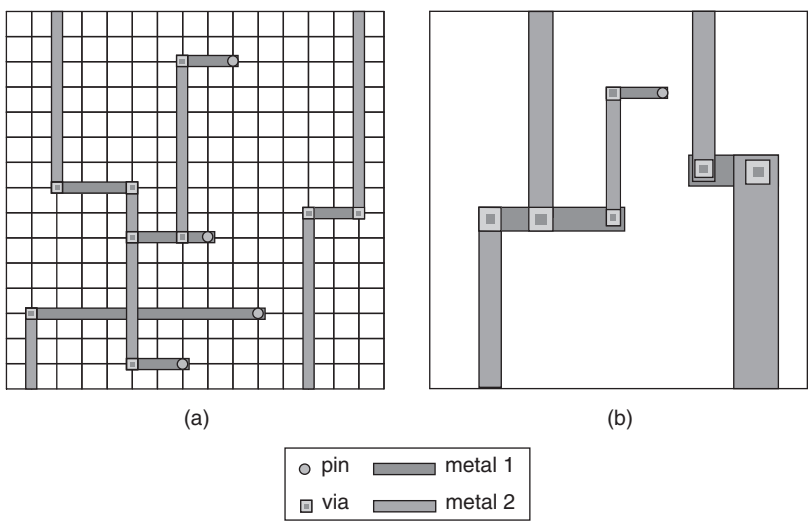

FIGURE 12.3

Two kinds of detailed-routing models: (a) Grid-based detailed routing. (b) Gridless detailed routing.

run horizontally/vertically for the reserved layer model, and switching from layer to layer is allowed only at the intersection of vertical and horizontal grid lines. In this way, the wires with the minimum width following the path in the grid would automatically satisfy the design rules. Therefore, grid-based detailed routing is much more efficient and easier for implementation.

The gridless detailed routing model (also called *shaped-based*) refers to any model that does not follow the grid-based model. A gridless detailed router does not follow the routing grid and thus can use different wire widths and spacing, as shown in the example in Figure 12.3b. Various gridless models have been proposed, such as the **connection graph** [Zheng 1996], the **implicit connection graph** [Cong 1999], the **implicit triple-line graph** [Chen 2007a], and **corner stitching** [Qusterhout 1984]. The main advantage of gridless routing lies in its greater flexibility; it can handle variable widths and spacing for wires and is, thus, more suitable for interconnect tuning optimization, such as wire sizing and perturbation. However, gridless detailed routing is generally much slower than the grid-based one because of its higher complexity.

Figure 12.4 illustrates an example of grid-based detailed routing for a two-pin net. After the global routing, we have a tile-to-tile global-routing path as shown in Figure 12.4a, and the detailed-routing graph is constructed only within the tiles of the global-routing path, as shown in Figure 12.4b. Then the final detailed-routing solution is found in the graph, as shown in Figure 12.4c. Constructing and searching the detailed-routing graph within the tiles of the global-routing path, the detailed router can substantially prune the searching space and thus reduce the routing time.

12.2.2 **Routing constraints**

The routing constraints can be classified into two major categories: (1) design-rule constraints and (2) performance constraints. The design-rule constraint is

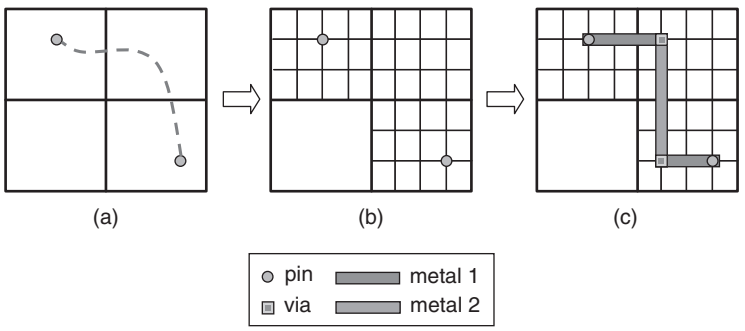

FIGURE 12.4

Detailed routing: (a) A tile-to-tile global-routing path connecting two pins on metal 1. (b) The detailed-routing graph is constructed within the tiles of the global-routing path. (c) A detailed-routing solution on the detailed-routing graph.

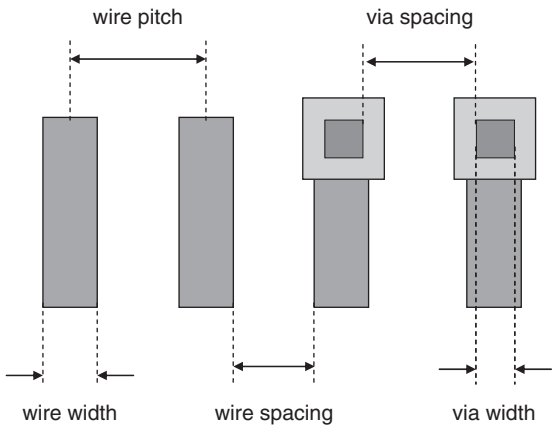

FIGURE 12.5

An example of design rules. Typical rules define wire width, wire spacing, wire pitch, via width, and via spacing on each layer.

often related with the manufacturing details during fabrication. To improve the manufacturing yield, connections of nets have to follow the rules provided by foundries. For example, in the 65-nm technology, the physical limitations of an optical lithography system would impose a constraint on a wire such that its width cannot be smaller than 65 nm.

Figure 12.5 illustrates a typical set of **design rules**. It defines the minimum widths of wires and vias, the minimum wire-to-wire spacing, and the minimum via-to-via spacing of a layer. The distance between two wires or routing tracks of the grid-based model is often called wire pitch. Other design rules of the manufacturing process, such as resistance and capacitance of each layer, are also included.

The objective of the performance constraint is to make the connections meet the performance specifications provided by chip designers. For example, the timing constraint is often the most important performance constraint for high-speed designs. The speed of a chip is limited by its **critical nets**, which have smaller timing budgets (or timing slacks) than others. To meet the performance constraint, it is desirable to carefully route these critical nets by proper routing topologies.

12.3 GENERAL-PURPOSE ROUTING

In Section 12.2.1, we modeled the routing resources by the global- and detailed-routing graphs. For global and detailed routing, we can perform a graph-search technique on these routing models. In the following, we introduce three popular graph-searching techniques, the **maze, line-search**, and **A*-search** routing

algorithms. Note that these algorithms are **general-purpose routing algorithms**, because they can be applied to both global and detailed routing problems on the general routing structure.

12.3.1 **Maze routing**

Perhaps the most widely used algorithm for finding a path between two points is the **maze-routing algorithm** (also called **Lee's algorithm**) [Lee 1961], which is based on the ***breadth-first-search*** (BFS) technique.

Maze routing adopts a two-phase approach of **filling** followed by **retracing**. The filling phase works in the "wave propagation" manner. Starting from the source node *S*, the adjacent grid cells are progressively labeled one by one according to the distance of the "wavefront" from *S* until the target node *T* is reached. Figures 12.6a and b illustrates the "wave propagation" when the labels of "wavefronts" reach 2 and 3, respectively. Once the target node *T* is reached, a shortest path is then retraced from *T* to *S* with decreasing labels during the retracing phase. Note that any such a path with decreasing labels gives a shortest path. However, we often prefer the one with the least detours for other practical concerns such as the number of bends (vias). Figure 12.7 illustrates the two phases of Lee's algorithm.

A nice property of Lee's algorithm is that it *guarantees to* find a path between two points if such a path does exist, and the path is the *shortest* one, even with obstacles. In practice, however, Lee's algorithm is slow and memory consuming. It has the time and space complexity of $O(mn)$, where

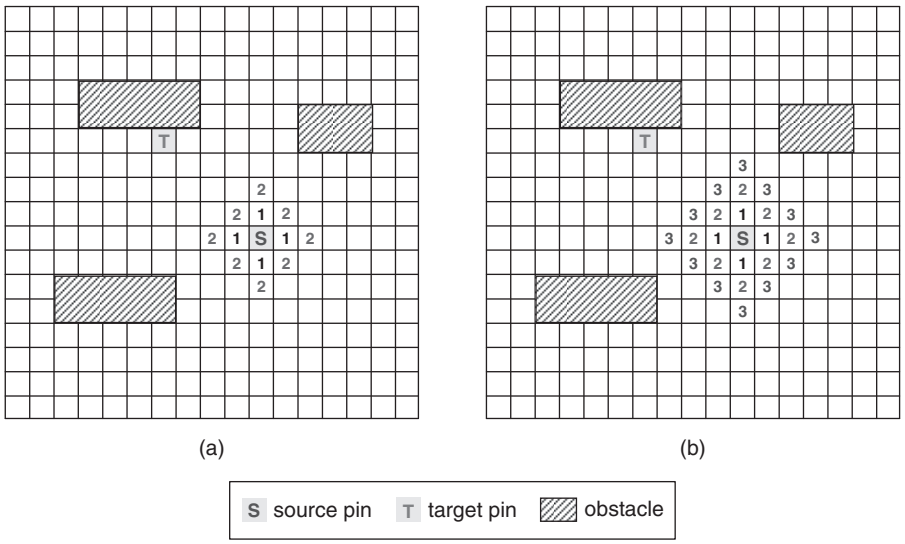

FIGURE 12.6

An example of the filling process: (a) The filling (wave propagation) when labels of the "wavefront" reach 2. (b) The next filling step of (a) when labels of the "wavefront" reach 3.

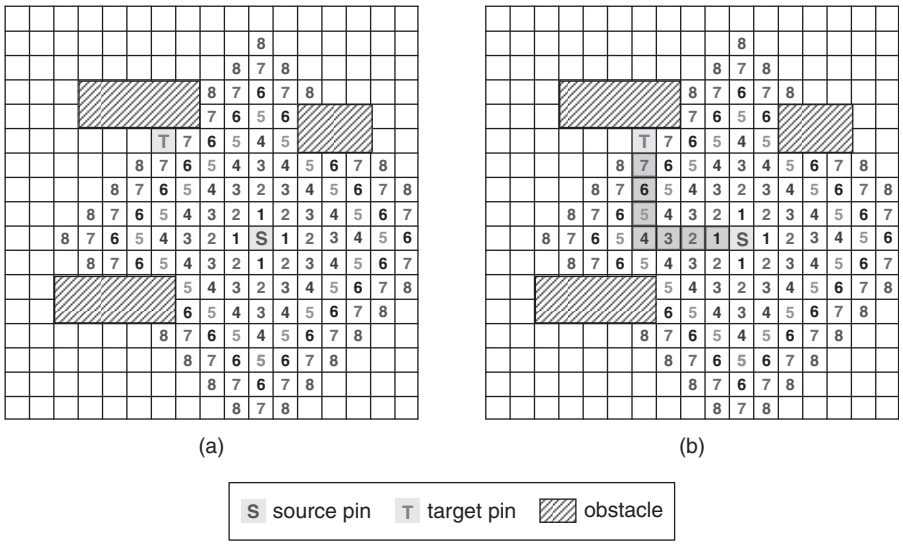

FIGURE 12.7

Lee's maze-routing algorithm: (a) The wave propagation phase. (b) The retracing phase.

m and n are the respective numbers of horizontal and vertical grid cells. Consequently, it is difficult to apply for large-scale dense designs directly.

Because of the pervasive use of Lee's maze-routing algorithm and its high time and space complexity, many methods have been proposed to reduce its running time and memory requirements. These popular optimization methods can be classified into three major categories: (1) coding scheme, (2) search algorithm, and (3) search space.

12.3.1.1 *Coding scheme*

Akers observed that adjacent labels for k are either $k-1$ or $k+1$ [Akers 1967]. To retrace the path, it suffices to have a labeling scheme such that each label has its preceding label different from its succeeding label. With the observations, Akers developed a 2-bit **coding scheme** to reduce memory requirement. The coding scheme uses 1 bit for filling by labeling the grid cells with the sequence 0, 0, 1, 1, 0, 0, 1, 1, ... In this way, for each label, its preceding label is different from its succeeding one, and thus retracing can work correctly. This coding scheme requires another bit to indicate whether a node is blocked or not. Therefore, this coding scheme needs only two bits for each grid cell to perform the maze routing. Another economical coding scheme is from [Hadlock 1977]; it uses the *detour* numbers for the labeling to reduce the search space and runtime.

12.3.1.2 *Search algorithm*

Soukup combined BFS and the ***depth-first-search*** (DFS) approaches to propagate wavefronts [Soukup 1978]. Depth-first (**line**) search is first directed from

the source *S* toward the target *T* until an obstacle or *T* is reached. BFS (as in Lee's algorithm) is then used to "bubble" around an obstacle if an obstacle is encountered. This algorithm has the same time and space complexity as that of Lee's algorithm, but is typically 10 to 50 times faster than Lee's algorithm. It can still find a path between *S* and *T* if such a path does exist, but it cannot guarantee a shortest path because of the DFS processing. Pure DFS (line-search) algorithms can further speed up the routing, at the cost of solution quality. See Section 12.3.2 for two line-search algorithms.

12.3.1.3 *Search space*

To reduce the running time for maze routing, techniques such as **starting point selection, double fanout**, and **framing** are in pervasive use [Sait 1999]. All the three techniques can substantially reduce the number of cells required to be labeled. The starting point selection is to choose the point closest to the chip boundary as the starting point for filling. In this way, we can discard more out-of-bound cells for labeling. Double fanout propagates waves from both the source and the target cells to reduce the area required for labeling. Framing searches only inside a rectangular region, say 10% larger than the bounding box formed by the source and the target. It needs to enlarge the rectangle and redo maze routing if the search fails. It is obvious that Lee's algorithm can no longer guarantee finding the shortest path with the framing heuristic.

12.3.2 Line-search routing

As mentioned earlier in Section 12.3.1, the major drawbacks of the maze-routing algorithm are the high memory use and long running time. The **line-search algorithm** alleviates these drawbacks by use of line segments to represent the routing space and paths at the cost of solution quality.

Mikami and Tabuchi proposed the first line-search algorithm (also called *line probe routing*) [Mikami 1968]. In contrast to the maze-routing algorithm, which mainly proceeds in a breadth-first manner, the line-search algorithm performs a depth-first search. The line-search algorithm initially sets the source *S* and the target *T* as **base points** and then generates four (two horizontal and two vertical) level-0 line segments passing through these base points. These line segments are extended until they hit the design boundary or obstacles. Then, each grid point of these line segments at level i are iteratively set as new base points, and a perpendicular line segment of level $i + 1$ is generated crossing each new base point. This process repeats until a segment generated from *S* intersects a segment generated from *T*, and a connection can then be found by tracing from this **intersection point** to both *S* and *T*. Figure 12.8 illustrates the Mikami-Tabuchi's line-search algorithm. The crossing points denote the base points, and the numbers denote the sequence of the search process. Like Lee's maze-routing algorithm, Mikami-Tabuchi's line-search algorithm also guarantees finding a path if one exists, but it may not always be the shortest. The line-search technique significantly reduces both memory requirements and execution times.

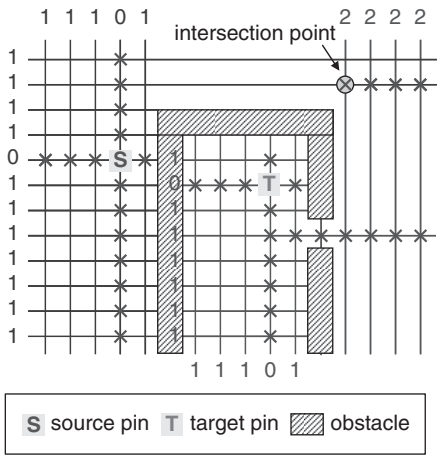

FIGURE 12.8

Mikami-Tabuchi's line-search algorithm.

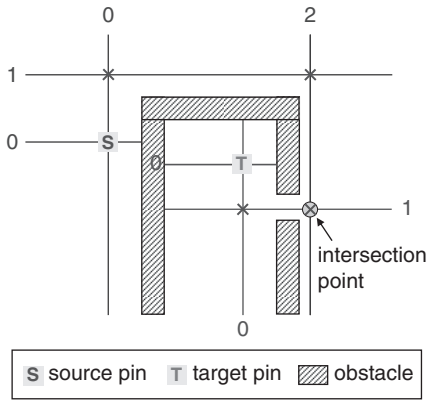

FIGURE 12.9

Hightower's line-search algorithm.

[Hightower 1969] proposed another line-search algorithm, which is similar to Mikami-Tabuchi's algorithm. The difference is that Hightower's algorithm only considers those line segments that are extendable beyond obstacles, and each line segment has at most two base points. Figure 12.9 illustrates Hightower's line-search algorithm. Because fewer line segments are considered, Hightower's algorithm has significantly more memory saving than Mikami-Tabuchi's algorithm. However, Hightower's algorithm might fail to find a path even if one exists. To remedy the deficiencies, it needs **backtracing** procedures to choose the right base points, and, therefore, the running time may not improve very much over Lee's maze-routing algorithm in practice.

12.3.3 **A*-Search routing**

As discussed in Section 12.3.1, the maze routing that adopts the BFS searching is generally slow, although it guarantees finding a shortest path. In the searching field, the maze search is also called **blind search,** because it searches the routing region in a blind way without any prioritized choices. Intuitively, if a router does not need to consider points that are not likely to be on the routing path, the running time would be improved.

In [Hart 1968], a general graph search algorithm called **A*-search** was proposed, which uses the function $f(x) = g(x) + h(x)$ to evaluate the cost of a path x, where $g(x)$ is the cost from the source node to the current node of x, and $h(x)$ is the *estimated* (or predicted) cost from the current node of x to the target node. Every time the algorithm selects a node with the lowest path cost to propagate (*i.e.,* the lower $f(x)$), the higher the priority for propagation. As a result, the A*-search is also called the **best-first search**, because at each decision making it first searches the routes that are most likely to lead toward the target. Note that generally speaking, the BFS is a special case of A*-search algorithm, where $h(x) = 0$ for all x.

The A*-search has a good property that if $h(x)$ is *admissible*, meaning that it never overestimates the actual minimal cost from the current node to the target node, then A*-search is optimal. Therefore, for the Manhattan routing (*i.e.*, only allow horizontal and vertical connections), $h(x)$ might be set as the Manhattan distance from the current node to the target, because it is the smallest possible distance between any two points in the Manhattan space.

The A*-search algorithm has many applications, such as in the field of ***artificial intelligence*** (AI). The **A*-search routing** introduced by [Clow 1984] for VLSI routing and [McMurchie 1995] for FPGA routing are pervasive in modern routers [Chao 2007; Pan 2007; Roy 2007; Chang 2008; Hsu 2008].

12.4 **GLOBAL ROUTING**

Traditional routing algorithms adopt the flat framework that finds paths for nets in the whole routing region directly. These algorithms can be classified into sequential and concurrent approaches, which are based on the general-purpose routing for 2-pin nets mentioned in Section 12.3 or a Steiner-tree algorithm for the multi-pin nets to be introduced in Section 12.4.3.

12.4.1 **Sequential global routing**

Perhaps the most straightforward strategy for routing is to select a specific net order and then to route nets sequentially in that order. However, this sequential approach often leads to a poor routing result, because an earlier routed net might block the routing for its subsequent nets. Therefore, the quality of the routing solution greatly depends on the net ordering.

Figure 12.10a illustrates a simple one-layer routing instance with two two-pin nets *A* and *B*. If we arbitrarily choose the net ordering as routing *A* first followed by B, net B might be blocked by net *A* and thus requires more longer wirelength to complete the routing (see Figure 12.10b). In contrast, if we route *B* first and then *A*, we can get a better routing result with shorter total wirelength (see Figure 12.10c). Therefore, it is desired to find a good net-ordering scheme for general routing instances. Unfortunately, such a universally good scheme is hard to find. In an earlier study, Abel concluded that there is no single net-ordering scheme that performs better than any other ordering scheme in all routing problems [Abel 1972], and finding the optimal net ordering has proven to be NP-hard, meaning that most likely no polynomial-time algorithm exists to solve this problem.

To remedy the deficiencies, today's sequential routing often applies a heuristic net ordering and conducts a **rip-up and reroute** process to further refine the solution. Here we give some popular net-ordering schemes: (1) Order the nets in the ascending order of the number of pins within their bounding boxes. If there are more pins inside the bounding box of a net, this net would tend to block the nets inside this bounding box. (2) Order the nets in the ascending (descending) order of their lengths if routability (timing) is the most critical metric. Research shows that routing shorter nets first often leads to better routability, because they usually have less routing flexibility than the longer ones. In this way, the shorter and straight nets would be routed without excessive detours, and the routing resource would be used more efficiently. In contrast, longer nets should be routed earlier for high-performance designs because they typically determine the overall timing. (3) Order the nets on the basis of their timing criticality. In addition to the net-ordering schemes, we can first analyze the net distribution over the routing region, identify the congested regions, and then route nets in the most congested regions first.

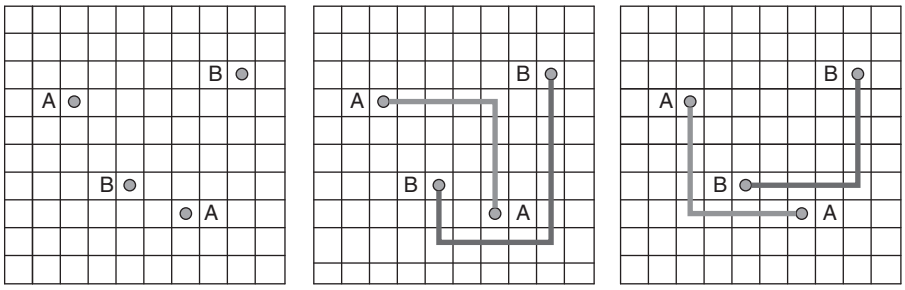

FIGURE 12.10

Routing based on different net orderings: (a) A one-layer routing instance with two two-pin nets *A* and *B*. (b) An inferior solution obtained by the net ordering of *A* followed by *B*. (c) A better solution resulted by the net ordering of *B* followed by *A*.

The rip-up and reroute process consists of two steps: (1) identify the bottleneck regions and rip up some already routed nets and (2) route the blocked connections and reroute the ripped-up connections. The process often performs iteratively until all nets are routed or a time limit is exceeded. Generally, it can lead to more desirable routing solutions. As the example of Figure 12.10b, if the router has observed that net *B* is blocked or its length is substantially increased because of net *A*, it can rip up net A, and reroute *B* and then *A* to improve the solution. McMurchie and Ebeling developed a **negotiation-based rip-up and reroute** algorithm called *PathFinder* for ***field-programmable gate array*** (FPGA) [McMurchie 1995], which reveals its superiority in recent leading academic global routers, such as *BoxRouter* [Cho 2007], *FastRoute* [Pan 2007], *FGR* [Roy 2007], *NTHU-Route* [Chang 2008], and *NTUgr* [Hsu 2008].

Chen *et al.* and Kastner *et al.* developed **pattern-routing** schemes [Ho 1990; Chen 1999; Kastner 2002] that use patterns such as **L-shaped** (1-bend) or **Z-shaped** (2-bend) routes to make connections (see Figure 12.11). The pattern routing gives the shortest path length between two points and enjoys very high speed and less memory use, because the search space followed by patterns is much smaller than the maze-routing algorithm. As a result, pattern routing is pervasively used for global-routing applications.

12.4.2 **Concurrent global routing**

The major drawback of the sequential approach is that it suffers from the net-ordering problem. Under any net ordering, it is more difficult to route the nets that are considered later, because they are subject to more blockages. In addition, if the sequential routing fails to find a feasible solution, it is not clear whether this is because of no existing feasible solution or because of a poor selection of net order. Moreover, when the sequential routing does find a feasible solution, we do not know whether or not this solution is optimal or how far it is from the optimal solution. These questions may be answered if we solve the routing problem with the concurrent approach.

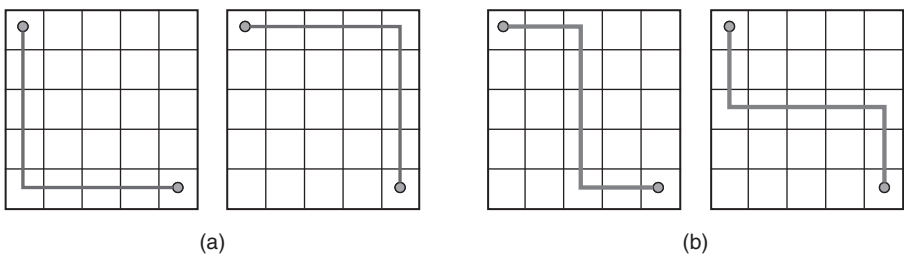

FIGURE 12.11

Pattern routing: (a) L-shaped (1-bend) routes. (b) Z-shaped (2-bend) routes.

One popular concurrent approach is to formulate global routing as a ***0-1 integer linear programming*** (0-1 ILP) problem. The layout is first modeled as a routing graph $G(V, E)$, where each node represents a tile and each edge denotes the boundary between two adjacent tiles. Each edge $e \in E$ is assigned a capacity, denoted by c_e, which represents the number of tracks crossing that boundary. Given a net, all of its possible routing patterns can be enumerated. Let the variable $x_{i,j} \in \{0,1\}$ indicate whether the routing pattern $r_{i,j}$ is selected from the set R_i of routing patterns of net n_i. Consequently, for a routing graph $G(V, E)$ with the netlist N, the congestion-driven global routing can be formulated as a 0-1 ILP problem as follows:

$$\begin{array}{lll} \text{Minimize} & \lambda & \\ \text{Subject to} & \sum_{r_{i,j} \in R_i} x_{i,j} = 1, & \forall n_i \in N \\ & x_{i,j} \in \{0,1\}, & \forall n_i \in N, \forall r_{i,j} \in R_i \\ & \sum_{i,j:e \in r_{i,j}} x_{i,j} \leq \lambda c_e, & \forall e \in E \end{array} \tag{12.1}$$

The first and the second constraints require that only one routing pattern can be chosen for each net, and the third constraint with the objective together ensure to minimize the maximum congestion. If a solution of $\lambda \leq 1$ exists, a global-routing solution with the maximum congestion being minimized can be achieved.

Because the 0-1 ILP is NP-complete, the high time complexity greatly limits the feasible problem size. An alternate approach to this problem is to first solve the continuous ***linear programming*** (LP) relaxation, obtained by replacing the second constraint with the real variable $x_{i,j} \in [0,1]$, because LP problems can be solved in polynomial time. Then, the resulting fractional solution can be transformed to integer solutions through rounding such as randomized rounding [Raghavan 1987]. However, this approximation could inevitably lose the optimality.

In practice, the 0-1 ILP concurrent routing technique is often embedded into a larger overall global routing framework with a hierarchical, divide-and-conquer manner, such as solving a subproblem, in which the complexity of computing the optimal solution is manageable. Another approach to divide a routing region into subregions such that the routing problem can be handled subregion by subregion to reduce the problem size is BoxRouter [Cho 2006], which is based on box expansion to push the congestion outward progressively.

12.4.3 Steiner trees

The algorithms we have described so far are mainly for two-pin nets. If all nets are two-pin ones, we can apply a general-purpose routing algorithm to handle the problem, such as maze, line-search, and A*-search routing described in Section 12.3.

For three or more multi-pin nets, one naive approach is to *decompose* each net into a set of two-pin connections, and then route the connections one-by-one. One popular decomposition method is to find a ***minimum spanning tree***

(MST) for pins of each net, which is a minimum-length tree of edges connecting all the pins. The MST can efficiently be computed in polynomial time by the **Kruskal** [Kruskal 1956] or **Prim-Dijkstra** [Prim 1957] algorithms. However, the routing result of this approach would depend on the decomposition and often leads to only suboptimal solutions. Figure 12.12 depicts an example 4-pin net decomposed by a rectilinear MST, where each segment runs horizontally or vertically.

A better and more natural method to route multi-pin nets is to adopt the **Steiner-tree–**based approach. Specifically, a ***minimum rectilinear Steiner tree*** (MRST) is used for routing a multi-pin net with the minimum wirelength. Given m points in the plane, an MRST connects all points by rectilinear lines, possibly via some extra points (called **Steiner points**), to achieve a minimum-wirelength tree of rectilinear edges. Let P and S denote the sets of original points and Steiner points, respectively. Then, we have the following relationship between MRST and MST.

$$\text{MRST}(P) = \text{MST}(P \cup S) \tag{12.2}$$

Figure 12.13b shows an example of the MRST with two Steiner points s_1 and s_2 for the four pins p_1, p_2, p_3, and p_4 in Figure 12.13a.

There could be an infinite number of Steiner points that need to be considered for the MRST construction. Fortunately, Hanan proved that for a set P of pins, there exists an MRST of P with *all* Steiner points chosen from the grid points of the **Hanan grid**, which is obtained by constructing vertical and horizontal lines through every pin in P. This is known as **Hanan's theorem** [Hanan 1966]. Figure 12.13c shows the Hanan grid for the four pins in Figure 12.13a. Both the Steiner points s_1 and s_2 of MRST in Figure 12.13b are on the grid points of the Hanan grid.

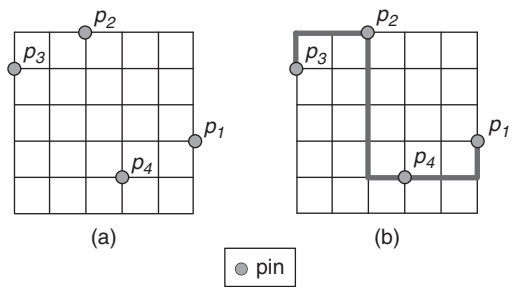

FIGURE 12.12

A 4-pin net decomposed by a minimum rectilinear spanning tree: (a) A net consisting of four pins: p_1, p_2, p_3, and p_4. (b) An MST of (a), which decomposes the net into three two-pin connections.

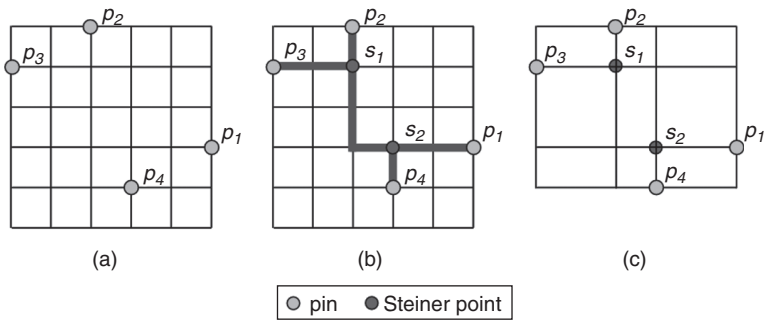

FIGURE 12.13

A minimum rectilinear Steiner tree (MRST) and its Hanan grid: (a) A net consisting of a set P of four pins: p_1, p_2, p_3, and p_4. (b) An MRST of (a) with the two Steiner points s_1 and s_2. (c) The Hanan grid of P. Note that all Steiner points s_1 and s_2 of MRST in (b) are chosen from the grid points on the Hanan grid.

The Hanan theorem greatly reduces the search space for the MRST construction from an infinite number of choices to only m^2-m candidates for the Steiner points, where $m=|P|$. However, the MRST construction is still an NP-hard problem [Garey 1977]. Therefore, many heuristics have been developed.

The relationship between MST and MRST can be stated by **Hwang's theorem** [Hwang 1976] as follows:

$$\frac{Wirelength(MST(P))}{Wirelength(MRST(P))} \leq \frac{3}{2} \tag{12.3}$$

Equation (12.3) gives a strong motivation for constructing an MRST by an MST-based approximation algorithm. Ho *et al.* constructed an MRST from an MST by maximizing monotonic (nondetour) edge (*e.g.*, L-shaped, Z-shaped) overlaps by dynamic programming [Ho 1990]. Kahng and Robins developed the **iterated 1-Steiner heuristic** [Kahng 1990] (see Algorithm 12.1). Starting with an MST, they iteratively select one Steiner point that can reduce the wirelength most and then add the Steiner point to the tree. The iterations continue until the wirelength cannot be further improved. Figures 12.14b–d illustrates the first, second, and third iterations after inserting Steiner points s_1, s_2, and s_3 into the initial MST in Figure 12.14a, respectively. Note that the iterated 1-Steiner heuristic may generate a "degenerate" Steiner point with the number of branches (degrees) $\leqq 2$, such as s_1 in Figure 12.14d. Therefore, we have to remove a degenerate Steiner point whenever it is created (see Figure 12.14e). Figure 12.14e shows the final MRST of Figure 12.14a.

On the basis of the **spanning graph** that contains an MRST in a sparse graph, [Zhou 2004] developed an efficient MRST algorithm with the worst-case time complexity of only $O(m \lg m)$ and solution quality close to that of the

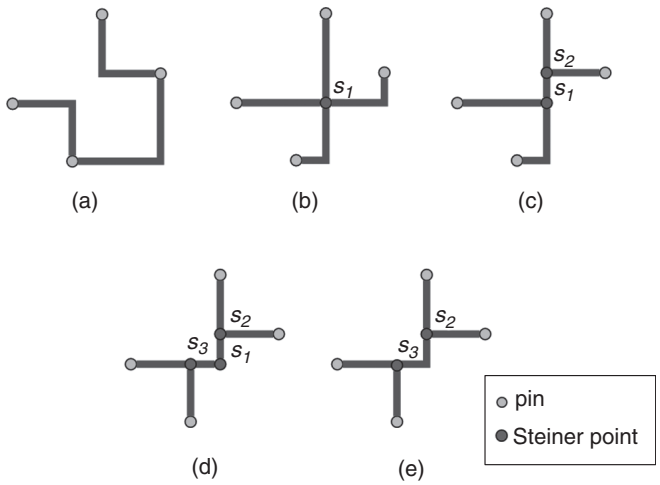

FIGURE 12.14

A step-by-step example of the iterated 1-Steiner heuristic for a 4-pin net: (a) The initial MST. (b) The MRST after the first iteration by inserting the Steiner point s_1. (c) The MRST after the second iteration by inserting the Steiner point s_2. (d) The MRST after the third iteration by inserting the Steiner point s_3. (e) The final MRST after removing the degenerate Steiner point s_1.

iterated 1-Steiner heuristic. [Chu 2004] developed the FLUTE package by use of precomputed lookup tables to efficiently and accurately estimate the wirelength for multi-pin nets. Lin *et al.* constructed a single-layer and a multi-layer obstacle-avoiding MRST to consider routing obstacles incurred from power networks, prerouted nets, IP blocks, and/or feature patterns for manufacturability/reliability improvements [Lin 2007, 2008]. Shi *et al*. constructed an obstacle-avoiding MRST based on a current-driven circuit model [Shi 2006].

Algorithm 12.1 Iterated 1-Steiner Algorithm

Input: P – a set of m pins.
Output: a Steiner tree on P.
1. $S \leftarrow \phi$;
   /*$H(P \cup S)$: set of Hanan points */
   /* $\Delta MST(A, B) = Wirelength(MST(A)) - Wirelength(MST(A \cup B))$ */
2. **while** ($Cand \leftarrow \{x \in H(P \cup S) \mid \Delta MST(P \cup S, \{x\}) > 0\} \neq \phi$) **do**
3. Find $x \in C$ *and* which maximizes $\Delta MST(P \cup S, \{x\})$;
4. $S \leftarrow S \cup \{x\}$;
5. Remove points in S which have degree ≤ 2 in $MST(P \cup S)$;
6. **end while**
7. **Output** $MST(P \cup S)$;

12.5 DETAILED ROUTING

Given global-routing paths, detailed routing determines the exact tracks and vias for nets. Here, we discuss the two most popular types of detailed routing: **channel routing** and **full-chip routing**.

In earlier process technologies when the maximum number of available metal layers was only two or three, channel routing was pervasively used, because most wires were routed in the free space (*i.e.*, **routing channel**) between a pair of logic blocks (cell rows); see Figure 12.15. In modern technologies, a chip typically contains six to ten metal layers, and the number of available metal layers is expected to increase steadily in the near future. With more metal layers, routing over the logic block (cell rows) is common (*i.e.*, **over-the-cell routing**). As a result, routing regions become more like channel-less regions. This trend drives the need of a full-chip routing method.

12.5.1 Channel routing

Channel routing is a special case of the routing problem in which wires are connected within the routing channels. To apply channel routing, a routing region is usually decomposed into routing channels. Note that there are often various ways to decompose a routing region. For example, Figure 12.16 shows two ways of decomposition for the T-shaped routing region. The routing region shown in Figure 12.16a is decomposed into one horizontal channel (channel 1) and one vertical channel (channel 2), whereas that in Figure 12.16b is decomposed into two horizontal channels (channels 1 and 2) and one vertical channel (channel 3).

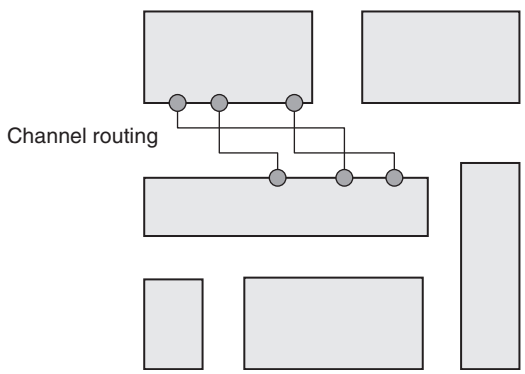

FIGURE 12.15

Channel routing between IC blocks.

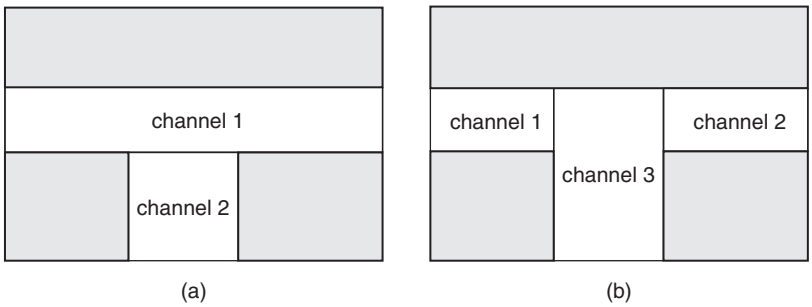

FIGURE 12.16

Two ways of routing region decomposition: (a) The routing region is decomposed into two channels. (b) The routing region is decomposed into three channels.

The order of routing regions significantly affects the channel-routing process. In Figure 12.16a, no conflicts occur in case of routing in the order of channel 2 and then channel 1. Instead, if channel 1 is routed first and all related wirings are fixed in the channel, channel 2 cannot be expanded if this channel cannot accommodate all the nets. In contrast, if channel 2 is routed first, we can still expand channel 1 for routing if needed. Note that it is not always possible to find a feasible channel ordering to avoid conflicts, for which we could resort to L-shaped channel routing to resolve the conflicts.

For modern chip routing, each routing layer typically has a preferred routing direction, either a horizontal or a vertical routing layer (*a.k.a.* **reserved routing model**). For example, the three-layer **HVH routing model** means that the preferred directions of the first, second, and third layers are horizontal, vertical, and horizontal, respectively. For the channel routing problem discussed in this section, we assume a two-layer HV routing model, unless stated otherwise.

We define some terminology of channel routing (see Figure 12.17 for an illustration). The inputs to a channel routing problem are two channel boundaries, the **upper boundary** and the **lower boundary**, with pin (terminal) numbers on columns of the channel boundaries. The pin number represents its unique net ID; pins of the same number belong to the same net and thus must be interconnected. The horizontal wire segments on the tracks are **trunks**, and the vertical wire segments connecting trunks to pins are **branches**. If the routing path of a net contains more than one trunk, this routing path is called a **dogleg**. The area of a routing channel is represented by the number of routing **tracks**, called **channel height**, inside the channel**.** Each column of a routing channel is associated with a **local density** to represent the total number of nets crossing the column. **Channel density**, the density of a routing channel, is then defined as the maximum local density inside the channel. It is obvious that channel density is a lower bound for the number of tracks required to complete the routing. The main objective of channel routing is to minimize the channel

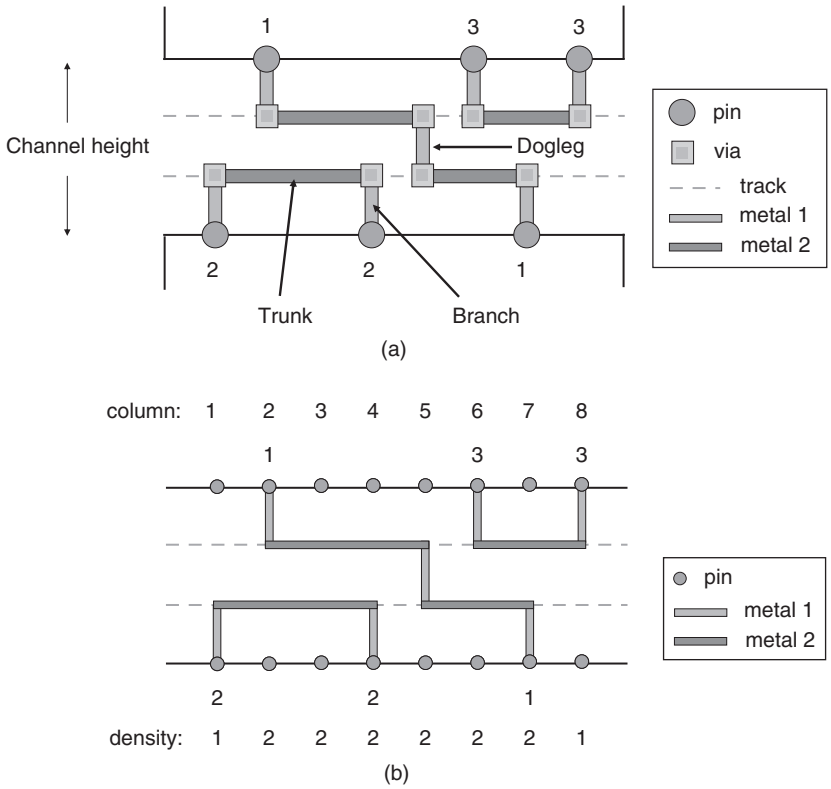

FIGURE 12.17

Channel routing illustration: (a) A channel routing configuration with two routing tracks. (b) A simplified illustration for (a).

height, which is directly related to the die size and thus the manufacturing cost. The general two-layer channel routing problem is NP-complete [Szymanski 1985], whereas some special cases of the problem can be solved optimally in polynomial time [Hashimoto 1971].

Figure 12.17a illustrates an example of two-layer channel routing that connects three nets with the pin numbers 1, 2, and 3, respectively. The channel height is two, and the connection of net 1 is a dogleg. For brevity, we would instead use the simplified illustration of Figure 12.17b throughout this chapter. As illustrated in Figure 12.17b, the routing channel contains eight columns with its local densities of 1, 2, 2, 2, 2, 2, 2, 1 for these columns (from left to right) and the channel density of 2.

To minimize the channel height, doglegs are commonly used to connect wire segments. For the same routing instance, the channel routing with doglegs shown in Figure 12.18b requires a channel height of only two tracks, whereas that without dogleg shown in Figure 12.18a needs four tracks to complete the routing.

In the following we introduce the **dogleg channel routing algorithm** [Deutsch 1976], which is an extension from the **constrained left-edge channel routing algorithm** [Hashimoto 1971]. The dogleg channel routing algorithm first decomposes multi-pin nets into two-pin connections and then assigns the trunk of each connection into a feasible track.

The dogleg channel routing algorithm contains three steps: (1) decompose each multi-pin net into 2-pin connections, (2) construct two constraint graphs to model the routing constraints, the ***horizontal constraint graph*** (*HCG*) and the ***vertical constraint graph*** (*VCG*), according to the locations of these connections, and (3) route each net without violating any constraints modeled in both HCG and VCG. As an example of the net decomposition, the 3-pin net 1 (represented by the interval [2, 7] because it spans from Column 2 to Column 7) is broken into two 2-pin connections, 1_a (interval [2, 5]) and 1_b (interval [5, 7]), as shown in Figure 12.19b.

The second step is to construct the *HCG* and *VCG* for the given routing instance. The *HCG* (*V, E*) is an *undirected* graph, where each node $v_i \in V$ represents a connection n_i, and an edge $(v_i, v_j) \in E$ exists if and only if a horizontal constraint exists between connections n_i and n_j (*i.e.*, the spans [intervals]) of n_i and n_j are overlapped) and thus n_i and n_j cannot share the same track or a circuit short would occur. In the example of Figure 12.19b, the spans of connections 2 and 4 ([1, 4] and [2, 4], respectively) are overlapped in the interval [2, 4], so there is a horizontal constraint in *HCG* between the nodes 2 and 4. Figure 12.19c depicts the *HCG* for the channel routing instance of Figure 12.19b. Note that there is no horizontal constraint between 1_a and 1_b, because they belong to the same net (net 1).

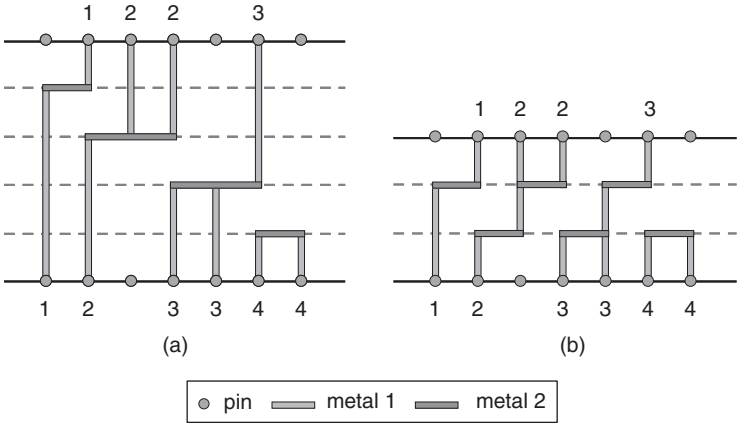

FIGURE 12.18

The effect of dogleg channel routing: (a) A channel routing solution without dogleg requires four tracks for routing completion. (b) A channel routing solution with dogleg only requires two tracks.

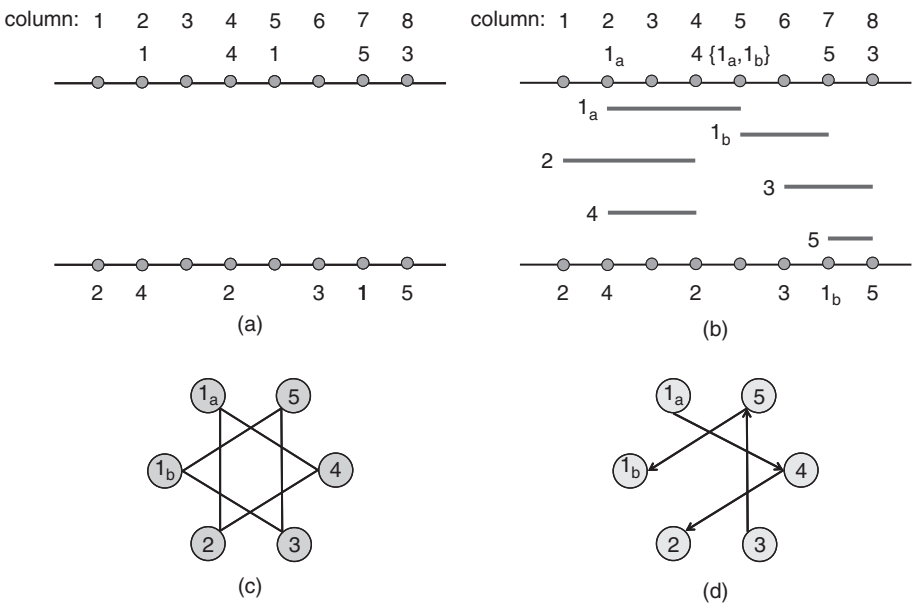

FIGURE 12.19

Constraint graph construction for dogleg channel routing: (a) A channel routing instance. (b) Multi-pin net decomposition. (c) The undirected horizontal constraint graph (HCG). (d) The directed vertical constraint graph (VCG).

The *VCG* (V, E) is a *directed* graph in which each node $v_i \in V$ represents a connection n_i, and a directed edge $(v_i, v_j) \in E$ exists if a vertical constraint exists between n_i and n_j (*i.e.*, the truck of n_i must be above that of n_j). The *VCG* can directly be constructed according to the pin locations in the upper and lower boundaries. For the example of Figure 12.19b, the pins in Column 4 of the upper and lower boundaries are 4 and 2, respectively; therefore, there is a directed edge (4, 2) in *VCG*. Figure 12.19d gives the *VCG* for the instance of Figure 12.19b.

The third step is to route each net under the constraints specified in both *HCG* and *VCG*. Suppose it routes nets to the routing tracks from top to bottom. In this step, the constrained left-edge algorithm [Hashimoto 1971] is applied. First, the algorithm treats each connection as an interval, and intervals are sorted according to their left-end x-coordinates. Then, the connections without any vertical constraint (*e.g.*, the nodes with zero in-degrees in the *VCG*) are routed one-by-one according to the order. For a connection, tracks in the channel are scanned from top to bottom, and the first track that can accommodate this connection is assigned to the connection. After all trunks (horizontal connections) are assigned to tracks, channel routing is completed by connecting the left ends and right ends of the trunks to the corresponding pins on the channel boundaries via branches. Note that the routing for a channel with no vertical

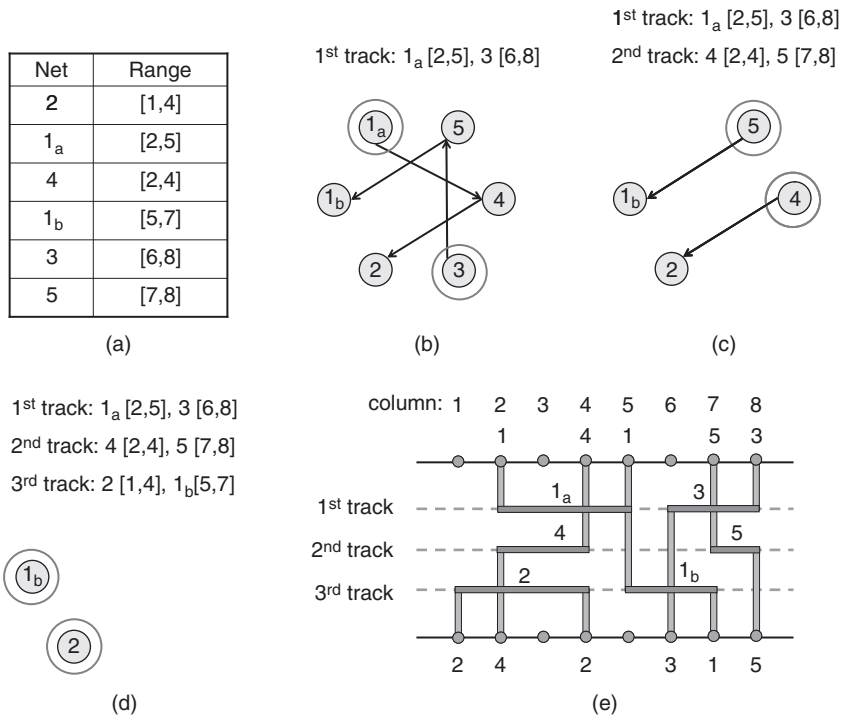

FIGURE 12.20

Dogleg channel routing for the instance of Figure 12.19a (unconstrained connections in the *VCG* are circled): (a) Connections are sorted by the left-end coordinates. (b) Connections 1_a and 3 are assigned one-by-one to the first track. (c) Connections 4 and 5 are assigned one-by-one to the second track. (d) Connections 2 and 1_b are assigned one-by-one to the third track. (e) The final routing solution with three tracks.

constraints (see the instance shown in Figure 12.17 for an example) can be solved optimally in polynomial time by the left-edge algorithm [Hashimoto 1971].

Figure 12.20 illustrates dogleg channel routing for the instance of Figure 12.19a, which has the channel density of three. Connections are first sorted as $<2, 1_a, 4, 1_b, 3, 5>$ according to their left-end coordinates (see Figure 12.20a). As shown in Figure 12.20b, there are two unconstrained connections 1_a and 3 in the *VCG*, and according to the order, 1_a and 3 are routed one-by-one. Both 1_a and 3 are assigned to the first track. Then the *VCG* is updated by deleting nodes 1_a and 3 and related edges (see Figure 12.20c). The resulting unconstrained connections in the *VCG* are 4 and 5. Similarly, 4 and 5 are routed one-by-one, and both trunks of 4 and 5 are routed on the second track. The *VCG* is then updated by deleting the nodes 4 and 5 and related edges (see Figure 12.20d). The resulting unconstrained connections in the *VCG* are 1_b and 2. Finally, 2 and 1_b are routed one-by-one, and both trunks

of 2 and 1_b are assigned to the third track. The final routing solution is then obtained (see Figure 12.20e) after connecting the left ends and right ends of each trunk to the pins on the corresponding channel boundaries via branches.

Note also that the dogleg channel routing algorithm introduced in [Deutsch 1976] applied two parameters to control the routing:

- Range: Determines the number of consecutive 2-pin connections of the same net that can be placed on the same track. This parameter would affect the number of doglegs and thus the number of vias.
- Routing sequence: Specifies the starting position and the direction of routing along the channel. The dogleg channel router assigns connections to the routing tracks from top to bottom, from bottom to top, or alternately with the two directions. Different routing sequences might result in different routing solutions. Note that the connections without any vertical constraint correspond to the nodes with zero out-degrees in the *VCG* if the routing sequence is from bottom to top.

12.5.2 Full-chip routing

Full-chip routing is typically a very complex combinatorial problem. To make it manageable, many routing algorithms adopt a two-stage technique of global routing followed by detailed routing. However, the continuously increasing design complexity imposes severe challenges for modern routers. The traditional **flat framework** does not scale well as the design size increases. A modern chip may contain billions of transistors and millions of nets. To cope with the scalability problem, routing frameworks are evolving, and the **hierarchical** and **multilevel frameworks** have become more and more popular for large-scale designs.

The hierarchical routing framework uses the divide-and-conquer approach by transforming a large and complicated routing problem into a series of smaller and simpler subproblems and then proceeds in a *top-down*, *bottom-up*, or *hybrid* manner, which can be applied to both global and detailed routing.

A top-down hierarchical global-routing framework has been proposed in [Burstein 1983]. The algorithm recursively divides the routing regions into successively smaller subregions, named **super cells**, and nets at each hierarchical level are routed sequentially or concurrently and are refined in the subsequent levels. Figure 12.21 illustrates an example of global routing for a 3-pin net by the top-down hierarchical approach, in which the routing region is recursively bisected into smaller super cells, and at each level, the net is routed in terms of these super cells at that level. This process is performed in a top-down manner until the sizes of super cells reduce to that of global-routing tiles.

A bottom-up hierarchical routing method is developed in [Marek-Sadowska 1984]. Initially, the routing region is partitioned into an array of super cells. At each hierarchical level, the routing is restrained within each super cell

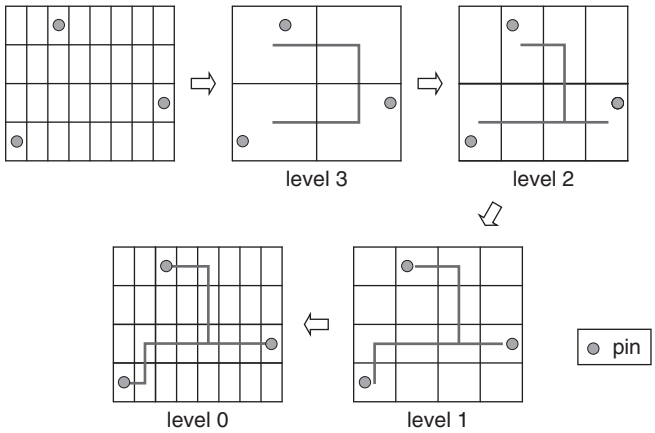

FIGURE 12.21

A level-by-level top-down hierarchical routing approach for a 3-pin net.

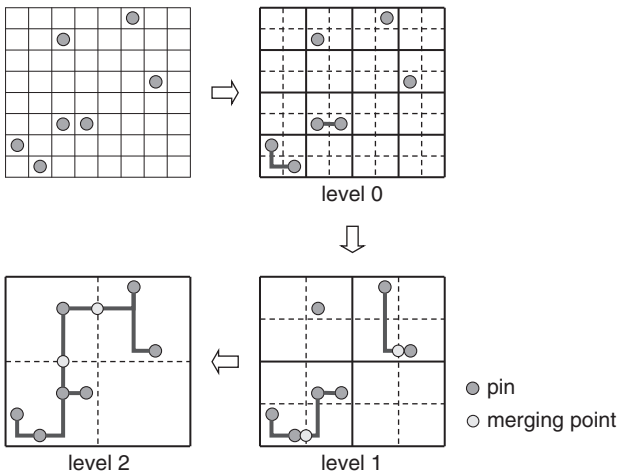

FIGURE 12.22

A level-by-level bottom-up hierarchical routing approach for a 7-pin net.

individually. When the routing at the current level is finished, every four super cells are merged to form a new larger super cell at the next higher level. This process continues until the top level containing the whole chip is reached. Figure 12.22 shows the process of bottom-up hierarchical routing for a 7-pin net, in which each solid rectangle represents a super cell, and the 2*2 dotted subregions of the previous level are merged together.

A major limitation in the top-down and the bottom-up hierarchical approaches is that the routing decision made at one hierarchical level may be suboptimal for

subsequent levels. To alleviate this problem, Lin, Hsu, and Tsai proposed a hybrid hierarchical approach that combines the bounded maze-routing algorithm with both the top-down and bottom-up hierarchical methods into a unified routing framework [Lin 1990]. Their algorithm consists of three phases: (1) neighboring propagation, (2) preference partitioning, and (3) bounded routing.

Phase 1 performs bounded maze routing by propagating W circles of waves out of each pin, where W is a user-defined parameter. If the connection is not found, Phase 2 recursively maps the pins and blockages onto the adjacent upper level (see Figure 12.23a) and calls the bounded maze-routing algorithm until a path is found. Then, the connected path is mapped back to the lower level to *preferred regions* (see Figure 12.23b). Phase 3 finds a routing path in the preferred regions (see Figure 12.23c). Compared with pure top-down or bottom-up hierarchical routing, the hybrid hierarchical approach has more global information to generate better routing solutions.

Although the hierarchical routing approach can scale to larger designs, it has the significant drawbacks that the interactions among different routing subregions are lacking and the routing decision at a level is irreversible (*i.e.*, cannot be refined at later stages), thus limiting the solution quality. To remedy the deficiencies, researchers have proposed the **multilevel framework** to handle large-scale routing problems. The multilevel frameworks were first developed in [Cong 2001, 2002] for global routing and in [Lin 2002] and [Chang 2004] for both global and detailed routing. In the following, we introduce the routability-driven Λ-**shaped multilevel routing framework** [Chang 2004].

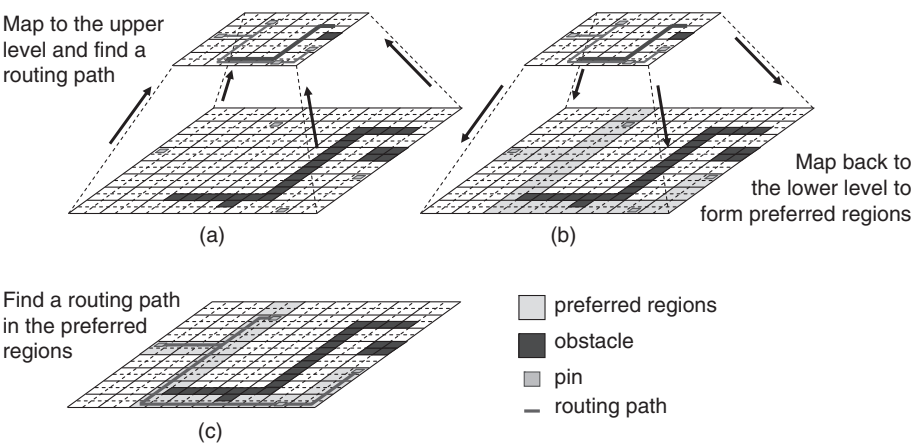

FIGURE 12.23

An example of global routing by use of the hybrid hierarchical approach: (a) Mapping pins and blockages up one level and then finding a routing path at the upper level. (b) Mapping the connection at the upper level to the lower level to form the preferred regions. (c) Finding a routing path in the preferred regions.

The multilevel routing framework models the routing resource as a **multilevel-routing graph**. At the beginning, the routing region is partitioned into an array of rectangular subregions, each of which may accommodate tens of routing tracks in each dimension (see Figure 12.24). These subregions are called ***global cells*** (*GCs*). A node in the routing graph represents a *GC* in the chip, whereas an edge denotes the boundary between two adjacent *GCs*. Each edge is assigned a capacity according to the physical area or the size of a *GC*. This routing graph is called the multilevel-routing graph of level 0, denoted by G_0, in which the subscript represents the level.

The Λ-shaped multilevel routing framework consists of bottom-up **coarsening** followed by top-down **uncoarsening**. The coarsening stage is a bottom-up approach that iteratively groups a set of *GCs* in the multilevel-routing graph. This process starts from the finest level (level 0) to the coarsest level; at each level k, four adjacent GC_k of G_k are merged into a larger GC_{k+1} of G_{k+1}, and at the same time it performs resource estimation for use at the $k+1$ level. Coarsening continues until the number of *GCs* at a level is below a threshold. In contrast, the uncoarsening stage iteratively ungroups a set of previously clustered *GCs* in a top-down manner. It proceeds from the coarsest level to the finest level; at each level k, a GC_k is decomposed into four smaller GC_{k+1}. Uncoarsening continues until the finest level is reached. Figure 12.25 illustrates the Λ-shaped multilevel framework.

Given a netlist, the multilevel routing first applies a minimum spanning tree (MST) algorithm to decompose each net into 2-pin connections. At each level k of the coarsening stage, global routing is first performed for the **local** 2-pin connections (those connections that entirely sit inside a GC_k), and then the detailed router is used to determine the exact wiring. Let the multilevel-routing graph of level 0 be $G_0 = (V_0, E_0)$, and the global-routing result for a local connection be $Re = \{e \in E_0 | e \text{ is the edge chosen for routing}\}$. For the congestion control, the cost function $\alpha : E_0 \rightarrow \Re$ is applied to guide the routing:

$$\alpha(R_e) = \sum_{e \in R_e} c_e \tag{12.4}$$

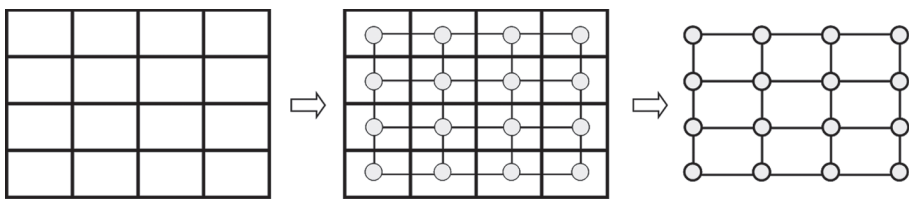

FIGURE 12.24

The multilevel-routing graph.

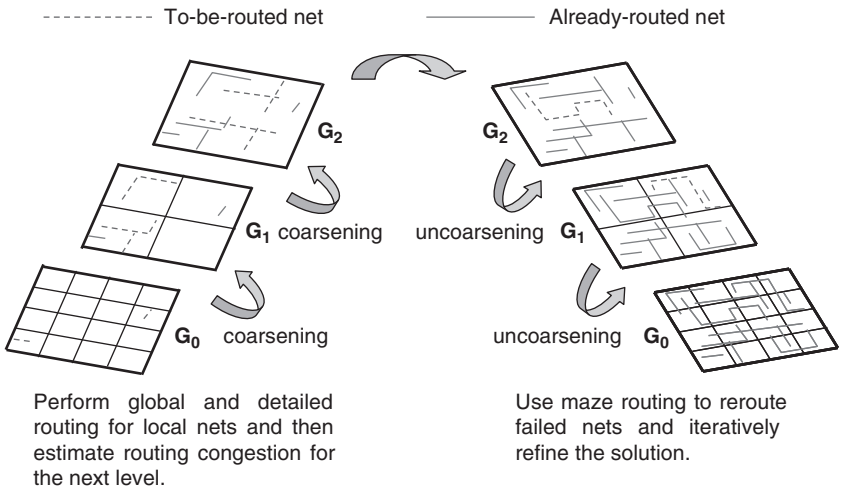

FIGURE 12.25

The Λ-shaped multilevel routing framework.

where c_e is the congestion of edge e and is defined by

$$c_e = 1/2^{(p_e - d_e)} \tag{12.5}$$

where p_e and d_e are the capacity and density associated with e, respectively. Note that we always search the shortest global-routing path between two pins in the coarsening stage therefore (*i.e.*, monotonic routes or no detours); therefore, the wirelength is the minimum, and thus the wirelength is not included in the cost function at the global routing stage. This cost function can guide the global router to select a path with smaller congestion.

After the global routing is completed, the detailed routing applies a ***simultaneous pathlength and via minimization*** (SPVM) algorithm to perform modified maze routing that simultaneously considers the pathlength and via minimization. For better circuit performance, it is desirable to minimize the number of vias used in a routing path, because vias typically have significantly larger RC delay than metal wires. The SPVM algorithm can find a shortest path with the minimum number of bends/vias, if such a path exists. It associates each *basic detailed routing region* u (could be a grid cell in grid-based routing or a basic routing region defined by the wire pitch in gridless routing) with two labels $d(u)$ and $b(u)$, where $d(u)$ is the distance of the shortest path from the source s to u, and $b(u)$ is the minimum number of bends/vias along the shortest path from s to u.

Initially, $d(s)$, $b(s) = 0$, and $d(u)$, $b(u) = \infty$, $\forall\ u \neq s$. In the filling phase of maze routing, the computation of label d is the same as the original maze-routing algorithm. Let u be a basic routing region on the wavefront of wave propagation and v a neighboring basic-routing region of u. The predecessor

routing region of u is the region from which the wavefront was propagated for obtaining the minimum $b(u)$. The propagation direction of u is the direction from the predecessor routing region of u to u. The computation of $b(v)$ is shown in Algorithm 12.2.

Algorithm 12.2 Computation of *b(v)* in the SPVM Algorithm

```
1. if (d(v) ≥ d(u) + 1) do
2.   if ((b(v) > b(u)) and
         (v is along the propagation direction of u)) do
3.     b(v) ← b(u);
4.     Record u as the predecessor routing region of v;
5. end if
6. if ((b(v)  > b(u) + 1) and
         (v is not along the propagation direction of u)) do
7.     b(v) ← b(u) + 1;
8.     Record u as the predecessor routing region of v;
9.   end if
10. end if
```

The basic idea is to compare the distance label d first and then compare the bend/via number label b. The value $b(v)$ of a neighboring routing region v with $d(v) < d(u)$ remains unchanged, because the path from s through u to v is not the shortest path between s and v. The retracing phase is the same as that of the original maze-routing algorithm. Note that there may be several shortest paths with different numbers of bends/vias. The wave-propagation phase always keeps track of the shortest path with the minimum bend/via number to allow the retracing phase to find such a path.

When the global and detailed routing is performed at level k, four adjacent GC_k are merged into a larger GC_{k+1} and at the same time resource estimation is performed for use at the next level $k + 1$. Because the global routing, detailed routing, and resource estimation are integrated together at each level, the routing resource estimation is more accurate, thus facilitating the solution refinement (*e.g.*, the rip-up and reroute processes) at the uncoarsening stage. Algorithm 12.3 gives the algorithm of the Λ-shaped multilevel routing framework [Chang 2004].

12.6 MODERN ROUTING CONSIDERATIONS

As the process geometries scale down to the nanometer territory, the IC industry faces severe challenges in **signal integrity, manufacturability**, and **reliability**. In this section, we address the routing problems considering these issues. Specifically, we discuss **crosstalk** for signal integrity-aware routing,

Algorithm 12.3 Λ-Shaped Multilevel Routing Algorithm

Input: *G* – partitioned layout;
N – netlist of multi-terminal nets.
Output: routing solutions for *N* on *G*

```
1. partition the layout and build MST's for N;
   //coarsening stage
2. for (each level at the coarsening stage) do
3.   Choose a local net n;
4.   if (n belongs to this level) do
5.     Global_Pattern_Routing(n);
6.     Detailed_Routing(n);
7.   end if
8. end for
   // uncoarsening stage
9. for (each level at the uncoarsening stage) do
10.  Choose a local net n;
11.  Global_Maze_Routing(n);
12.  Detailed_Routing(n);
13. end for
14. Output_Result();
```

optical proximity correction (OPC) and ***chemical-mechanical polishing*** (CMP) for manufacturability-aware routing, and **antenna effect** avoidance and **double-via insertion** for reliability-aware routing.

12.6.1 Routing for signal integrity

As the fabrication technology advances, on-chip minimum feature sizes continue to decrease, clock rates keep increasing, and devices and interconnection wires are placed in closer proximity to reduce interconnection delay and routing area. Consequently, increasing the aspect ratios of wires and decreasing interconnect spacing make the **coupling capacitance** larger than self-capacitance. In fact, the ratio of coupling capacitance is reported to be even as high as 70% to 80% of the total wiring capacitance, even in the 0.25-μm technology. As a result, **crosstalk** becomes a key issue for signal integrity.

12.6.1.1 *Crosstalk modeling*

Noise is an unwanted variation that makes the behavior of a manufactured circuit deviate from the expected response. The deleterious influences of noise can be classified into two categories. One is malfunctioning, which makes the logic values of gates differ from what we desire; the other is timing

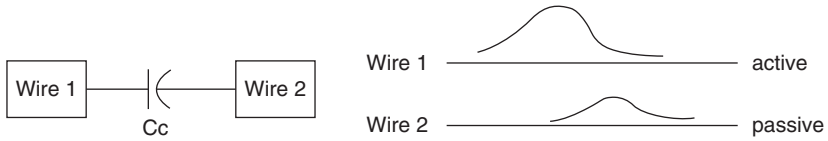

FIGURE 12.26

The crosstalk effect.

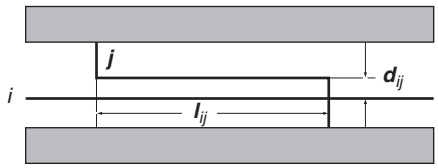

FIGURE 12.27

The capacitive crosstalk computation between two wires *i* and *j*.

change, which is caused by switching behavior. The main noise comes from the **crosstalk** effect, which is mostly caused by the coupling capacitance between interconnection wires. As an example shown in Figure 12.26, because of the coupling capacitance C_c between wires 1 and 2, wire 2 would induce an undesirable pulse when wire 1 is activated by a positive signal. If the unexpected pulse is larger than a threshold, the functionality of the circuit may fail. More precisely, the crosstalk between two wires switching in different directions would increase signal delays and decrease signal integrity; on the contrary, the crosstalk would decrease signal delays and increase signal integrity if the two wires switch in the same direction.

In general, the crosstalk between two wires is proportional to their coupling capacitance, which is determined by the relative positions of these wires. The coupling capacitance between orthogonal wires is negligible compared with that between adjacent parallel wires in current technology. Consequently, the crosstalk can be approximated by considering only adjacent parallel wires.

Figure 12.27 illustrates an instance with two wires i and j belonging to different nets. The coupling capacitance c_{ij} between i and j can be approximated as follows [Sakurai 1983]:

$$c_{ij} = a\frac{l_{ij}}{(d_{ij})^k} \tag{12.6}$$

where a is a technology-dependent constant, k is a constant between 1 and 2 (and close to 2), l_{ij} is the overlapping length of wires i and j, and d_{ij} is the distance between wires i and j. On the basis of Equation (12.6), we can see that

the coupling capacitance between two parallel wires is proportional to their coupling length and is inversely proportional to the distance between them. More accurate crosstalk modeling can be found in [Vittal 1999; Jiang 2000; Cong 2001].

12.6.1.2 *Crosstalk-aware routing*

Routing with minimum crosstalk has been extensively studied in the literature [Gao 1996; Zhou 1998; Ho 2005, 2007]. Gao and Liu applied a mixed ILP (integer linear programming) formulation to permute the routing tracks in a given channel routing solution to minimize crosstalk [Gao 1996]. Zhou and Wong minimized crosstalk during global routing on the basis of a Steiner tree formulation and Lagrangian relaxation [Zhou 1998]. Chaudhary, Onozawa, and Kuh proposed a wire-spacing adjusting algorithm after detailed routing to reduce crosstalk [Chaudhary 1993]. However, it might not be easy to handle crosstalk during global routing or detailed routing. It might be too early to handle crosstalk during global routing, because the relative positions and ordering of nets are not determined at this stage; consequently, the best that one can possibly do is to use rough statistical estimators that discourage nets from entering unwanted proximity regions. Conversely, it might be too late for detailed routing to handle crosstalk, because detailed routers may encounter unsolvable rip-up/re-route problems when trying to embed a late-routing net into a dense region with conflicting aggressor or victim nets.

To address these problems, Ho *et al.* incorporated a **layer/track assignment** heuristic for crosstalk optimization in the intermediate stage of the Λ-shaped multilevel routing framework [Ho 2005], as shown in Figure 12.28.

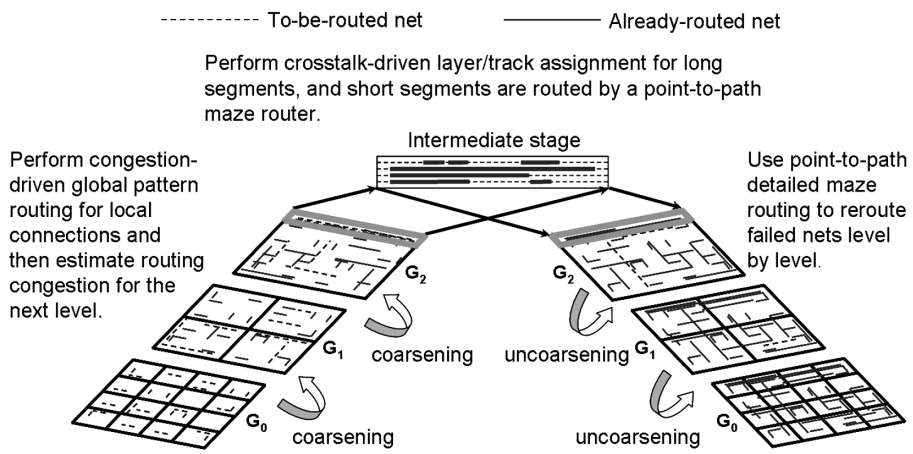

FIGURE 12.28

The Λ-shaped multilevel routing framework with an intermediate stage for crosstalk minimization.

The layer/track assigner works on a full row or column of the global cell array at a time, where a row (column) is called a ***panel***. In the layer/track assignment, the segments spanning more than one complete global cell in a row or a column are processed, and short segments are routed during detailed routing.

First, a horizontal constraint graph $HCG(V, E)$ is built for all segments in the panel. Each vertex $v \in V$ corresponds to a segment in the panel. Two vertices v_i and v_j are connected by an edge $e \in E$ if and only if these segments belong to two different nets and their spans overlap. The edge cost of $e = (v_i, v_j) \in E$ represents the coupling length if v_i and v_j are assigned to adjacent tracks. The crosstalk-driven layer assignment can be formulated as the max-cut, k-coloring (MC) problem. However, the general MC problem is NP-complete [Garey 1979]. Thus, a simple yet efficient heuristic is applied by constructing a maximum spanning tree of *HCG* followed by the k-coloring method to spread all segment into k layers. After k-coloring, the nodes are assigned to layers one-by-one in a decreasing order of their costs (coupling lengths).

After the crosstalk-driven layer assignment, the crosstalk-driven track assignment is applied. Let T be the set of tracks inside a panel. Each track $\tau \in T$ can be represented by the set of its constituent contiguous intervals. Denote these intervals by x_i. A segment $r \in S$ (set of segments) is said to be **assignable** to $\tau \in T$, $\tau \equiv \cup x_i$, if x_i is either a free interval or is an interval occupied by a segment of the same net.

After layer assignment, most of the edges with larger costs in an *HCG* are eliminated, and the *HCG* is decomposed into k subgraphs $subHCG_1$, $subHCG_2, \ldots,$ $subHCG_k$ if there are k layers. Figure 12.29 shows an example of the track assignment problem for a *subHCG*, where $S = \{a, b, c, d, e, f\}$, $T = \{1, 2, 3, 4\}$, and obstacles on tracks are shaded in grey (*e.g.*, the two obstacles on tracks 3 and 4). A bipartite assignment graph is used to indicate the assignability of segments to tracks. For example, as shown in Figure 12.29b, edges between node a and nodes 1, 2, and 3 are introduced, because segment a can be assigned to track 1, 2, or 3, but not track 4. For easier implementation, the *subHCG* and the bipartite assignment graph are merged into a combination graph, as shown in Figure 12.29c.

Because each vertex $v \in V$ corresponds to a segment and each edge $e \in E$ corresponds to the coupling cost in $HCG(V, E)$, the crosstalk-driven track assignment can be formulated as the Hamiltonian path problem which is NP-complete [Garey 1979]. Here is a heuristic for this problem. The heuristic starts by finding the maximal sets of conflicting segments. This is equivalent to finding the largest clique V_c in the subgraph $subHCG_i$. The algorithm first assigns one maximal subset of conflicting segments at a time by starting from the largest clique. Then the longest segment in the clique is chosen as the source s and assigned to the uppermost available track. Then, the minimum-cost edge (s, i) (and thus the minimal coupling) is chosen, and the segment associated with i is assigned to the first available track. If all tracks are occupied, the net associated with i is marked as a failed net that will be reconsidered at the uncoarsening stage.

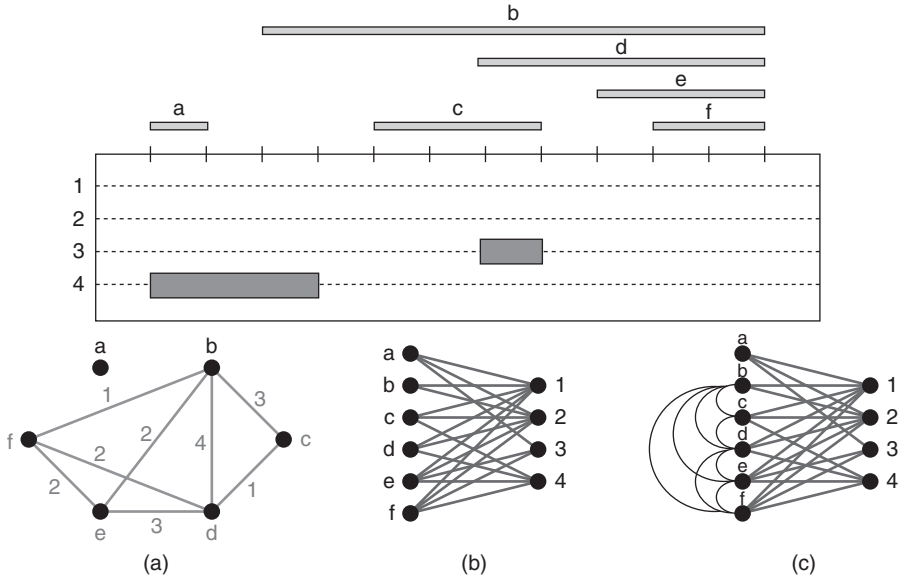

FIGURE 12.29

Constraint graph modeling for track assignment: (a) *SubHCG* for a given instance. (b) The corresponding bipartite assignment graph. (c) The combination graph.

The procedure is repeated by finding the minimum-cost edge (i, j) for further processing, where j is an unvisited node.

Figure 12.30 illustrates the track assignment process for the instance of Figure 12.29. The maximum clique in the *subHCG* is $\{b, d, e, f\}$, and the longest segment in the clique is b. Thus, the segment b is assigned to the uppermost available track, which is track 1. See Figure 12.30b for the updated combination graph after assigning b to track 1. Then, the heuristic makes b the source for constructing the Hamiltonian path for the clique. The minimum-cost edge $e = (b, f)$ incident on b is chosen, and f is assigned to the first available track. See Figure 12.30c for the updated combination graph after assigning f to track 2. The process is repeated until all nodes in the clique are visited. The final track assignment solution is shown in Figure 12.30a.

12.6.2 Routing for manufacturability

For manufacturability, OPC and CMP are two most important concerns for modern chip designs. The former adds or subtracts feature patterns to a **mask** to enhance the layout resolution and thus the **printability** of the mask patterns on the wafer, whereas the latter improves layout uniformity and chip planarization to achieve higher manufacturing yield.

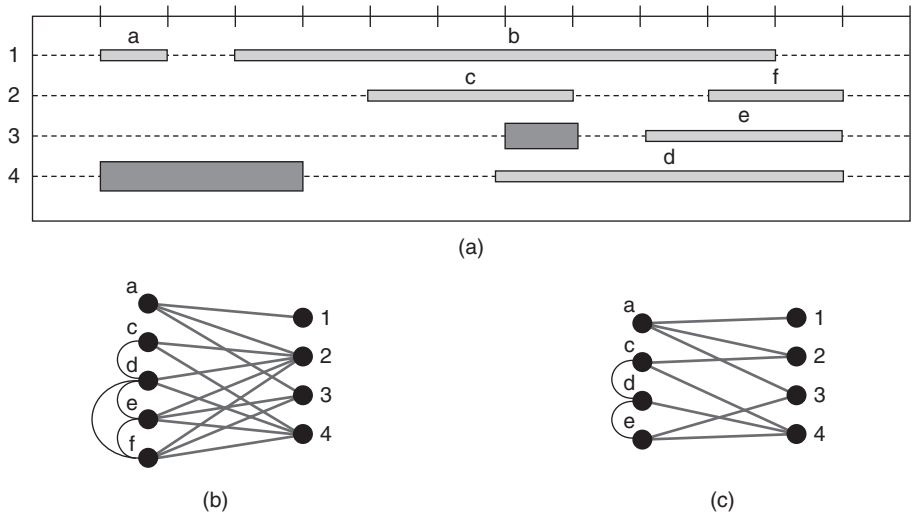

FIGURE 12.30

Process of track assignment: (a) Final track assignment for the instance of Figure 12.29. (b) The resulting combination graph after assigning *b* to track 1. (c) The resulting combination graph after assigning *f* to track 2.

12.6.2.1 *OPC-aware routing*

We will first introduce the manufacturing process. The process uses an **optical lithography system** and goes through many cycles of processing, each of which consists of two major steps: **exposure** followed by **etching**.

Figure 12.31 illustrates a basic optical lithography system. In the exposure step, it transfers the patterns on a **mask** to the light-sensitive **positive** or **negative photoresist** coated on the top of the wafer, which is performed by an intense ultraviolet light emitted from the light source through the apertures of the mask. Exposed by the light, the positive photoresist becomes soluble to the photoresist developer, whereas the negative photoresist becomes insoluble. This chemical change allows some of the photoresist to be removed by a special solution. In the etching step, a chemical agent removes the uppermost layer of the wafer in the areas that are not protected by photoresist to form the designed patterns on the wafer.

With the continuous shrinking of the minimum feature size, IC foundries have to use an optical lithography system with a larger wavelength of light to print a feature pattern with a much smaller size on a wafer, which is called the **sub-wavelength lithography gap** (see Figure 12.32). For the modern process technology, for example, we might need to print a 45-nm feature pattern by use of the light of 193-nm wavelength. The sub-wavelength lithography gap might lead to unwanted large shape *distortions* for the printed patterns on

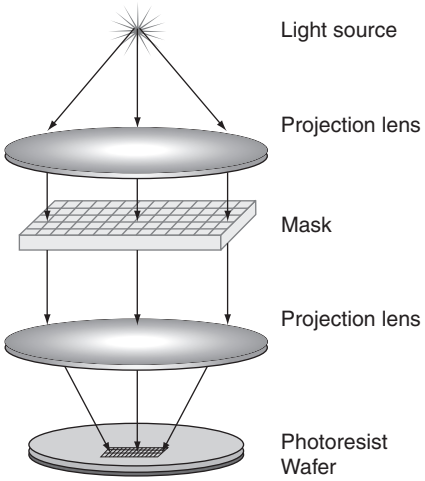

FIGURE 12.31

A typical optical lithography system.

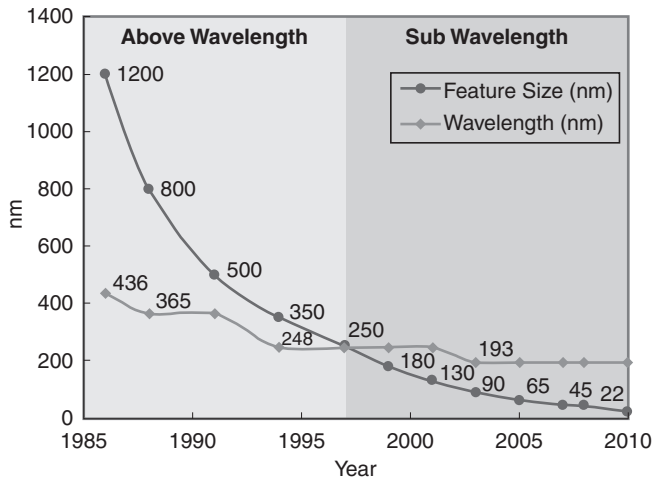

FIGURE 12.32

The sub-wavelength lithography gap: the printed feature size is smaller than the wavelength of the light shining through the mask.

the wafer. Physically, when a light with the wavelength λ passes through an aperture of the size d, the wavefronts of the light behave differently according to the relation between λ and d. When λ is much smaller than the aperture size d on the mask, the wavefronts of the light remain straight, as illustrated in Figure 12.33a. However, when λ is close to or larger than d, the light behaves

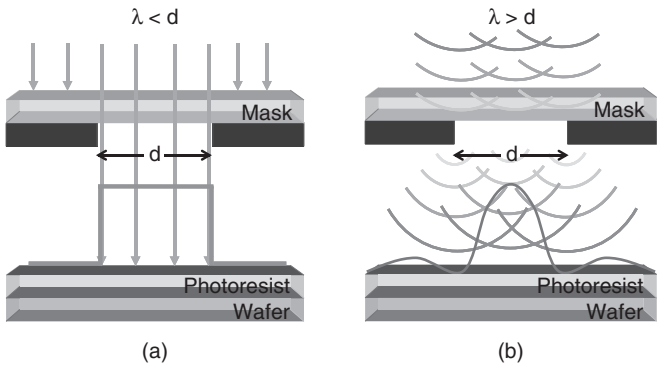

FIGURE 12.33

When a light with the wavelength λ passes through an aperture of the size *d*, the wavefronts of the light behave differently according to the relation between λ and *d*: (a) When λ is much smaller than *d*, the wavefronts remain straight. (b) When λ is much larger than *d*, diffracted wavefronts might occur.

like *waves* (instead of particles) and diffraction occurs (see Figure 12.33b), making the pattern on the wafer not exactly the same as that on the mask. As a result, intensive use of costly ***resolution enhancement techniques*** (RETs) to improve the layout accuracy becomes inevitable.

Many RETs are adopted at the post-layout stage to enhance the printability and thus the yield. The increasing design complexity, however, leaves very limited space for post-layout optimization. Therefore, it is desirable to consider the manufacturability earlier in the design flow, such as RET-aware routing.

Among the RETs, ***optical proximity correction*** (OPC) is the most popular in industry. OPC is the process of modifying the layout patterns on the mask (drawn by the designers) to compensate for the non-ideal properties of the lithography process and thus to enhance the layout printability. Figure 12.34 illustrates an example of OPC enhancement. Without OPC, the printed patterns on the wafer would be distorted from the designed pattern on the mask because of the sub-wavelength lithography. In contrast, if the patterns on the mask are enhanced by OPC, the printed patterns on the wafer could well match the original designed patterns.

However, OPC might incur a large number of extra pattern features, implying larger memory requirements to record these features and thus higher mask-making costs, such as mask synthesis, writing, and inspection verification. If a router can consider the optical effects, the number of pattern features on the final mask can greatly be reduced.

Chen and Chang proposed a **rule-based OPC-aware multilevel router** to reduce the requirements for OPC-pattern feature [Chen 2007a]. They classify the pattern distortions into three major types: **corner rounding, line-end shortening**, and **line-width shrinking**, as illustrated in Figures 12.35a–c.

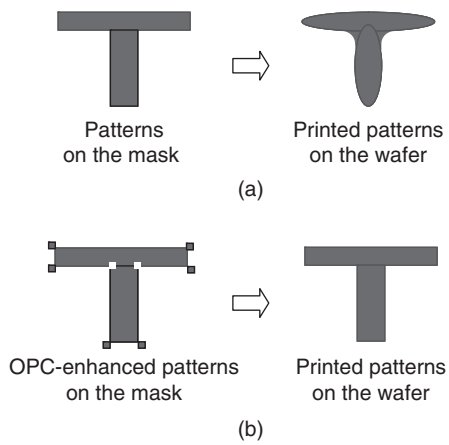

FIGURE 12.34

The effects of OPC: (a) Without OPC, the printed patterns on the wafer incur large distortions from the patterns on the mask. (b) With OPC enhancement, the printed patterns could well match the original patterns.

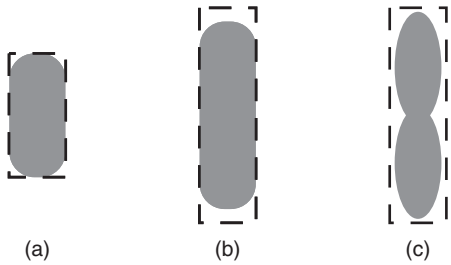

FIGURE 12.35

Three major types of pattern distortions (the dashed lines represent the ideal pattern shapes): (a) Corner rounding. (b) Line-end shortening. (c) Line-width shrinking.

For each type of distortion, the pattern features required for compensation are identified on the basis of some geometry rules, for example, the **serifs** added at corners to make the angles sharper, the **hammerheads** added at line ends to compensate for line-end shortenings, and the **line biasing** added along line sides to compensate for line-width shrinking (see Figure 12.36).

The number of pattern features required for OPC is then modeled as a cost for routing the connection. For example, as shown in Figure 12.36a, four serifs are required at the four corners to increase the fidelity of images for a line. Also, when the length of a line increases, the ends of the line become shortened, as illustrated in Figure 12.36b; therefore, two hammerheads are required at the line ends for a long line. Besides, a wider line is easier to be affected by neighboring lines than a narrower one, making the sides of a line shrink more seriously.

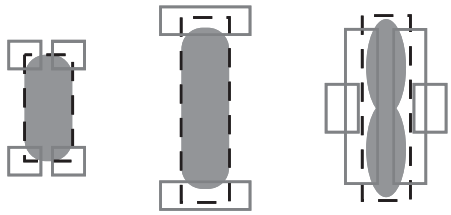

FIGURE 12.36

Three major OPC compensation pattern features: (a) Serif. (b) Hammerhead. (c) Line biasing.

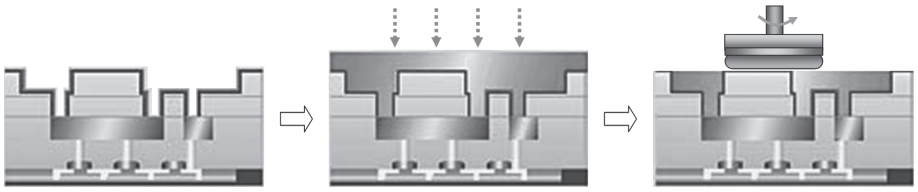

FIGURE 12.37

Damascene process: (a) Open trenches. (b) Electroplating (ECP) deposits Cu on the trenches. (c) Chemical-mechanical polishing (CMP) removes Cu that overfills the trenches.

Therefore, as shown in Figure 12.36c, some line biasing in the line sides is required for a wide line. Therefore, the total number of additional features for a line can be modeled as a function of the length and width of the line. With this function, we can incorporate the OPC cost into the original routability and wirelength costs for a router to obtain a rule-based OPC-aware routing method.

Chen, Liao, and Chang considered the OPC effects during routing to alleviate the cost of post-layout OPC operations [Chen 2008b]. They developed an analytical formula for the intensity computation from **model-based OPC** (which involves complicated simulations of various process effects) and a post-layout OPC modeling on the basis of an **inverse lithography technique**, and then incorporated the OPC costs into an OPC-friendly router. Huang *et al.* and Wu *et al.* also addressed OPC-friendly maze routing [Huang 2004b; Wu 2005b].

12.6.2.2 *CMP-aware routing*

In the modern metallization process, copper (Cu) has replaced the traditional aluminum (Al) because of its better properties, such as higher current-carrying capability, lower resistance, and lower cost. However, the process of copper is significantly different from that for traditional aluminum. The modern copper metallization process applies the **dual-Damascene process** [Luo 2005], which consists of ***electroplating*** (ECP) followed by the ***chemical-mechanical***

polishing (CMP). The ECP deposits the copper on the trenches, whereas the CMP removes the copper that overfills the trenches, as shown in Figures 12.37a–c.

Figure 12.38 shows a schematic diagram of the CMP process. Abrasive and corrosive chemical **slurry** that can dissolve the wafer layer is deposited on the surface of a polishing pad. Then, the polishing pad and wafer are pressed together by a dynamic, rotating polishing head. Combined with both the chemical reaction and the mechanical force, the CMP process can remove materials on the surface of the wafer and tends to make the wafer planar.

However, because of the difference in the hardness between copper and dielectric materials, the CMP planarizing process might generate topography *irregularities*, which might incur significant yield loss of copper interconnects. The studies of the CMP process have indicated that the post-CMP dielectric thickness is highly correlated to the layout pattern density, because during the polishing step, the dielectric removal rates are varied with the pattern density. A non-uniform feature density distribution on each layer might cause CMP to over polish or under polish, as illustrated in Figure 12.39.

These post-CMP thickness variations need to be carefully controlled, because the variation in one metal layer could be progressively transferred to subsequent layers during manufacturing, and finally the accumulative variation could be

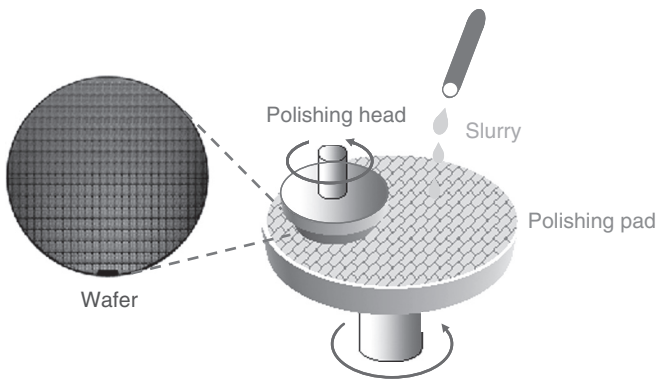

FIGURE 12.38

Schematic diagram of the CMP polisher.

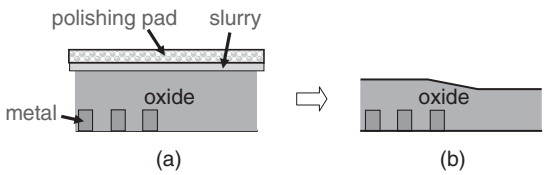

FIGURE 12.39

Layout-dependent thickness variations: (a) Pre-CMP layout. (b) Post-CMP thickness variation.

significant on the upper metal layer, which is often called the **multilayer accumulative effect** [Tian 2000].

To improve the CMP quality, modern foundries often impose recommended layout density rules (or even density gradient rules) on each layer and fill **dummy features** into layouts to reduce the variations on each layer. However, these filled dummy features might incur unwanted effects at 65nm and successive technology nodes [White 2005]. For example, they may induce high coupling capacitances to nearby interconnects and thus incur crosstalk problems. Moreover, dummy fills also significantly increase the data volume of mask, lengthening the time of the mask-making processes and thus the mask cost. Especially, these filled features would significantly increase the input data in the following time-consuming RETs, such as the OPC process.

Wire density greatly affects dummy feature filling. The layout pattern (consisting of wires and dummy features) density strongly depends on the wire density distribution, as reported in [Cho 2006]. Therefore, controlling wire density at the routing stage can alleviate the problems induced by aggressive dummy feature filling. In addition, good wire distribution can reduce the random particle short defects and also benefit the post-layout redundant-via insertion (see Section 12.6.3.2), which can translate into yield gain.

The density uniformity in different routing stages for CMP variation control has been addressed in the literature [Cho 2006; Chen 2007b; Li 2007]. Cho *et al.* considered CMP variation during global routing [Cho 2006]. They empirically showed that the number of inserted dummy features can be predicted by the wire density and observed that a path with higher pin density may not get much benefit from the wire density optimization, because there is little room for improvement (it is destined to have high wire density from the beginning). Therefore, they proposed a **minimum pin-density global-routing** algorithm to reduce the maximum wire density.

Figure 12.40 illustrates the minimum pin-density global-routing algorithm. A net from the source S to the target T to be routed is shown in

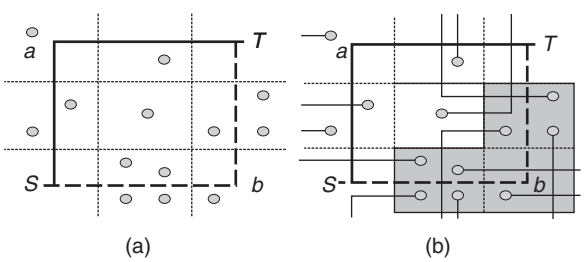

FIGURE 12.40

Minimum pin-density global routing [Cho 2006]: (a) There are two possible 1-bend paths *a* and *b* from the source *S* to the target *T*. (b) The path *a* with smaller pin density is better than the path *b*.

Figure 12.40a with a pin distribution. If only the *L*-shaped (1-bend) routing paths are allowed, there are two possible paths, *a* and *b*, with the same wirelength, but different pin densities. Because the existence of a pin implies at least one connection to other pins, a path with higher pin density like *b* would tend to have higher wire density eventually as shown in Figure 12.40b, resulting in higher final wire densities. Therefore, a path with the minimum pin density (like path *a*) leads to better wire density distribution.

Figure 12.41 shows the two-pass, top-down planarization-driven routing framework presented in [Chen 2007b], which consists of four major stages: (1) Prerouting: identify the potential density hot spots on the basis of the pin distribution and wire connection to guide the following global routing; (2) Global routing: apply prerouting-guided planarization-aware global pattern routing for nets and iteratively refine the solution; (3) Layer/track assignment: perform density-driven layer/track assignment for long segments panel by panel; and (4) Detailed routing: use segment-to-segment detailed maze routing to route short segments and reroute failed nets level by level. By handling longer nets first, the routing density for CMP can be better optimized, because the longer nets have higher density impact than the shorter ones.

In the prerouting stage, a density critical area analysis algorithm (on the basis of Voronoi diagrams [Preparata 1985]) is performed to identify the potential density hot spots. The identified density information of pins and wire connection is then used to guide the subsequent routing.

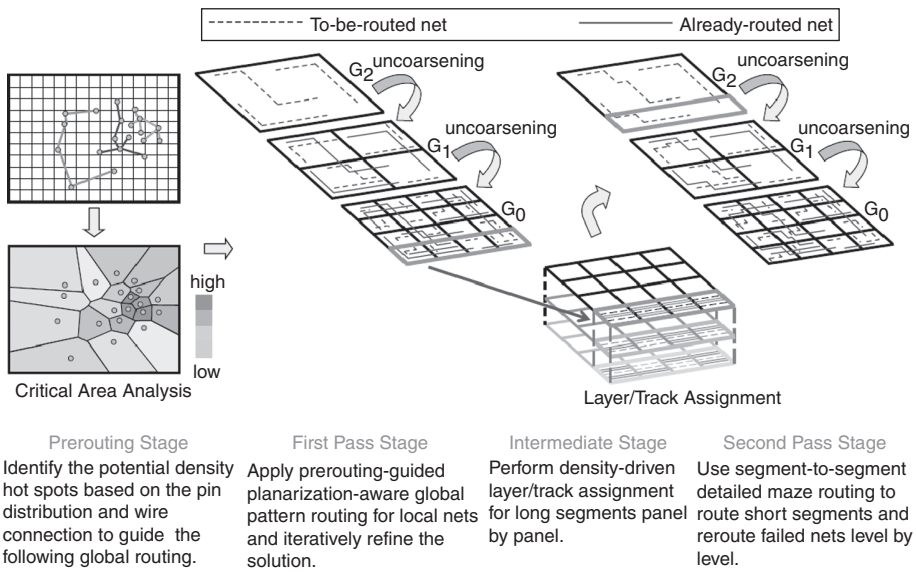

FIGURE 12.41

The two-pass, top-down planarization-driven routing framework.

In the first top-down (global-routing) stage, a planarization-aware global router is used to consider the density lower and upper bounds while minimizing the density gradient among global tiles. The planarization-aware cost Φ_t for each global tile t is defined as follows:

$$\Phi_t = \tilde{d}_t + \begin{cases} \kappa_p, & \text{if } d_t \geq B_u \\ \beta(2^{d_t} - 1) + (1-\beta)(d_t - \overline{d}_t)^2, & \text{if } B_l \leq d_t < B_u \\ \kappa_n, & \text{if } d_t < B_l \end{cases} \tag{12.7}$$

where d_t is the wire density of t, $\tilde{d}_t$ is the predicted hot spot cost calculated in the prerouting stage, $\overline{d}_t$ is the average wire density of tiles adjacent to t, B_l and B_u are respective density lower and upper bounds specified in foundry's density rules, and β, $0 \leq \beta \leq 1$, is a user-defined parameter. κ_p and κ_n are constants, where κ_p is a positive penalty that discourages routing through dense global tiles, and κ_n is a negative reward that encourages routing through sparse tiles. The second equation simultaneously considers the local tile density and minimizes the density gradient among adjacent regions.

The intermediate stage tries to preserve more flexibility for wire density arrangement. It consists of two phases: (1) a density-driven layer assigner evenly distributes the segments in a **panel** (row of global tiles) into layers, and (2) a density-driven track assigner balances the segment density of each track on the basis of incremental **Delaunay triangulation** (DT) [Preparata 1985]. First, the **flexibility** of a segment s_i is defined as follows:

$$\xi(s_i) = t_i + \frac{1}{\ell_i} \tag{12.8}$$

where t_i is the number of assignable tracks of s_i, and l_i is the length of s_i. If the flexibility of s_i is smaller, s_i might have a longer length or less space to insert and thus should be assigned first. Therefore, segments are inserted into tracks in the nondecreasing order of their flexibilities. Then, each segment or obstacle is represented by three points: its left-end, center, and right-end points, and then the resulting DT is analyzed. The segment is assigned to a track such that the resulting area difference among all triangles is minimized. Figure 12.42 shows a density-driven track-assignment example by inserting three segments s_1, s_2, and s_3 into tracks with obstacles O_1 (see Figure 12.42). Note that the *artificial segments* lying on the boundary are used to model the distribution of segments and obstacles in the neighborhood.

After the track assignment, the actual track position of a segment is known. Thus, classical segment-to-segment maze detailed routing is performed in the second top-down (detailed-routing) stage to connect shorter nets, and the whole routing process is finished.

12.6.3 **Routing for reliability**

Manufacturing reliability and yield in VLSI designs are becoming a crucial challenge as the feature sizes shrink into the nanometer scale. Both the antenna

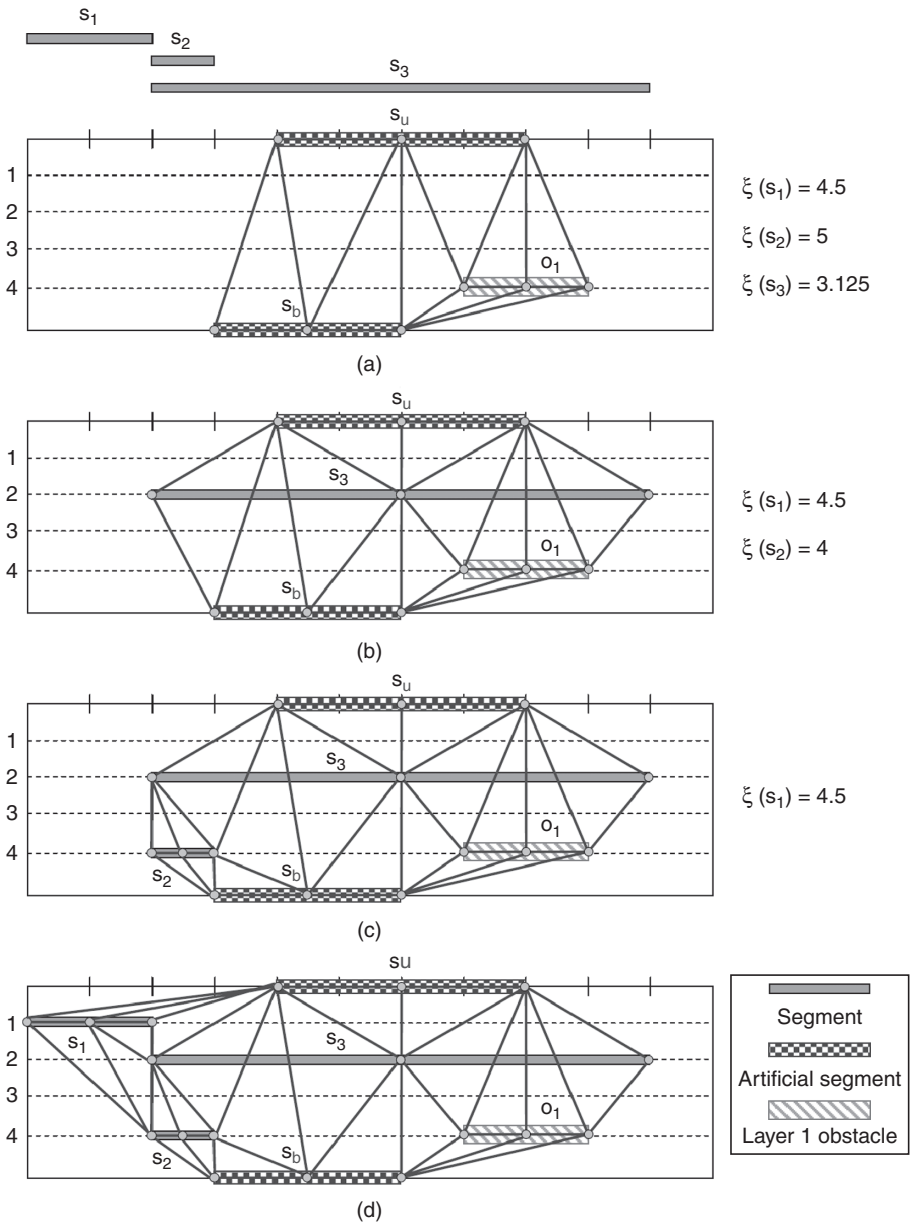

FIGURE 12.42

A density-driven track assignment example: (a) The initial Delaunay triangulation. (b) Track assignment for segment s_3. (c) Track assignment for segment s_2. (d) Track assignment for segment s_1.

effect arising in the plasma process and the via-open defect are important issues for achieving a higher reliability and yield.

12.6.3.1 *Antenna-avoidance routing*

The **antenna effect** is caused by the charges collected on the floating interconnects, which are connected to only a gate oxide. During the metallization, long floating interconnects act as temporary capacitors and store charges gained from the energy provided by fabrication steps such as plasma etching and CMP. If the collected charges exceed a threshold, the ***Fowler-Nordheim*** (F-N) tunneling current will discharge through the thin oxide and cause gate damage. On the other hand, if the collected charges can be released before exceeding the threshold through a low impedance path, such as diffusion, the gate damage can be avoided.

For example, considering the routing in Figure 12.43a, the interconnects are manufactured in the order of poly, metal 1, and metal 2. After manufacturing metal 1 (see Figure 12.43b), the collected charges on the right metal 1 pattern

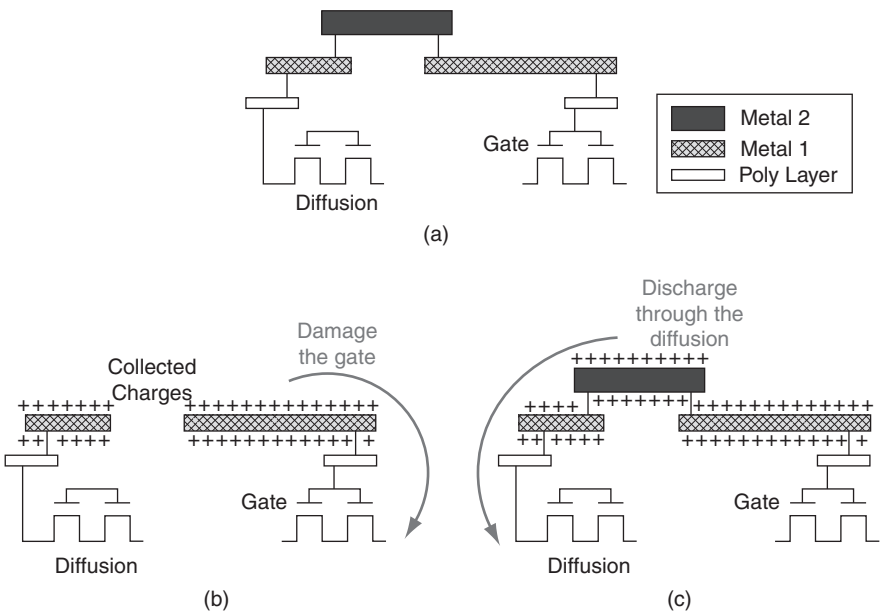

FIGURE 12.43

Illustration of the antenna effect: (a) A routing example. (b) Late stage of metal-1 pattern etching of (a), where the collected charges on the right side of the metal-1 pattern may cause damage to the connected gate oxide. (c) Late stage of metal-2 pattern etching of (a), where all the collected charges can be released through the connected diffusion on the left side.

may cause damage to the connected gate oxide. The discharging path is constructed after manufacturing metal 2 (see Figure 12.43c), and thus the charges can be released through the connected diffusion on the left side.

There are three kinds of solutions to reduce the antenna effect [Chen 2000]:

1. **Jumper insertion**: Break only signal wires with antenna violation and route to the highest level by jumper insertion. This reduces the charge amount for violated nets during manufacturing.
2. **Embedded protection diode**: Add protection diodes on every input port for every standard cell. Because these diodes are embedded and fixed, they consume unnecessary area when there is no violation at the connecting wire.
3. **Diode inserting after placement and routing**: Fix those wires with antenna violations that have enough room for "under-the-wire" diode insertion. During wafer manufacturing, all the inserted diodes are floating (or ground). One diode can be used to protect all input ports that are connected to the same output ports. However, this approach works only if there is enough room for diode insertion.

Jumper insertion is a popular way to solve the antenna problem. To avoid/fix the antenna violation, it is required that the total effective conductor connecting to a gate be less than or equal to a threshold, L_{max}. The threshold could be the wirelength limit, the wire area limit, the wire perimeter limit, the ratio of antenna strength (length, area, perimeter, etc.) to the gate size, or any model of the strength of antenna effect caused by conductors. As the example shown in Figure 12.44, we have a two-terminal net in which *a* is the source node and *b* is

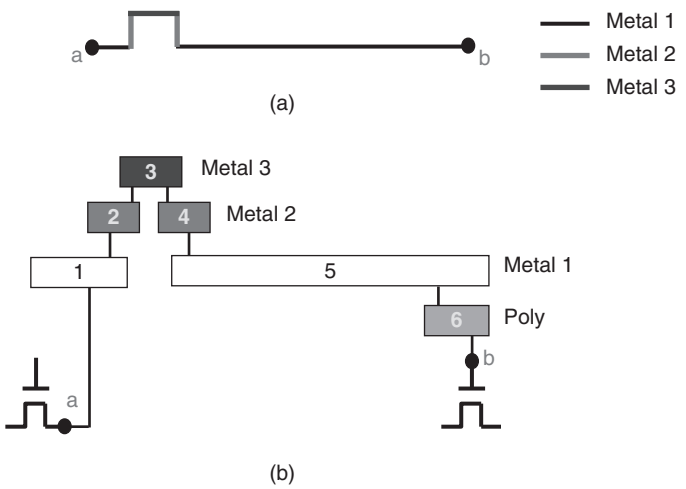

FIGURE 12.44

(a) A two-pin net. (b) The cross-sectional view.

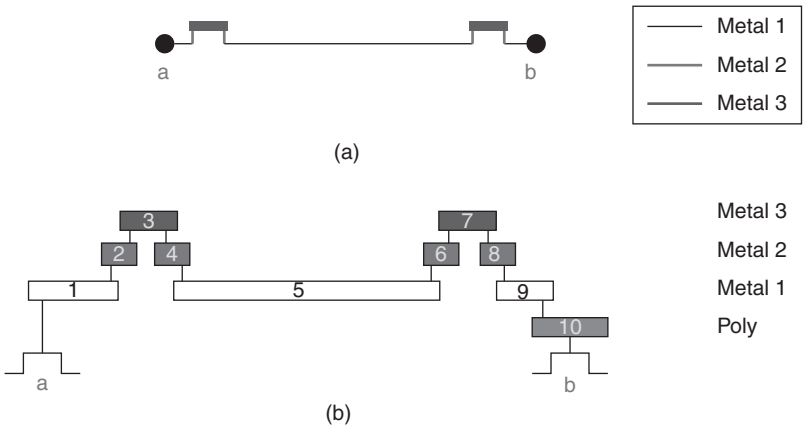

FIGURE 12.45

(a) A two-pin net with jumper insertion. (b) The cross-sectional view.

the terminal node. In this case, the antenna charge weight of b is the sum of the antenna charge weight of segments 4, 5, and 6, which may violate L_{max}. Note that once segment 3 is manufactured, a discharging path is established through segment 1 and the diffusion of the transistor a (see Figure 12.44b). If we add a jumper at the long segment 5 (see Figure 12.45), the antenna charge weight of b is just the sum of the length of segments 8, 9, and 10, which will not violate L_{max}. Thus, if we add jumpers appropriately, the antenna problem can be easily solved.

Antenna avoidance by jumper insertion has been extensively studied in the literature (*e.g.*, [Ho 2004, 2007; Wu 2005a; Su 2007]). Ho, Chang, and Chen proposed multilevel routing considering antenna effects by bottom-up jumper insertion [Ho 2004]. The work inserts jumpers only beside gate terminals, and its optimality of the use of the least jumpers to satisfy the antenna rule holds only for this special condition of inserting jumpers right beside gate terminals. Wu, Hu, and Mahapatra extended the work [Ho 2004] to handle the problem [Wu 2005a]. To fix the antenna violation of a gate terminal, the work first removes all subtrees around the node that violate the antenna rules. After all such subtrees are removed, if the sink still violates the antenna rule, the work will continually remove the heaviest branch from the sink until the antenna rules are satisfied. This approach still cannot guarantee optimal solutions under some special cases.

Su and Chang formulated the general jumper insertion for antenna avoidance (applicable at the routing stage) and/or fixing (applicable at the post-layout stage) as a **tree-cutting** problem on a routing tree and presented the first optimal algorithm for the general tree-cutting problem [Su 2007]. As usual, a net is modeled as a routing tree, where a node in the tree denotes a circuit terminal/junction (a gate, diffusion, or a junction of interconnects), and an edge denotes the interconnection between two circuit terminals or junctions. Because the

interconnection connecting to a diffusion terminal will not cause any antenna violation, the algorithm focuses on those connecting to gate terminals.

Let $L(u)$ denote the sum of edge weight (could be wirelengths, wire area, wire perimeter limit, the ratio of antenna strength, etc.) between the node u and all its neighbors. The problem of jumper insertion on a routing tree for antenna avoidance/fixing can be formulated as a tree-cutting problem as follows:

Jumper Insertion on a Routing Tree for Antenna Avoidance Problem: Given a routing tree $T = (V, E)$ and an upper bound L_{max}, find the minimum set C of cutting nodes, $e \neq u$ for any $c \in C$ and $u \in V$, so that $L(u) \leq L_{\max}$, $\forall u \in V$.

As the routing-tree example shows in Figure 12.46a, u_1 and u_2 are two sink nodes, the number beside each edge denotes the antenna charge weight, and L_{max} is assumed to be 10. For this case, three jumpers suffice to solve the antenna violations; see the jumpers c_1, c_2, and c_3 shown in Figure 12.46b.

The algorithm performs in a bottom-up manner by dealing with *leaf nodes* first followed by *sub-leaf nodes* of the tree. Here, a leaf node is a node with no children, whereas a sub-leaf node is a node for which all its children are leaf nodes, and if any of its children is a gate terminal, the edges between it and its children all have weights $\leq L_{max}$. Let $p(u)$ denote the parent node of node u, and $l(e)$ (or $l(u, v)$) be the antenna charge weight of the edge $e = (u, v)$ in the routing tree.

For a leaf node u, if $l(u, p(u)) \leq L_{max}$ or u is not a gate terminal, then u satisfies the antenna rule and thus it does not need to insert any cutting nodes. However, if $l(u, p(u)) > L_{max}$ and u is a gate, then $l(u, c) = L_{max}$ gives the best position for inserting the cutting node c, as illustrated in Figure 12.47. After adding jumper c, the edge $e(u, c)$ is cut from the tree.

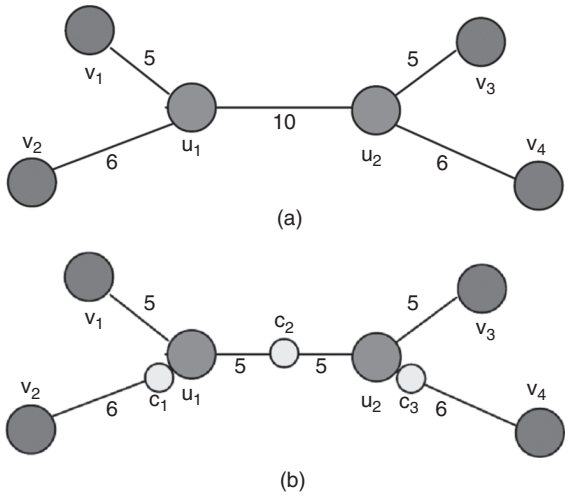

FIGURE 12.46

(a) A routing tree with two sink nodes u_1 and u_2. (b) Three jumpers c_1, c_2, c_3 are inserted to satisfy the antenna rule.

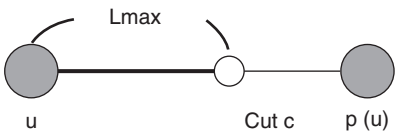

FIGURE 12.47

Optimal jumper insertion for a leaf node. The cutting node *c* is the optimal one among the nodes on edge *e*(u, p(u)).

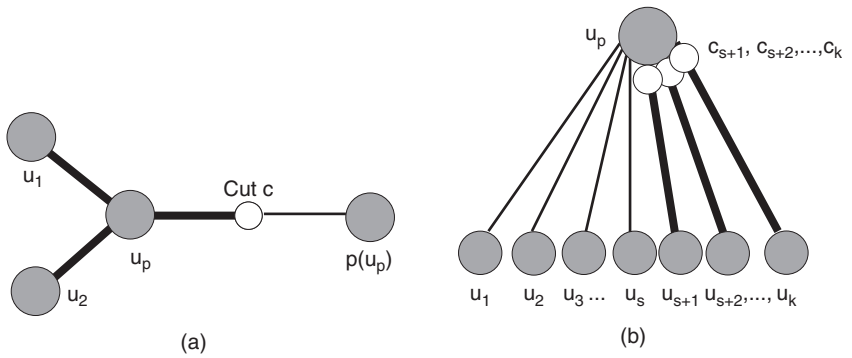

FIGURE 12.48

(a) Optimal jumper insertion for a sub-leaf node. (b) Illustration for the case of *total_len* $> L_{max}$.

For a sub-leaf node u_p and its children $u_i, \forall\, 1 \leq i \leq k$, let $total\_len = \sum_{i=1}^{k} l(u_i, u_p)$.

There are two cases:

Case 1: $total\_len \leq L_{max}$

If $(total\_len + l(u_p, p(u_p)) > L_{max}$, the cutting node c with $l(c, u_p) + total\_len = L_{max}$ gives the best position, as shown in Figure 12.48a. After adding c, all u_p's children from the original tree are cut from the tree.

Case 2: $total\_len > L_{max}$

First sort $l(u_p, u_i)\ \forall\, 1 \leq i \leq k$ in non-decreasing order and find the maximum s such that $\sum_{i=1}^{s} l(u_p, u_i) \leq L_{max}$. Then add the cutting nodes $c_{s+1}, \ldots, c_k$ as shown in Figure 12.48b.

For the embedded protection diode, Huang *et al.* solved the diode insertion and routing problem by a minimum-cost network-flow based algorithm, called ***Diode Insertion and Routing by Min-Cost Flow*** (DIRMCF) [Huang 2004a]. As shown in Figure 12.49, the antenna-violating wires, the routing grids, and the feasible diode positions are transformed into a flow network, and then the problem is solved by the minimum-cost network-flow algorithm. Both the positions of inserted diodes and the required routing can be determined through the resulting flow.

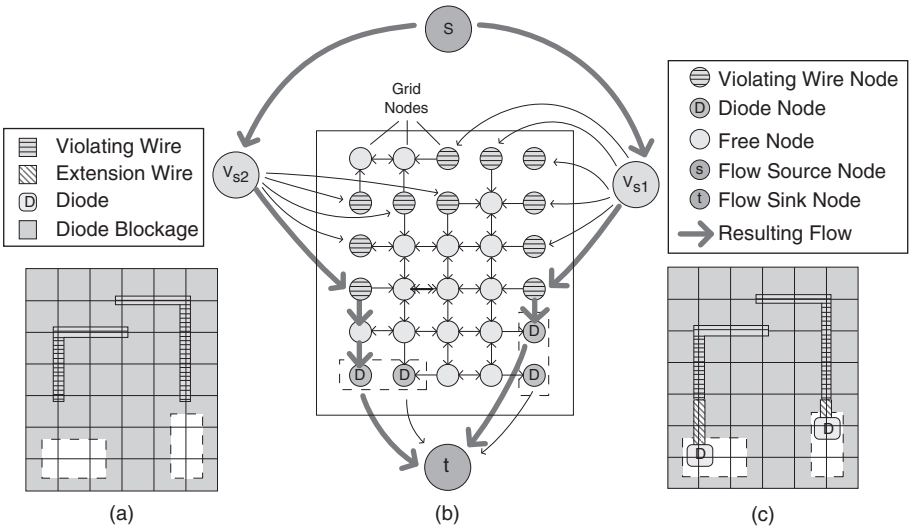

FIGURE 12.49

An example of the DIRMCF algorithm: (a) The violating wires and the routing grids. (b) The transformed flow network and the resulting flow after applying the minimum cost network-flow algorithm. (c) The inserted diodes and their corresponding routing.

Besides, because the vias of jumper insertion and the routing wires for diode insertion will both increase the driving load of the antenna violating wires (and thus the incurred *RC* delay will reduce the circuit performance), it is desirable to perform diode and jumper insertion simultaneously and consider the interaction between them to find a smaller performance degradation for the antenna fixing. Jiang and Chang [Jiang 2008] proposed a minimum-cost network-flow–based algorithm to solve the simultaneous diode/jumper insertion problem. The proposed algorithm first computes the jumper cost to fix each violating wire. Then it constructs the flow network in a similar way as the DIRMCF algorithm but integrates the jumper cost into the network. Finally, the antenna-fixed layout with the optimal fixing cost is found by applying the minimum-cost network-flow algorithm.

12.6.3.2 *Redundant-via aware routing*

In the nanometer technology, via-open defects are one of the important failures. A via may fail because of various reasons such as **random defects, electromigration, cut misalignment**, and/or **thermal stress**–induced voiding effects. The failure significantly reduces the manufacturing yield and chip performance.

To improve via reliability and yield, **redundant-via insertion** is a highly recommended technique proposed by foundries. If a via fails, a redundant via can serve as a fault-tolerant substitute for the failing one. As shown in Figure 12.50, a redundant

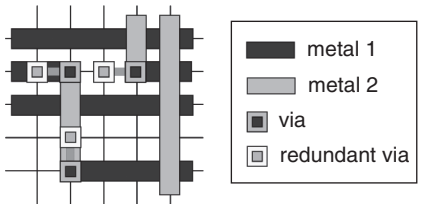

FIGURE 12.50

Double-via insertion. Each via is paired with a redundant via to form a double-via pair.

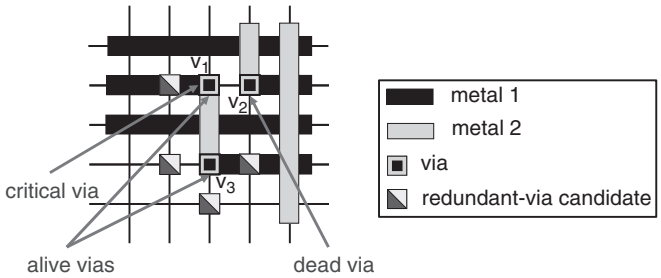

FIGURE 12.51

Illustration of redundant-via candidates, dead vias, alive vias, and critical vias. Vias v_1, v_2, and v_3 have one, zero, and three redundant-via candidates, respectively. Both v_1 and v_3 are alive vias, v_2 is a dead via, and v_1 is also called a critical via.

via can be inserted adjacent to each via to form a double-via pair. Double vias typically lead to 10 to 100 smaller failure rates than single vias.

The following gives some terminologies about vias. For a via, a **redundant-via candidate** is its adjacent position where a redundant via can be inserted. For the example shown in Figure 12.51, via v_1 has one redundant-via candidate on its left side, and via v_3 has three candidates around it. According to the number of redundant-via candidates, vias can be classified as **dead, alive**, or **critical vias**. If a via has at least one redundant-via candidate, it is an alive via; otherwise, it is called a dead via. Note that if an alive via has *exactly* one redundant-via candidate, it is also called a critical via. As shown in Figure 12.51, both vias v_1 and v_3 are alive vias, v_2 is a dead via, and v_1 is also a critical via.

Traditionally, redundant-via insertion is performed at the post-layout stage, which can be formulated as a ***maximum independent set*** (MIS) problem [Lee 2006], 0-1 integer linear programming (ILP) [Lee 2008], or maximum bipartite matching [Yao 2005; Chen 2008a]. However, it has been reported that if the router can minimize the number of dead and critical vias, the post-layout double-via insertion rate can be significantly improved. The reason is that the dead vias cannot be paired with redundant vias, and critical vias may not be paired because of the competition with other vias. For a routing instance from the source S to the target T shown in Figure 12.52a, an inferior routing path as

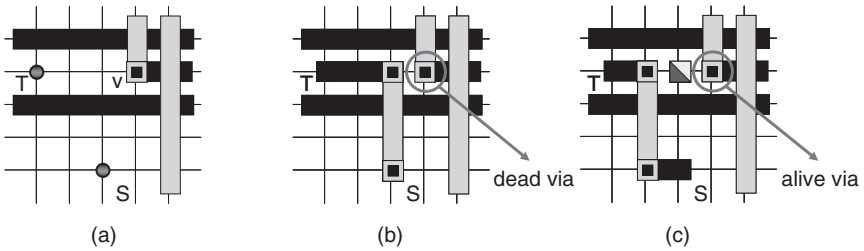

FIGURE 12.52

Redundant-via aware routing benefits the post-layout double-via insertion: (a) A detailed-routing instance for a 2-pin connection from the source *S* to the target *T*. (b) If an inferior routing path is selected, via *v* would become a dead via and cannot be paired. (c) For a better routing path, via *v* would remain alive for double-via insertion.

shown in Figure 12.52b would make via v a dead via and cannot be paired with any redundant vias. In contrast, for the better routing result as shown in Figure 12.52c, via v still remains alive for double-via insertion. Therefore, it is desirable to consider the redundant-via insertion at the routing stage to facilitate and preserve more flexibility for the post-layout double via insertion, as pointed out by [Xu 2005].

Chen *et al.* developed a redundant-via aware detailed-routing algorithm [Chen 2008a]. For each redundant-via candidate r_i of a via v, the *redundant-via cost* of r_i, cost(r_i), is set as

$$\text{cost}(r_i) = \frac{1}{DoF_v} \tag{12.9}$$

where DoF_v stands for the **degree of freedom** of v and equals the number of redundant-via candidates of v. The redundant-via penalty for a connection path p is calculated as the summation of the redundant-via costs of these redundant-via candidates on p.

Figure 12.53 illustrates the routing algorithm. Figure 12.53a shows a detailed-routing instance connected from the source *S* to the target *T*. The redundant-via costs of redundant-via candidates are shown in Figure 12.53b. The router can find a better routing path by choosing one with smaller redundant-via penalty, as shown in Figure 12.53c. Finally, the routing solution would be more redundant-via friendly as shown in Figure 12.53d, which contains more alive vias and preserves more redundant-via candidates to benefit the post-layout redundant-via insertion.

12.7 CONCLUDING REMARKS

Routing is one of the most fundamental steps in the physical design flow and is typically a very complex optimization problem. Effective and efficient routing

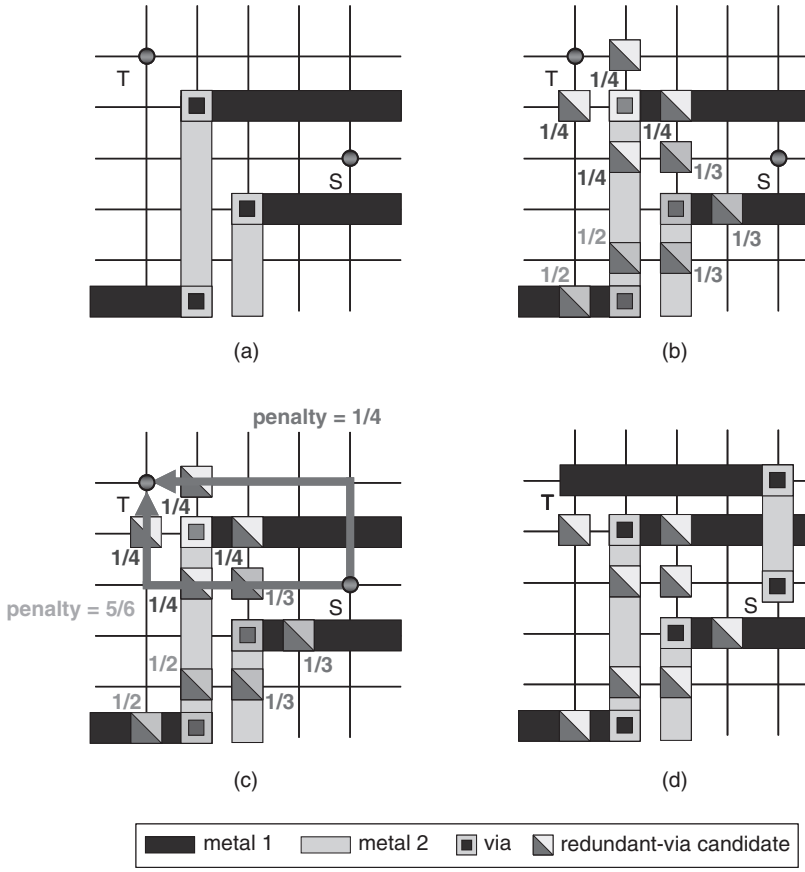

FIGURE 12.53

Redundant-via aware detailed routing: (a) A detailed-routing instance connected from the source *S* to the target *T*. (b) The redundant-via costs of redundant-via candidates. (c) The router can find a better routing path with smaller redundant-via penalty. (d) The routing solution would be more redundant-via friendly by preserving more redundant-via candidates.

algorithms are essential to handle the challenges arising from the fast growing scaling of IC integration. Traditionally, the routing problem is usually solved by a two-stage approach of global routing followed by detailed routing to tackle its high complexity.

In this chapter, we have first formulated the global and detailed routing as graph-search problems and examined the general-purpose routing algorithm, which includes the maze, line-search, and A*-search routing and can be applied to both global and detailed routing. Then we have discussed the global-routing algorithms, including sequential, concurrent, and tree-based approaches. For the detailed routing, we have covered channel routing and full-chip routing and

discussed the flat, hierarchical, and multilevel routing frameworks. Last, we have addressed routing for some important nanometer effects, including signal integrity, manufacturability, and reliability. As the technology nodes keep shrinking, all these effects should be considered in the earlier design stages. Considering the tradeoff between optimization flexibility and layout-information availability, routing seems to be the best stage to handle these effects.

"Old routers never die; they just fade away." With emerging design challenges (such as manufacturability, reliability, complexity, new chip architectures, and technologies), routers will keep evolving, with key techniques still remaining. It would be necessary to develop new data structures, algorithms, frameworks, and/or methods for the next-generation routers to handle the severe challenges yet to come.

12.8 EXERCISES

12.1. (General-Purpose Routing) Consider the chessboard shown in Figure 12.54. Some squares are shaded, denoting blockages. We intend to find a shortest path, if one exists, that starts at the square designated by *s*, after visiting the minimum number of squares, and ends at the square designated by *t*. The path must not pass through any shaded square. Formulate this problem as a graph-search routing problem and give an efficient algorithm to solve this problem. What is the time and space complexity of your algorithm?

12.2. (Concurrent Global Routing) You are asked to derive a routing algorithm for large-scale circuit designs on the basis of ***integer linear programming*** (ILP). ILP is typically very time-consuming for such large-scale designs. Instead of processing the whole routing region at one time, give at least two systematic approaches to divide

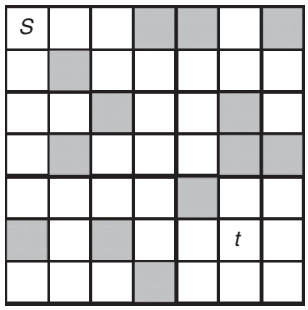

FIGURE 12.54

The graph-search problem of Exercise 12.1.

the routing region into subregions such that your algorithm can handle the routing problem subregion by subregion to reduce the problem size.

12.3. **(Routing Tree)** Given a net n with the four pins $p_1 = (6, 3)$, $p_2 = (3, 6)$, $p_3 = (1, 5)$, and $p_4 = (4, 2)$, let the estimated wirelengths by use of the ***minimum rectilinear spanning tree*** (MST) and the ***minimum rectilinear Steiner tree*** (MRST) be p and q, respectively. Find p and q.

12.4. **(Routing Tree)** Give an $O(E \lg V)$-time algorithm to find a minimum spanning tree T of an undirected graph $G = (V, E)$ so that the maximum edge weight of T is minimum over all spanning trees of G. Analyze the time complexity of your algorithm.

12.5. **(Line-Search Routing)** For the Hightower line-search router, there is no guarantee that we can find a path if such a path exists. Give an example routing configuration for this situation.

12.6. **(Channel Routing)** Given the channel-routing instance shown in Figure 12.55,

- **(a)** Draw the horizontal constraint graph (HCG) and the vertical constraint graph (VCG) for the given instance.
- **(b)** Determine a tight lower bound on the channel height from the HCG.
- **(c)** Route the instance by the dogleg channel routing algorithm. What is the final channel height?
- **(d)** Route the instance by the constrained left-edge channel routing algorithm. What is the final channel height?

12.7. **(Channel Routing)** Design an efficient algorithm to produce optimal routing solutions for 3-layer channel routing with the VHV model. (In the VHV model, the top and the bottom layers are reserved for vertical wires, and the middle layer is reserved for horizontal wires.)

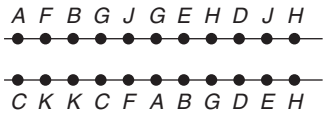

FIGURE 12.55

The channel routing instance of Exercise 12.6.

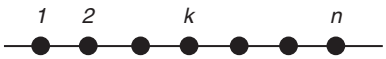

FIGURE 12.56

The routing instance of Exercise 12.8.

12.8. (Channel Routing) Label the terminals of the channel boundary $1, 2, \ldots, n$ in Figure 12.56, starting from the left and to the right. Let $N(i, j)$ denote the maximum number of nonintersecting connections between terminals i and j that can be routed on a single layer. Assume that there is a connection between terminals 1 and k. Give the recurrence $N(i, j)$ for the maximum number of nonintersecting connections in the layer in terms of the indices k and n. Apply dynamic programming to compute $N(1, n)$.

12.9. (Multilevel Routing) Given a netlist $N = \{[(1, 1), (2, 2)], [(2, 10), (2, 14)], [(6, 2), (10, 10)], [(6, 10), (10, 14)], [(10, 2), (14, 2)]\}$, where $[(p, q), (r, s)]$ denotes a route from the coordinate (p, q) to (r, s), you are asked to apply a 3-level routing (Λ-shaped multilevel routing with three levels) to route the instance N on a 16×16 chip plane. Suppose only straight and L-shaped routes are allowed during the coarsening stage, whereas maze routing is applied during uncoarsening. Also, all wire spacing (including point-to-wire spacing) must be at least 4 units. Show step by step how you obtain the routing solution.

12.10. (Maze Routing) Explain how you will extend the maze router for the X-architecture on which vertical, horizontal, 45°, and 135° routes are allowed for routing.

12.11. (Programming) This programming problem is modified from the 2007 ACM ISPD (*Int. Symp. on Physical Design*) Global Routing Contest [Nam 2007]. This programming assignment asks you to write a global router that can route 2-pin nets. To simplify the problem, we have some simplifications as follows:

1. Consider only two layers (layer 1 is for horizontal routes, and layer 2 is for vertical ones).
2. Consider only 2-pin nets.
3. Consider only tile-based coordinates. All lower left corners of the global routing regions are (0, 0). The tile width and height are ignored, because all X and Y are tile-based.
4. Consider only fixed wire width and spacing. All wire widths, wire, and via spacing are equal to 1.

(1) Input/output specification

Input format

The file format for the global routing contest is shown, with comments in italics (actual input files do not contain these comments). The example below gives an instance with two routing layers. The first line gives the problem size in terms of the number of horizontal and vertical tiles and the number of routing layers. Each global-routing tile (tile in short) has a capacity on each of its four boundaries to measure the available space.

The default capacity value of each layer is given in the second and third lines, which represents the maximum number of routing paths allowed to pass through a tile boundary. For example, the tile boundary with capacity 10 can accommodate up to 10 routing paths. The file format is as follows:

grid # # # *//number of horizontal tiles, vertical tiles, and layers*
vertical capacity # # *//vertical capacity by default on each layer*
horizontal capacity # # *//horizontal capacity by default on each layer*

num net # *//number of nets*
net_name net_id number_of_pins
x y layer
x y layer
. . .
[*repeat for the appropriate number of nets*]

Output format

All the routes in the output could only be horizontal lines, vertical lines, or via connections. For example (18, 61, 1)-(19, 62, 1) is not acceptable, because it is diagonal. All the nets are written in the output file in the same order as the input file. The output file format is shown as follows:

Net net_name net_id
([x11], [y11], [z11])-([x12], [y12], [z12])
([x21], [y21], [z21])-([x22], [y22], [z22])
. . .
!
[*repeat for the appropriate number of nets*]

(2) Problem statement

Given the problem size (the number of horizontal and vertical tiles and layers), horizontal and vertical capacities on each layers, and a netlist, the global router routes all nets in the routing region. The main objective is to minimize the total number of overflows, and the second objective is to minimize the total wirelength. Here the overflow on a tile boundary is calculated as the amount of demand that exceeds the given capacity (*i.e.*, *overflow* = max(*0, demand-capacity*)).

Following is an example of Input/Output files for Figure 12.57 with two routing layers.

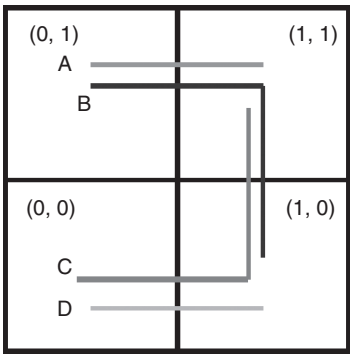

FIGURE 12.57

A routing problem and its solution.

Input file:

```
grid 2 2 2
vertical capacity 0 2
horizontal capacity 2 0

num net 4
A 0 2
0 1 1
1 1 1
B 1 2
0 1 1
1 0 1
C 2 2
0 0 1
1 1 1
D 3 2
0 0 1
1 0 1
```

Output file:

```
A 0
(0, 1, 1)-(1, 1, 1)
!
B 1
(0, 1, 1)-(1, 1, 1)
(1, 1, 1)-(1, 1, 2)
(1, 1, 2)-(1, 0, 2)
(1, 0, 2)-(1, 0, 1)
!
C 2
(0, 0, 1)-(1, 0, 1)
(1, 0, 1)-(1, 0, 2)
(1, 0, 2)-(1, 1, 2)
(1, 1, 2)-(1, 1, 1)
!
D 3
(0, 0, 1)-(1, 0, 1)
!
```

The total overflow is 0, and the total wirelength is 10.

ACKNOWLEDGMENTS

We thank Dr. Laung-Terng Wang of SynTest Technologies, Professor Cheng-Kok Koh of Purdue University, Professor Chris Chu of Iowa State University, Professor Ting-Chi Wang of National Tsing Hua University, Professor Hung-Ming Chen of National Chiao Tung University, the National Taiwan University students in the 2008 Physical Design class, and the EDA Laboratory for their very careful review of this chapter. We also thank the authors of [Nam 2007] for their help with the formulation of the programming assignment in Exercise 12.11, and Mr. Zhe-Wei Jiang of National Taiwan University for his help with Section 12.6.3.1.

REFERENCES

R12.0 Books

[Garey 1979] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness,* W. H. Freeman & Co., New York, 1979.

[Ho 2007] T.-Y. Ho, Y.-W. Chang, and S.-J. Chen, *Full-Chip Nanometer Routing Techniques,* Springer, Dordrecht, The Netherlands, 2007.

[Preparata 1985] F. P. Preparata and M. I. Shamos, *Computational Geometry: An Introduction,* Springer-Verlag, New York, 1985.

[Sait 1999] S. M. Sait and H. Youssef, VLSI *Physical Design Automation: Theory and Practice,* World Scientific, Singapore, 1999.

[Saxena 2007] P. Saxena, R. S. Shelar, and S. S. Sapatnekar, *Routing Congestion in VLSI Circuits: Estimation and Optimization,* Springer, New York, 2007.

R12.2 Problem Definition

[Chen 2007a] T.-C. Chen and Y.-W. Chang, Multilevel full-chip gridless routing with applications to optical proximity correction, *IEEE Trans. on Computer-Aided Design*, 26(6), pp. 1041–1053, June 2007.

[Cong 1999] J. Cong, J. Fang, and K.-Y. Khoo, An implicit connection graph maze routing algorithm for eco routing, in *Proc. IEEE/ACM Int. Conf. on Computer-Aided Design*, pp. 163–167, November 2005.

[Qusterhout 1984] J. K. Qusterhout, Corner stitching: A data structuring technique for VLSI layout tools, *IEEE Trans. on Computer-Aided Design*, 3(1), pp. 87–100, January 1984.

[Zheng 1996] S. Q. Zheng, J. S. Lim, and S. S. Iyengar, Finding obstacle avoiding shortest paths using implicit connection graphs, *IEEE Trans. on Computer-Aided Design*, 15(1), pp. 103–110, January 1996.

R12.3 General-Purpose Routing

[Akers 1967] S. B. Akers, A modification of Lee's path connection algorithm, *IEEE Trans. on Electronic Computers*, 16(1), pp. 97–98, February 1967.

[Clow 1984] G. W. Clow, A global routing algorithm for general cells, in *Proc. ACM/IEEE Design Automation Conf.*, pp. 45–51, June 1984.

[Hadlock 1977] F. O. Hadlock, A shortest path algorithm for grid graphs, *Networks*, 7(4), pp. 323–334, Winter, 1977.

[Hart 1968] P. E. Hart, N. J. Nilsson, and B. Raphael, A formal basis for the heuristic determination of minimum cost paths, *IEEE Trans. on Systems Science and Cybernetics*, 4(2), pp. 100–107, July 1968.

[Hightower 1969] D. Hightower, A solution to line routing problems on the continuous plane, in *Proc. ACM/IEEE Design Automation Conf.*, pp. 1–24, June 1969.

[Lee 1961] C. Y. Lee, An algorithm for path connection and its application, in *IRE Trans. on Electronic Computer*, 10, pp. 346–365, 1961.

[McMurchie 1995] L. McMurchie and C. Ebeling, PathFinder: A negotiation-based performance-driven router for FPGAs, *in Proc. Int. ACM Symp. on Field-Programmable Gate Arrays*, pp. 111–117, February 1995.

[Mikami 1968] K. Mikami and K. Tabuchi, A computer program for optimal routing of printed circuit connectors, in *Proc. Int. Federation for Information Processing*, pp. 1475–1478, November 1968.

[Soukup 1978] J. Soukup, Fast maze router, in *Proc. ACM/IEEE Design Automation Conf.*, pp. 100–102, June 1978.

R12.4 Global Routing

[Abel 1972] L. C. Abel, On the ordering of connections for automatic wire routing, *IEEE Trans. on Computers*, 21(11), pp. 1227–1233, November 1972.

[Chang 2008] Y.-J. Chang, Y.-T. Lee, and T.-C. Wang, A fast and stable global router, in *Proc. IEEE/ACM Int. Conf. on Computer-Aided Design*, November 2008.

[Chen 1999] H.-M. Chen, H. Zhou, F. Y. Young, D. F. Wong, H. H. Yang, and N. Sherwani, Integrated floorplanning and interconnect planning, in *Proc. IEEE/ACM Int. Conf. on computer-Aided Design*, pp. 354–357, November 1999.

[Cho 2007] M. Cho, K. Lu, K. Yuan, and D. Z. Pan, BoxRouter 2.0: Architecture and implementation of a hybrid and robust global router, in *Proc. ACM/IEEE Design Automation Conf.*, pp. 503–508, June 2007.

[Cho 2006] M. Cho, D. Z. Pan, H. Xiang, and R. Puri, Wire density driven global routing for CMP variation and timing, in *Proc. IEEE/ACM Int. Conf. on Computer-Aided Design*, pp. 487–492, November 2006.

[Chu 2004] C. Chu, FLUTE: fast lookup table based wirelength estimation technique, in *Proc. IEEE/ACM Int. Conf. on Computer-Aided Design*, pp. 696–701, November 2004.

[Garey 1977] M. R. Garey and D. S. Johnson, The rectilinear Steiner tree problem is NP-complete, *SIAM Journal Applied Mathematics*, 32(4), pp. 826–834, June 1977.

[Hanan 1966] M. Hanan, On Steiner's problem with rectilinear distance, *SIAM Journal on Applied Mathematics*, 14(2), pp. 255–265, March 1966.

[Ho 1990] J.-M. Ho, C. K. Vijayan, and C. K. Wong, New algorithms for the rectilinear Steiner tree problem, *IEEE Trans. on Computer-Aided Design*, 9(2), pp. 185–193, February 1990.

[Hsu 2008] C.-H. Hsu, H.-Y. Chen, and Y.-W. Chang, Multi-layer global routing considering via and wire capacities, in *Proc. IEEE/ACM Int. Conf. on Computer-Aided Design*, November 2008.

[Hwang 1976] F. K. Hwang, On Steiner minimal tree with rectilinear distance, *SIAM Journal on Applied Mathematics*, 30(1), pp. 104–114, January 1976.

[Kahng 1990] A. B. Kahng and G. Robins, A new class of Steiner tree heuristics with good performance: the iterated 1-Steiner approach, in *Proc. IEEE/ACM Int. Conf. on Computer-Aided Design*, pp. 428–431, November 1990.

[Kastner 2002] R. Kastner, E. Bozorgzadeh, and M. Sarrafzadeh, Pattern routing: use and theory for increasing predictability and avoiding coupling, *IEEE Trans. on Computer-Aided Design*, 21(7), pp. 777–790, November 2002.

[Kruskal 1956] J. B. Kruskal, On the shortest spanning subtree of a graph and the traveling salesman problem, in *Proc. the American Mathematical Society*, 7(1), pp. 48–50, February 1956.

[Lin 2008] C.-W. Lin, S.-Y. Chen, C.-F. Li, Y.-W. Chang, and C.-L. Yang, Obstacle-avoiding rectilinear Steiner tree construction based on spanning graphs, *IEEE Trans. Computer-Aided Design*, 27(4), pp. 643–653, April 2008.

[Lin 2007] C.-W. Lin, S.-L. Huang, K.-C. Hsu, M.-X. Lee, and Y.-W. Chang, Efficient multi-layer obstacle-avoiding rectilinear Steiner tree construction, in *Proc. IEEE/ACM Int. Conf. on Computer-Aided Design*, pp. 380–385, November 2007.

[McMurchie 1995] L. McMurchie and C. Ebeling, PathFinder: A negotiation-based performance-driven router for FPGAs, in *Proc. Int. ACM Symp. on Field-Programmable Gate Arrays*, pp. 111–117, February 1995.

[Pan 2007] M. Pan and C. N. Chu, FastRoute 2.0: A high-quality and efficient global router, in *Proc. IEEE/ACM Asian and South Pacific Design Automation Conf.*, pp. 250–255, January 2007.

[Prim 1957] R. C. Prim, Shortest connection networks and some generalizations, *Bell System Technical Journal*, 36, pp. 1389–1401, 1957.

[Raghavan 1987] P. Raghavan and C. D. Thompson, Randomized rounding: A technique for provably good algorithms and algorithmic proofs, in *Proc. Combinatorica*, pp. 365–374, December 1987.

[Roy 2007] J. A. Roy and I. L. Markov, High-performance routing at the nanometer scale, in *Proc. IEEE/ACM Int. Conf. on Computer-Aided Design*, pp. 496–502, November 2007.

[Shi 2006] Y. Shi, T. Jing, L. He, Z. Feng, and X. Hong, CDCTree: novel obstacle-avoiding routing tree construction based on current driven circuit model, in *Proc. IEEE/ACM Asia and South Pacific Design Automation Conf.*, pp. 630–635, January 2006.

[Zhou 2004] H. Zhou, Efficient Steiner tree construction based on spanning graphs, *IEEE Trans. on Computer-Aided Design*, 23(5), pp. 704–710, May 2004.

R12.5 Detailed Routing

[Burstein 1983] M. Burstein and R. Pelavin, Hierarchical wire routing, *IEEE Trans. on Computer-Aided Design*, 2(4), pp. 223–234, October 1983.

[Chang 2004] Y.-W. Chang and S.-P. Lin, MR: A new framework for multilevel full-chip routing, *IEEE Trans. on Computer-Aided Design*, 23(5), pp. 793–800, May 2004.

[Cong 2001] J. Cong, J. Fang, and Y. Zhang, Multilevel approach to full-chip gridless routing, in *Proc. IEEE/ACM Int. Conf. on Computer-Aided Design*, pp. 396–403, November 2001.

[Cong 2002] J. Cong, M. Xie, and Y. Zhang, An enhanced multilevel routing system, in *Proc. IEEE/ACM Int. Conf. on Computer-Aided Design*, pp. 51–58, November 2002.

[Deutsch 1976] D. N. Deutsch, A "dogleg" channel router, in *Proc. ACM/IEEE Design Automation Conf.*, pp. 425–433, June 1976.

[Hashimoto 1971] A. Hashimoto and J. Stevens, Wire routing by optimizing channel assignment within large apertures, in *Proc. ACM/IEEE Design Automation Conf.*, pp. 155–169, June 1971.

[Lin 2002] S.-P. Lin and Y.-W. Chang, A novel framework for multilevel routing considering routability and performance, *in Proc. IEEE/ACM Int. Conf. on Computer-Aided Design*, pp. 44–50, November 2002.

[Lin 1990] Y.-L. Lin, Y.-C. Hsu, and F.-S. Tsai, Hybrid routing, *IEEE Trans. on Computer-Aided Design*, 9(2), pp. 151–157, February 1990.

[Marek-Sadowska 1984] M. Marek-Sadowska, Global router for gate array, in *Proc. IEEE Int. Conf. on Computer Design*, pp. 332–337, October 1984.

[Szymanski 1985] T. G. Szymanski, Dogleg channel routing is NP-complete, *IEEE Trans. on Computer-Aided Design*, 4(1), pp. 31–41, January 1985.

R12.6 Modern Routing Considerations

[Chaudhary 1993] K. Chaudhary, A. Onozawa, and E. S. Kuh, A spacing algorithm for performance and crosstalk reduction, in *Proc. IEEE Int. Conf. on Computer-Aided Design*, pp. 697–702, November 1993.

[Chen 2008a] H.-Y. Chen, M.-F. Chiang, Y.-W. Chang, L. Chen, and B. Han, Full-chip routing considering double-via insertion, *IEEE Trans. on Computer-Aided Design*, 27(5), pp. 844–857, May 2008.

[Chen 2007a] T.-C. Chen and Y.-W. Chang, Multilevel full-chip gridless routing with applications to optical proximity correction, *IEEE Trans. on Computer-Aided Design*, 26(6), pp. 1041–1053, June 2007.

[Chen 2007b] H.-Y. Chen, S.-J. Chou, S.-L. Wang, and Y.-W. Chang, Novel wire density driven full-chip routing for CMP variation control, in *Proc. IEEE/ACM Int. Conf. on Computer-Aided Design*, pp. 831–838, November 2007.

[Chen 2008b] T.-C. Chen, G.-W. Liao, and Y.-W. Chang, Predictive formulae for OPC with applications to lithography-friendly routing, in *Proc. ACM/IEEE Design Automation Conf.*, pp. 510–515, June 2008.

[Chen 2000] P. H. Chen, S. Malkani, C.-M. Peng, and J. Lin, Fixing antenna problem by dynamic diode dropping and jumper insertion, in *Proc. IEEE Int. Symp. on Quality Electronic Design*, pp. 275–282, March 2000.

[Cho 2006] M. Cho, D. Z. Pan, H. Xiang, and R. Puri, Wire density driven global routing for CMP variation and timing, in *Proc. IEEE/ACM Int. Conf. on Computer-Aided Design*, pp. 487–492, November 2006.

[Cong 2001] J. Cong, D. Z. Pan, and P. V. Srinivas, Improved crosstalk modeling for noise constrained interconnect optimization, in *Proc. IEEE/ACM Asia and South Pacific Design Automation Conf.*, pp. 373–378, January 2001.

[Gao 1996] T. Gao and C.-L. Liu, Minimum crosstalk channel routing, *IEEE Trans. on Computer-Aided Design*, 15(5), pp. 465–474, May 1996.

[Ho 2004] T.-Y. Ho, Y.-W. Chang, and S.-J. Chen, Multilevel routing with antenna avoidance, in *Proc. ACM Int. Symp. on Physical Design*, pp. 34–40, April 2004.

[Ho 2005] T.-Y. Ho, Y.-W. Chang, S.-J. Chen, and D.-T. Lee, Crosstalk- and performance-driven multilevel full-chip routing, *IEEE Trans. on Computer-Aided Design*, 24(6), pp. 869–878, June 2005.

[Huang 2004a] L.-D. Huang, X. Tang, H. Xiang, D. F. Wong, and I.-M. Liu, A polynomial time-optimal diode insertion/routing algorithm for fixing antenna problem, *IEEE Trans. on Computer-Aided Design*, 23(1), pp. 141–147, January 2004.

[Huang 2004b] L.-D. Huang and D. F. Wong, Optical proximity correction (OPC)-friendly maze routing, in *Proc. ACM/IEEE Design Automation Conf.*, pp. 186–191, June 2004.

[Jiang 2008] Z.-W. Jiang and Y.-W. Chang, An optimal network-flow-based simultaneous diode and jumper insertion algorithm for antenna fixing, *IEEE Trans. on Computer-Aided Design*, 27(6), pp. 1055–1065, June 2008.

[Jiang 2000] I. H.-R. Jiang, Y.-W. Chang, and J.-Y. Jou, Crosstalk-driven interconnect optimization by simultaneous gate and wire sizing, *IEEE Trans. on Computer-Aided Design*, 19(9), pp. 999–1010, September 2000.

[Lee 2008] K.-Y. Lee, C.-K. Koh, T.-C. Wang, and K.-Y. Chao, Optimal post-routing redundant via insertion, in *Proc. ACM Int. Symp. on Physical Design*, pp. 111–117, April 2008.

[Lee 2006] K.-Y. Lee and T.-C. Wang, Post-routing redundant via insertion for yield/reliability improvement, in *Proc. ACM/IEEE Asia and South Pacific Design Automation Conf.*, pp. 303–308, January 2006.

[Li 2007] K. S.-M. Li, C.-L. Lee, Y.-W. Chang, C.-C. Su, and J. E. Chen, Multilevel full-chip routing with testability and yield enhancement, *IEEE Trans. on Computer-Aided Design*, 26(9), pp. 1625–1636, September 2007.

[Luo 2005] J. Luo, Q. Su, C. Chiang, and J. Kawa, A layout dependent full-chip copper electroplating topography model, in *Proc. IEEE/ACM Int. Conf. on Computer-Aided Design*, pp. 133–140, November 2005.

[Sakurai 1983] T. Sakurai and K. Tamaru, Simple formulas for two and three dimensional capacitance, *IEEE Trans. on Electronic Devices*, 30(2), pp. 183–185, February 1983.

[Su 2007] B.-Y. Su, Y.-W. Chang, and J. Hu, An optimal jumper insertion algorithm for antenna avoidance/fixing, *IEEE Trans. on Computer-Aided Design*, 26(10), pp. 1818–1929, October 2007.

[Tian 2000] R. Tian, D. F. Wong, and R. Boone, Model-based dummy feature placement for oxide chemical-mechanical polishing manufacturability, in *Proc. ACM/IEEE Design Automation Conf.*, pp. 667–670, June 2000.

[Vittal 1999] A. Vittal, L. H. Chen, M. Marek-Sadowska, K.-P. Wang, and S. Yang, Crosstalk in VLSI interconnections, *IEEE Trans. on Computer-Aided Design*, 18(12), pp. 1817–1824, December 1999.

[White 2005] D. White and B. Moore, An 'intelligent' approach to dummy fill, in *EE Times*, January 3, 2005.

[Wu 2005a] D. Wu, J. Hu, and R. Mahapatra, Coupling aware timing optimization and antenna avoidance in layer assignment, in *Proc. ACM. Int. Symp. on Physical Design*, pp. 20–27, April 2005.

[Wu 2005b] Y.-R. Wu, M.-C. Tsai, and T.-C. Wang, Maze routing with OPC consideration, in *Proc. ACM/IEEE Asia and South Pacific Design Automation Conf.*, pp. 198–203, January 2005.

[Xu 2005] G. Xu, L.-D. Huang, D. Z. Pan, and F. D. Wong, Redundant-via enhanced maze routing for yield improvement, in *Proc. ACM/IEEE Asia and South Pacific Design Automation Conf.*, pp. 1148–1151, January 2005.

[Yao 2005] H. Yao, Y. Cai, X. Hong, and Q. Zhou, Improved multilevel routing with redundant via placement for yield and reliability, in *Proc. ACM Great Lakes Symp. on VLSI*, pp. 143-146, April 2005.

[Zhou 1998] H. Zhou and D. F. Wong, Global routing with crosstalk constraints, in *Proc. ACM/IEEE Design Automation Conf.*, pp. 374–377, June 1998.

R12.8 Exercises

[Nam 2007] G.-J. Nam, M. Yildiz, D. Z. Pan, and P. H. Madden, ISPD placement contest updates and ISPD 2007 global routing contest, in *Proc. ACM Int. Symp. on Physical Design*, p. 167, April 2007.